SECTest: An Integrated Testing Platform for QoS in Satellite Edge Clouds

Guogen Zeng, Juan Luo, *Member, IEEE*, Yufeng Zhang, Ying Qiao, Shuyang Teng, and Keqin Li, *Fellow, IEEE*

***Abstract*—With the advancement of satellite computing capabilities, the diversity of satellite communication services imposes varied quality of service (QoS) requirements. Limited satellite resources necessitate remote deployment and updates of running services for QoS testing, increasing testing difficulty. Existing testing tools are limited in functionality or reliant on specific infrastructures, failing to meet the QoS testing needs of edge cloud services in mobile satellite scenarios. In this paper, we present SECTest, an integrated testing platform for QoS in satellite edge clouds. More precisely, SECTest can integrate changes in satellite network topology, create and manage satellite edge cloud cluster testing environments on heterogeneous edge devices, customize experiments for users, support deployment and scaling of various integrated testing tools, provide test data persistence function to manage data life cycle and store data hierarchically, and publish and visualize test results. We have built a real satellite edge cloud cluster based on Kubernetes, integrating both physical and virtual machines, and deploying a variety of integrated testing tools using containerization technology. Currently, we have evaluated the quality of service in terms of processing latency, packet drop rate, throughput, and average response time for object detection microservice applications, web microservice applications, and data transfer tasks. To demonstrate SECTest's scalability in testing network communication protocols, we evaluated the performance of HTTP and gRPC in microservice communication within the cluster. Our experimental results validate SECTest's ability to test key service quality metrics in a real satellite edge cloud cluster.**

***Index Terms*—Satellite edge clouds, integration testing, quality of service, microservices, testing platform.**

I. Introduction

As THE demand for satellite computing services continues to grow, satellites are poised to offer a wider range of service types than ever before. Multiple satellites are interconnected through various communication devices to form satellite clusters, enabling the provision of diverse services [2]. Different services will have varying quality of service (QoS) requirements. With advancements in satellite hardware performance, cloud computing, and virtualization technologies have been widely adopted in satellite systems. The edge cloud systems designed using cloud-native technologies are widely deployed on satellites to manage various microservice applications [3]. However, a prevailing issue is that a large number of users making repeated service requests can often lead to network congestion and performance degradation [4]. Additionally, devices providing software services within satellite clusters may face resource constraints or be affected by high-speed mobility, potentially resulting in QoS degradation or unresponsive services [5]. Therefore, effective QoS testing is critical for request-response services within a satellite edge cloud that require fast processing and have specific latency requirements. Additionally, bidirectional data streaming services, which are often constrained by packet loss and limited bandwidth, also benefit from such testing. Ensuring easy and effective QoS tests before or during service deployment is essential for optimal performance. These tests ensure the reliability and performance of the service under varying conditions, thereby improving the service's resilience and stability in the satellite edge cloud environment.

Cloud-native satellites can already provide various types of services to ground users, such as satellite-based smart city applications, emergency communications, and disaster detection [6]. When satellite edge cloud clusters offer computing services to the ground, unlike coordinated computing with ground networks, satellite networks are significantly affected by external environmental factors [7]. This is intolerable for computing services with high QoS stability requirements. Optimizing only satellite networks to ensure QoS requirements cannot satisfy the need for rapid detection and verification of actual QoS conditions [8]. Therefore, a targeted QoS testing platform is still necessary in the context of satellite edge cloud clusters.

Testing the interactions between modules in microservice applications in a satellite edge cloud cannot be accomplished by isolating the service to specific modules in the cluster alone [9]. Therefore, testing the interactions of microservice applications between nodes can only be accomplished by running end-to-end tests or extensive integration tests, which require deploying microservice applications across the cluster [10]. Performing

Received 19 July 2024; revised 5 March 2025; accepted 13 April 2025. Date of publication 9 May 2025; date of current version 12 June 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62372163, and in part by the Science and Technology Innovation Program of Hunan Province under Grant 2024RC1033. An earlier version of this paper was presented in part at the 21st IEEE International Conference on Web Services (ICWS 2024) [DOI: 10.1109/ICWS62655.2024.00132]. *(Corresponding author: Juan Luo.)*

Guogen Zeng, Juan Luo, Yufeng Zhang, Ying Qiao, and Shuyang Teng are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan 410012, China (e-mail: zengguogen@hnu.edu.cn; juanluo@hnu.edu.cn; yufengzhang@hnu.edu.cn; qy2020@hnu.edu.cn; S2110Z0004@hnu.edu.cn).

Keqin Li is with the College of Computer Science and Electronic Engineering, Hunan University, Hunan 410082, China, also with the National Supercomputing Center in Changsha, Hunan 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TSC.2025.3565368

end-to-end testing on satellites is exceedingly complex, time-consuming and costly. There are two ways to perform extensive integration testing of microservice applications in a satellite edge cloud: inside or outside the satellite cluster. However, extensive integration testing of microservice applications from outside the satellite cluster has unavoidable drawbacks that make it as inefficient as end-to-end testing [11], [12]. For example, most satellite services lack public interfaces that allow access from outside the cluster. Therefore, the standard approach to extensive integration testing of microservices on satellites is to test them from within the cluster. For integration testing of microservices from within the satellite edge cloud cluster, certain limitations exist with some testing tools. Some testing tools lack key functionalities for testing service quality, or they are challenging to deploy in the cluster and unable to accurately assess the test content. For instance, Iperf3 primarily focuses on network-layer performance testing, while microservices often involve business logic and data processing, thus Iperf3 may not directly test the business logic performance or data processing capabilities of microservices [13]. Furthermore, Jmeter and LoadRunner have difficulty in effectively scheduling and allocating cluster resources on edge cloud clusters, which may lead to imbalance resource allocation or waste during testing, thereby affecting the accuracy and effectiveness of test results [14]. Complex and difficult deployment of Jmeter and LoadRunner on edge cloud clusters is also one of the problems [15]. To this end, in view of satellite edge cloud service quality testing and the limitations of existing testing tools, we present a new service quality integrated testing platform suitable for edge cloud microservice applications in satellite scenarios.

This paper is an extended version of a conference paper [1]. We have evolved the testing framework into a comprehensive testing platform. This platform retains the core functionalities of the initial framework while introducing enhancements to improve scalability, usability, and performance. In this paper, our main contributions are as follows:

- We have implemented the SECTest,a testing platform for service quality integration testing on satellite edge clouds.
- SECTest possesses the capability to integrate inter-satellite network topology changes, it also can create and manage satellite edge cloud cluster testing environments on heterogeneous edge devices.
- SECTest can integrate and expand multiple testing tools into a unified platform, supporting customizable input parameters for testing and publishing test results.
- Tested within a real satellite edge cloud system, SECTest evaluates critical service quality metrics in terms of processing delay, packet drop rate, throughput, and average response time, demonstrating the feasibility and effectiveness of the SECTest.
- Finally, we conducted a functional comparison between SECTest and the latest related research efforts, highlighting SECTest's advantages over existing satellite edge cloud testing platforms.

The rest of this paper is organized as follows. Section II provides a detailed overview of the SECTest platform design, testing process, and how testing tools are integrated and deployed within SECTest. Section III describes the selected microservice applications for testing, the testing methods employed, and the QoS evaluation metrics. Section IV elaborates on the experimental setup, results, and analysis. Section V reviews related work in the field, comparing SECTest with the latest research efforts and highlighting its advantages. Section VI addresses the limitations of SECTest and explores future research directions. Finally, Section VI concludes the paper.

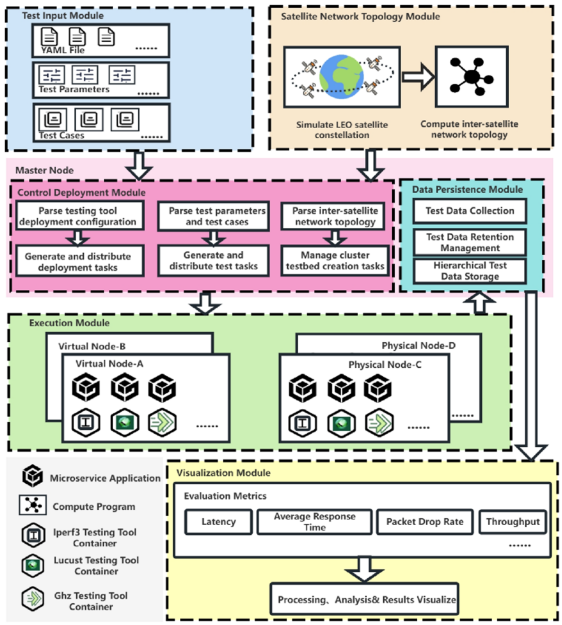


Fig. 1. The overview and workflow of SECTest.

II. SECTEST

A. The Platform SECTEST

Fig. 1 provides a detailed illustration of the platform and interaction process within SECTest. SECTest is divided into six main modules: test input module, satellite network topology module, control deployment module, execution module, data persistence module and visualization module.

Test Input Module: This module is equipped with functionalities for user interaction, responsible for customizing test parameters and test case inputs, as well as uploading YAML configuration files for deploying testing tools. The types of test parameters supported by this module are determined by the testing content and the testing tools integrated within SECTest. Currently, the test input module of SECTest supports various parameter types, including text parameters, numerical parameters, and file parameters, as well as custom parameter. This design emphasizes enabling efficient configuration and execution of tests while maintaining compatibility with testing tools and methodologies.

Satellite Network Topology Module: Within a specific timeframe set in Satellite Tool Kit (STK), the STK are utilized to simulate changes in the Low-Earth-Orbit (LEO) satellite network topology and the trajectories of satellite operations,

thereby obtaining data on inter-satellite distances and communication ranges. Subsequently, we have developed a program to retrieve and process the simulated data from STK, calculating the network topology of the satellites within the defined timeframe. This data is then passed to the control deployment module for managing the number of edge cloud clusters.

Control Deployment Module: The test cluster generated by SECTest follows a Master -Worker architecture. And the control deployment module is positioned on the master node, which primarily serves the following functions: 1) It is responsible for the integrated deployment and configuration of test tools, parsing them into specific deployment tasks, and distributing them to each worker node in the execution module. 2) It is capable of parsing test parameters and test cases, generating test tasks, and uploading them to the test tool container of the worker node for execution. 3) The control deployment module parses the acquired satellite network topology and generates a satellite edge cloud test cluster. Additionally, it manages the scope of the satellite edge cloud cluster and performs tasks such as adding or removing nodes in the cluster. It's important to note that the generated test cluster nodes encompass both physical servers and virtual servers. A more detailed description of the test tool integration deployment process is provided in Section II-C.

Execution Module: Worker nodes receive tasks to create cluster test environments. The test cluster comprises virtual and physical computing nodes, with each node capable of containing multiple built test tools and microservice applications. At the execution module, edge devices are designated as worker nodes to execute test tasks. Due to Kubernetes' potential to better integrate distributed edge devices with the cloud, we have chosen to use it to manage and execute the deployment tasks of microservice applications on the edge nodes [16]. SECTest does not concern itself with the basic settings and deployment of microservice applications within the satellite edge cloud. Moreover, the execution module is responsible for the construction and deployment of the test tool container. Finally, upon completion of the testing task, the test results are returned to the visualization module.

Data Persistence Module: Its primary functions include test data collection, lifecycle management, and tiered storage. A NoSQL database is deployed on the master node to support data persistence. Each worker node executing test tasks generates test data, which is then collected by the master node and stored in the database. However, due to limited storage resources on satellites, the data persistence module provides support for managing the lifecycle of test data and implements tiered storage based on data access frequency. This tiered storage mechanism effectively optimizes the use of storage resources, ensuring that frequently accessed data can be retrieved quickly, while infrequently accessed data is stored in lower-priority tiers, thereby improving the overall efficiency of accessing test data.

Visualization Module: Responsible for retrieving and processing test data from each running test container, this module can publish test results on the SECTest frontend page, presenting processed results intuitively to the user. To broaden the capabilities of the visualization module, we have reserved a common interface in the visualization module of SECTest to support users' custom development. This interface is designed for acquiring and processing diverse types of test data, enabling tailored display of test results according to individual requirements.

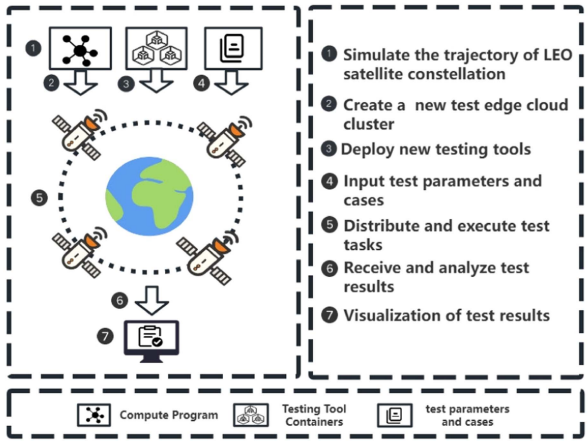


Fig. 2. Test process of SECTest.

B. Testing Process of SECTest

The key process of our integrated testing on the satellite edge cloud cluster is illustrated in Fig. 2. In the initial stage, for a completely new test, users simulate the operation trajectories of satellite clusters over a certain period using STK. The satellite network topology module then computes the network topology of the satellite constellation based on this simulation. Subsequently, the computed results are transmitted to the control deployment module. Upon receiving and parsing the data, the control deployment module distributes tasks to build the test cluster to the execution module. At the execution module, the Worker nodes execute the construction tasks and generate an edge cloud test cluster. If there is a change in the network topology of the satellite cluster, the control deployment module dynamically adjusts the cluster size accordingly. Otherwise, this step is not necessary when building upon an existing edge cloud infrastructure. Similarly, on the testing input module, since SECTest supports the expansion of testing tools, users can decide whether to deploy new testing tools at the start of the test. When users configure test parameters and test cases in the test input module, the data will be transmitted to the control deployment module. The control deployment module will parse the test parameters and allocate test tasks to the test tool containers on the worker nodes in the cluster, which will then execute the test tasks. Finally, the visualization module collects test results from each testing node, processes and analyzes them, and then publishes the results.

C. Integrated Deployment Testing Tools

One of the key features of SECTest is the ability for users to easily define performance testing tasks and deploy testing tools in configuration files. SECTest utilizes Dockerfiles to build testing tool images. These images are then uploaded to an image repository, from which each edge cloud node pulls the

image and creates testing tool containers. SECTest allows for the deployment of containers for different testing tools into the same namespace as the pods running the microservices under test. Additionally, it supports the deployment of testing tool containers in any Kubernetes namespace without affecting their functionality. Integration tasks for testing tools are specified in YAML files, and the choice of YAML is driven by two reasons: 1) Due to its simple format, YAML does not require additional complex configuration files, and deployment tasks can be completed solely through YAML configuration. 2) Users can embed test code and commands directly into YAML files.

III. Methodology

A. SECTest Applications and Task

We have selected three classic satellite service applications: object detection applications, web microservice applications, and inter-satellite data transmission tasks. With the enhancement of satellite high-resolution camera performance, image processing and classification in-space have become quite prevalent [17]. Additionally, some satellite edge computing frameworks and edge cloud platforms are opening up to more service applications. Currently, these applications are typically executed in the cloud. As a result, it is of great significance to study the service quality of such applications.

Web Microservice Application: We developed two simple Web microservice applications with identical functionalities using Python and Golang. These applications were deployed to each edge cloud node in the test cluster by building images and creating containers. In this context, we used Python to simulate the POST and GET request methods in the HTTP protocol, and Golang to simulate the POST and GET request methods in the gRPC protocol. Locust was utilized to simulate multiple users executing numerous service requests on the web microservice applications within the cluster, subsequently distributing the request load to the microservice applications on other nodes.

Object Detection Application: In the training phase, we use three object detection models, YOLOv5 [18], Faster-RCNN [19] and RetinaNet [20], and train them using the DOTA dataset [21]. The trained model is then built into an image and created as a container, which is deployed to the edge cloud cluster through a YAML file. From the input module of SECTest, users input image test cases. The deployment control module then uploads the object detection tasks to the satellite edge cloud, distributing the testing tasks to each edge cloud node for execution.

Data Transmission Task: Given the inter-satellite communication transmission task, a satellite can potentially communicate with multiple satellites within any given period. Multiple satellites can collaborate to perform data transmission tasks. During actual inter-satellite data transmission, additional communication overhead may also occur. Consequently, based on the simulated satellite network topology, we select satellite nodes that can communicate within a certain period of time to execute the data transmission task.

SECTest is compatible with the diversity of microservice application types to be tested, supporting users to develop microservice applications using any programming language. Nevertheless, to complete the testing of microservice applications on SECTest, certain prerequisites need to be met. Customized microservices must be successfully deployed to edge devices, ensuring that their microservice applications are compatible with the hardware and software environment of the edge devices, and can start and run normally. Only when these prerequisites are met can the true performance and behavior of microservice applications be demonstrated through testing on SECTest.

B. QoS Evaluation Metrics

SECTest not only can be adjusted according to specific test content, but it can also customize a diverse range of metrics based on the characteristics and requirements of the integrated testing tools. This flexible and adaptable feature enables SECTest to demonstrate powerful application potential in various testing scenarios. For QoS testing of microservices, selecting appropriate evaluation metrics is crucial. In order to comprehensively evaluate the performance of microservice architecture, we specifically focus on these metrics, including latency, throughput, average response time, and packet drop rate. When evaluating the quality of testing services for microservice applications in satellite edge clouds, each selected metric carries unique significance. The testing metrics collectively provide a comprehensive view of the performance, scalability, user experience, network reliability, and data integrity of microservice applications under stress within satellite edge clouds. This holistic perspective is instrumental in optimizing service quality in complex, distributed satellite cloud environments.

Total Latency: Whenever a satellite performs a communication task, it attempts to establish contact with neighboring satellites. Therefore, when neighboring satellites come into communication range and need to complete communication tasks, direct communication connections are established. Transmission delay may occur during inter-satellite communication, where transmission delay is defined as [22]:

$$T_{trans} = \frac{D_{trans}}{B \log_2 \left(1 + \frac{P_t G_{\max}^2}{k_B \tau L(xy)}\right)} \tag{1}$$

where D_{trans} represents the transmission packet size between satellites x and y. B denotes the channel bandwidth in Hertz, while P_t is the transmission power. $G_{\max}$ refers to the peak gain of both antennas of satellite x in the direction of their main lobe. k_B is the Boltzmann constant, and τ represents the thermal noise in Kelvin. Finally, $L(xy)$ is the path loss for an inter-satellite link between satellites x and y [22]. Inference delay refers to the total time spent by LEO satellites from the start of target recognition to the completion of the task. Therefore, the total latency of an image inference application task is:

$$T_{total} = T_{trans} + T_{infer} \tag{2}$$

Throughput: TP represents the throughput of successfully transmitted data packets through the communication bandwidth channel B. Taking into account the possibility of inter-satellite communication failures, data packets are transmitted within reach between satellites x and y. This process continues until a communication mechanism cannot be established between the satellites within a time period $T_{connect}$. The communication latency between satellites is denoted as T_{trans}. We define *TP*

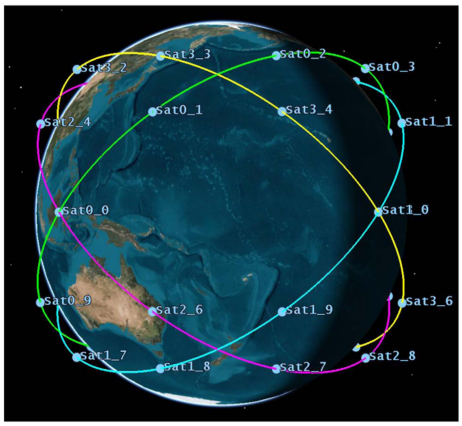


Fig. 3. The example of the StarLink constellation includes four orbits, each with ten satellites.

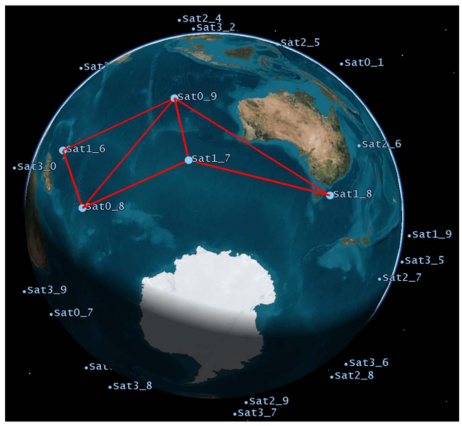


Fig. 4. Network topology for communication satellites within five minutes.

as [23]:

$$TP = \frac{\sum_{i=0}^{n}(T_{connect}) \cdot B}{\sum_{i=0}^{n}(T_{connect} + T_{trans})} \tag{3}$$

Packet Drop Rate: We define the probability of data packet D experiencing data reduction during transmission from satellite x to satellite y, as the percentage calculated between the total lost data packets $D_{totalPL}$ and the total successfully transmitted data packets $D_{totalPT}$, and is defined as [23], [24]:

$$PDR(x, y) = \frac{D_{totalPL}}{D_{totalPT}} \tag{4}$$

Average Response Time: During the satellite service request process, a complete request cycle involves satellite x sending a service request to satellite y, and upon receiving the request, satellite y sends a response message back to satellite x. We define the average response time of the edge cloud system's microservices requests as:

$$\bar{R} = \frac{\sum_{i=0}^{n}(t_{req} + t_{resp})}{N} \tag{5}$$

where t_{req} is the time when satellite x sends a request to y, t_{resp} is the time when satellite y sends a response back to x after receiving the request, and N is the total number of requests.

IV. Experiment

A. Experimental Setup

To simulate the operational trajectories of the satellite cluster, we utilized the STK system toolkit to model the topological changes in the StarLink satellite network. As illustrated in Fig. 3, the orbital altitude was set at 550 kilometers with an inclination angle of 45 degrees. Four orbits from the Starlink constellation were selected, each containing 10 LEO satellites, resulting in a total of 40 satellites [25]. We randomly selected a satellite node and calculated its connectivity in the satellite network topology changes within a five-minute interval. The resulting satellite network topology, as shown in Fig. 4, indicates that within this five-minute interval, this satellite can form a communicative cluster with its adjacent four satellites. The specific experimental parameters are detailed in Table I. Carrier bandwidth, Thermal noise, and EIRP plus receiver antenna gain are utilized for calculating communication delays.

TABLE I
Experimental Parameters for Simulating the Starlink Constellation

Parameter	Value
LEO orbital altitude	550 km
Orbit inclination	45°
Number of satellite per plane	10
Number of plane	4
Carrier bandwidth	[50,100,500] MHz
Thermal noise	354.81 K

B. The Real-World Satellite-Based Edge Cloud Cluster

The ideal environment for evaluating the QoS in SECTest is a real-world edge cloud cluster composed of small LEO satellites. According to weight standards, satellites weighing up to 180 kg are classified as small LEO satellites, which are primarily made up of the platform and computational payload [26]. The microservices for SECTest evaluation primarily run on the computational payload of these satellites. Currently, commercial computing systems aboard small LEO satellites must meet several key requirements. These include low power consumption, high performance, and ease of system integration [27]. Additionally, the weight of these commercial systems is significantly lower than the limitations imposed on the satellite's computational payload. We have fully considered the energy consumption and weight constraints of computational payloads on real operational small LEO satellites. Moreover, we have taken into account the characteristics of the heterogeneous computing environments in LEO satellite constellations.

Hence, we constructed a realistic satellite edge cloud cluster environment consisting of 5 physical nodes and 5 virtual nodes. The physical nodes included 3 Atlas 200I DK A2 and 2 Phytium D2000 nodes. The 5 virtual nodes were created and deployed on a single Intel i7 CPU to dynamically adjust the cluster's scale. One of the Atlas nodes served as the master node, while the remaining nodes functioned as worker nodes. The schematic diagram of the built edge cloud cluster is depicted in Fig. 5.

The computing architectures of the Atlas, Phytium, and i7-CPU nodes are different, making programming in a unified

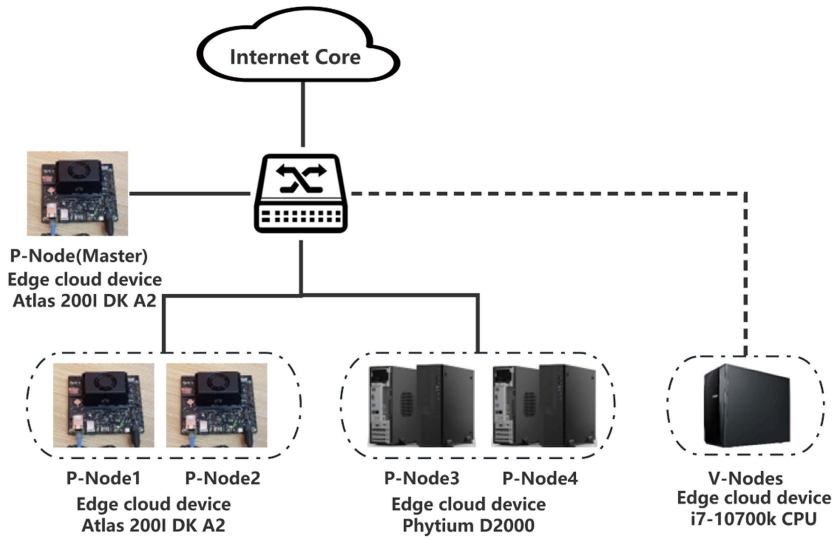


Fig. 5. Architecture of real satellite edge cloud cluster.

TABLE II
THE PARAMETERS OF REAL COMPUTATION NODES

Nodes	RAM	Cores	Computation Capacity	Consumed Power
Atlas 200I DK A2	8G	4 Cores	8 TFLOPS	24W
Phytium D2000	8G	8 Cores	2.4 GHz	60W
i7-10700k CPU	64G	8 Cores	5.10 GHz	125W

TABLE III
PERFORMANCE COMPARISON OF THREE DIFFERENT MODELS IN DOTA DATASET

Model	FLOPS (G)	Params (M)
YOLOv5 [31]	17.5	7.6
Faster-RCNN [32]	116.86	41.7
RetinaNet [33]	36.42	215.92

TABLE IV
COMPARISON OF INFERENCE DELAY OF DIFFERENT MODELS AT DIFFERENT NODES

Model	Node Name	Latency (s)
YOLOv5	P-Node1	0.53
	P-Node2	0.53
	P-Node3	0.44
	P-Node4	0.32
Faster-RCNN	P-Node1	10.74
	P-Node2	12.99
	P-Node3	5.61
	P-Node4	5.65
RetinaNet	P-Node1	13.12
	P-Node2	12.10
	P-Node3	5.73
	P-Node4	5.88

environment challenging. The Atlas and virtual nodes were installed with the Ubuntu-v22.04 system, while the Phytium nodes were installed with the CentOS 7 system. The parameters of the nodes and those used in the experiments are detailed in Table II.

C. Integrated Testing Tools

Based on the aforementioned QoS evaluation metrics, we have selected iperf3, Locust and ghz as our testing tools. These tools will be deployed by creating containers and deploying them into the same namespace on each node.

Iperf3 [28]: This tool provides functionality to measure network throughput and packet loss. It will help in evaluating the network performance of our satellite edge cloud cluster.

Locust [29]: This tool is used for performance testing of web microservice applications. It simulates varying workloads of HTTP servers with different numbers of users, enabling us to evaluate the performance of our web microservice application.

Ghz [30]: This is an open-source grpc benchmarking and load testing tool. It can simulate multiple worker threads on a single server and initiate concurrent requests to these threads simultaneously, offering various load strategy options. We use ghz to evaluate the performance of different computing capacity servers in handling high-concurrency requests.

D. Result

1) Latency: We have deployed YOLOv5, Faster-RCNN, and RetinaNet models in different pods with the same namespace on worker physical nodes. The master node is used for distributing testing tasks, while the Worker nodes are used for executing object detection testing tasks. As shown in Tables III and V, compared to Faster-RCNN and RetinaNet, YOLOv5 has significantly fewer parameters and FLOPS. Using the lightweight

TABLE V
COMPARISON OF COMMUNICATION DELAY BETWEEN DIFFERENT NODES AT DIFFERENT BANDWIDTHS

Bandwidth	Node Name	Latency (s)
50M	P-Node1	0.562
	P-Node2	0.554
	P-Node3	0.567
	P-Node4	0.577
100M	P-Node1	0.281
	P-Node2	0.279
	P-Node3	0.283
	P-Node4	0.288
500M	P-Node1	0.056
	P-Node2	0.055
	P-Node3	0.057
	P-Node4	0.058

YOLOv5 model, each node can reduce the average image inference latency by 27 times and 29 times, respectively, compared to the other two models. On the other hand, the computing capability of the nodes in the edge cloud cluster also affects the processing latency. As illustrated in Tables II and III, the Phytium D2000 provides a more powerful computing capability than the Atlas 200I DK A2. Therefore, the processing latency of P-Node3 and P-Node4 is lower than that of P-Node1 and P-Node2 across the three different models. Additionally, as shown in Table IV, in the simulated satellite cluster, the inter-satellite communication transmission latency of different nodes varies under different bandwidth conditions due to their different positions in orbit.

2) Throughput and Packet Drop Rate: The actual service scenarios of satellite edge clouds extend beyond remote areas with poor network conditions, encompassing conventional networks and large-scale network transmission environments as well. To comprehensively evaluate the throughput performance of satellite edge cloud clusters under resource-constrained conditions, conventional network settings, and when confronted with high-load or high-bandwidth demands, we conducted tests on packet drop rate and throughput for 5 physical edge cloud nodes within a 5-minute timeframe, across bandwidths of 50 Mbps, 100 Mbps, and 500 Mbps. The master node served as the data sender, distributing data to the other Worker nodes in the cluster. The results of throughput and packet drop rate tests for the worker nodes are illustrated in Figs. 6 and 7. The experimental findings indicate that an increase in bandwidth leads to an expected increase in throughput for each edge cloud node. Simultaneously, with the ability to transmit more data within the same timeframe, the likelihood of data loss during transmission also significantly increases, resulting in a rise in the packet drop rate. Interestingly, except for P-Node1, the overall packet drop rate at 100 Mbps bandwidth is higher than that at 500 Mbps bandwidth. This may be attributed to the heavier network load during data transmission on the 100 Mbps bandwidth network.

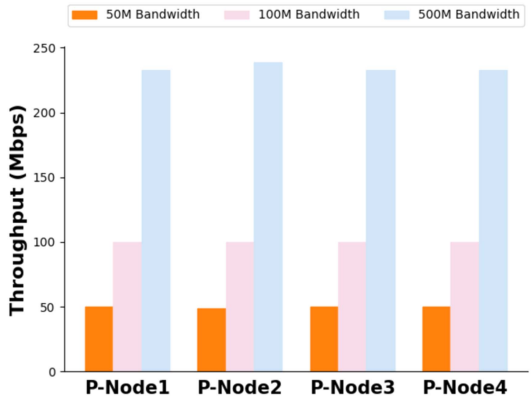


Fig. 6. Throughput of different nodes at different bandwidths.

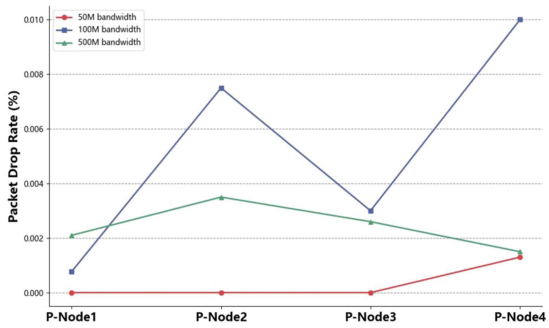


Fig. 7. Packet drop rate of different nodes at different bandwidths.

3) Average Response Time: In SECTest, the locust container on the edge cloud Master node simulates user requests and distributes them to the web microservices on each node, then collects the response times. We selected 5 edge cloud physical nodes and 5 virtual nodes, testing the response times of the edge cloud cluster services with different combinations of node types and quantities. At the start of the load test, we simulated HTTP requests from 50 users, distributed across the nodes via locust. To assess the satellite edge cloud cluster's response capability and ability to maintain service stability during sudden peaks in request volume, we randomly selected time points within a 5-minute period to perform two abrupt increases in the number of simulated users. The increases were by 100 and 500 users, respectively. We followed the same testing procedure when invoking the gRPC network communication protocol in the test cluster microservices. For reference, we repeated the testing process multiple times. Figs. 8 and 9 shows our test results.

The response time curves in Figs. 8 and 9 depict the response trends of edge cloud clusters formed by different combinations of node types and quantities when facing sudden spikes in requests. When there is a sudden surge in user requests, the response times of all node combinations in the edge cloud cluster increase correspondingly for a short period. However, subsequent load balancing of these requests distributes them across various nodes, gradually restoring stability, and thereby reducing the response times of microservices over time.

Based on the experimental results from Tables VI and VII, we analyze the concurrent request response performance of edge

TABLE VI
RESPONSE TIME FOR LOAD TESTING OF HTTP AND GRPC PROTOCOLS IN A SATELLITE EDGE CLOUD CLUSTER COMPRISING PHYSICAL AND VIRTUAL NODES

Node combinations	HTTP			gRPC		
	Total Average Response Time (ms)	Total Min Response Time (ms)	Total Max Response Time (s)	Total Average Response Time (ms)	Total Min Response Time (ms)	Total Max Response Time (ms)
5P-Node	6.225	1.556	0.525	1.223	0.273	4.433
4P-Node+1V-Node	6.935	1.426	0.246	1.387	0.232	7.776
2P-Node+3V-Node	6.827	1.432	0.29	1.733	0.262	8.248
1P-Node+4V-Node	7.091	1.464	0.311	1.647	0.371	14.047
5V-Node	7.242	1.474	0.203	1.667	0.375	8.706

TABLE VII
RESPONSE TIME FOR LOAD TESTING OF HTTP AND GRPC PROTOCOLS IN A SATELLITE EDGE CLOUD CLUSTER COMPRISING PURELY PHYSICAL OR PURELY VIRTUAL NODES

Node combinations	HTTP			gRPC		
	Total Average Response Time (ms)	Total Min Response Time (ms)	Total Max Response Time (s)	Total Average Response Time (ms)	Total Min Response Time (ms)	Total Max Response Time (ms)
3P-Node	10.507	2.749	0.272	1.505	0.582	4.641
4P-Node	7.908	2.776	0.208	1.513	0.239	4.079
5P-Node	6.225	1.556	0.525	1.223	0.272	4.433
5V-Node	7.242	1.474	0.203	1.667	0.375	8.706

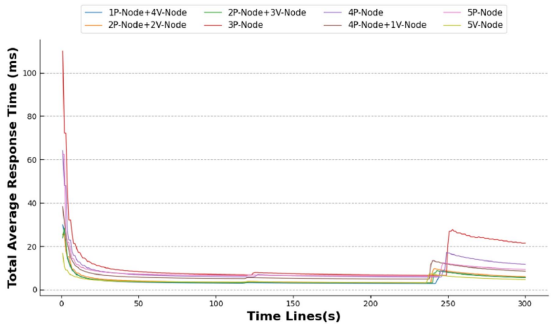


Fig. 8. Average response time variation for HTTP protocol testing across different cluster sizes.

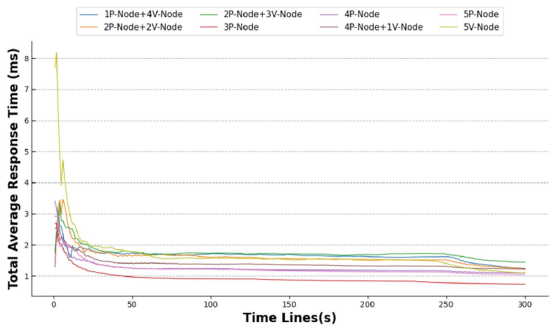


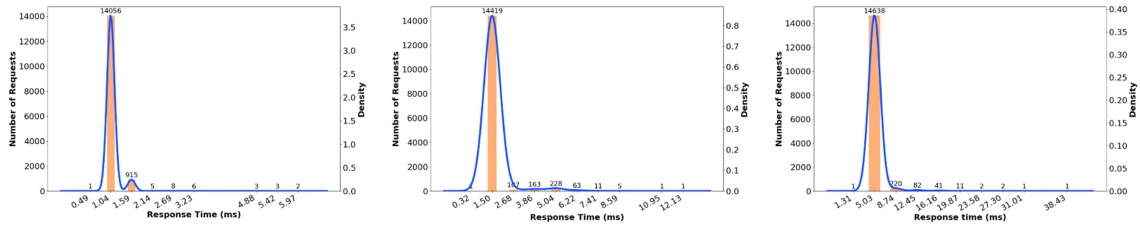
Fig. 9. Average response time variation for gRPC protocol testing across different cluster sizes.

cloud clusters. This analysis focuses on three key aspects: node type, node quantity, and the communication protocol used for requests.

Node Type: Under the condition of the same number of nodes in the cluster, different types of node combinations imply varying computational power for the cluster. An edge cloud cluster composed solely of physical nodes exhibits faster response times when handling concurrent HTTP and gRPC protocol requests. When virtual nodes are added, the response time for handling concurrent requests increases, but not significantly. Conversely, an edge cloud cluster composed solely of virtual nodes has relatively weaker high-concurrency processing capabilities. However, this is not absolute. A cluster made entirely of virtual nodes demonstrated a lower overall average response time for gRPC requests compared to a cluster composed of two physical nodes and three virtual nodes.

Node Quantity: Increasing the number of nodes can potentially enhance the system's ability to handle concurrent requests, but this is not always the case. The experimental results indicate that increasing the number of nodes does not always lead to reduced response times. For instance, the minimum response time for handling gRPC requests in an edge cloud cluster composed of five physical nodes is still higher than that of a cluster composed of four nodes. This could be due to load balancing strategies, inter-node communication overhead, or other system bottlenecks. As shown in Table VII, with an increase in the number of nodes, the system should be able to handle more concurrent requests, thereby reducing the average response time. Nevertheless, when the number of nodes reaches a certain point, improvements may become less noticeable due to network latency, inter-node communication overhead, and the complexity of load balancing algorithms.

Request Type: In the satellite edge cloud cluster, the response time for handling concurrent gRPC requests is significantly



(a) Phytium D2000 (b) Intel i7-10700k CPU (c) Atlas 200I DK A2

Fig. 10. Histogram and density plot of response time distribution for single-point load testing of the gRPC protocol on Phytium D2000, i7-10700 k CPU, and Atlas 200I DK A2 over 5 minutes.

TABLE VIII
RESPONSE TIME FOR SINGLE-POINT LOAD TESTING OF THE GRPC PROTOCOL ON PHYTIUM D2000, I7-10700 K CPU, AND ATLAS 200I DK A2 OVER 5 MINUTES

Nodes	Total Max Response Time (ms)	Total Min Response Time (ms)	Total Average Response Time (ms)
Phytium D2000	5.97	0.49	0.72
Intel i7-10700k CPU	12.13	0.32	0.83
Atlas 200I DK A2	38.43	1.31	2.07

shorter than for handling HTTP requests. This is because gRPC uses the HTTP/2 protocol, which supports multiplexing and bidirectional streaming, allowing for more efficient use of network resources and reduced latency. In Tables VI and VII, the response time for gRPC requests is markedly shorter than for HTTP requests, confirming the performance advantages of the gRPC protocol.

Overall, the pure physical node combination has the shortest overall response time and the lowest peak response time compared to other combinations, indicating that pure physical nodes have better recovery capabilities when dealing with sudden loads. The recovery time for mixed nodes is slightly longer, but their overall recovery capability is still good. When virtual nodes are added, the mixed node combinations take slightly longer to process requests under heavy load, but their overall performance remains satisfactory. This suggests that an appropriate mix of node types can achieve a balance between performance and resource utilization. The response time for pure virtual nodes is relatively high, indicating that pure virtual nodes have weaker recovery capabilities after sudden load spikes. Overall, all types of node combinations are able to stabilize the system after handling sudden surges in requests, providing an acceptable quality of service.

E. Single-Point Load Testing of the gRPC Protocol

To analyze and explain in detail how hardware performance impacts the processing of requests by microservice applications in satellite edge clouds, we used ghz to to conduct single-node load testing of the gRPC protocol on nodes with varying computational capacities. We deployed the testing tool ghz on the nodes of Atlas 200I DK A2, Phytium D2000, and Intel i7-10700 k CPU. On each node, we simulated 5 working threads and set the initial gRPC request load to 50 requests per second. The load strategy involved increasing the number of requests per second incrementally until reaching 500 requests per second. The test duration was 5 minutes, aimed at evaluating the response time of the microservice applications under a continuously increasing request load.

In Fig. 10 Table VIII, Phytium D2000 exhibited stronger computational capability when handling high-concurrency requests. Compared to other nodes, its average response time was faster, enabling it to handle and respond to concurrent requests more quickly, thereby demonstrating outstanding overall performance in microservice request handling. The outcome further validates the Phytium's advantage in processing microservice requests. During the load testing on a single virtual machine deployed on Intel i7-10700 k CPU, the average response time showed a significant decrease compared to the load testing conducted on a cluster composed entirely of virtual nodes. This can be attributed to several factors. Firstly, reducing communication overhead between multiple virtual nodes lowers latency and minimizes performance loss caused by network communication. Secondly, enabling individual virtual nodes to access more CPU resources enhances their ability to process requests. Therefore, optimizing resource allocation and minimizing unnecessary communication overhead is crucial for improving overall performance in resource-constrained satellite edge clouds.

V. RELATED WORK

A. Satellite Network Testing Platforms and Measurement

With the significant reduction in satellite launch speed and cost, research in satellite application testing has advanced considerably. Currently, satellite network testing can be categorized into the following types: (1) testing platforms relying on theoretical modeling and simulation; (2) platforms using actual satellite systems for measurement; (3) simulation testing based on real data.

Celestial [34] is a microVM-based LEO edge virtual testing platform that effectively simulates individual satellites and their motion, along with ground station servers under real network conditions and application-agnostic environments. This ensures network testing accuracy while maintaining scalability. StarryNet [35], as a network testing framework, can flexibly construct experimental network testing environments, simulating satellite dynamics and the network behavior of large ISTNs. StarryNet achieves constellation consistency, network system realism, and flexibility while evaluating network performance. StarPerf [36] is a simulation testing platform for evaluating and optimizing the performance of mega constellations. It can simulate the high mobility impacts of large satellite networks and analyze network performance under different topology options. As a type of simulation satellite testing platform, its core capability lies in the ability to simulate the real motion trajectories of satellite clusters. The advantages of simulation testing platforms include high operability, flexibility, and lower testing costs.

Platforms based on actual satellite systems [37], [38], [39], [40], [41] offer the advantage of high accuracy and precision in testing, as they rely on real data. The extensive coverage of real satellite constellations also allows for a broader range of testing types. Unlike modeling and simulation testing platforms, testing on simulation systems based on real satellite data [42], [43], [44], [45], [46] ensures a high degree of authenticity and accuracy in the test data. Its advantages include high safety and efficiency in testing. Simulation testing can be conducted in a virtual environment, avoiding the risks and potential losses associated with directly operating real satellites. Additionally, simulation testing can quickly simulate various complex scenarios and conditions, shortening the testing cycle and reducing time costs.

B. Microservice Testing in Satellite Edge Clouds

Microservice applications have become an integral part of satellite edge clouds computing architectures, offering services with lower latency and enhanced computational capabilities. With the evolution of cloud-native satellite systems, there's a growing demand for testing microservice performance and validating the effectiveness of optimization algorithms. SatEC [47], inspired by the constellation model of the Iridium communication system, constructs a testing environment on the ground for simulation purposes, aiming to assess the communication range and network performance of satellite networks. Tiansuan constellation [40], deploys AI-related image inference microservices based on the Kubeedge platform within real cloud-native satellite systems, primarily testing inference accuracy and computational latency. Huang et al. [48] developed an open-source platform combining STHN and EC based on KubeEdge, altering traditional microservice deployment methods. They deployed three interconnected neural network microservices and tested the effectiveness of their microservice scheduling algorithm in reducing computational latency.

C. Microservice Integration Testing on Edge Clouds

In cloud environments, microservices integration testing is primarily categorized into two types: integrating different microservices to test the same metrics and integrating various testing tools to evaluate different metrics. Camilli et al. [49] proposed the MIPaRT method and platform, which automates the performance and reliability testing of microservices operations and can be integrated into the DevOps cycle, supporting continuous testing and monitoring. By validating on open-source benchmarks, they demonstrated insights into the performance and reliability behavior of microservices. Giallorenzo et al. [50] illustrated how to utilize Jolie, a programming language for microservices composition, to integrate and design applications based on microservices architecture. Raith et al. [51] extended the open-source tool Galileo to propose a distributed load testing framework for edge cloud cluster benchmarking, primarily for end-to-end testing between different components. In the context of multi-cloud storage platforms, Cao et al. [52] designed a framework that supports unit testing and can isolate testing environments, integrating Tempest and Mock for testing the storage functionalities of edge devices. Additionally, Reile et al. [12] introduced an integration testing tool for microservices deployment in the cloud, which supports extensive microservices integration testing and the publication of test results. However, this tool has limitations in narrow integration testing and load testing. These studies and tools provide diverse solutions for microservices integration testing in cloud environments but also reveal their respective limitations, highlighting areas for optimization under different testing requirements.

D. Comparison of Satellite Network Testing Platforms and Frameworks

With the gradual enhancement of satellite computing resources and capabilities, deploying and managing AI inference services on satellites using cloud environments has become more convenient. Solely relying on theoretical analysis and simulation platforms, without validation of actual systems and real environments, may lead to significant disparities between theoretical derivation results and actual test outcomes. Traditional simulation platforms for network measurement in the cloud are no longer suitable for current scenarios.

Table IX provides a brief overview of the latest satellite network testing platforms and frameworks, comparing them based on common features and the distinct functionalities supported by SECTest. Compared to other research, SECtest possesses core capabilities in satellite constellation testing, suitable for satellite edge clouds, and capable of simulating actual satellite cluster trajectories. Unlike other works, SECtest dynamically generates and manages test clusters by computing network topology changes during satellite cluster operation, rendering it more adaptive and practical. One of SECtest's primary advantages lies in its ability to integrate and extend multiple testing tools, enhancing its network performance testing capabilities to support testing across multiple network protocols. Additionally, SECtest supports customization and extension of testing metrics, providing greater flexibility compared to traditional simulation testing platforms. In terms of testing scope, SECtest not only evaluates microservices' QoS across the entire satellite cluster but also conducts end-to-end testing under various conditions.

In SECtest, the microservice applications used for testing are written in Python and Golang. Requests to the services are

TABLE IX
COMPARISON WITH THE MOST POPULAR SATELLITE NETWORK TEST PLATFORMS AND FRAMEWORKS

	SECTest	**Celestial** [34]	**StarryNet** [35]	**StarPerf** [36]
Purposes	Platform	Platform	Framework	Platform
Integrated testing tools	✔	✖	✖	✖
Test of various types of network protocols	✔	✖	✖	✖
Test in the satellite cloud	✔	✔	✔	✔
Generated test bed	✔	✖	✖	✖
Customized and extended test metrics	✔	✖	✖	✖
Support for satellite cluster network testing	✔	✖	✖	✖
Support for end-to-end network testing between satellites	✔	✔	✔	✔
Development in multiple programming languages	✔	✖	✖	✖

✔ Supported. ✖ No support.

associated with running containers, implying that microservice programs can be defined in any programming language and executed through containers. All applications can be deployed and managed on SECtest using YAML files.

VI. DISCUSSION

We developed SECTest as a testing platform to evaluate QoS in satellite edge cloudss, supporting user-defined experiments and the integration of testing tools. We demonstrated the extensibility of SECTest in integrating testing tools and customizing test metrics. The experiments showed that SECTest can not only assess key QoS metrics of satellite edge clouds but also test the QoS variations of individual nodes. This extensibility allows developers to customize and execute test cases. Additionally, SECTest enables services on the satellite edge clouds to benefit from comprehensive integrated testing, reducing deployment and update costs, minimizing the effort required to modify test code, and significantly lowering the cost of QoS testing for microservices in satellite edge clouds.

For future research directions, SECTest can be improved in several areas. Currently, SECTest relies on manual design and execution of test cases, even though this approach provides solid support for integrated testing on satellite edge clouds. Nonetheless, expanding the functionality of SECTest to enable automatic generation of test cases and achieve higher test coverage represents one of our future research directions. Additionally, SECTest has demonstrated robust extensibility in integrating testing tools, thus expanding its testing capabilities and scope. However, careful consideration must be given when deploying testing tools on SECTest. Currently, there is no mechanism to resolve potential conflicts between different testing tools that might run concurrently. Furthermore, redundancy in the functionalities provided by integrated testing tools needs to be considered. In the future, we aim to establish standards for integrating testing tools more flexibly to meet the QoS testing requirements of services on the satellite edge clouds.

In the SECTest for evaluating the QoS of satellite microservices, the data link layer and network layer of satellite communication protocols are involved. These layers ensure reliable data transmission in environments with high bit error rates and dynamic topologies. Data loss during inter-satellite transmission can be recovered through automatic repeat request (ARQ) or forward error correction (FEC). However, retransmission mechanisms in high-latency links may lead to prolonged recovery times, and redundant transmissions can increase bandwidth consumption. SECTest will simulate scenarios with high bit error rates and signal attenuation to test the performance of ARQ and FEC. Additionally, satellite communication links typically use frame protocols like high-level data link control or point-to-point protocol for data encapsulation and synchronization. In high-speed satellite environments, the Doppler effect can complicate frame synchronization, and the overhead of frame protocols can impact bandwidth utilization. SECTest will further evaluate the performance of different frame protocols under packet loss conditions. MoreOver, in LEO satellite networks, the dynamic topology requires inter-satellite routing protocols to quickly adapt to changes. Routing protocols need to converge rapidly to prevent data loss, but frequent routing updates can increase computational and communication overhead. To address this, routing paths can be pre-calculated based on satellite orbit information. SECTest will support real-time routing adjustments to handle topology changes and help optimize routing protocol design by assessing routing convergence time and computational overhead.

One important aspect of QoS testing on satellite edge clouds that deserves consideration is the cost of using cloud resources. The computing resources and energy required to maintain operations on satellites are limited. Long-term testing on the satellite edge cloud can incur significant energy consumption and computing resource costs. As developers, we aim to complete the entire testing process efficiently and swiftly to ensure no waste of satellite resources. Given the test objectives and metrics, developers can deploy testing tools to the edge cloud cluster on SECTest quickly and cost-effectively, executing scalable tests manually. However, SECTest currently lacks effective solutions for optimizing energy consumption and reducing the computing resources needed for prolonged testing on the satellite edge cloud. In future research, SECTest will better address and resolve these issues.

VII. Conclusion

We have designed and implemented SECTest, a QoS integration testing platform for satellite edge cloud environments. This platform holds significant and guiding implications for testing service quality in satellite edge cloud clusters, addressing key challenges in testing service quality within satellite edge cloud environments. SECTest provides a structured and efficient method to assess the reliability and performance of these services. SECTest can integrate with changes in satellite network topology. It supports the management of satellite edge cloud cluster scales. Additionally, it integrates and extends multiple testing tools into a unified platform. The platform also allows for custom test parameter input and the publication of test results. We have built a real satellite edge cloud cluster based on Kubernetes and, furthermore, utilized containerization technology to integrate and deploy various testing tools within it. Subsequently, we evaluated the key service quality indicators of different types of microservice applications, thereby demonstrating the feasibility and effectiveness of SECTest. In the future, we plan to expand the range of testing tools to provide comprehensive testing of satellite edge cloud environments from multiple perspectives. Additionally, we aim to support automated testing on SECTest, which is another avenue of our research.

References

[1] G. Zeng, J. Luo, Y. Zhang, Y. Qiao, and S. Teng, "A framework for QoS of integration testing in satellite edge clouds," in *Proc. 2024 IEEE Int. Conf. Web Serv.*, 2024, pp. 1109–1111.

[2] C. Wang, Y. Zhang, Q. Li, A. Zhou, and S. Wang, "Satellite computing: A case study of cloud-native satellites," in *Proc. 2023 IEEE Int. Conf. Edge Comput. Commun.*, 2023, pp. 262–270.

[3] X. Gao, R. Liu, A. Kaushik, and H. Zhang, "Dynamic resource allocation for virtual network function placement in satellite edge clouds," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 4, pp. 2252–2265, Jul./Aug. 2022.

[4] L. Hao, P. Ren, and Q. Du, "Satellite QoS routing algorithm based on energy aware and load balancing," in *Proc. 2020 Int. Conf. Wireless Commun. Signal Process.*, 2020, pp. 685–690.

[5] X. Wang, H. Liy, W. Yao, T. Lany, and Q. Wu, "Content delivery for high-speed railway via integrated terrestrial-satellite networks," in *Proc. 2020 IEEE Wireless Commun. Netw. Conf.*, 2020, pp. 1–6.

[6] S. Wang and Q. Li, "Satellite computing: Vision and challenges," *IEEE Internet Things J.*, vol. 10, no. 24, pp. 22514–22529, Dec. 2023.

[7] Y. Zhang, Q. Wu, Z. Lai, and H. Li, "Enabling low-latency-capable satellite-ground topology for emerging leo satellite networks," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1329–1338.

[8] T. Pfandzelter and D. Bermbach, "QoS-aware resource placement for leo satellite edge computing," in *Proc. 2022 IEEE 6th Int. Conf. Fog Edge Comput.*, 2022, pp. 66–72.

[9] Y. Turk and E. Zeydan, "Satellite backhauling for next generation cellular networks: Challenges and opportunities," *IEEE Commun. Mag.*, vol. 57, no. 12, pp. 52–57, Dec. 2019.

[10] O. Liubimov, "Use of micro-services architecture and containerization for the fast development and testing of the cubesat nanosatellites software," *J. Rocket-Space Technol.*, vol. 31, no. 4, pp. 128–137, 2023.

[11] Y. Sun, M. Peng, S. Zhang, G. Lin, and P. Zhang, "Integrated satellite-terrestrial networks: Architectures, key techniques, and experimental progress," *IEEE Netw.*, vol. 36, no. 6, pp. 191–198, Nov./Dec. 2022.

[12] C. Reile, M. Chadha, V. Hauner, A. Jindal, B. Hofmann, and M. Gerndt, "Bunk8s: Enabling easy integration testing of microservices in kubernetes," in *Proc. IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, 2022, pp. 459–463.

[13] C. Dumitrache, G. Predusca, G. Gavriloaia, N. Angelescu, D. Circiumarescu, and D. C. Puchianu, "Comparative analysis of routing protocols using GNS3, wireshark and IPerf3," in *Proc. 2022 14th Int. Conf. Electron. Comput. Artif. Intell.*, 2022, pp. 1–6.

[14] T. F. Düllmann, A. V. Hoorn, V. Yussupov, P. Jakovits, and M. Adhikari, "CTT: Load test automation for TOSCA-based cloud applications," in *Proc. Companion 2022 ACM/SPEC Int. Conf. Perform. Eng.*, 2022, pp. 89–96.

[15] R. Mukherjee and K. S. Patnaik, "A survey on different approaches for software test case prioritization," *J. King Saud Univ.- Comput. Inf. Sci.*, vol. 33, no. 9, pp. 1041–1054, 2021.

[16] Y. Han, S. Shen, X. Wang, S. Wang, and C. M. V. Leung, "Tailored learning-based scheduling for kubernetes-oriented edge-cloud system," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[17] B. Zhang et al., "Progress and challenges in intelligent remote sensing satellite systems," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 15, pp. 1814–1822, 2022.

[18] G. Jocher et al., "ultralytics/yolov5: v3.1 - bug fixes and performance improvements," Oct. 2020.

[19] S. Ren, K.R. HeGirshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[20] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2980–2988.

[21] G.-S. Xia et al., "DOTA: A large-scale dataset for object detection in aerial images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 3974–3983.

[22] I. Leyva-Mayorga, B. Soret, and P. Popovski, "Inter-plane inter-satellite connectivity in dense leo constellations," *IEEE Trans. Wireless Commun.*, vol. 20, no. 6, pp. 3430–3443, Jun. 2021.

[23] P. Tam, S. Math, and S. Kim, "Intelligent massive traffic handling scheme in 5 G bottleneck backhaul networks," *KSII Trans. Internet Inf. Syst.*, vol. 15, no. 3, pp. 874–890, 2021.

[24] M. Esmaeilzadeh, N. Aboutorab, and P. Sadeghi, "Joint optimization of throughput and packet drop rate for delay sensitive applications in TDD satellite network coded systems," *IEEE Trans. Commun.*, vol. 62, no. 2, pp. 676–690, Feb. 2014.

[25] F. Wang, D. Jiang, Z. Wang, J. Chen, and T. Q. S. Quek, "Seamless handover in LEO based non-terrestrial networks: Service continuity and optimization," *IEEE Trans. Commun.*, vol. 71, no. 2, pp. 1008–1023, Feb. 2023.

[26] Q. Li et al., "Exploring real-time satellite computing: From energy and thermal perspectives," in *Proc. 2024 IEEE Real-Time Syst. Symp.*, pp. 161–173, 2024.

[27] G.M. Capez et al., "On the use of mega constellation services in space: Integrating leo platforms into 6G non-terrestrial networks," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 12, pp. 3490–3504, Dec. 2024.

[28] P. Loreti et al., "SRv6-PM: A cloud-native architecture for performance monitoring of SRv6 networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 611–626, Mar. 2021.

[29] G. R. Russo, T. Mannucci, V. Cardellini, and F. L. Presti, "Serverledge: Decentralized function-as-a-service for the edge-cloud continuum," in *Proc. 2023 IEEE Int. Conf. Pervasive Comput. Commun.*, 2023, pp. 131–140.

[30] K. Bojan, "Ghz," 2024. Accessed: Jun. 16, 2024. [Online]. Available: https://github.com/bojand/ghz

[31] X. Wang, J. Lin, J. Zhao, X. Yang, and J. Yan, "Eautodet: Efficient architecture search for object detection," in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 668–684.

[32] T. Chen, R. Li, J. Fu, and D. Jiang, "Tucker bilinear attention network for multi-scale remote sensing object detection," *IEEE Geosci. Remote Sens. Lett.*, to be published, doi: 10.1109/LGRS.2023.3296984.

[33] Y. Pang, Y. Zhang, Q. Kong, Y. Wang, B. Chen, and X. Cao, "SOCDet: A lightweight and accurate oriented object detection network for satellite on-orbit computing," *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 5608115.

[34] T. Pfandzelter and D. Bermbach, "Celestial: Virtual software system testbeds for the leo edge," in *Proc. 23 rd ACM/IFIP Int. Middleware Conf.*, 2022, pp. 69–81.

[35] Z. Lai et al., "StarryNet: Empowering researchers to evaluate futuristic integrated space and terrestrial networks," in *Proc. 20th USENIX Symp. Networked Syst. Des. Implementation*, 2023, pp. 1309–1324.

[36] Z. Lai, H. Li, and J. Li, "STARPERF: Characterizing network performance for emerging mega-constellations," in *Proc. 2020 IEEE 28th Int. Conf. Netw. Protoc.*, 2020, pp. 1–11.

[37] S. Narayana, R. V. Prasad, V. Rao, L. Mottola, and T. V. Prabhakar, "Hummingbird: Energy efficient gps receiver for small satellites," in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–13.

[38] V. Singh, A. Prabhakara, D. Zhang, O. Yağan, and S. Kumar, "A community-driven approach to democratize access to satellite ground stations," in *Proc. 27th Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 1–14.
[39] A. Singla, "SatNetLab: A call to arms for the next global internet testbed," 2021.
[40] S. Wang, Q. Li, M. Xu, X. Ma, A. Zhou, and Q. Sun, "Tiansuan constellation: An open research platform," in *Proc. 2021 IEEE Int. Conf. Edge Comput.*, 2021, pp. 94–101.
[41] R. Xing et al., "Deciphering the enigma of satellite computing with cots devices: Measurement and analysis," in *Proc. 30th Annu. Int. Conf. Mobile Comput. Netw.*, 2024, pp. 420–435.
[42] D. Vasisht, J. Shenoy, and R. Chandra, "L2D2: Low latency distributed downlink for leo satellites," in *Proc. 2021 ACM SIGCOMM 2021 Conf.*, 2021, pp. 151–164.
[43] D. Vasisht and R. Chandra, "A distributed and hybrid ground station network for low earth orbit satellites," in *Proc. 19th ACM Workshop Hot Topics Netw.*, 2020, pp. 190–196.
[44] J. GP Rodrigues and A. Aguiar, "Extracting 3 D maps from crowdsourced gnss skyview data," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–15.
[45] C. Ge et al., "QoE-assured live streaming via satellite backhaul in 5G networks," *IEEE Trans. Broadcast.*, vol. 65, no. 2, pp. 381–391, Jun. 2019.
[46] J. Zhang, S. Dong, H. Yuan, W. Li, Y. Guo, and J. Zhang, "Interface verification and satellite-ground testing of high-precision time comparison system between leo satellites and beidou satellites," in *Proc. 2023 5th Int. Conf. Geosci. Remote Sens. Mapping*, 2023, pp. 120–126.
[47] L. Yan et al., "SatEC: A 5G satellite edge computing framework based on microservice architecture," *Sensors*, vol. 19, no. 4, 2019, Art. no. 831.
[48] Y. Huang and X. Zhang, "Microservice scheduling for satellite-terrestrial hybrid network with edge computing," in *Proc. 2022 IEEE/CIC Int. Conf. Commun. China*, 2022, pp. 24–29.
[49] M. Camilli, A. Guerriero, A. Janes, B. Russo, and S. Russo, "Microservices integrated performance and reliability testing," in *Proc. 3rd ACM/IEEE Int. Conf. Automat. Softw. Test*, 2022, pp. 29–39.
[50] S. Giallorenzo, F. Montesi, M. Peressotti, F. Rademacher, and N. Unwerawattana, "JoT: A jolie framework for testing microservices," in *Proc. Int. Conf. Coordination Lang. Models*, 2023, pp. 172–191.
[51] P. Raith, T. Rausch, P. Prüller, A. Furutanpey, and S. Dustdar, "An end-to-end framework for benchmarking edge-cloud cluster management techniques," in *Proc. 2022 IEEE Int. Conf. Cloud Eng.*, 2022, pp. 22–28.
[52] R. Cao, Z. Tang, C. Liu, and B. Veeravalli, "A scalable multicloud storage architecture for cloud-supported medical Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 1641–1654, Mar. 2020.

Guogen Zeng received the MS degree in software engineering from Hunan University, China, in 2022. He is currently working toward the doctoral degree in energy and power with Hunan University, China. His research interests include software testing, highly trusted software, edge computing, and artificial intelligence.

Juan Luo (Member, IEEE) received the bachelor's degree in electronic engineering from the National University of Defense Technology, Changsha, Hunan, China, in 1997, and the master's and PhD degrees in communication and information system from Wuhan University, Wuhan, Hubei, China, in 2000 and 2005, respectively. From 2008 to 2009, she was a visiting scholar with the University of California at Irvine, Irvine, CA, USA. She is currently a professor and vise dean with the College of Computer Science and Electronic Engineering, Hunan University, Changsha. She has published more than 100 papers. Her research interests include focused on the Internet of Things, cloud computing, and AI. She is a member of ACM and SIGCOM. She is also a Distinguished Member of CCF.

Yufeng Zhang received the PhD degree from the College of Computer, National University of Defense Technology, Changsha, China, in 2013. He is an associate professor with the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include artificial intelligence and software engineering.

Ying Qiao received the BE degree from Xi'an University, Xi'an, China, in 2017, and the MSc degree in software programming from Central South University, Changsha, China, in 2020. She is currently working toward the PhD degree in computer science and technology with Hunan University, Changsha, China. Her current research interests include mobile edge computing and satellite IoT.

Shuyang Teng received the bachelor's degree in digital media technology from Hunan University, China, in 2021. He is currently working toward the doctoral degree in computer science and technology with Hunan University, China. His research interests include Edge computing and satellite Internet.

Keqin Li (Fellow, IEEE) received a BS degree in computer science from Tsinghua University, in 1985, and the PhD degree in computer science from the University of Houston in 1990. He is a SUNY Distinguished professor with the State University of New York and a National Distinguished professor with Hunan University (China). He has authored or coauthored more than 1110 journal articles, book chapters, and refereed conference papers. He holds more than 75 patents announced or authorized by the Chinese National Intellectual Property Administration. Since 2020, he has been among the world's top few most influential scientists in parallel and distributed computing regarding single-year impact (ranked 2) and career-long impact (ranked 4) based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars (2023-2024) and is among the top 0.02% out of more than 20 million scholars worldwide based on top-cited publications. He is listed in ScholarGPS Highly Ranked Scholars (2022-2024) and is among the top 0.002% out of over 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He was a recipient of the 2022-2023 International Science and Technology Cooperation Award and the 2023 Xiaoxiang Friendship Award of Hunan Province, China. He is a member of the SUNY Distinguished Academy. He is an AAAS fellow, an AAIA fellow, an ACIS fellow, and an AIIA fellow. He is a member of the European Academy of Sciences and Arts. He is a member of Academia Europaea (Academician of the Academy of Europe).