Hindawi
Security and Communication Networks
Volume 2020, Article ID 8825749, 17 pages
https://doi.org/10.1155/2020/8825749

Research Article

High-Performance Routing Emulation Technologies Based on a Cloud Platform

Jianyu Chen, Xiaofeng Wang, Leiting Tao, and Yuan Liu

School of Artificial Intelligence and Computer Science, Jiangnan University, Wuxi 214122, China

Correspondence should be addressed to Xiaofeng Wang; wangxf@jiangnan.edu.cn

Received 14 August 2020; Revised 3 October 2020; Accepted 10 October 2020; Published 24 October 2020

Academic Editor: Xiaolong Xu

Currently, the emergence of edge computing provides low-latency and high-efficiency computing for the Internet of Things (IoT). However, new architectures, protocols, and security technologies of edge computing need to be verified and evaluated before use. Since network emulation based on a cloud platform has advantages in scalability and fidelity, it can provide an effective network environment for verifying and evaluating new edge computing technologies. Therefore, we propose a high-performance emulation technology supporting the routing protocol based on a cloud platform. First, we take OpenStack as a basic network environment. To improve the performance and scalability of routing emulation, we then design the routing emulation architecture according to the software-defined network (SDN) and design the cluster scheduling mechanism. Finally, the design of the Open Shortest Path First (OSPF) protocol can support communication with physical routers. Through extensive experiments, we demonstrate that this technology not only can provide a realistic OSPF protocol but also has obvious advantages in the overhead and performance of routing nodes compared with those of other network emulation technologies. Furthermore, the realization of the controller cluster improves the scalability in the emulation scale.

1. Introduction

The Internet of Things (IoT) technology is an extension of the Internet and refers to the billions of physical devices around the world that are now connected to the Internet. Because of the arrival of super low-cost computer chips and the development of wireless networks [1], it is possible to turn anything into a part of the IoT, such as wearable Point-of-Care Testing (POCT) systems [2], which provides convenience for immediate diagnostic results. With the rapid growth of the IoT, a large amount of edge data has been explosively generated. Edge computing has emerged out of necessity to process these data [3–5]. Edge computing avoids problems of the slowdown in broadband expansion and delays in data transmission between central cloud servers and edge devices. A large amount of edge data can be processed at the edge-layer, which realizes low-latency and high-efficiency data processing.

At present, an increasing number of protocols and architectures for edge computing have been proposed [6–8]. Therefore, there is an urgent need for a testing platform for edge computing that provides network virtualization and computing realism at a low cost to test and verify these new protocols and architectures. Zeng et al. [9] show that a testing platform for edge computing mainly is composed of two aspects: the network and computing. It also proposes EmuEdge, a hybrid emulator based on Linux *netns* and Xen for full-stack edge computing emulation. Coutinho et al. [10] present a framework architecture to create virtualized fog environments that help researchers test and evaluate fog applications.

Although the above two methods can provide solutions for evaluating and testing edge computing, they have performance bottlenecks in network virtualization. In [9], the approach uses Linux *netns* and Xen to provide the basic environment for a testing platform for edge computing. However, the Xen virtualization is not as convenient and efficient as the cloud platform [11]. Coutinho et al. [10] use Mininet [12] to build an emulation network for fog virtualized environments. Compared with the cloud platform,

Mininet may yield erroneous results if they inadequately manage the interaction between the emulation environment and the operating system that lies beneath the application and the virtual links [13].

Cloud platforms [14, 15] can provide a high-fidelity basic network environment for network emulation. OpenStack has become the standard for cloud platforms because it is open-source, scalable, and flexible [16–18]. Therefore, network emulation based on OpenStack is a research hotspot. At present, there are two routing emulation technologies based on OpenStack. One is to use the "qrouter" component of OpenStack to realize the routing emulation. Mengdong et al. [19] proposes a high-throughput routing emulation solution based on the "qrouter," but due to the lack of routing protocol design, its application scenarios are limited. Another is to integrate virtualization technologies, such as Kernel-based Virtual Machine (KVM) [20] and Docker [21], with routing software technologies, such as Quagga [22], XORP [23], and Click [24], to realize the routing emulation. The solution adopts a closed router architecture closely coupled with physical resources, operating systems, and network applications. Therefore, each routing node requires independent system space and relies on a virtual machine, which means that these solutions need to occupy a large number of physical resources to deploy a large-scale network emulation, thereby increasing costs. In terms of performance, the communication between virtual machines in OpenStack requires multiple forwarding of Linux-bridge [25] and Open vSwitch (OVS) [26], which leads to the problem of poor link performance of the routing emulation solution.

In this case, to reduce the overhead and improve the performance, this paper combines the software-defined network (SDN) [27] and OpenStack technologies to propose a high-performance routing emulation technology. OpenStack technology can provide a low-cost and realistic basic network emulation environment. SDN technology can optimize the architecture of this routing emulation technology. The routing node of the optimized architecture is responsible only for processing data packets according to flow rules distributed by the controller. Therefore, each routing node does not need an independent system space, which reduces the emulation overhead. In terms of the performance, because the routing node of the optimized architecture is implemented by the OVS bridge, it has high forwarding performance. We design the Open Shortest Path First (OSPF) protocol, which is a dynamic protocol and transmits routing information. In addition, we also design a cluster scheduling mechanism to schedule multiple controllers to provide the control service. The main contributions of this paper are summarized as follows:

(1) We propose a high-performance routing emulation architecture. This architecture can effectively reduce emulation overhead and improve emulation performance. In addition, the design of the load balancing of links can improve the throughput in the situation of congestion.

(2) We realize the OSPF protocol for this routing emulation technology. The realization of the OSPF protocol not only enables the routing node to communicate with the physical router but can also be used to analyse the OSPF attack.

(3) We design the controller cluster to strengthen the ability of control. The controller cluster ensures that all routing nodes can be controlled by multiple controllers simultaneously, which significantly improves the scalability in the emulation scale.

The rest of the paper is organized as follows. Section 2 describes some related work about network emulation. In Section 3, we introduce the architecture, routing function, and OSPF function. We introduce the load balancing of links in Section 4 and the load balancing of controllers in Section 5. Section 6 evaluates its function and performance. We conclude the paper in Section 7.

2. Related Work

Network emulation is one aspect of a testing platform for edge computing. The related routing emulation solutions are as follows.

Zeng et al. [9] present a hybrid emulator based on Linux *netns* and Xen for full-stack edge computing emulation, which is called EmuEdge. EmuEdge adopts *netns* for network-bounded node virtualization and provides full-system virtualization with Xen to combine the emulation of both the computation and network plane in an edge computing platform. Coutinho et al. [10] present a framework architecture to create fog testbeds in virtualized environments, and it uses Mininet to build an emulation network. Because the two solutions do not combine the cloud platform, they lack convenience, scalability, and scale.

NS3 [28], OPNET [29], and Glomosim [30] are widely adopted to simulate the network environment. Though these solutions can provide reproducible and convenient network simulation, they have defects in fidelity and ignore some essential details when acting. Therefore, these network simulators cannot offer realistic network results [31].

The cloud platform can provide an excellent essential environment for network emulation. In [19], an emulation technology is proposed based on a centralized routing engine and distributed router deployment. The technology uses the "qrouter" of Neutron [32] to construct a complex emulation network. However, this solution does not support the standard dynamic routing protocol, which leads to the inability of connected physical routers to learn routings from each other through the routing protocol and thus limits its application. Routing emulation software provides essential support for the routing emulation solutions based on cloud computing and virtualization technology. In [33], Quagga is deployed on a virtual machine, providing various routing function emulations. Huang et al. [34] use Linux Containers (LXC) [35] to provide a running environment for virtual routers and use the Xtensible Open Router Platform (XORP) or Click to build a routing node. However, the routing emulation solutions in [33, 34] build routing nodes by deploying routing emulation software on virtual machines,

which leads to defects in overhead and performance of routing nodes in the routing emulation network.

3. Routing Emulation Design

3.1. Architecture Design. OpenStack, as the mainstream cloud platform technology, can provide outstanding operation and a basic network environment for network emulation with its high scalability, high fidelity, and low cost [36]. SDN technology optimizes the routing emulation architecture by separating the forwarding layer and the control layer, effectively reducing the emulation overhead and improving the emulation performance [27]. Therefore, we propose a high-performance routing emulation architecture based on OpenStack and SDN technology. The architecture consists of a 5-layer structure of the interconnection layer, data forwarding layer, control layer, decision layer, and cluster scheduling layer.

This multilayer structure can refine tasks and improve the scalability and reliability of the system [37]. The interconnection layer is the bottom layer of the architecture and is responsible for interconnecting with the OpenStack virtual network. The data forwarding layer is responsible for processing the data packets according to the flow rules. The control layer and decision layer are a vital part of the architecture. The control layer processes the information submitted by the data forwarding layer according to the fine-grained requirements specified by applications of the decision layer and distributes flow rules to each routing node. The decision layer provides various functions for the system through various applications, including monitoring, routing, and OSPF. The cluster scheduling layer relieves the pressure of the controller by scheduling multiple controllers to provide control services at the same time and improves the scalability of the emulation scale.

Figure 1 shows the architecture of routing emulation technology. The detailed introduction is as follows.

(1) Interconnection layer: routing emulation technology realizes communication with the OpenStack virtual network through the interconnection layer. With Neutron API [38], the interconnection layer uses an OpenStack virtual network to add a network interface for routing nodes. The routing nodes can send data packets to the OpenStack virtual network through this network interface.

(2) Data forwarding layer: this layer is composed of OVS bridge devices, and each bridge device represents a routing node in the emulation network. The network interface of each routing node is connected to the OVS of Neutron through the interconnection layer. Each node processes the data packets submitted by the interconnection layer according to the flow table rules distributed by the upper layer and submits information such as port status, traffic conditions, and network requests to the control layer.

(3) Control layer: the routing emulation technology manages the whole emulation network through centralized control. As an open-source SDN controller based on Python, Ryu [39] facilitates the study and research of researchers. Therefore, we adopt Ryu as the controller of the control layer. Ryu is responsible for processing the information submitted by the data forwarding layer according to the fine-grained requirements specified by applications of the decision layer and distributing flow rules to each routing node.

(4) Decision layer: the decision layer is crucial in the routing emulation architecture and consists of a network awareness app, network monitoring app, routing app, OSPF app, and a database. Each app performs its duties to achieve different functions. The network awareness app obtains the topology information of the emulation network and writes it into the database. The network monitoring app monitors the traffic and flow rules information of each routing node in real time and writes it into the database. The routing app is responsible for calculating all forwarding paths in the emulation network and generating flow rules based on these paths. The OSPF app is responsible for providing standard OSPF dynamic routing protocol [40] and supporting routing exchanges with physical routers. The network awareness app and routing app work together to provide the routing function, which will be described in Section 3.2. In Section 3.3, we will introduce how to realize the OSPF protocol with the OSPF app. Section 4 will introduce the load balancing of links realized by the network awareness app, network monitoring app, and routing app.

(5) Controller cluster: the control layer and decision layer compose the routing controller, and a group of routing controllers composes the controller cluster. Each routing controller will monitor its load information (including the number of connected nodes and the number of managed IP addresses) and submit it to the cluster scheduling layer through the message queue of RabbitMQ [41].

(6) Cluster scheduling layer: according to the load information of all routing controllers, the cluster scheduling centre can provide the load balancing of controllers, which will be introduced in Section 5.

3.2. Realization of High-Performance Routing Function. The routing function is the primary function of the routing node, which allows different networks to communicate with each other. Therefore, we realize the routing function through the network awareness app and routing app of the decision layer. The workflow of each app is shown in Figure 2.

According to Figure 2, the network awareness app is responsible for drawing the topology of the whole network, and the routing app is responsible for generating and distributing flow rules. First, the network awareness app obtains connection information by polling each routing node and draws the topology of the whole network based on this

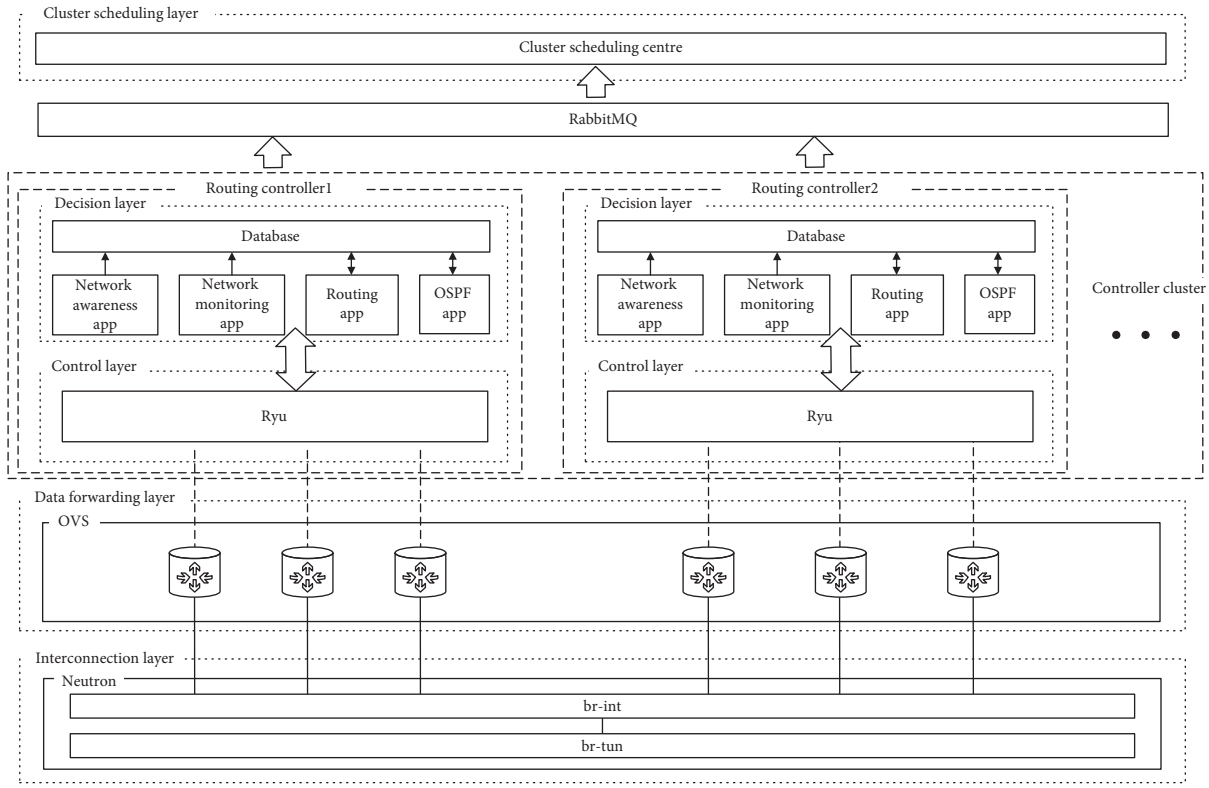

Figure 1: Architecture of routing emulation technology.

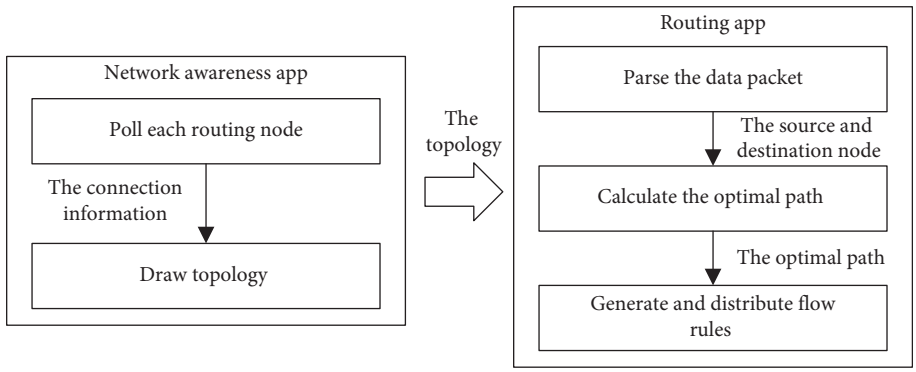

Figure 2: Workflow of the network awareness app and routing app.

information. Then, the routing app parses the data packet to obtain the information of the source and destination node and calculates the optimal path with the Dijkstra algorithm [42] between the two nodes according to the topology. Finally, the routing app generates flow rules based on the path and distributes them to the routing nodes in the path. Figure 3 shows the process of routing and forwarding.

As seen from Figure 3, when data packets arrive at the interconnection layer, they will be submitted to the data forwarding layer to match the flow rules (I in Figure 3). At the data forwarding layer, data packets will be matched to the flow rules based on the destination network, protocol type, and other fields. If the packets fail to match, the data forwarding layer will submit the header information of the packet to the control layer (II in Figure 3). Then, the control layer calculates the optimal path and generates flow rules with the network awareness app and routing app of the decision layer (III in Figure 3). Finally, the control layer distributes flow rules to routing nodes (IV in Figure 3). If the packets succeed to match, the TTL of the packets will be reduced by 1 to prevent the packets from generating loops in the network. The source media access control (MAC) of packets is modified to the MAC of the exit port. The destination MAC is modified to the MAC of the next hop. Then, the packets are forwarded to the exit port (V in Figure 3).

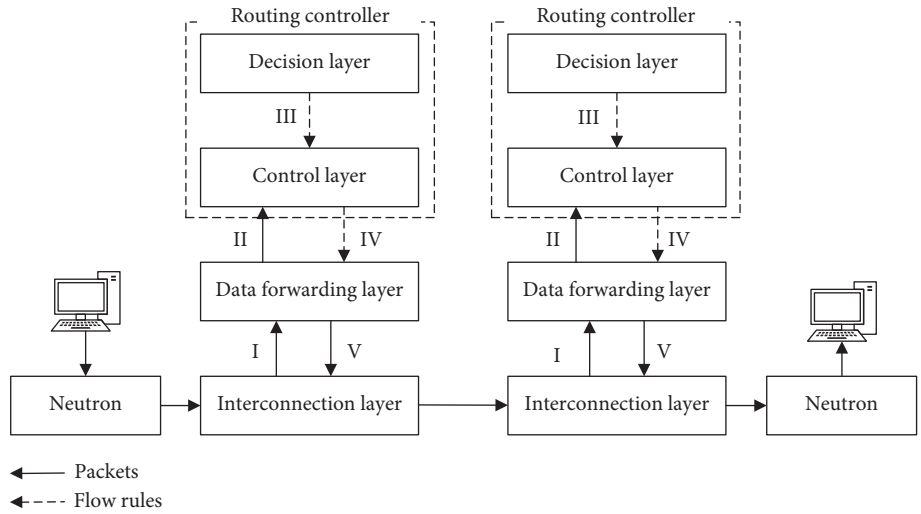

Figure 3: Process of routing and forwarding.

3.3. Realization of the OSPF Protocol. To support the construction of a complex emulation network with other emulation routers or physical routers through the OSPF protocol, we design the OSPF app at the decision layer. The OSPF protocol relies on five different types of messages to establish an OSPF neighbour adjacency and exchange routing information. The messages include Hello messages, Database Descriptor (DBD) messages, Link-State Request (LSR) messages, Link-State Update (LSU) messages, and Link-State Acknowledgement (LSAck) messages [40]. Therefore, the OSPF app also designs these five messages. Through neighbour discovery, database information exchange, routing calculation, and the flow rule distribution of the four steps, the app uses these messages to establish an OSPF neighbour adjacency and exchange routing information. The workflow of the OSPF app is shown in Figure 4.

The OSPF app calls different modules according to different message types. According to Figure 4, after the app parses the data packet, it will determine the type of the message. If it is a Hello message, the app will call the Hello module to respond to the message. If it is a DBD message, the app first calls the DBD module to support the exchange of link status information and then calls the LSR module to request detailed link status information. If it is an LSU message, the app will call the Daemon module to store the link status information and call the LSAck module to reply to the message. If it is an LSAck message, the app will call the Daemon module to record the message. Figure 5 shows the establishment process of OSPF.

When the routing node receives OSPF packets, the node will submit the packets to the routing controller for processing and reply to the neighbour router with the OSPF packets issued by the routing controller (I in Figure 5). When the routing controller receives OSPF packets, the OSPF app will construct corresponding reply packets according to the type of the OSPF message and issue them to the routing node (II in Figure 5). The specific process is as follows:

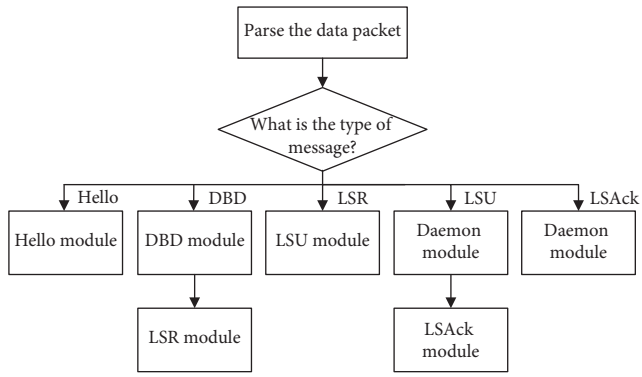

Figure 4: Workflow of the OSPF app.

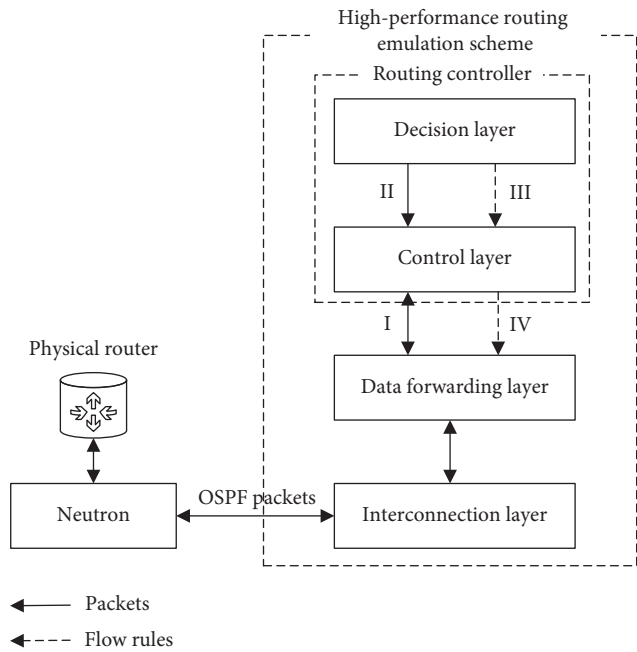

Figure 5: Establishment process of OSPF.

(1) Neighbour discovery: when the routing controller receives the neighbour Hello packets, the OSPF app constructs new Hello packets, and the control layer issues them to the routing node. Then, both sides establish an OSPF neighbour adjacency.

(2) Database information exchange: the OSPF app constructs DBD packets according to the link-state advertisement (LSA) information stored in the database to maintain the exchange of LSA information between the two sides. Then, according to the DBD packets of neighbour router, the app constructs LSR packets to request detailed LSA information.

(3) Routing calculation: according to all LSA information in the database, the OSPF app calculates all shortest path routings by the SPF algorithm [43] and stores them in the database (III in Figure 5).

(4) Flow rules distribution: after traversing the OSPF routes in the database, each route is resolved into forwarding flow table rules and distributed to routing nodes (IV in Figure 5).

Through the above four steps, the routing node establishes an OSPF neighbour adjacency and exchanges routing information with the physical router.

4. Realization of Load Balancing of Links

This paper proposes a load balancing of links mechanism to ensure that the emulation network can still obtain a high-throughput transmission path in the situation of congestion. The mechanism realizes load balancing of links through the network awareness app, network monitoring app, and routing app. The workflow of each app is shown in Figure 6.

According to Figure 6, the network awareness app is responsible for providing the topology of the emulation network. The network monitoring app is responsible for monitoring the load information of the emulation network (including the used bandwidth and the number of flow rules of a routing node) and calculating the load value of each path according to the topology and the evaluation model of the link. The routing app is responsible for calculating the optimal path according to the congestion avoidance algorithm based on the topology and the load value of each path. In Section 4.1, we will introduce the evaluation model of the link. The congestion avoidance algorithm will be introduced in Section 4.2.

4.1. Evaluation Model of the Link. In the process of network communication, bandwidth is the key factor affecting the transmission quality. The number of flow rules maintained by each routing node can reflect the load of the routing node. Therefore, we use the bandwidth and number of flow rules as two parameters of the model.

The bandwidth directly affects the speed of data transmission on the link. The ratio of used bandwidth can effectively reflect the busy degree of the link. In general, selecting the link with lower used bandwidth can yield higher-quality data transmission. The bandwidth used in the link can be obtained by the network monitoring app polling each routing node with the OpenFlow protocol [44]. We use (1) and (2) to obtain the bandwidth load value $B_{(i,j)}$ of the link (i,j) (i, j represent routing nodes i, j, and they are two adjacent nodes), where bandwidth_i and bandwidth_j, respectively, represent the port bandwidth of the nodes i and j, $\text{BW}_{(i,j)}$ represents the bandwidth of link (i,j) (since one link has a node on the left and right and the port bandwidth of each node is assumed to be different, we need to obtain the minimum of them as the bandwidth of the link), $\text{BW}_{\text{actual}(i,j)}$ represents the bandwidth actually used of the link (i,j), $\text{BW}_{(x,y)}$ represents the bandwidth of link (x, y) (x, y represent two adjacent nodes), and {nodes} is the set of all routing nodes:

$$\text{BW}_{(i,j)} = \min\left(\text{bandwidth}_i, \text{bandwidth}_j\right), \quad i \neq j, \tag{1}$$

$$B_{(i,j)} = 1 - \frac{\text{BW}_{(i,j)} - \text{BW}_{\text{actual}(i,j)}}{\sum_{x,y \in \{\text{nodes}\}, x \neq y \text{ and } x \text{ is adjacent to } y} \text{BW}_{(x,y)}}, \quad 0 \leq \text{BW}_{\text{actual}(i,j)} \leq \text{BW}_{(i,j)}. \tag{2}$$

Another parameter is the number of flow rules. The more the flow rules are maintained by the routing node, the more the time packets spend matching them. We use (3) and (4) to obtain the flow rule load value $F_{(i,j)}$ of the link (i, j) (i, j represent routing nodes i, j, and they are two adjacent nodes), where f_i and f_j represent the number of flow rules maintained by the routing nodes i, j (one link has a node on the left and right, so we take the average value of both nodes as $fn_{(i,j)}$ for convenience of calculation), fn_x represents the number of flow rules maintained by the routing node x, and {nodes} is the set of all routing nodes:

$$fn_{(i,j)} = \frac{fn_i + fn_j}{2}, \quad 0 \leq fn_i, fn_j, i \neq j, \tag{3}$$

$$F_{(i,j)} = \frac{fn_{(i,j)}}{\sum_{x \in \{\text{nodes}\}} fn_x}. \tag{4}$$

We use (5) to obtain the load value of the link (i, j) named as $L_{(i,j)}$ by weighting the sum of $B_{(i,j)}$ and $F_{(i,j)}$, where the weights are k_1 and k_2:

$$L_{(i,j)} = k_1 * B_{(i,j)} + k_2 * F_{(i,j)}, \quad k_1 + k_2 = 1. \tag{5}$$

4.2. Congestion Avoidance Algorithm. In this section, we will introduce the congestion avoidance algorithm. First, we obtain the topology of the emulation network through the network awareness app and load values (calculated by (5)) through the network monitoring app. We then use them as the input parameters of the congestion avoidance algorithm to calculate the optimal path in the situation of congestion. The pseudocode of the algorithm flow is shown in Algorithm 1.

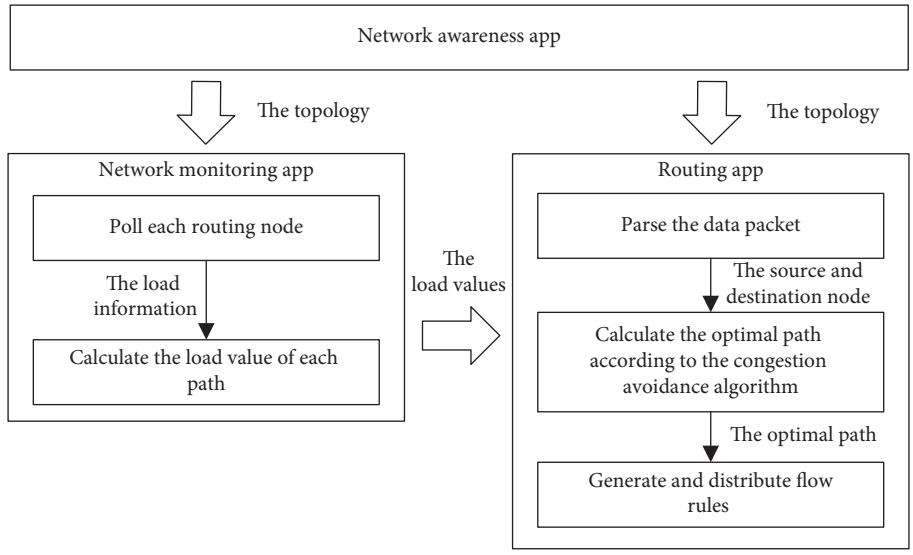

Figure 6: Workflow of the network awareness app, network monitoring app, and routing app.

Input:
nodes, weights, s, d;
//The nodes represents the set of all nodes, such as nodes = $\{R_1, \ldots, R_i, \ldots, R_n\}$. The weights represents the set of weights of all links, which are calculated by (5), such as weights = $\{(R_i, R_j) : L_{(Ri,Rj)}\}$. The s represents the source node. The d represents the destination node.
Output:
result;
//The result is a Boolean value, representing whether the optimal path is found.
Description:
Begin:
(1) result ← **True**;
(2) dict ← Dijkstra(nodes, weights, s);
(3) //nodes and s are used to initialize the sets S and U in the Dijkstra algorithm, S is the initial set of vertices (the initial set is $\{s\}$), and U is the set of vertices to be selected. Weights are used to calculate the weight between two nodes (if there is no edge between the two nodes, the weight is infinite). The set of shortest paths from node s to all other nodes can be obtained by the Dijkstra algorithm.
(4) **If** d **not in** dict **Then**
(5) result ← **False**;
(6) **Else**
(7) path ← dict[d];
(8) //Get the shortest path from node s to node d.
(9) Traverse all nodes in path and distribute flow rules;
(10) **End if**
(11) **Return** result;

Algorithm 1: Congestion avoidance algorithm.

5. Cluster Scheduling Mechanism

This paper adopts a routing emulation architecture that separates the forwarding layer and the control layer. The routing controller undertakes most of the work, including network awareness, network monitoring, and OSPF. Therefore, the pressure on the routing controller will increase as the number of routing nodes controlled by it increases. To solve this problem, this paper adopts the idea of distributed control and proposes a cluster scheduling mechanism. The mechanism balances multiple routing controllers to control more routing nodes simultaneously, which increases the scalability of the emulation network.

First, Section 5.1 describes how to realize the cluster scheduling mechanism. Then, we propose the evaluation model of the routing controller in Section 5.2 and propose the scheduling algorithm in Section 5.3.

5.1. Cluster Scheduling Centre. The cluster scheduling centre can manage multiple routing controllers simultaneously and evenly distributes the routing nodes in the emulation network to routing controllers for management.

As shown in Figure 7, routing controllers submit the load information (I in Figures 7 and 8) to the cluster scheduling centre through RabbitMq. The load information includes the number of routing nodes controlled by the controller, the number of IP addresses managed by the controller (because the routing node composed of the OVS bridge cannot set the IP address, the controller needs to manage the IP address of each access routing node), and the traffic of the southbound (it is the interactive traffic between the routing controller and routing nodes). The terminal is the entry point for creating a routing node. Whenever a routing node needs to be created, the terminal will send a request (II in Figures 7 and 8) to the cluster scheduling centre through API to obtain the idlest controller.

It can be seen from Figure 8 that when the load information is received, the centre will calculate the idlest controller according to the scheduling algorithm. In addition, it will return this controller information when receiving the request information.

5.2. Evaluation Model of the Routing Controller. We propose an evaluation model of the routing controller and calculate the load situation of each controller according to the load information submitted by the routing controller.

The routing controller needs to provide the Transmission Control Protocol/IP (TCP/IP) stack [45] and routing calculation function to connect routing nodes. The more routing nodes are connected to the controller, the busier the controller is. Therefore, we define the value RM_i, which represents the load situation of the ith controller on the number of routing nodes. The formula is shown as (6), where $controller_i$ represents the number of routing nodes connected to the ith controller and N represents that there are N controllers:

$$RM_i = \frac{controller_i}{\sum_{j=1}^{N} controller_j}, \quad 1 \le i \le N. \tag{6}$$

In the emulation network, the IP address of routing nodes is managed by the routing controller. The controller is responsible for answering ARP, ICMP, TCP, and UDP packets of these IP addresses. When the number of IP addresses is too large, it will reduce the speed of packet processing. Therefore, we define the value AM_i that represents the load situation of the ith controller on the number of IP addresses. The formula is shown as (7), where $address_i$ represents the number of IP addresses managed by the ith controller and N represents that there are N controllers:

$$AM_i = \frac{address_i}{\sum_{j=1}^{N} address_j}, \quad 1 \le i \le N. \tag{7}$$

When the southbound traffic is too heavy, the communication between controller and routing nodes will be blocked. Therefore, we define the value SF_i that represents the load situation of the ith controller on the southbound traffic. The formulas are shown as (8) and (9), where ST_i represents the southbound traffic of the ith controller, and it is calculated from the packet size passed within 30 seconds ($packet_t$ represents the packet size at t). SF_{max} represents the maximum size of the specified southbound traffic:

$$ST_i = \frac{packet_{t+\Delta t} - packet_t}{\Delta t}, \quad \Delta t = 30_{seconds}, \tag{8}$$

$$SF_i = \frac{ST_i}{SF_{max}}, \quad 0 \le ST_i \le SF_{max}. \tag{9}$$

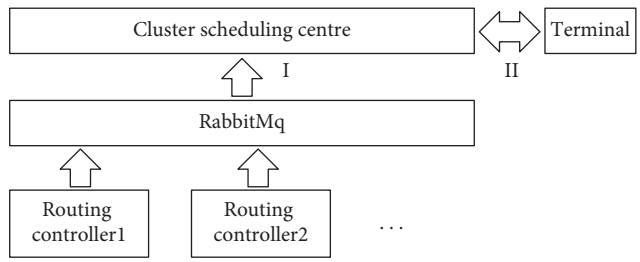

Figure 7: Process of information exchanging.

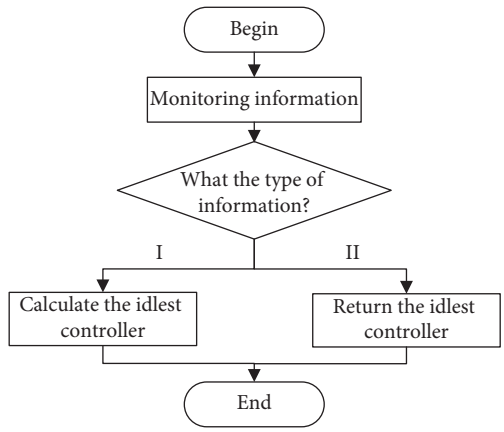

Figure 8: Workflow of the cluster scheduling centre.

5.3. Scheduling Algorithm. This section elaborates on a scheduling algorithm. Combined with the evaluation model in the previous section, we can calculate the current idlest controller. The pseudocode of the algorithm flow is shown in Algorithm 2.

In the process of Algorithm 2, we calculate the busyness of each controller by using (10), where busyness = {id_1: value(id_1), . . ., id_i: value(id_i), . . ., id_n: value(id_n)}:

$$\text{value}(id_i) = k_1 * RM(id_i) + k_2 * AM(id_i) + k_3 * SF(id_i), \quad \sum_{j=1}^{3} k_j = 1. \tag{10}$$

Finally, we traverse busyness to find the least busy controller and return it.

Input:
controllers, nodes_num, addresses_num, southbound_flow;
//controllers contains all the controllers' ID, such as {id_1, ..., id_i, ..., id_n}. nodes_num contains the number of routing nodes controlled by each controller, such as {id_1: N_1, ..., id_i: N_i, ..., id_n: N_n}, where N_i represents the number of routing nodes. addresses_num contains the number of IP addresses managed by each controller, such as {id_1: A_1, ..., id_i: A_i, ..., id_n: A_n}, where A_i represents the number of IP addresses. southbound_flow contains the size of southbound traffic of each controller, such as {id_1: T_1, ..., id_i: T_i, ..., id_n: T_n}, where T_i represents the size of southbound traffic.
Output:
free_controller;
//free_controller is a controller's ID that is the idlest among all controllers.
Description:
Begin:
(12) free_controller ← **None**;
(13) total_nodes ← SUM(nodes_num);
(14) //Sum the number of routing nodes.
(15) total_addresses ← SUM(addresses_num);
(16) //Sum the number of IP addresses.
(17) **for** con **in** controllers **do**
(18) A ← nodes_num[con];
(19) //Get the number of routing nodes controlled by the controller con from nodes_num.
(20) Compute the RM according to A and total_nodes based on formula (6);
(21) B ← addresses_num[con];
(22) //Get the number of IP addresses managed by the controller con from addresses_num.
(23) Compute the AM according to B and total_addresses based on the formula (7);
(24) C ← southbound_flow[con];
(25) //Get the size of the southbound traffic of the controller con from southbound_flow.
(26) Compute the SF according to C based on the formulas (8) and (9);
(27) Use formula (10) to obtain busyness according to RM, AM, and SF;
(28) **end for**
(29) free_controller ← MIN (busyness);
(30) //Get the least busy controller.
(31) **return** free_controller;

Algorithm 2: Controller scheduling algorithm.

6. Evaluation

In this section, we verify and evaluate the routing emulation technology of this paper. First, we verify the routing and forwarding function of routing nodes and OSPF function through experiments in Section 6.2. Then, we compare the routing emulation technology in this paper (named as *high-performance technology*), in [33, 34] (named as *virtualization technology*), and in [19] (named as *high-throughput technology*). Section 6.3 compares the cost of the three routing emulation technologies, and Section 6.4 compares the performance. Finally, in Sections 6.5 and 6.6, we, respectively, evaluated the effects of load balancing of links and load balancing of controllers, respectively.

6.1. Experimental Environment. In this paper, we use four Dell servers to build the OpenStack (version Queens) and use it as the basic environment of the experiment. The environment is shown in Figure 9, and the specifications of the servers are shown in Table 1.

The operating system of all the nodes is Ubuntu 16.04. We deploy the *high-performance technology* on the Compute1 node and deploy the *virtualization technology* or *high-throughput technology* on the Compute2 node for comparisons with our *high-performance technology* separately. R6 and R7 are Cisco's physical routers.

6.2. Verification of Routing Function Emulation. We build the emulation network, as shown in Figure 10, on the Compute1 node.

In Figure 10, VM1, VM2, and VM3, which represent clients, have IP addresses of 192.168.1.3/24, 192.168.6.3/24, and 10.1.9.3/24, respectively. One net represents a virtual network, and the details are shown in Table 2.

The emulation network is divided into a *high-performance technology* network and a physical network. The physical network is composed of two Cisco physical routers and connects to the *high-performance technology* network through the *net7* virtual network. The two networks use OSPF to exchange routing.

We take VM1 and VM2 as an example and use the traceroute tool [46] to test whether they can communicate with each other through forwarding routing nodes and whether the path the packet passes through is correct.

From Figure 11, we know that VM1 can communicate with VM2 and the path is correct. In the *high-performance technology* network, any two nodes can also communicate through the forwarding routing nodes to build an interconnected routing emulation network on OpenStack.

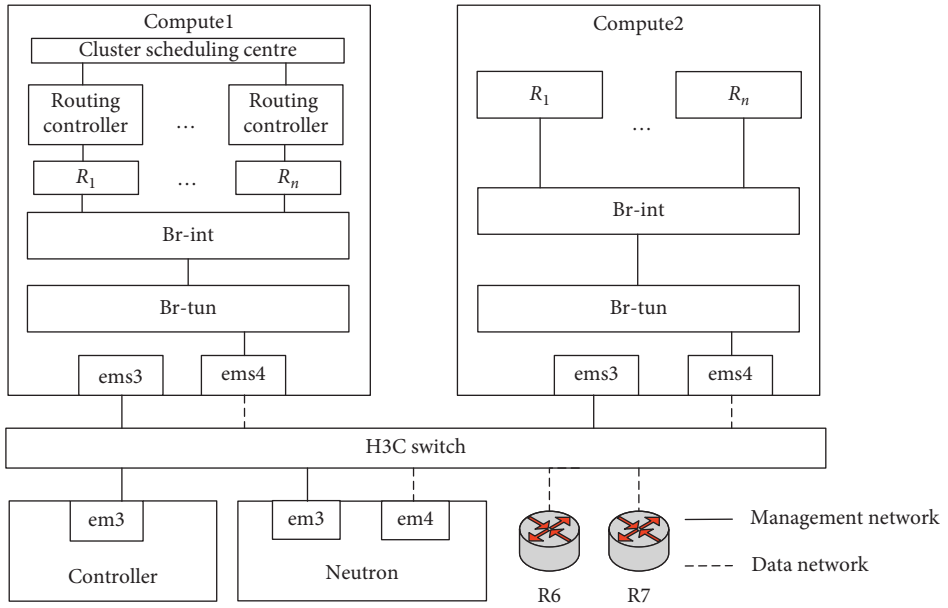

Figure 9: Experimental environment.

Table 1: Specifications of Dell servers.

Type	Controller node	Neutron node	Compute1 and Compute2 nodes
Model	R730	R730	R730
Core	8	6	6
Memory	64 GB	16 GB	64 GB
Disk	1 TB	1 TB	2 TB

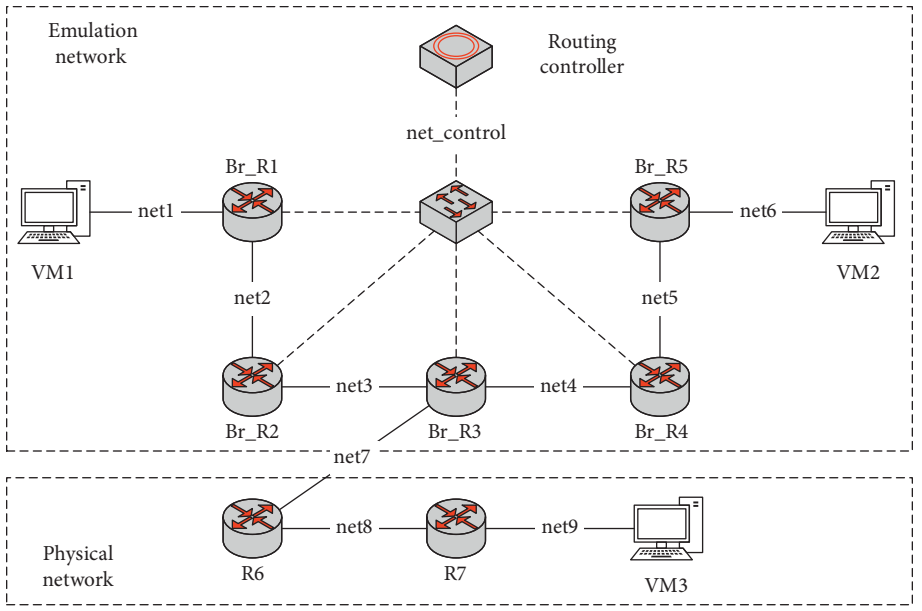

Figure 10: Topology for testing function.

We take VM1 and VM3 as an example and use the traceroute tool to test whether they can communicate over two kinds of networks and whether the path the packet passes through is correct.

From Figure 12, we know VM1 can communicate with VM3, and the path is correct. It can be proved that the emulation network constructed by the *high-performance technology* can exchange routings with the physical network by OSPF.

Table 2: Virtual network information.

Name	Segment
net1	192.168.1.0/24
net2	192.168.2.0/24
net3	192.168.3.0/24
net4	192.168.4.0/24
net5	192.168.5.0/24
net6	192.168.6.0/24
net7	10.1.7.0/24
net8	10.1.8.0/24
net9	10.1.9.0/24

```
root@vm1:~# traceroute 192.168.6.3
traceroute to 192.168.6.3 (192.168.6.3), 30 hops max, 60 byte packets
 1  host-192-168-1-4.openstacklocal (192.168.1.4)  24.889 ms  25.386 ms  25.389 ms
 2  192.168.2.4 (192.168.2.4)  24.761 ms  24.764 ms  24.759 ms
 3  192.168.3.4 (192.168.3.4)  8.722 ms  8.726 ms  8.720 ms
 4  192.168.4.4 (192.168.4.4)  25.237 ms  25.214 ms  25.234 ms
 5  192.168.5.4 (192.168.5.4)  25.269 ms  25.231 ms  25.237 ms
 6  192.168.6.3 (192.168.6.3)  2.307 ms  0.515 ms  0.522 ms
```

Figure 11: Traceroute path between VM1 and VM2.

```
root@vm1:~# traceroute 10.1.9.3
traceroute to 10.1.9.3 (10.1.9.3), 30 hops max, 60 byte packets
 1  host-192-168-1-4.openstacklocal (192.168.1.4)  17.850 ms  17.822 ms  17.819 ms
 2  192.168.2.4 (192.168.2.4)  17.754 ms  17.742 ms  17.735 ms
 3  192.168.3.4 (192.168.3.4)  10.034 ms  10.038 ms  10.031 ms
 4  10.1.7.4 (10.1.7.4)  0.476 ms  0.480 ms  0.480 ms
 5  10.1.8.4 (10.1.8.4)  0.868 ms  0.859 ms  0.856 ms
 6  10.1.9.3 (10.1.9.3)  1.014 ms  0.777 ms  0.765 ms
```

Figure 12: Traceroute path between VM1 and VM3.

In this section, we also verify whether the routing emulation technology in this paper supports OSPF protocol attack analysis. The adjacency attack [47] is a typical OSPF protocol attack method. We use VM1 and the routing node Br-R3 as an example to simulate this attack method.

The adjacency attack tampered with the routing table by simulating the virtual neighbour node of the OSPF router. Figure 13 shows that VM1 is simulated as the adjacent node of BR-R3 (192.168.3.100) and successfully establishes an OSPF neighbour with BR-R3 by sending OSPF messages, which verifies that the routing emulation technology in this paper supports OSPF protocol attack analysis.

6.3. Comparisons of Overhead in Routing Emulation. The overhead of the routing emulation technology determines the size of the network that can be emulated in the same physical resource. In this section, we compare the CPU and memory consumption of the virtualization technology, high-throughput technology, and high-performance technology.

Tables 3 and 4 show the results of comparing how much CPU and memory are occupied when routing nodes are forwarding packets. In terms of CPU, compared with the *virtualization technology*, *high-performance technology* and *high-throughput technology* have a significant advantage. When there are 10 nodes, the *virtualization technology* takes up 45.6% of the CPU, which is 50.6 times of the *high-performance technology*. In terms of memory, *high-performance technology* has a significant advantage. When there are 10 nodes, the *high-performance technology* takes up 78 MB of the memory, which is 97.9% less than the *virtualization technology* and 92.3% less than the *high-throughput technology*. According to the above comparison results, using *high-performance technology* to build an emulation network can effectively reduce the costs.

6.4. Comparisons of Performance in Routing Emulation. The throughput, delay, and packet loss rate determine the performance of a routing emulation system. In this section, we compare the *virtualization technology*, *high-throughput technology*, and *high-performance technology*.

To more accurately verify the advantages of the *high-performance technology* on the performance of the routing node, two experimental scenarios are designed: the multihop situation and concurrent situation. We design a linear topology with 10 routing nodes, as shown in Figure 14, and design a concurrent topology with 5 clients, as shown in Figure 15. We use the iPerf3 [48] tool to test the throughput and packet loss rate and use the Ping tool to test [49] the delay.

23 49.470000	192.168.3.100	192.168.3.4	OSPF	82 Hello Packet
24 49.643000	192.168.3.4	224.0.0.5	OSPF	94 Hello Packet
26 51.483000	192.168.3.100	192.168.3.4	OSPF	66 DB Description
28 53.500000	192.168.3.100	192.168.3.4	OSPF	66 DB Description
29 55.513000	192.168.3.100	192.168.3.4	OSPF	66 DB Description
30 56.678000	192.168.3.3	224.0.0.5	OSPF	94 Hello Packet
32 57.549000	192.168.3.100	192.168.3.4	OSPF	66 DB Description
34 58.761000	192.168.3.4	224.0.0.5	OSPF	94 Hello Packet
35 59.568000	192.168.3.100	192.168.3.4	OSPF	66 DB Description
36 59.603000	192.168.3.100	192.168.3.4	OSPF	82 Hello Packet
38 60.629000	192.168.3.100	192.168.3.4	OSPF	110 LS Update

Figure 13: Process of the adjacency attack.

Table 3: CPU consumption comparison.

Number of nodes	Virtualization technology (%)	High-throughput technology (%)	High-performance technology (%)
2	9	0.3	0.2
4	19.1	0.5	0.4
6	30.6	0.6	0.5
8	39.3	0.8	0.7
10	45.6	1.0	0.9

Table 4: Memory consumption comparison.

Number of nodes	Virtualization technology (MB)	High-throughput technology (MB)	High-performance technology (MB)
2	907	852	78
4	1712	877	80
6	2519	998	81
8	3229	1054	83
10	4000	1092	84

Figure 14: Multihop topology for testing.

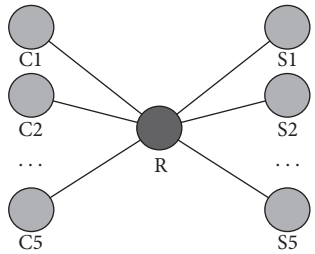

Figure 15: Concurrent topology for testing.

Figure 16 shows the comparison of the throughput, delay, and packet loss rate of the three routing emulation technologies in the multihop situation.

In terms of the throughput, in the 2-hop situation, the throughput of the virtualization technology is 11.8 Gbit/s and that of the high-throughput technology is 14.6 Gbit/s. However, the throughput of the *high-performance technology* is 17.2 Gbit/s, which is 1.46 times that of the *virtualization technology* and 1.18 times that of the *high-throughput technology*. With the increase in the number of hops, the *high-performance technology* still has advantages. In the 10-hop situation, the throughput of the *high-performance technology* is 2.49 times of the *virtualization technology* and 1.24 times that of the *high-throughput technology*.

In terms of the delay time, in the 2-hop situation, the delay time of the *virtualization technology* is 1.071 ms and that of the *high-throughput technology* is 0.73 ms. However, the delay time of the *high-performance technology* is 0.456 ms, which is reduced by 57% compared to the *virtualization technology* and reduced by 38% compared to the *high-throughput technology*. With the increase in the number of hops, the delay time of the *high-throughput technology* and *high-performance technology* appears to be stable, but that of the *virtualization technology* increases linearly.

In terms of the packet loss rate, the high-throughput technology and high-performance technology tend towards 0, and with the increase in the number of hops, the packet loss rate is still stable at 0. However, in the 2-hop situation, the packet loss rate of the virtualization technology is 10%, and it will increase as the number of hops increases.

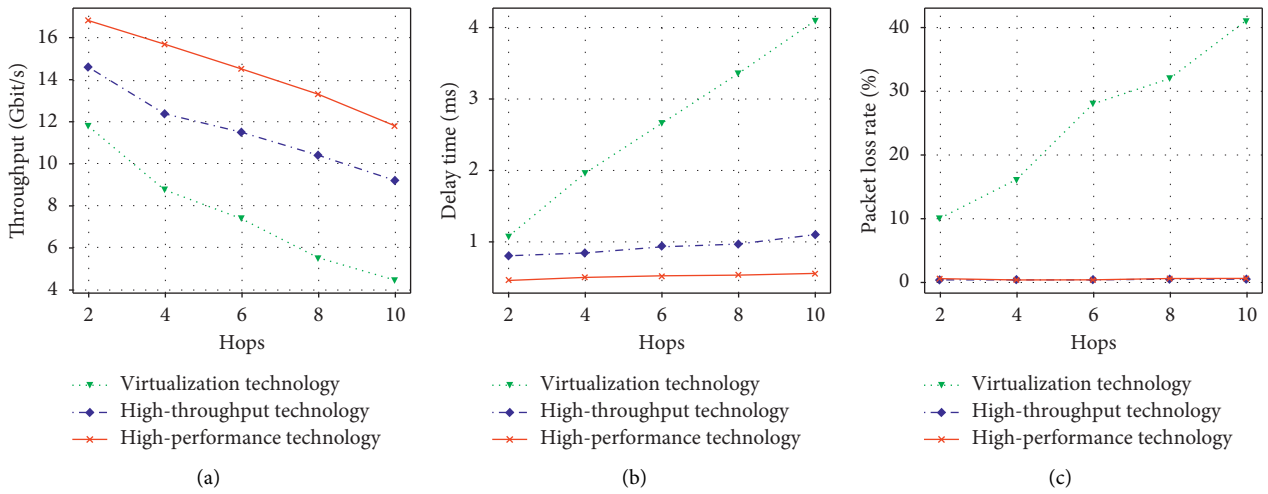

Figure 16: Comparison of performance in the multihop situation.

Therefore, in the multihop situation, the high-performance technology performs well in terms of the throughput, delay, and packet loss rate.

Figure 17 shows the comparison of the throughput, delay, and packet loss rate of the three routing emulation technologies in the concurrent situation.

In terms of the throughput, with the increase in the number of concurrencies, the throughput of the *high-throughput technology* and *high-performance technology* shows a linear growth, while that of the *virtualization technology* shows little change. When the number of concurrencies is 5, the throughput of the *virtualization technology* is 11.92 Gbit/s and that of the high-throughput technology is 39.7 Gbit/s. The throughput of the high-performance technology is 53.5 Gbit/s, which is 4.49 times that of the virtualization technology and 1.35 times that of the high-throughput technology.

In terms of the delay time, the three technologies are stable. The virtualization technology is stable at 0.9 ms, the high-throughput technology is stable at 0.7 ms, and the high-performance technology is stable at 0.46 ms. The delay time of the high-performance technology is reduced by 49% compared to that of the *virtualization technology* and reduced by 34% compared to that of the *high-throughput technology*.

In terms of the packet loss rate, *high-throughput technology* is stable at 0.4%. The *high-performance technology* will increase slowly as the number of concurrencies increases, but the *virtualization technology* always has a high packet loss rate.

Therefore, in the concurrent situation, *high-performance technology* performs well in terms of throughput and delay. However, because the *high-throughput technology* uses namespace [50], it can process packets more stably in the concurrent situation, and it has a lower packet loss rate than *high-performance technology*.

In conclusion, compared with the *virtualization technology* and *high-throughput technology*, *high-performance technology* has obvious performance advantages in multihop and concurrent situations.

6.5. Comparisons in Situations of Link Congestion. In the same environment, we compare the *virtualization technology*, *high-throughput technology*, and *high-performance technology* (with and without the load balancing of links). We design a testing topology, as shown in Figure 18. When C1 transmits many packets to C3 to block the communication link, we test the communication path and throughput of C2 and C4. The results are shown in Table 5 and Figure 19.

From Table 5 and Figure 19, we can see that, compared with the *virtualization technology*, *high-throughput technology*, and *high-performance technology* without the load balancing of links, the *high-performance technology* with the load balancing of links can select the relatively idle link as the forwarding path in the situation of congestion. Because the congestion link is selected for forwarding paths, the throughput of the *virtualization technology* is almost 0 Gbit/s, that of the *high-throughput technology* is 4.35 Gbit/s, and that of the *high-performance technology* without the load balancing of links is 9.8 Gbit/s. However, the throughput of the *high-performance technology* with the load balancing of links is 12.3 Gbit/s. Therefore, *high-performance technology* can provide a high-quality communication path in the situation of congestion through the load balancing of links.

6.6. Verification of Advantages of the Controller Cluster. In this section, we compare the average time for the single controller and the controller cluster (it contains three controllers) to process all packets. The shorter the time is, the more the routing nodes that can be controlled. We design a scenario as Figure 20 and use a single controller or a controller cluster to control all routing nodes.

In Figure 20, all the clients send packets to the controller of the emulation network, so the controller needs to process these packets and return them. Each client calculates the

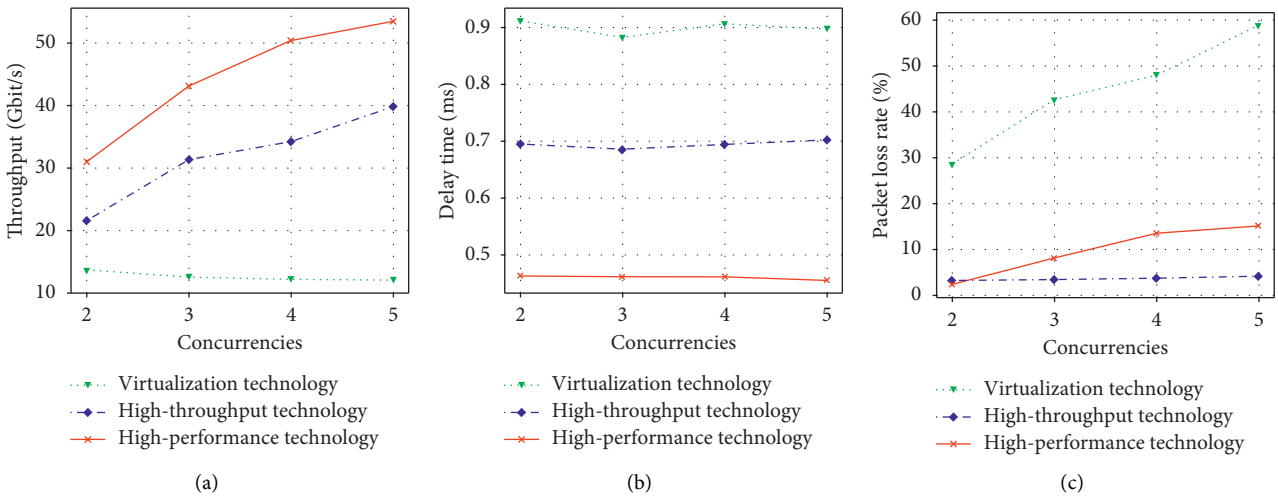

Figure 17: Comparison of performance in the concurrent situation.

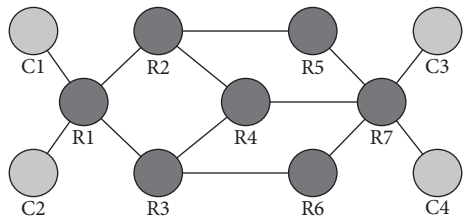

Figure 18: Topology for testing link load balancing.

Table 5: Route before and after congestion.

Routing emulation technology	The path of C1 to C3	The path of C2 to C4
Virtualization technology	[C1, R1, R2, R5, R7, C3]	[C2, R1, R2, R5, R7, C4]
High-throughput technology	[C1, R1, R2, R5, R7, C3]	[C2, R1, R2, R5, R7, C4]
Without the load balancing of links	[C1, R1, R2, R5, R7, C3]	[C2, R1, R2, R5, R7, C4]
With the load balancing of links	[C1, R1, R2, R5, R7, C3]	[C2, R1, R3, R6, R7, C4]

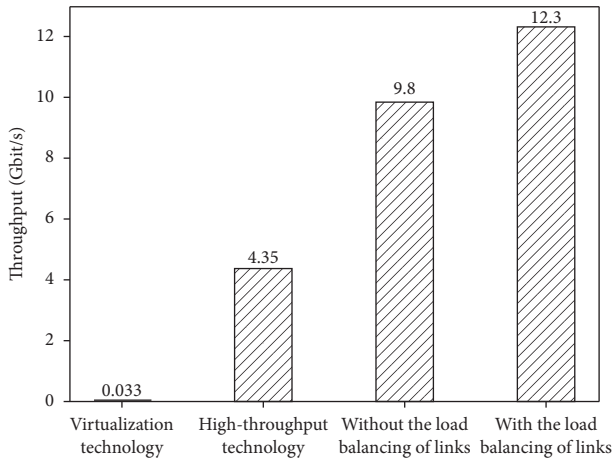

Figure 19: Comparison of throughput in the situation of congestion.

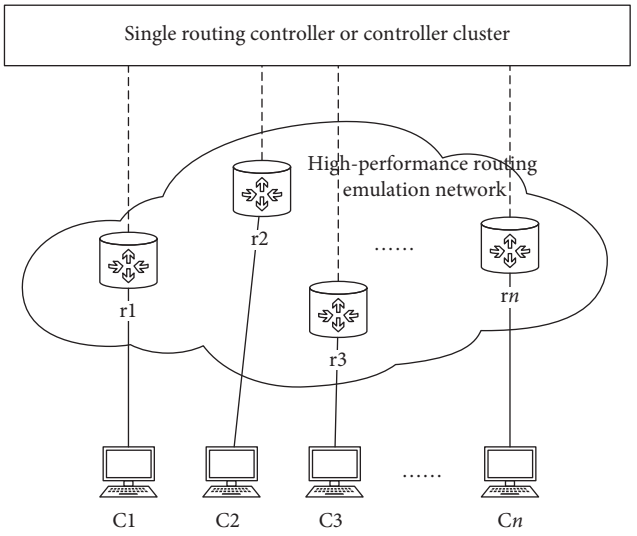

Figure 20: Scenario for verifying advantages of the controller cluster.

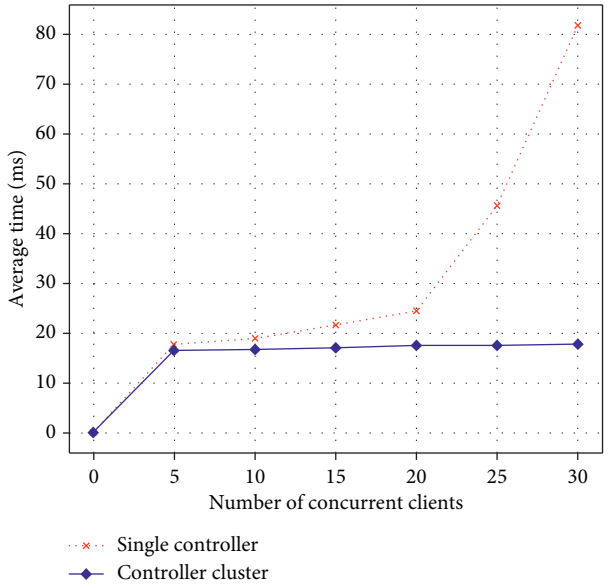

Figure 21: Comparison of processing time.

round-trip delay of the packets and obtains the average time T_{avg} by

$$T_i = \text{packet\_delay}_i - \text{link\_delay}_i, \tag{11}$$

$$T_{avg} = \frac{\sum_{i=1}^{N} T_i}{N}. \tag{12}$$

In (11), packet_delay_i represents the round-trip time of the packet sent by the ith client, and link_delay_i represents the link delay time from the ith client to the controller. The time for the controller to process the packet of the ith client can be obtained by subtracting link_delay_i from packet_delay_i. In (12), N represents the number of clients. Figure 21 shows the comparison of the value T_{avg} between a single controller and a controller cluster in the case of different numbers of concurrent clients.

As the number of concurrent clients increases, the time needed for the single controller to process packets increases dramatically. However, the time needed for the controller cluster to process packets increases slowly. Therefore, the controller cluster can control more routing nodes and improve the scalability of the emulation scale.

7. Conclusions

This paper proposes a high-performance routing emulation technology based on a cloud platform that provides a network environment for edge computing to verify and evaluate new architecture, protocol, and security technologies. First, we combine OpenStack and SDN technology to propose a high-performance routing emulation architecture. Then, we implement the routing function, OSPF protocol, and load balancing of links through apps of the decision layer. Finally, we propose a distributed control method and improve the scalability of the emulation scale with the controller cluster. Experiments show that, compared with other routing emulation technologies, this technology achieves less overhead, higher performance, and a realistic OSPF protocol. The controller cluster can also control more routing nodes than the single controller.

However, this paper studies only how to support the OSPF protocol, and other dynamic routing protocols (such as the Routing Information Protocol (RIP) and the Border Gateway Protocol (BGP)) need to be studied further. In addition, how to realize the hot backup of the controller will be the direction of the follow-up research.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant nos. 61672264 and 61972182) and National Key R&D Program of China (Grant no. 2016YFB0800801).

References

[1] D. Acarali, M. Rajarajan, N. Komninos, and B. B. Zarpelão, "Modelling the spread of botnet malware in IoT-based wireless sensor networks," *Security and Communication Networks*, vol. 2019, Article ID 3745619, 13 pages, 2019.

[2] C. Zhou, A. Li, A. Hou et al., "Modeling methodology for early warning of chronic heart failure based on real medical big data," *Expert Systems with Applications*, vol. 151, Article ID 113361, 2020.

[3] X. Xu, Q. Huang, X. Yin, M. Abbasi, M. R. Khosravi, and L. Qi, "Intelligent offloading for collaborative smart city services in edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 7919–7927, 2020.

[4] W. Zhong, X. Yin, X. Zhang et al., "Multi-dimensional quality-driven service recommendation with privacy-preservation in mobile edge environment," *Computer Communications*, vol. 157, pp. 116–123, 2020.

[5] X. Xu, X. Zhang, X. Liu, J. Jiang, L. Qi, and M. Z. A. Bhuiyan, "Adaptive computation offloading with edge for 5G-envisioned internet of connected vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2020.

[6] J. Li, T. Cai, K. Deng, X. Wang, T. Sellis, and F. Xia, "Community-diversified influence maximization in social networks," *Information Systems*, vol. 92, pp. 1–12, 2020.

[7] L. Wang, X. Zhang, R. Wang, C. Yan, H. Kou, and L. Qi, "Diversified service recommendation with high accuracy and efficiency," *Knowledge-Based Systems*, vol. 204, Article ID 106196, 2020.

[8] L. Wang, X. Zhang, T. Wang et al., "Diversified and scalable service recommendation with accuracy guarantee," *IEEE Transactions on Computational Social Systems*, pp. 1–12, 2020.

[9] Y. Zeng, M. Chao, and R. Stoleru, "EmuEdge: a hybrid emulator for reproducible and realistic edge computing experiments," in *Proceedings of the 2019 IEEE International Conference on Fog Computing (ICFC)*, June 2019.

[10] A. A. T. R. Coutinho, F. Greve, and C. Prazeres, "An architecture for fog computing emulation," in *Proceedings of the Anais do XV Workshop em Clouds e Aplicações*, SBC, Belém, Brazil, May 2017.

[11] Y. Xing and Y. Zhan, "Virtualization and cloud computing," *Future Wireless Networks and Information Systems*, pp. 305–312, Springer, Berlin, Germany, 2012.

[12] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda et al., "Using mininet for emulation and prototyping software-defined networks," in *Proceedings of the 2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pp. 1–6, IEEE, Bogota, Colombia, June 2014.

[13] J. D. Beshay, A. Francini, and R. Prakash, "On the fidelity of single-machine network emulation in linux," in *Proceedings of the 2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 19–22, IEEE, Atlanta, GA, USA, October 2015.

[14] X. Xu, S. Fu, W. Li, F. Dai, H. Gao, and V. Chang, "Multi-objective data placement for workflow management in cloud infrastructure using NSGA-II," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 5, pp. 605–615, 2020.

[15] X. Xu, R. Mo, F. Dai, W. Lin, S. Wan, and W. Dou, "Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6172–6181, 2020.

[16] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[17] R. Kumar, N. Gupta, S. Charu et al., "Open source solution for cloud computing platform using openstack," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 5, pp. 89–98, 2014.

[18] T. Rosado and J. Bernardino, "An overview of openstack architecture," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, pp. 366-367, Porto, Portugal, July 2014.

[19] Z. Mengdong, J. Xin, and W. Xiaofeng, "Research on high-throughput routing simulation based on open-stack," *Computer Engineering and Applications*, vol. 54, no. 22, pp. 74–79, 2018.

[20] T. Hirt, *Kvm-the Kernel-Based Virtual Machine*, Red Hat Inc., Raleigh, NC, USA, 2010.

[21] Docker: storagedriver, https://docs.docker.com/storage/storagedriver/.

[22] P. Jakma and D. Lamparter, "Introduction to the quagga routing suite," *IEEE Network*, vol. 28, no. 2, pp. 42–48, 2014.

[23] M. Handley, O. Hodson, and E. Kohler, "XORP: an open platform for network research," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 53–57, 2003.

[24] J. Martins, M. Ahmed, C. Raiciu et al., "ClickOS and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, USENIX Association, Seattle, WA, USA, April 2014.

[25] N. Varis, "Anatomy of a linux bridge," in *Proceedings of the Seminar on Network Protocols in Operating Systems*, p. 58, Espoo, Finland, September 2012.

[26] B. Pfaff, J. Pettit, T. Koponen et al., "The design and implementation of open vswitch," in *Proceedings of the 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 117–130, Oakland, CA, USA, May 2015.

[27] M. K. Shin, K. H. Nam, and H. J. Kim, "Software-defined networking (SDN): a reference architecture and open APIs," in *Proceedings of the 2012 International Conference on ICT Convergence (ICTC)*, pp. 360-361, IEEE, Jeju, South Korea, October 2012.

[28] P. S. Katkar and D. V. R. Ghorpade, "Comparative study of network simulator: NS2 and NS3," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 6, no. 3, pp. 608–612, 2016.

[29] X. Chang, "Network simulations with opnet," in *Proceedings of the 31st Conference on Winter Simulation: Simulation—A Bridge to the Future-Volume 1*, pp. 307–314, Phoenix, AZ, USA, December 1999.

[30] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks," *ACM SIGSIM Simulation Digest*, vol. 28, no. 1, pp. 154–161, 1998.

[31] B. Heller, *Reproducible Network Research with High-Fidelity Emulation*, Stanford University, Stanford, CA, USA, 2013.

[32] O. Tkachova, M. J. Salim, and A. R. Yahya, "An analysis of SDN-openstack integration," in *Proceedings of the 2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, pp. 60–62, IEEE, Kharkiv, Ukraine, October 2015.

[33] J. Chen, H. Song, X. Wang et al., "Emulation of router functions based on a cloud platform," in *Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 271–276, IEEE, Beijing, China, October 2019.

[34] M. H. Huang, Y. X. Zhang, and X. U. Fei, "Design of virtualization platform for router emulation," *Journal of System Simulation*, vol. 26, no. 8, pp. 1672–1677, 2014.

[35] N. Memari, S. J. B. Hashim, and K. B. Samsudin, "Towards virtual honeynet based on LXC virtualization," in *Proceedings of the 2014 IEEE Region 10 Symposium*, pp. 496–501, IEEE, Kuala Lumpur, Malaysia, April 2014.

[36] L. Lian, Y. Zhang, H. Zhang et al., "Constructing virtual network attack and defense platform based on openstack," in *Proceedings of the 2015 International Conference on*

Automation, Mechanical Control and Computational Engineering, Changsha, China, April 2015.

[37] D. Huang, B. He, and C. Miao, "A survey of resource management in multi-tier web applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1574–1590, 2014.

[38] L. Qi, Q. He, F. Chen, X. Zhang, W. Dou, and Q. Ni, "Data-driven web APIs recommendation for building web applications," *IEEE Transactions on Big Data*, p. 1, 2020.

[39] S. Y. Wang, H. W. Chiu, and C. L. Chou, "Comparisons of SDN OpenFlow controllers over EstiNet: Ryu vs. NOX," in *Proceedings of the ICN 2015*, p. 256, San Francisco, CA, USA, September 2015.

[40] J. Moy, *OSPF Version 2*, 1998.

[41] M. Rostanski, K. Grochla, and A. Seman, "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, pp. 879–884, IEEE, Warsaw, Poland, September 2014.

[42] Y. Deng, Y. Chen, Y. Zhang, and S. Mahadevan, "Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment," *Applied Soft Computing*, vol. 12, no. 3, pp. 1231–1237, 2012.

[43] Z. Wang and J. Crowcroft, "Shortest path first with emergency exits," in *Proceedings of the ACM Symposium on Communications Architectures & Protocols*, pp. 166–176, Philadelphia, PA, USA, September 1990.

[44] Open Networking Foundation, 2012, https://www.opennetworking.org/.

[45] C. Hunt, *TCP/IP Network Administration*, O'Reilly Media, Inc., Newton, MA, USA, 2002.

[46] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari et al., "Reverse traceroute," in *Proceedings of the NSDI*, vol. 10, pp. 219–234, Boston, MA, USA, April 2010.

[47] B. Al-Musawi, P. Branch, M. F. Hassan, and S. R. Pokhrel, "Identifying OSPF LSA falsification attacks through non-linear analysis," *Computer Networks*, vol. 167, Article ID 107031, 2020.

[48] A. Agusriandi and E. Elihami, "Developing delay jitter, throughput, and package lost IPERF3 for learning Islamic education," *Jutkel: Jurnal Telekomunikasi, Kendali dan Listrik*, vol. 2, no. 1, pp. 23–30, 2020.

[49] C. Pelsser, L. Cittadini, S. Vissicchio et al., "From Paris to Tokyo: on the suitability of ping to measure latency," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, pp. 427–432, Barcelona, Spain, October 2013.

[50] E. W. Biederman and L. Networx, "Multiple instances of the global linux namespaces,"vol. 1, pp. 101–112, in *Proceedings of the Linux Symposium*, vol. 1, pp. 101–112, Citeseer, Ottawa, Canada, July 2006.