EasiEI: A Simulator to Flexibly Modeling Complex Edge Computing Environments

Xiao Su⁰, Jianpeng Qi⁰, Jiahao Wang⁰, Rui Wang⁰, and Yuan Yao

Abstract—In edge computing scenarios, there is a need for modeling dedicated features and heterogeneous devices functions, as well as integrating multiple complex scenarios with diverse objectives and frequent interactions. However, existing platforms modeling for the whole device ignores the independence between functional components resulting in limited scenario support. We propose an open-source simulator named EasiEI. EasiEI addresses the need for higher level feature replaceability and independence in modeling complex edge scenarios through independent functional component-level modeling and microkernel architecture. This approach enables users to assemble independent functional components in a plug-and-play manner for heterogeneous devices or different application requirements. EasiEI is fully compatible with all the existing built-in modules in NS3 (a powerful network discrete event simulator). To verify the flexibility and extensibility of EasiEI, we implement several centralized and decentralized computing paradigms cases in a step-by-step way. These cases restore and simulate the performance state of various real devices in real time, meeting the requirements for verifying the edge computing ideas such as task scheduling in a distributed manner. Results show that the simulations have well reflected the characteristics of the real world and can construct complex environment flexibly.

Index Terms—Complex environment, edge computing, microkernel architecture, modeling and simulation, resource management.

I. INTRODUCTION

E DGE computing refers to performing various functions at the edge of network and focuses more toward the thing side [1] which are devices with functional components, such as sensing, communicating, data collecting, managing, and decision making [2].

Recent IoT growth projections suggest that by the year 2030 the number of connected on this planet will reach approximately 30 billion [3]. With the increasing types of device functions, manifestations of resources, and the amount of resources, managing them is becoming more and more complex and facing significant challenges [2]. It also complicated the strategies in maintaining the system reliability

Xiao Su, Jianpeng Qi, Jiahao Wang, and Rui Wang are with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China (e-mail: wangrui@ustb.edu.cn).

Yuan Yao is with the Medical Innovation Research Department, Chinese PLA General Hospital, Beijing 100853, China (e-mail: yaoyuan301@sina.cn). Digital Object Identifier 10.1109/JIOT.2023.3289870

and performance, no matter in distributed, centralized, or single-machine scenarios.

A notable challenge under those complex characteristics is that prototype and idea validation are not always feasible, especially when the scenarios are diverse. Edge scenario faces a challenge similar to the insect order paradox [4], where the variety of edge devices and scenario needs is comparable to the variety of insect species, making it difficult to effectively represent them. However, existing edge computing simulation platforms are limited to specific application scenarios during development, resulting in tight coupling between device functional components and models, making it impossible to replace one feature with another that has similar functionality without affecting the behavior of the overall system. We also make a full list of popular edge computing open-source simulators and frameworks on GitHub [5]. In reality, devices or entities often have overlapping features, but the internal features of the entities are independent of each other. When manufacturers produce new edge devices, they often adjust the feature based on existing chips and reproduce new devices to adapt to different scenarios.

One of the main challenges introduced by the reality is the lack of feature independence and isolation. Modeling at the device level as the finest granularity cannot achieve the above-mentioned feature independence and replaceability, which leads to highly coupled functional components, limited scenario support, and difficulty in extending simulators.

"A simulation model should always be developed for a particular set of objectives. In fact, a model that is valid for one objective may not be for another," Law [6] says. To make the "objective" more general and to improve the scalability and precision of edge computing simulator, in complex edge scenarios, we design and implement an open-source simulator EasiEI (https://gitlab.com/Mirrola/ns-3-dev), a simulator for complex edge scenarios that focuses on resource management and utilization. In EasiEI, different functional components can be combined flexibly to meet different requirements. The major contribution of our work is listed as follows.

 In order to conform to the independence of functional components, improve the scalability of the underlying components, and reduce the time cost of development. We crack the simulation granularity from the device type level to the function level. To do that, after examining various scenarios and cluster trace data sets, including Google, Alibaba, and others, we extract four essential functional components from the multifarious and multifunctional edge and cloud devices, including Task

2327-4662 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Manuscript received 15 May 2023; revised 22 June 2023; accepted 22 June 2023. Date of publication 27 June 2023; date of current version 25 December 2023. This work was supported by the National Natural Science Foundation of China under Grant 62173158. (*Xiao Su and Jianpeng Qi are co-first authors.*) (*Corresponding authors: Rui Wang; Yuan Yao.*)

Generator, Task Receiver, Task Orchestrator, and Task Sender.

2) To better address the challenges in edge computing scenarios, we propose a microkernel architecture that enhances the independence and replaceability of components in complex edge scenario. This architecture allows for flexible feature combinations through interchangeable and pluggable operations, which can effectively simulate heterogeneous devices and complex scenarios and conduct extensive experiments and scenarios to show that EasiEI is easy to use.

The remainder of this article is organized as follows. We first discuss the background and then introduce the related work and motivation in Section II. We then introduce the design and implementation and analyze the architecture of EasiEI in Section III. In Section IV, we implement three cases to prove the flexibility and extensibility of EasiEI. Section V discusses the conclusion and several notable future directions.

II. BACKGROUND AND MOTIVATION

A. Requirements When Designing Simulator

Two aspects reflect the needs of complex scenarios in edge computing systems: 1) simple scenarios require modeling dedicated features (we use feature and function interchangeably hereafter) and heterogeneous devices and 2) complex scenarios consist of several different simple scenarios embedded with diverse objectives metrics and frequent interactions. Using a smart city as a case study, it comprises a multitude of simple interconnected scenarios, including but not limited to smart traffic management, intelligent lighting systems, efficient waste management, and advanced public safety measures. Each of these scenarios possesses unique objectives and performance metrics, necessitating frequent interaction amongst them. In order to effectively model such complex scenarios, it is imperative to achieve a high degree of feature independence and replaceability.

1) Feature Replaceability in Edge Simulation Platforms: In practice, the features required in a simulation platform depend on the specific objectives of a given study. Different edge scenarios have unique feature requirements, resulting in an infinite range of potential applications with different needs [7]. For instance, fog computing [8], mobile-edge computing [9], and edge cloud [10] have distinct architectures and resource management goals, which necessitate fine-grained modeling of functions, such as task offloading [11] and energy consumption monitoring [12] within the framework. Meanwhile, each specific application in a scenario has unique performance metrics, including Quality of Service (QoS) [13], energy consumption [14], reliability [15], and system throughput [16]. Due to the wide range of scenarios involved in edge computing, it is impossible to fully cover the evolving requirements after the platform is built, and also difficult to meet all existing or future requirements. To achieve dynamic scalability and diversity in edge computing scenarios, it is crucial to consider the various features of each scenario and ensure the replaceability of their functions. The ability to replace features is

essential for simulation platforms to quickly switch between different verification scenarios.

2) Feature Independence in Edge Simulation Platforms: In edge computing, each edge device typically has independent functionality, and the level of feature isolation plays a critical role in determining the visibility of feature integrity to other features and devices. A higher level of feature isolation allows for more flexible and scalable modeling of edge computing systems. According to Law [17], a system's state is defined as a collection of variables, including features and entities, that are necessary to describe the system at a particular time. In such systems, the features of each edge node are usually independent of each other. For example, one feature of a node might be responsible for data acquisition, while another feature might be responsible for data processing or storage. This means that each feature can be modeled and simulated independently without affecting the behavior of other devices or features in the system.

3) Need for Low-Coupling and Feature-Independent Simulation Platform: Simulating edge computing scenarios using existing simulators can lead to highly coupled scenarios that are difficult to reconfigure or dissolve. To accurately model complex real-world scenarios in edge computing, it is essential to recognize the importance of feature independence and replaceability in simulation platforms.

The resources and functional features in existing simulators are often confined within device models, leading to inflexible resource representation and limited adaptability to diverse resources and scheduling requirements in complex scenarios. The lack of feature replaceability and independence can cause inflexibility of the simulation platform, requiring a redesign and implementation of the entire system for any change. This constraint limits the platform's scalability and reusability, making it challenging to add new features or replace existing ones. Moreover, high coupling between features can cause changes in one feature to affect the behavior of other features, resulting in unpredictable outcomes.

B. Related Edge Computing Simulators

Edge computing simulation platforms are generally built on three frameworks: 1) computing-based simulators; 2) networking-based simulators; and 3) greenfield simulator (refers to a new project that has no existing infrastructure or legacy code to utilize and needs to be built from scratch). However, the computing resources and functional component modeling and organization of most platforms are insufficient to support the simulation of complex scenarios in cloud computing or edge computing.

1) Computing-Based Simulators: CloudSim [18] is capable of modeling and simulating heterogeneous devices in largescale cloud computing infrastructures, including data centers, service agents, scheduling, and allocation policies. It supports independent virtualized resource modeling and organization methods and can establish and manage multiple independent and collaborative virtualized resource scheduling services on data center nodes.

TABLE I							
Related	SIMULATORS						

	Modeling of Network Protocol Stack	Resource Feature Replaceability	Function Feature Replaceability	Resource Feature Independence	Function Feature Independence
CloudSim [18]		√		√	
iFogSim [19]		√		√	
EdgeCloudSim [20]		✓		✓	✓
NS3 [21]	√				
OMNeT++ [22]	√				
Mininet [23]	√				
FogNetSim++ [24]	√		√		√
ns-3-DCE [25]	√			√	✓
MaxiNet [26]	√				
FogBed [27]	 ✓ 			√	✓
PureEdgeSim [28]				↓ ✓	√
EasiEI	 ✓ 	√	√	√	✓

iFogSim [19] inherits the low-coupling virtualization computational design of CloudSim in modeling edge computing devices and adds instances of fog computing devices, along with a collaborative scheduling framework. Although it can simulate heterogeneous devices, it lacks a modular feature combination architecture, making it difficult to flexibly replace features in a single device, and the device model is too large to be developed, which cannot meet the needs of complex edge computing scenarios.

EdgeCloudSim [20] inherits the resource virtualization of CloudSim and builds an edge computing scheduling architecture in the cloud–edge–end. EdgeCloudSim adopts a modular design for load generation features, But there are too many manual controls for access between devices, making feature replacement difficult, which is not conducive to the expansion and migration of distributed scenarios.

PureEdgeSim [28] can simulate heterogeneous devices and has functionally independent components but adopts a deviceas-component approach, resulting in each device having only a single function and lacking modular design. Although devices are heterogeneous, their functions are too singular to support complex scenario simulations.

2) Networking-Based Simulators: NS3 [21], OMNeT++ [22], and Mininet [23] are commonly used network simulation and modeling tools that provide rich network protocols and components for simulating different network scenarios. NS3 provides basic network protocols and components, including LTE, named data networking (NDN), wireless sensor networks, 5G networks, WiFi, WiMax, TCP/IP, and more. OMNeT++ provides support for vehicular communication, wireless network simulation frameworks, and mobile ad hoc networks. Mininet simulates network interactions by creating software-defined network (SDN) element nodes, including hosts, switches, controllers, and links.

FogNetSim++ [24], based on the data model of OMNeT++, models fog computing networks and encapsulates the simulation of protocols such as MQTT. It provides network application layer receive/send functionality replaceability but lacks physical characteristic modeling of computing power and heterogeneous devices. Compared to OMNeT++, NS3 adopts a Linux-like architecture design, with its internal interface (network to device driver) and application programming interface (socket) well mapping the way modern computers are built. It has network application layer protocol replaceability but lacks the modeling of computing ability, making it unable to simulate edge computing scenarios. ns-3-DCE [25] provides the functionality of executing entities of network protocols or applications in both user space and kernel space in NS3. It adds basic edge computing components for modeling, and the basic components can be combined and cooperate but lack replaceability.

MiniNet is a process-based lightweight virtual machine simulation platform that can create SDNs on a single physical device. MiniNet simulates network elements by creating network interactions between hosts, switches, controllers, and links. MaxiNet [26] expands the scene based on MiniNet and can interconnect multiple MiniNet scenes. Both have independent and replaceable scenes but lack replaceability at the device function level due to its coarse granularity, making it difficult to customize scenes for dynamically changing scenario requirements.

FogBed [27] combines Containernet [29] and MaxiNet, allowing users to use Docker containers as hosts and create virtual instances to simulate resource configurations. This approach enhances MaxiNet's fine-grained replaceability from scene level to edge device level, and independent modeling of computational resources. However, due to docker's dependency on the underlying operating system, it is not possible to modularly replace device functionalities and computational resources. Additionally, the simulation is constrained by the limited resources of the host machine and cannot emulate large-scale scenarios.

3) Greenfield Simulation Platform: SimEdgeIntel [30] is a platform that focuses on the mobile features, edge caching, and switching strategies in edge computing. The platform has interchangeable algorithms for the aforementioned functionalities, which allow users to validate different strategies for their scenarios. However, lacks modeling for network protocal stack and computing resources.

RECAP [31] is a platform that considers the characteristics of complex edge scenarios and designs architectures for resource utilization scenarios. However, the modeling precision of the platform is limited to the device level, and it lacks independent functionality and interchangeability, making it difficult to expand scenarios.

While edge computing simulation platforms can take advantage of the computing modeling and scheduling architecture of cloud computing platforms, cloud computing mainly focuses on centralized resource management under the assumption of sufficient network resources. In cloud computing modeling, computing resources are finely modeled, whereas network resources are only modeled at the physical layer with parameters such as bandwidth and the number of links, without modeling the multilayer communication protocols.



Fig. 1. EasiEI architecture.

Computing-based platforms possess computing modeling and scheduling frameworks but lack precise network modeling. Networking-based platforms have high-precision network modeling of multilayer protocols but lack scheduling frameworks. Greenfield simulators have lower modeling accuracy in both computing and network aspects. However, due to their small platform size, they have the advantage of facilitating rapid process validation.

Based on the literature review, it is evident that many existing simulators used in edge computing scenarios suffer from high coupling and low flexibility due to the lack of feature replaceability and independence. Therefore, it is crucial to develop a simulation platform that provides a higher level of feature isolation to support complex real-world scenarios.

III. EASIEI DESIGN AND IMPLEMENTATION

A. EasiEI Architecture

EasiEI is developed on top of the NS3 simulator (core engine), which provides real-time event scheduling, C++/Python runtime environment, networking, and logging components. Furthermore, based on the built-in modules of NS3 and the feature replaceability of the EasiEI architecture, various functional components related to NS3's features, such as energy consumption and mobility, can be also flexibly recombined.

EasiEI adheres to the Single Responsibility Principle by dividing edge devices into independent functional components. Four essential functional components, namely, Task Generator, Task Receiver, Task Orchestrator, and Task Sender, have been extracted from the diverse set of edge and cloud device functionalities. Each component is responsible for executing its own tasks without interfering with other components. Fig. 1 highlights various types of components that are triggered by different tasks to initiate their respective processes. The task-driven architecture, based on feature-level modeling, caters to complex edge computing scenarios by satisfying the requirements of feature independence and replaceability. It allows for flexible instantiation and combination of features to build complex scenarios, transcending the constraints of individual simple scenarios, conforming to the Open-Closed Principle. As a result, it can extend its functionality or adapt to different scenarios by adding or replacing functional components without modifying the existing kernel code. Furthermore, the platform adheres to the Dependency Inversion Principle and defines a unified abstract interface for various types of functional components, allowing highlevel modules (device-level functions) to access different functional components without understanding their underlying implementation details.

The microkernel-based EasiEI architecture exhibits several significant advantages over traditional architectures. First, its microkernel only needs to maintain the task table, resulting in a much simpler and more scalable architecture. This allows for the insertion or removal of functional components as needed to realize heterogeneous device simulation, significantly reducing complexity. Second, the architecture's scalability is enhanced by its plug-in-based design, which enables users to customize new functional components and add existing equipment models. Third, different types of task requirements signify diverse edge scenarios, and the task-driven microkernel architecture enables scene expansion and migration through the addition of new types of tasks or adjustments to existing tasks while allowing different types of tasks to coexist in the kernel simultaneously. This greatly enhances the platform's flexibility compared to traditional architectures. Users can leverage edge computing-oriented engineering principles, such as [32], to partition functional components based on specific scenario requirements. This approach enables a more efficient and tailored implementation of edge computing solutions, ultimately enhancing overall system performance.

B. Fine-Grained Resource Model Enables Functional Components Higher Accuracy

Task modeling is an essential component of the architecture kernel, and the definition of features is largely dependent on the modeling of resources and tasks. Different types of tasks can reflect diverse scenario requirements, which can impact the definition and modeling of feature components. Modeling tasks can assist the platform in defining scenarios. Furthermore, the level of resource modeling represents the refinement degree of feature components. Therefore, modeling tasks and resources can ensure the scalability and refinement of features in scenarios.

1) Task Representation: The task entity information is shown in Fig. 2, including time information, execution information, location information, and maintenance entities. The task requests virtualized resources modeled in Resource Units as shown in Fig. 3 and task's execution process is simulated by sleeping the Task Orchestrator of the current device for an equivalent duration as the task's execution time. EasiEI conducts two distinct task execution simulations



Resource Unit Resource Unit Resource pool Resource pool Computation device

Fig. 3. Resource combination.



using task time information and task execution information, respectively.

In the first scenario, EasiEI leverages time information, which encompasses the generation time, maximum waiting time, and completion time of tasks. If the data set provides the complete execution process of a task, task execution can be simulated by configuring the generation time and completion time accordingly. If the task's execution time exceeds the maximum waiting time, the task is considered to have failed.

In the second scenario, EasiEI utilizes task execution information, which includes computational resource requirements such as resource units, task priority, and the current state of the task. In scenarios that necessitate dynamic alignment between task workloads and device performance, users are responsible for conducting their own performance modeling. The resulting task load from the modeling process is designated as the computility demand, while the device performance is defined as the Resource Unit. The runtime for task execution on different computational devices is determined by the quotient obtained by dividing these two values.

Both approaches can coexist within the same scenario, with the time information approach being given higher priority. Furthermore, in the above two particular scenarios, the utilization of task priority and task status can aid in validating different scheduling algorithms. During the task execution process, location information is maintained by a maintenance entity. The source address identifies the device that generates the task, the current address identifies the intermediate device where the task is currently located, and the destination address identifies the ultimate destination where the task will be executed. The transition of a task from start to finish in a device involves the transition between multiple states and operations. The workflow of tasks and platform will be discussed in Section III-E.

2) Computation Resource Representation: Allocation and recording of state changes of computing resources is done through the compute resource class. The compute resource class is responsible for modeling the amount of compute resources in a computing device. For different types of compute nodes in complex edge scenarios, EasiEI provides two types of resource scaling: 1) resource unit and 2) resource pools. A single resource unit can be assigned to each device. By increasing the resource capacity of a resource unit, edge



Fig. 4. Task generator design.

devices with sufficient computational resources are modeled. Meanwhile, multiple resource units can be combined into a resource pool to be allocated to a single device, and the resource ratio between different resource units can be adjusted to simulate a cloud device with high computing resources. Therefore, users can experimentally test the combination and scheduling of resources based on different resource combination methods for different devices, such as edge computing devices, edge servers, and cloud servers, as shown in Fig. 3.

C. Functional Components

As shown in Fig. 1, task generator can flexibly set frequency, resource requirements, load, and category. Task receiver provides the ability to define the task receiving frequency, fault tolerance policies, and caching algorithm to fit various receiving scenarios. Task orchestrator (or scheduler) defines different scheduling policies, resource allocation policies, and offloading policies. Task sender allows users to customize task sending frequency and destination selection algorithm. EasiEI functional component design adopts a lowcoupling mode, which facilitates the expansion, migration, and simulation of complex edge scenarios.

1) Task Generator: As the basic unit for requesting and allocating resources, tasks are generated by the Task Generator class, as shown in Fig. 4. Users can import real-world data sets through the GetTaskInfo function, which includes information about resource requirements, start time, end time, priority, and destination, to generate task base data. Alternatively, specific distributions of task resources and time can be simulated and imported as task information using ProbDistr. Then, the Generate function is called to generate a task instance. The task instance information is evaluated, and if the task's destination is equal to the local machine ID, the task is saved in Ptr(LocalTable) and waits for the task scheduler to schedule it.





Fig. 6. Orchestrator design.

Fig. 5. Receiver design.

Otherwise, it is saved in Ptr(SendTable) and waits for the task sender to forward and uninstall it.

2) Task Receiver: In a complex edge scenario, task offloading, forwarding, and receiving policies change continuously with many factors, including task arrival rate in the queue, processing speed of the device, and transmission rate of the link. These factors finally affect the queue latency of a task.

Application is responsible for implementing the encapsulation of the communication protocol, as shown in Fig. 5. The nodes in the network topology interact with each other through the Application. Nodes are bound to Application and ResourcePool by Aggregator, and Node has direct access to the application and resource components installed in this node. The task instance is then stored in Ptr(ReceiveTable).

Therefore, *FrequencyCtr* and *Synchronize* are implemented in *Receiver*, where *Synchronize* is responsible for controlling the frequency of synchronization between the local task table $Ptr\langle LocalTable \rangle$ and the receive cache $Ptr\langle ReceiveTable \rangle$. The frequency synchronization policy is bundled into a task instance, which can not only synchronize the frequency between local components but also serialize the synchronization between network nodes. This method verifies data consistency, cache misses, and other scenarios.

3) Task Orchestrator: Task Orchestrator class is responsible for allocating computing resources based on task requirements and scenario context, as shown in Fig. 6. It schedules tasks based on the scheduling model defined by users. In EasiEI, by default, task operations and resource allocation are manipulated in a first-in-first-served (FIFS) and round robin (RR) order which is implemented by *ChooseTask*. For simultaneous arrival tasks, we use a nonpreemptive high-priority scheduling policy. When allocating resources, Task Orchestrator traverses the resource unit in the *Ptr*(*ResourcePool*) by order. EasiEI also allows custom implementation of resource scheduling techniques by *ResourceAllocate* to accommodate complex edge scenario requirements [33].

4) Task Sender: Users can verify different types of loadtransfer policies by using the Task Sender class, as shown in Fig. 7. By default, there are two ways to send tasks. The first is centralized scheduling, which picks the task destination. When a unique destination device name is given in a task, Task Sender determines the IP address of the unique item by checking its local DNS and forwards the task to the destination. The second is distributed scheduling, which picks



Fig. 7. Sender design.

the bound address to send the task. These two ways make EasiEI highly scalable for various scenarios and allow users to build diverse distributed/centralized scheduling frameworks according to their needs.

D. Microkernel Architecture Makes EasiEI More Scalable and Devices Easy to Communicate With

To reduce the coupling between functional components, we assume there is no direct information transfer among the components. Therefore, we introduce the kernel part of the microkernel architecture, task status table (TST), as shown in Fig. 1. The above four functional components accomplish information exchange by performing CRUD (create, read, update, and delete) operations on the TST in the device. The kernel is relatively stable and will not be constantly modified due to the expansion of scenario capabilities, and the plug-in modules can be continuously extended. This kernel design can efficiently shield the functional heterogeneity of the physical device.

Fig. 8 shows the details of TST. In order to make the scheduling simulation more realistic and to allow users to track the state of the target task in real time during the simulation, EasiEI provides the TST class to assist in multistate switching of tasks and to save multiple states of all tasks on the current device. Tasks with different states are stored in different task state tables. TST maintains task instances in each status using double circular linked list that are consumed by functional components. EasiEI allows functional components to control the state of an independently configurable number of tasks. Life cycles of the tasks in TST are divided into the following four phases: 1) *submit:* a task was submitted to TST by task generator or received from network by task receiver; 2) *pending:* a task is queued waiting for a specific event to



Fig. 8. Kernel: status table design.

occur; 3) *running:* a task was scheduled by task orchestrator; and 4) *dead:* a task was canceled, de-scheduled, or competed normally.

According to the capabilities and scenario requirements of the four queues, each task queue can plug-in different functional components.

- Submit list needs to wait for the task generator and task receiver to pass tasks. Under the influence of task generator's task probability distribution, task receiver's cache strategy, and network status, submit list sorts the tasks according to the time stamp of their arrival in the list.
- 2) Pending lists have many uses in task scheduling, especially in task interrupt handling and task synchronization. Tasks must often wait for certain events to occur, such as waiting for device to release resources, waiting for time to pass a fixed interval, waiting for a parent task to finish, waiting to offload to another edge device, and so on. The pending list is preordered according to different scheduling policies, and the task orchestrator only needs to pick the first task in the pending list and perform state transition and resource allocation when scheduling.
- The volume of the running list is affected by the computing capacity and concurrency of the current edge device.
- 4) The dead list is sorted by task end time, and the capacity can be adjusted according to the platform's log strategy. Tasks processed by different device functions require state transitions and list maintenance by the dominant device function.

In addition, users can use a building block approach to simulate functionally heterogeneous edge devices for different combinations of functional components.

A Tiny Example: Consider a simple example of a laboratory as shown in Fig. 9. The IoT devices in this scenario are temperature and humidity sensors installed in the lab to monitor the room temperature and humidity in real time and adjust the air conditioner intelligently. However, these sensors may not be embedded in the air conditioner, a classic way is using wireless access to connect them with the control center. Some other devices, such as an automatic door embedded a distance sensor to intelligently open or close the door, may not require data



Fig. 9. Real topology of example.



Fig. 10. Abstract topology and EasiEI device model of example.

communication and therefore do not require networking. Due to limited batteries, the IoT device needs to be integrated with an energy module. Edge computing devices are smartphones and computers, where computers are wired-connected and smartphones are wireless-connected. Smartphones also need to be combined with mobility components. High-performance labs are equipped with multiple servers to provide private cloud computing resources, and the servers are wired to the network. Given the limited computational capabilities of edge devices, collaborative scenarios involving data sharing, task offloading, and collaborative computing are observed among phones, computers, and private clouds.

To simulate it, sensors, devices, and cloud servers need to be modeled. For functionally heterogeneous devices, EasiEI's loosely coupled build form is shown in Fig. 10. The construction of cloud-edge-end devices can be accomplished by assembling functional components. Networked temperature and humidity sensors have at least three components: task receiving, task sending, and task generation. Non-networked sensors have only task generation and a simple task scheduling component. For functionally complex edge computing devices and server resources, four functional components are included: 1) task generation; 2) task acceptance; 3) task scheduling; and 4) task sending. By virtue of the inherent independence of the functional components, it is possible to dynamically add or remove the combination of these components within the device model, thereby achieving a dynamic simulation of various device types.



Fig. 11. Overall workflow of EasiEI platform.

Moreover, by inheriting and modifying the implementation methods of these components, a wide range of cross-device collaborative scenarios can be realized. Users have the ability to modify the generator and sender of sensors to accomplish environmental monitoring logging to the cloud. Phones, computers, and cloud systems possess the capability to simulate collaborative scenarios by making adjustments to the orchestrator and sender components. Furthermore, the cloud can leverage the Resource Unit to validate various resource combinations and allocation strategies.

E. Workflow of Functional Components in EasiEI

To better illustrate the workflow of the platform as shown in Fig. 11, the scenario setup and running process of EasiEI is described as follows.

- 1) Read the network topology data and configuration information, create the edge computing scenario network topology, which consists of multiple nodes and links. Then initialize the parameters of the five-layer network protocol stack according to the network configuration information.
- 2) Based on the complex scenario requirements, combine independent functional components on each node of the topology and encapsulate the nodes into heterogeneous edge computing devices with functions, such as sensing, storage, computation, and transmission.
- 3) Read the computing power configuration data of the heterogeneous edge devices in the topology and initialize the computing power deployment of the edge devices.
- 4) Read the task information in the data set, generate tasks on the edge devices, and drive the simulation running process by task management and operation.

To facilitate a comprehensive understanding of task management and operation in the simulation scenario, it is critical to have a clear grasp of the workflow involved in task processing. This workflow comprises several critical stages, such as task generation, distribution, scheduling, and execution, and is fundamental to ensure the efficient and effective execution of tasks in a distributed environment as depicted in Fig. 12. The colored blocks in the diagram represent the task entity information used during the state transition process as illustrated in Fig. 2. During the task state transition process, the maintenance entity responsible for task processing uses abbreviations of functional component names to identify which component is responsible for maintaining tasks and TSTs.

Tasks can be generated locally or received remotely and transition from the unsubmit state to the pending state through



Fig. 12. Task lifecycle process diagram.

the submit operation. When generated locally, the device reads the task time and execution information through the task generator and creates the task entity. When received remotely through the receiver, if the maximum waiting time has been exceeded, the task will be discarded. Both generation methods require an address judgment step. If the destination address matches the current device address, the task will be saved to the main task chain in the TST on the current device. Otherwise, the task will be forwarded to the destination address.

Then, orchestrator can judge the task's execution and time information based on the scheduling algorithm. Tasks that meet the scheduling algorithm will transition to the running state through the orchestrate behavior. If local computing resources are scarce or the waiting time for the task exceeds the maximum, orchestrator will transition the task from the pending state to the dead state through the fail operation. Orchestrator completes task execution simulation by sleeping and simulating the execution time difference between task end and start time. When Orchestrator judges that the task has executed for an equivalent simulated execution time, the task transitions to the dead state through the accomplish operation.

For tasks that have not been able to complete the execution process normally and are in the dead state, the time and execution information of the task can be extracted and reinput to the generator through the resubmit operation to reconvert the task to the pending state. Alternatively, after the generator has finished generating, the task can be temporarily stored in the auxiliary task table in the TST, sent to a remote node by the sender through the send operation, and initialized to the unsubmit state.

IV. CASE STUDIES

Computing paradigms in edge scenarios can be classified into three categories: 1) centralized; 2) semi-decentralized or semi-centralized; and 3) decentralized. The semi-decentralized can actually be directly inherited from the centralized by adding several domain coordinators, such as Cloudlet [34]. In this article, we implement the centralized and decentralized paradigms. The experiment configuration for our platform consists of Ubuntu 18.04 as the underlying operating system. Additionally, it needs support for the native NS3 environment and the Boost Serialization Library. We first provide a case to verify the ground truth of EasiEI in Section IV-A, which belongs to the centralized. Then, we further implement a semidecentralized scenario in Section IV-B. a similar concept can refer to the Computing First Networking [35]. In Section IV-C,



Fig. 13. Google scenario topology.

we also implement a Monte Carlo simulation coming from the reliability analysis realm to prove the flexibility of EasiEI. All subsequent experimental scenarios and functional component operations follow the same workflow as described in Section III-E.

A. Case Study One: Centralized Computation Resource Scheduling in Dynamic Scenario

To verify the ground truth of EasiEI, we adopt Google Cluster traces that consists of 12 500 physical machines connected via a high-speed wired network [36]. We select the top ten machines from the stable machine cluster with the most valid information and no updates or deletions as the stable task receivers. Like many cloud systems, Google cluster architecture consists of a single scheduling host (device 0) and many service nodes (SNs) as shown in Fig. 13. Each SN receives job requests from scheduling host that contain meta-information required for single-machine scheduling. In this experiment, a scheduling strategy of FIFS, and high-priority response was used for all SNs. Moreover, the transmission delay between devices was set to 2 ms. The task arrival density within the data set is depicted in Fig. 14, revealing a noticeable spike in cluster requests around the 1500-second mark.

The task execution time was obtained by processing the task state, and the local task execution time was obtained by subtracting the termination timestamp from the running timestamp. The simulation process assumed that task execution time was stable once resource requests were fulfilled. Tasks that failed to receive scheduling within the timeout period or failed due to insufficient resources upon receiving scheduling were deemed failures and were discarded.

Based on the initialization information of the devices in the data set, the ratios of the total CPU resource and memory resource were set to 0.5 and 0.2493, respectively. These resource quantities were normalized based on the device with the maximum capacity in the data set, which was scaled to 1.0. Task resource requests were represented as a binary tuple of CPU and memory, and the density plot of resource requests is



Fig. 14. Arrival density of task.



Fig. 15. Task resource request.

depicted in Fig. 15. The majority of the requested tasks were memory-intensive, requiring a significant amount of memory resources during task execution.

Fig. 16 displays the resource changes during the simulation process. At 2000 s, nodes 2, 6, 7, 8, and 9 had their memory reduced to near zero. Additionally, there was often a surplus of CPU resources while memory repeatedly reached its lowest threshold during the simulation. When the memory resource of the devices was increased to 0.5 while keeping the CPU resource constant, the task failure statistics for the two experiments are shown in Fig. 17. The graph records the total number of task failures in each time zone during the simulation, with a statistical interval of 250 s. Both experiments reached the peak of failed tasks at 2000 s. However, when the total memory resources of the devices were increased while the CPU remained constant, the number of task failures at the peak significantly decreased.

B. Case Study Two: Semi-Decentralized Service Provision in Dynamic Scenario

Compared with the centralized cloud computing, the edge computing scenario lacks stable resource support with high



Fig. 16. Remaining computation resources of nodes.



Fig. 17. Task failure comparison under different memory limits.

computing power, which makes it difficult to realize a global centralized scheduling scheme. The semi-decentralized scheduling scheme divides the global area into multiple domain, like edge cloud implemented in iFogSim [19]. Each domain contains a coordinator that monitors the domestic information, and the SN status is synchronized between domains through the coordinators. From the perspective of global information, it consists of multiple domains that are scheduled through distributed scheduling. From the intradomain information point of view, the domains are controlled by the coordinators through a centralized scheduling scheme. To reflect the complexity of the scenario in complex edge scenarios and the scalability of EasiEI for the simulation of multiple scheduling frameworks, this experiment is performed for



Fig. 18. Network topology.



Fig. 19. EasiEI abstract devices.

the simulation comparison of semi-decentralized scheduling frameworks.

Fig. 18 shows this scenario's topology whose data is adopted from the real-word ISP network, i.e., rocketfuel [37]. It contains three types of nodes: 1) SN (green double-circle nodes); 2) user (red nodes, service requesters); and 3) coordinators (Coord, blue double-circle nodes). Each domain is formed by combining a single Coord and multiple SN nodes. Users' requests follow the shortest path to access the services maintained on SN.

We select nodes 8, 125, 147, and 164 from this topology as Coord nodes and label them as CoordC. 12, 51, 78, 90, 93, 116, 196, 198, 216, and 226 are selected as SN nodes and labeled as SCN in Fig. 18. In this simulation, all users access the service ten times per minute, obeying Poisson distribution. And the time interval of the next visit obeys exponential distribution. To benchmark performance, the network latency between each pair of hosts is set to 2 ms and each host's bandwidth is 1.5 Gb/s. The task load is 8 Mb, and the default execution is 5 ms. All peers start at the same time and run for 100 simulated seconds. Max concurrency represents the maximum number of concurrency that can be supported by the node's service capacity. Deadline threshold represents the maximum waiting time for the task. A task fails if the total delay from the time it is generated by the user to the time it is scheduled by the SN exceeds the threshold.

EasiEI uses four functional components to form each of the three types of nodes into three functional types of devices, as shown in Fig. 19.



Fig. 20. Task execution statistics.

By redefining the four functional components of these three types of nodes, perception, and collaboration within the scenarios can be achieved. In the experiment, Coord use a latency threshold to shape the domain range, which means users out of the "latency circle" can be judged to out-ofservice. First, SN nodes monitor and generate heartbeat data in 325 ms intervals via task generator which obtain their own available computing resource status and packetize them into packets and send them by sender to the Coord in the domain using UDP protocol. All Coord have decision capabilities. After receiving heartbeat by receiver, the Coord node decodes the information and calculates the transmission delay of the packet to estimate the service delays. The Coord node then saves this delay information in the sender's sending table to provide an indicator when users accessing the services. When a task is received from a user via receiver, Coord looks up the send table in sender and completes the distribution of the task and load balancing of the network according to the link bandwidth and computing resources of SN. When the SN node receives a task forwarded by Coord through receiver, SN node allocates computational resources for the task. The execution statistics of the tasks are shown in Fig. 20.

To investigate the impact of different computation support, perception frequency, and task time constraints on the number of task successes and to assess the scalability of the distributed framework, two sets of experiments were conducted separately. The results of the first set of experiments, shown in Fig. 20, demonstrate the fluctuation of task success with changes in the maximum task waiting time under different concurrency



Fig. 21. Task success variation under different perception frequencies.

degrees. Concurrency, as one of the bottlenecks of the scheduling framework, has a significant impact on task scheduling and response. Increasing the concurrency of the device can significantly improve the system's service capability. However, the number of task successes plateaus when the maximum task wait time reaches 23 ms. As the longest waiting time increases, the latency constraint on task response decreases, resulting in computation resource contention in the service circle, which in turn reduces the number of successes.

To investigate the effect of perceived frequency on service response, a second set of experiments was conducted, and the results are shown in Fig. 20. As the maximum waiting time grows to 23 ms, the effect of perceived frequency on the number of task successes gradually decreases. The primary reason for this is that the data freshness of the Coord node has a more significant impact on service response when the service constraint is more stringent. As the service constraint gradually relaxes as shown in Fig. 21, the scheduling system can increase efficiency and reduce costs by reducing the sensing frequency.

In conclusion, the results of our experiments show that concurrency, latency, and data freshness are essential factors that significantly impact the success of task scheduling and response in a distributed framework. Therefore, when designing and implementing such a framework, it is crucial to carefully consider these factors and adjust the deployment parameters accordingly. With the help of EasiEI, these parameters can be fine-tuned to improve service quality while reducing service costs, resulting in a more scalable and efficient service delivery system. This research provides valuable insights into the optimization of distributed frameworks, which can be applied to a wide range of applications, including edge computing, IoT, and cloud computing.

C. Case Study Three: Monte Carlo Support

In the edge scenario, in addition to the limited computing resources, the communication resources are also changing in real time, which leads to the high computing delay of the services. In particular, each edge node is usually responsible for more than one service and service provider, which means that there are many factors causing the change of its available



Fig. 22. Real topology.



Fig. 23. EasiEI abstract topology.

resources. Calculating service reliability given a precise model in this scenario becomes vital. However, centralized methods in this scenario are not flexible due to frequently synchronizing of the massive edge nodes. Also, searching space becomes tremendous due to the resource changing states.

According to [15], a broadly used technology to improve the reliability of a collaborative service is service redundancy. Fig. 22 gives the topology of the scenario, where each path from IoT sources to the Cloud contains three sequential subservices or subtasks of a same service. The simplified topology of one path based on the EasiEI architecture is shown in Fig. 23. The dashed link indicates that the devices are connected via a wireless network while the solid link indicates a wired network. The setup of a chained task scheduling scenario was achieved through modifications made to the orchestrator.

In this example, the sensor on the left generates task data and sends three copies of the task to the three links connected with it. Accept tuning as the task goes through the computation device required in the directed cycling graph. Otherwise, continue to forward until the task ends. If the task is completed within the maximum time threshold, it is considered successful; otherwise, it fails.

Algorithm 1 gives the flow of the Monte Carlo simulation implemented in EasiEI.

To reflect the flexible network scenario simulation and diverse heterogeneous device simulation. In this experiment, bandwidth distribution sampling is adopted to represent the variation of available bandwidth which in view of the dynamic nature of network. In view of the dynamic nature of computing resources, the probability distribution of availability is summarized through historical data modeling. The network bandwidth table and available computation resource table are shown in Tables II and III.

Table II describes the probability distribution of bandwidth available to each link in the current network. Table III shows the probability distribution of computing resources available to each computing device.

The reliability or success rate table for each round of 4000 tasks is shown in Table IV, where T is the given time

Algorithm 1 Monte Carlo Simulation

Require:

Initial *Computation resource*, *Bandwidth distribution*; Input *Task*;

Ensure: Reliability;

- 1: $t = \sum lead$ time + trans time + computing time;
- 2: for k = 1 : 4000 do
- 3: Create *TASK*₀ at the client's (*task generator* at sensor)
- 4: Decide whether need transmission or computing according to device category, computing resource and bandwidth distribution (*task receiver* at edge devices or transmission site);
- Orchestrate task according to task requirement and computing resources(*task orchestrator* at edge devices and cloud);
- Offload task according to DAG requirement (*task sender* at edge devices);
- 7:
- 8: until finish the sequential task chain of a size three (*task orchestrator* at cloud);
- 9: Recording time cost *t* in total (*task orchestrator* at cloud);
- 10: **if** $t \leq Maximum$ time threshold **then**

11: Success + = 1

12: **else**

13:

Defeat + = 1

- 14: **end if**
- 15: end for

16: $Reliability = \frac{Success}{Success+Defeat};$

TABLE II Bandwidth Capacity Probability

branch	land time	capaci	capacity				
Drahen	ieau time -	0	1	2	3	4	5
i_1	2	0.02	0.01	0	0.04	0	0.93
i_2	1	0.01	0.03	0.02	0.94	0	0
i_3	1	0.01	0.02	0	0.04	0.93	0
i_4	1	0.03	0.01	0	0.06	0	0.9
i_5	2	0.01	0.01	0.02	0	0.96	0
i_6	1	0.04	0.01	0.95	0	0	0
i_7	1	0.02	0.02	0	0.02	0.94	0
i_8	1	0.03	0.01	0.01	0.95	0	0
i_9	2	0.03	0	0.01	0.03	0.93	0
i_{10}	1	0.02	0	0.01	0.02	0.95	0

TABLE III Available Nodes Capacity Probability

nodes	0/І	capacity						
noues	0/1	0	1	2	3	4	5	6
s_0	0.7	0.0	0.01	0.09	0.26	0.37	0.20	0.07
s_{11}	1.3	0.0	0.01	0.09	0.26	0.37	0.20	0.07
s_{12}	1.0	0.0	0.01	0.09	0.26	0.37	0.20	0.07
s_{13}	$\rightarrow 0$	0.0	0.01	0.09	0.26	0.37	0.20	0.07
s_{21}	1.3	0.0	0.01	0.09	0.26	0.37	0.20	0.07
s_{22}	$\rightarrow 0$	0.0	0.01	0.09	0.26	0.37	0.20	0.07
s_{31}	1.3	0.0	0.01	0.09	0.26	0.37	0.20	0.07
s_{32}	$\rightarrow 0$	0.0	0.01	0.09	0.26	0.37	0.20	0.07

threshold and C is the data transmission size. Results show that simulation results are roughly equal to the theoretical results.

TABLE IV EASIEI MONTE CARLO SIMULATION RESULTS

EasiEI	C=14	C=15	C=16
T=21	0.85650	0.76025	0.67125
T=22	0.91300	0.86800	0.82575
T=24	0.97600	0.94875	0.94000
T=25	0.98050	0.97550	0.97100

V. CONCLUSION AND FUTURE WORK

In edge computing scenarios, there are various complexity factors, such as scenario metrics and coordination. In such complex environments, simulation can efficiently evaluate deployment strategies. However, current simulation platforms mostly focus on modeling simple scenarios and still rely on device-level modeling with high coupling among functions. This fails to consider the need for functional independence and replaceability in complex scenarios, making it difficult to support complex edge scenario simulation.

To address these issues, we propose the EasiEI simulation platform. EasiEI elevates the modeling granularity to the level of device functionality and adopts a microkernel architecture design that considers the independence and replaceability of resources and functions. The platform can flexibly combine different functional components to meet the simulation needs of heterogeneous devices and enables functional replacement through a plug-and-play approach. Furthermore, users can customize new functional components and embed them into the microkernel architecture to meet the requirements of complex scenarios.

To better address the challenges of task real execution and complex relationships between executable task entities in complex edge computing scenarios, EasiEI will design a framework for managing executable task entities and task execution statuses. Leveraging the microkernel architecture design at the current functional level, the framework will seamlessly integrate with the existing platform, avoiding the impact of reconstructing from a simulator to a mixed execution architecture.

REFERENCES

- W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
- [2] K. K. Patel and S. M. Patel, "Internet of Things-IOT: Definition, characteristics, architecture, enabling technologies, application & future challenges," *Int. J. Eng. Sci. Comput.*, vol. 6, no. 5, pp. 6123–6131, 2016.
- [3] L. S. Vailshery. "Number of IoT connected devices worldwide 2019– 2021, with forecasts to 2030." 2022. [Online]. Available: https://www. statista.com/statistics/1183457/iot-connected-devices-worldwide/
- [4] L. Chao, X. Peng, Z. Xu, and L. Zhang, "Ecosystem of things: Hardware, software, and architecture," *Proc. IEEE*, vol. 107, no. 8, pp. 1563–1583, Aug. 2019.
- [5] J. Qi. "A curated list of awesome edge computing, including frameworks, simulators, tools." 2022. [Online]. Available: https://github.com/ qijianpeng/awesome-edge-computing
- [6] A. M. Law, "How to build valid and credible simulation models," in Proc. Winter Simulat. Conf. (WSC), 2019, pp. 1402–1414.
- [7] S. Svorobej et al., "Simulating fog and edge computing scenarios: An overview and research challenges," *Future Internet*, vol. 11, no. 3, p. 55, 2019.
- [8] M. Iorga et al., Fog Computing Conceptual Model. Gaithersburg, MD, USA: Special Publication Nat. Inst. Standards Technol. (NIST SP), 2018.

- [9] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.
- [10] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, Feb. 2018.
- [11] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.
- [12] C. Jiang et al., "Energy aware edge computing: A survey," Comput. Commun., vol. 151, pp. 556–580, Feb. 2020.
- [13] H. Gao, W. Huang, and Y. Duan, "The cloud-edge-based dynamic reconfiguration to service workflow for mobile eCommerce environments: A QoS prediction perspective," ACM Trans. Internet Technol., vol. 21, no. 1, pp. 1–23, 2021.
- [14] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energyefficient offloading for DNN-based smart IoT systems in cloud-edge environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 683–697, Mar. 2022.
- [15] L. Chen, J. Qi, X. Su, and R. Wang, "REMR: A reliability evaluation method for dynamic edge computing network under time constraint," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 4281–4291, Mar. 2023.
- [16] X. Deng, J. Li, L. Shi, Z. Wei, X. Zhou, and J. Yuan, "Wireless powered mobile edge computing: Dynamic resource allocation and throughput maximization," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 2271–2288, Jun. 2022.
- [17] A. M. Law, W. D. Kelton, and W. D. Kelton, Simulation Modeling and Analysis. New York, NY, USA: McGraw-Hill, 2007.
- [18] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.
- [19] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw. Pract. Exp.*, vol. 47, no. 9, pp. 1275–1296, 2017.
- [20] C. Sonmez, A. Ozgovde, and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of edge computing systems," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 11, 2018, Art. no. e3493.
- [21] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the NS-3 simulator," *SIGCOMM Demonstration*, vol. 14, no. 14, p. 527, 2008.
- [22] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proc. 1st Int. Conf. Simulat. Tools Techn. Commun.*, *Netw. Syst. Workshops*, 2008, pp. 1–10.
- [23] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping softwaredefined networks," in *Proc. IEEE Colombian Conf. Commun. Comput.* (COLCOM), 2014, pp. 1–6.
- [24] T. Qayyum, A. W. Malik, M. A. K. Khattak, O. Khalid, and S. U. Khan, "FogNetSim++: A toolkit for modeling and simulation of distributed fog environment," *IEEE Access*, vol. 6, pp. 63570–63583, 2018.
- [25] D. Camara, H. Tazaki, E. Mancini, T. Turletti, W. Dabbous, and M. Lacage, "DCE: Test the real code of your protocols and applications over simulated networks," *IEEE Commun. Mag.*, vol. 52, no. 3, pp. 104–110, Mar. 2014.
- [26] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "MaxiNet: Distributed emulation of software-defined networks," in *Proc. IFIP Netw. Conf.*, 2014, pp. 1–9.
- [27] A. Coutinho, F. Greve, C. Prazeres, and J. Cardoso, "FogBed: A rapidprototyping emulation environment for fog computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2018, pp. 1–7.
- [28] C. Mechalikh, H. Taktak, and F. Moussa, "PureEdgeSim: A simulation toolkit for performance evaluation of cloud, fog, and pure edge computing environments," in *Proc. Int. Conf. High Perform. Comput. Simulat.* (HPCS), 2019, pp. 700–707.
- [29] M. Peuster, J. Kampmeyer, and H. Karl, "ContainerNet 2.0: A rapid prototyping platform for hybrid service function chains," in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, 2018, pp. 335–337.
- [30] C. Wang, R. Li, W. Li, C. Qiu, and X. Wang, "SimEdgeIntel: A open-source simulation platform for resource management in edge intelligence," J. Syst. Archit., vol. 115, May 2021, Art. no. 102016.
- [31] J. Byrne et al., "RECAP simulator: Simulation of cloud/edge/fog computing scenarios," in *Proc. Winter Simulat. Conf. (WSC)*, 2017, pp. 4568–4569.

- [32] C. Savaglio and G. Fortino, "A simulation-driven methodology for IoT data mining based on edge computing," ACM Trans. Internet Technol., vol. 21, no. 2, pp. 1–22, 2021.
- [33] G. Kecskemeti, G. Casale, D. N. Jha, J. Lyon, and R. Ranjan, "Modelling and simulation challenges in Internet of Things," *IEEE Cloud Comput.*, vol. 4, no. 1, pp. 62–69, Jan./Feb. 2017.
- [34] Z. Á. Mann, "Decentralized application placement in fog computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3262–3273, Dec. 2022.
- [35] M. Król, S. Mastorakis, D. Oran, and D. Kutscher, "Compute first networking: Distributed computing meets ICN," in *Proc. 6th ACM Conf. Inf.-Centric Netw.*, 2019, pp. 67–77. [Online]. Available: https://doi.org/ 10.1145/3357150.3357395
- [36] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format+ schema," Google, Inc., Mountain View, CA, USA, White Paper, 2011.
- [37] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with Rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2004.



Jiahao Wang received the B.S. degree in Internet of Things engineering from the University of Science and Technology Beijing, Beijing, China, in 2022, where he is currently pursuing the master's degree in computer science and technology under his advisor Prof. R. Wang.

His research interests include edge computing, simulation, and machine learning.



Xiao Su received the B.S. degree in information and computational science from Beijing Information Science and Technology University, Beijing, China, in 2020, and the M.S. degree in computer technology from the University of Science and Technology Beijing, Beijing, in 2023, under his advisor Prof. R. Wang.

His research interests include edge computing, simulation, and machine learning.



Rui Wang received the Ph.D. degree in pattern recognition and intelligent systems from Northwestern Polytechnical University, Xi'an, China, in 2007.

He is currently a Professor with the Department of Computer Science and Technology, School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China. His research interests include edge intelligence, mobile and ubiquitous computing, and distributed systems.



Jianpeng Qi received the Ph.D. degree in computer science and technology from the University of Science and Technology Beijing, Beijing, China, in 2023.

He is currently a Postdoctoral Fellow with the College of Computer Science and Technology, Ocean University of China, Qingdao, China. His research interests include edge computing, compute first networking, and distributed systems.



Yuan Yao received the Ph.D. degree in health management from the Academy of Military Medical Sciences, Beijing, China, in 2015.

She is currently the Deputy Director of the Institute for Hospital Management Research, Chinese PLA General Hospital, Beijing. Her research interests include healthcare management, medical intelligence, mobile health based on social networks, and medical big data analysis.