sensors

Article

Research on Computing Resource Measurement and Routing Methods in Software Defined Computing First Network

Xiaomin Gong, Shuangyin Ren, Chunjiang Wang * and Jingchao Wang *

Academy of Systems Engineering, AMS, Beijing 100141, China; gongxiaomin04@163.com (X.G.)
* Correspondence: wangcj61@sina.com (C.W.); wangjc.2000@tsinghua.org.cn (J.W.)

Abstract: Computing resource measurement and computing routing are essential technologies in the computing first network (CFN), serving as its foundational elements. This paper introduces a Software Defined Computing First Network (SD-CFN) architecture. Building upon this framework, a Dynamic-Static Integrated Computing Resource Measurement Mechanism (DCRMM) is proposed, incorporating methods such as the entropy weight method and K-Means clustering. The DCRMM algorithm outperforms the Maximum-closest Static Algorithm (MSA) and Maximum Closest Dynamic Algorithm (MDA) in terms of node stability, node utilization, and node matching accuracy. Additionally, a Reinforcement Learning and Software Defined Computing First Networking Routing (RSCR) algorithm is presented as a software-defined computing routing solution within the SD-CFN. RSCR introduces a knowledge plane responsible for computing routing calculations. It comprehensively considers factors such as link latency, available bandwidth, and packet loss rate. Simulation experiments conducted on the GÉANT topology demonstrate that RSCR outperforms the OSPF algorithm in terms of link latency, packet loss rate, and throughput. DCRMM and RSCR offer innovative solutions for computing resource measurement and computing routing in computing first networks.

Keywords: software defined network; computing first network; computing routing; computing resource measurement; reinforcement learning

Citation: Gong, X.; Ren, S.; Wang, C.; Wang, J. Research on Computing Resource Measurement and Routing Methods in Software Defined Computing First Network. *Sensors* **2024**, *24*, 1086. https://doi.org/10.3390/s24041086

Academic Editor: Hazer Inaltekin

Received: 6 December 2023
Revised: 12 January 2024
Accepted: 22 January 2024
Published: 7 February 2024

1. Introduction

As artificial intelligence, big data, and other technologies continue to advance, computing-intensive and latency-sensitive tasks such as facial recognition, object detection, and autonomous driving have emerged in large numbers. This trend has driven the further development of technologies such as cloud computing and edge computing, enabling end-users to more conveniently access distributed computing resources at the edge. However, the deployment of edge computing still faces significant challenges. From a network perspective, the computing capabilities of individual nodes in edge computing scenarios are limited. Edge nodes cannot perceive each other and are unable to collaborate effectively. The challenge lies in the inability to efficiently schedule computing tasks to the optimal edge node. From a business requirements perspective, there is a need to decouple existing business application layers from the network. The application layer cannot accurately grasp the network's status. Addressing results dominated by the application layer may not achieve optimal overall performance and could potentially lead to network load imbalance. Services may not be efficiently arranged to the best edge node, resulting in impaired business operations [1]. Therefore, it is crucial to address how to efficiently and flexibly schedule computing resources between nodes, thereby improving the utilization of computing power [2,3].

To more efficiently utilize ubiquitous distributed computing resources and expedite the processing of computing tasks, the convergence of computing and networking has become imminent. With the impetus from telecom operators and equipment vendors, the Computing First Networking (CFN) paradigm has emerged [4,5]. CFN represents a new paradigm of cloud-network integration, aiming to achieve the interconnection and coordinated scheduling of ubiquitous distributed computing, storage, and other resources through the network. This enables on-demand and real-time invocation of massive computing and storage resources, facilitating global optimization and efficient utilization of computing and network resources. As businesses transition to the cloud and the consumer internet evolves into the industrial internet, the convergence of computing and networking has become a pivotal force in the digital transformation of social and economic activities.

The CFN proposes a new network architecture and protocols, unifying the scheduling of computing and network resources, enabling global optimization and collaborative scheduling of these resources. Existing CFN architectures can be broadly categorized into two types: centralized architecture and distributed architecture. The centralized architecture, often combined with Software Defined Networking (SDN), separates the data plane from the control plane, allowing the controller to have a global view of computing and network resources for routing computation and decision-making. The distributed architecture achieves the synchronization of computing and network resource information through information exchanges between adjacent routing nodes, with each routing node handling forwarding and control decisions. Whether centralized or distributed, achieving comprehensive scheduling of computing resources across the entire network requires a routing algorithm that schedules CFN information. Similarly, the prerequisite for routing implementation is the establishment of a unified computing resource measurement and modeling mechanism.

This paper introduces a centralized computing first network architecture, namely the Software Defined Computing First Network (SD-CFN). Leveraging the advantages of control-plane separation and centralized control, this architecture forms a global view of the computing resources and networking resources. Additionally, based on this architecture, we combine methods such as entropy weight and K-Means clustering to construct a Dynamic-Static Integrated Computing Resource Measurement Mechanism (DCRMM). According to the measurement scheme and task requirements, this mechanism identifies the most suitable computing nodes for the computation. Simultaneously, we augment SDN with a knowledge plane and design a Reinforcement Learning and Software Defined Compute First Networking Routing (RSCR) algorithm within this plane. This algorithm, utilizing interactions with the environment, RL intelligence, and the global computing and networking view provided by SDN, considers factors such as link latency, bandwidth, packet loss rate, and computing resource distribution. After finding appropriate computing nodes, it rapidly calculates the computing route from the task initiator to the target computing node. Simulation results using real traffic demonstrate that RSCR outperforms traditional OSPF algorithms significantly in terms of link throughput, latency, and packet loss rate.

This article makes two main contributions:

(1) Proposes a dynamic-static integrated computing resource measurement modeling mechanism.
(2) Proposes an RL-based proactive computing routing algorithm in the SD-CFN environment.

The rest of the paper is organized as follows: Section 2 introduces related work. Section 3 provides a detailed explanation of RSCR. Section 4 details DCRMM. Section 5 introduces the RSCR routing algorithm. Section 6 analyzes the simulation results of the RSCR algorithm. The final section concludes the paper and suggests future directions for research.

2. Related Work

Routing is a fundamental function in the Internet, responsible for forwarding data packets from source to destination addresses. It plays a crucial role in ensuring Quality of Service (QoS) [6]. Traditional routing strategies, like Open Shortest Path First (OSPF) routing [7], may lead to issues such as network congestion and low link utilization. Compared to optimal routing methods, traditional approaches may exhibit performance differences of up to 5000 times [8]. Routing optimization has been a topic of in-depth research [9]. Existing solutions include those based on analytical optimization [10] and those leveraging machine learning [11,12] and deep learning [13,14]. Papers [9,15–17] also proposed various SDN routing optimization methods. In this section, we review SDN routing optimization work based on Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL). Unlike model-based routing optimization algorithms, RL-based methods are model-free. Moreover, while machine learning algorithms require dataset labeling and training, RL algorithms directly interact with the network environment to learn strategies.

The key application of reinforcement learning in routing problems focuses on modeling the forwarding process using the Markov Decision Process (MDP). This process involves setting appropriate states, actions, and reward functions based on specific network scenarios. The intelligent agent adapts to various dynamic changes through interaction with the environment [18]. RL can play a crucial role in SDN routing optimization [19]. As early as 1993, Boyan et al. [20] proposed using the Q-routing algorithm to avoid network congestion, marking the first academic use of reinforcement learning for routing optimization. Although this algorithm was not applied in an SDN architecture, it provided a new reference direction for subsequent routing optimization algorithms. Lin et al. [21] introduced multi-layer SDN QoS-aware adaptive routing, combining the work of Hassas [22] and McCauley [23], introducing a distributed control plane architecture, and achieving a reliable SDN infrastructure with minimal signal latency. Rischke et al. [24] proposed the QR-SDN algorithm to create multiple paths between source and destination addresses, maintaining flow integrity using Q-Learning, and employing the softmax function as an exploration-exploitation strategy. However, this algorithm can only guarantee lower flow latency than Shortest Path First (SPF) in small networks.

Houda et al. [25] used the Q-Learning algorithm to address the delay minimization problem in multipath SDN networks. They employed two different exploration strategies, namely ϵ-greedy, and softmax, and evaluated the performance of the Q-Learning algorithm based on average flow latency and convergence time for different load levels. Casas-Velasco et al. [26] proposed the RSIR algorithm, which defined an active routing algorithm based on link-state, exhibiting lower overall latency and better load balancing compared to the traditional Dijkstra algorithm. Building on this work, they also introduced Deep Reinforcement Learning (DRL) and defined the DRSIR algorithm [27] in SDN. DRSIR considers path state indicators to generate active, efficient, and intelligent routing to adapt to dynamic traffic changes. The evaluation of DRSIR was conducted through simulation using real and synthetic traffic matrices. Chen et al. [9] also introduced DRL into the routing process and proposed the RL-Routing algorithm to address Traffic Engineering (TE) issues in SDN regarding throughput and latency. They performed comprehensive experiments based on Fattree, NSFNet, and ARPANet topologies, demonstrating that experience-driven artificial intelligence has advantages over traditional algorithms in solving TE problems. Yu et al. [28] proposed a DDPG routing optimization solution, which, unlike the previous Q-table-based mechanism, saves time and storage by using a neural network. Zhao et al. [29] positioned the optimal multicast routing problem in SDN as a multi-objective optimization problem. They designed an intelligent multicast routing algorithm, DRL-M4MR, to construct a multicast tree in SDN. After DRL-M4MR agent training, the SDN controller installs multicast flow entries by reverse traversing the multicast tree to SDN switches, achieving intelligent multicast routing.

Currently, RL routing optimization algorithms based on the SDN architecture are primarily centered around Q-Learning. This paper also adopts the Q-Learning algorithm to design the reward function and adjust the strategy based on link available bandwidth, latency, and packet loss rate. The proposed RSCR algorithm exhibits superior performance compared to the traditional OSPF algorithm.

3. DCRMM

This section primarily introduces the unified modeling and measurement of dynamic and static performance indicators of computing nodes by DCRMM and explains in detail how to select the optimal computing node based on the measurement results. The dynamic performance indicators of computing nodes refer to the node's idle CPU cores (cores) c_f, idle memory (GB) m_f, and idle storage (GB) s_f , while static performance indicators refer to the node's total CPU cores (cores) c_t, total memory (GB) m_t, and total storage (GB) s_t. The workflow of DCRMM mainly consists of the following steps: a. Use the entropy weight method to calculate the static performance baseline score of computing nodes; b. Segment the baseline scores using the K-Means clustering algorithm; c. Using the entropy weight method to calculate the weights of dynamic indicators, assigning these weights to various indicators of computing tasks (i.e., the required CPU cores (cores) t_c, memory (GB) t_m, storage (GB) t_s for computing tasks), scoring the computing tasks based on these weights, and classifying them into the corresponding static performance score intervals according to the scores. d. Within the qualifying interval, calculate the Euclidean distance between the dynamic performance of computing nodes and the required performance for computing tasks, and the node with the shortest distance is considered the optimal computing node. The reasons for using the entropy weight method are the following two points: a. The basis for determining the weight of the entropy weight method comes from the data, which are highly objective and reduce the impact of subjectivity on the decision-making results. b. The calculation logic of the entropy weight method is simple and clear, has good operability, and is easy to implement. To prevent a single indicator from occupying a high weight for a long time in the actual application process, we can reasonably set the update cycle according to environmental changes to cope with the dynamic updates of different indicators of the computing power node as the computing tasks are processed. Compared to traditional algorithms such as the Maximum-closest Static Algorithm (MSA), which considers only static indicators, and the Maximum closest dynamic algorithm (MDA), which considers only dynamic indicators, DCRMM performs better in terms of stability, node utilization, and accuracy in node selection.

3.1. Calculating the Basic Score

Firstly, the entropy weight method [30] is employed to analyze the static indicators of nodes, yielding the foundational performance scores. Drawing inspiration from the concept of information entropy, the entropy weight method's greatest advantage lies in mitigating the influence of human factors on indicator weights, ensuring that the results of comprehensive evaluations are more objective [31].

Assuming there are n evaluation indicators and m objects, the raw data can be represented as a matrix X:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} \tag{1}$$

Here, x_{ij} represents the numerical value of the j-th indicator for the i-th object. Since various evaluation indicators may have different dimensions, normalization is required before calculation to ensure that the results fall within the $[0,1]$ range. Therefore, we normalize the matrix X column-wise, where X_i represents a value in the column where the standardization occurs. The normalization function is given by:

$$x'_{ij} = \frac{x_{ij} - min(X_i)}{max(X_i) - min(X_i)} \tag{2}$$

To avoid the impact of zero in the data on subsequent calculations, it is common to apply an offset to the data. The normalized data can be represented as a matrix X':

$$X' = \begin{bmatrix} x'_{11} & \cdots & x'_{1n} \\ \vdots & \ddots & \vdots \\ x'_{m1} & \cdots & x'_{mn} \end{bmatrix} \tag{3}$$

Let the weight of the j-th indicator in the i-th object be denoted as p_{ij}:

$$p_{ij} = \frac{x'_{ij}}{\sum_{i=1}^{n} x'_{ij}}, (0 \leqslant p_{ij} \leqslant 1) \tag{4}$$

Subsequently, the weight matrix P for the original data is obtained as follows:

$$P = \begin{bmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{m1} & \cdots & p_{mn} \end{bmatrix} \tag{5}$$

In the entropy weight method, the entropy value of the j-th indicator is given by [31]:

$$e_j = -\frac{1}{\ln m} \sum_{i=1}^{n} p_{ij} \ln p_{ij}, (0 \leqslant e_j \leqslant 1) \tag{6}$$

The entropy value e_j indicates that the larger its numerical value, the higher the differentiation degree of the j-th indicator, suggesting the derivation of more information. Therefore, a higher weight should be assigned to this indicator. Thus, the calculation method for the weight w_j is given by:

$$w_j = \frac{1 - e_j}{\sum_{j=1}^{m} (1 - e_j)} \tag{7}$$

Therefore, the comprehensive score F_i for the i-th object can be represented as:

$$F_i = \sum_{j=1}^{m} p_{ij} w_j \tag{8}$$

The static performance indicators of the computing nodes can be defined as a triplet $I = (c_t, m_t, s_t)$, and through the entropy weight method, the weight triplet corresponding to each indicator is obtained $W = (w_1, w_2, w_3)$. Since only numerical calculations are considered when calculating the base score, the impact of different units can be ignored. The basic score S_n for the static performance of the computing node can be represented as:

$$S_n = c_t \ w_1 + m_t \ w_2 + s_t \ w_3 \tag{9}$$

S_n represents the static performance score of the computing node. A higher value indicates that the node has stronger basic performance, making it more capable of handling computing tasks with high demands for computation, storage, and memory performance.

3.2. Segmenting the Basic Score

Next, the K-Means clustering algorithm is applied to segment the basic score. The K-Means algorithm is a commonly used clustering method that divides data points into different clusters or groups, so that similar data points are close to each other, while dissimilar data points are farther apart. This algorithm is an unsupervised learning method, as it does not require prior knowledge of the categories of data points but automatically identifies and groups them. By using the K-Means algorithm, nodes are divided into levels according to performance, and tasks are divided into corresponding performance nodes according to computing task resource requirements, which can reduce the amount of calculation of Euclidean distance.

Typically, the K-Means algorithm can be broken down into the following six steps:

a. **Inputs:** Input the dataset $S_n = \{S_1, S_2, \ldots, S_i, \ldots, S_m\}$ (S_n represents the set of all basic scores) and the maximum number of iterations *max_interations* are provided.

b. **Initialization:** Define the number of clusters K (In this article, K = 3), and initialize the cluster centroids $C^{(0)} = \{C_1^{(0)}, C_2^{(0)}, \ldots, C_K^{(0)}\}$. The initial cluster of centroids can be randomly chosen.

c. **Assignment:** For each data point S_i, calculate its distance to each cluster centroid C_j, typically using the Euclidean distance measured by the formula:

$$d(S_i, C_j) = \sqrt{\sum_{k=1}^{n} (S_{ik} - C_{jk})^2} \tag{10}$$

Subsequently, assign each data point S_i to the cluster with the nearest centroid C_j where j is determined by the following formula:

$$j = arg\ min_j\ d(S_i, C_j) \tag{11}$$

d. **Update Centroids:** For each cluster j, calculate the new cluster center $C_j^{(t+1)}$ as the mean of all data points in that cluster:

$$C_j^{(t+1)} = \frac{1}{|C_j|} \sum_{S_i \in C_j} x_i \tag{12}$$

where $|C_j|$ represents the number of data points in the cluster j, and represents the iteration number.

e. **Convergence Check:** Usually, K-means iterates until a stopping condition is met, such as when the cluster centers no longer undergo significant changes (or changes are below a certain convergence threshold ϵ), or when the maximum number of iterations *max_interations* is reached. This can be expressed as:

$$\left\| C_j^{(t+1)} - C_j^{(t)} \right\| < \epsilon \tag{13}$$

f. **Outputs:** The final result of the K-Means algorithm includes the ultimate cluster centers $C^{(t)} = \{C_1^{(t)}, C_2^{(t)}, \ldots, C_K^{(t)}\}$ and the set of data points assigned to each cluster $Z_i \subset \{Z_1, Z_2, \ldots, Z_n\}$.

Finally, all basic scores are divided into K categories $C^{(t)} = \{C_1^{(t)}, C_2^{(t)}, \ldots, C_K^{(t)}\}$, and each category has the following scores $Z_i \subset \{Z_1, Z_2, \ldots, Z_n\}$.

3.3. Matching the Cluster

Subsequently, evaluate the resource demand indicators of the computing tasks, identifying the cluster of static performance scores corresponding to their scores. The resource demand of a computing task can be defined as a triplet $T = (t_c, t_m, t_s)$. The score of the tasks can be calculated using the weighted triplet $W' = (w_1', w_2', w_3')$ obtained during the computation of the dynamic performance score of the computing nodes. Therefore, the score of the computing task can be expressed as S_t :

$$S_t = t_c\ w_1' + t_m\ w_2' + t_s\ w_3' \tag{14}$$

If $Z_i min < S_t < Z_i max$, then the matching of computing nodes only needs to be performed within the set Z_i of computing tasks. This reduces the number of nodes to be matched, improving the matching time.

3.4. Selecting a Computing Node

Compute the Euclidean distance between the dynamic performance of each computing node within the interval and the computing task's resource requirements to find the most suitable computing node. To avoid selecting a computing node based on a single Euclidean distance that cannot execute the computing task, it is also necessary to exclude computing nodes in the set Z_i that do not meet the requirements. Suppose the static performance indicator triplets of a computing node, denoted as $I = (c_t, m_t, s_t)$, have at least one indicator not less than the corresponding triplet in the computing task's resource triplets $T = (t_c, t_m, t_s)$. In that case, the new matching interval can be defined as $Z_i' = \{z_i | z_i.c_t \geqslant t_c, z_i.m_t \geqslant t_m, z_i.s_t \geqslant t_s\}$.

The dynamic performance of a computing node can be defined as a triplet $J = (c_f, m_f, s_f)$. Considering that the dynamic performance values are continuously changing, if we denote a computing task as *Task* and a computing node as *Node*, the smaller the distance between *Task* and *Node*, the smaller the difference between them. In other words, the computing node (*Node*) better satisfies the conditions of the computational task (*Task*). Therefore, this paper adopts the n-dimensional Euclidean distance [32] as a measurement standard to assess the dynamic performance and select computing nodes suitable for computational tasks.

Suppose the computational task's resource requirements are denoted as $T_i = (t_{ci}, t_{mi}, t_{si})$ and the dynamic performance of the computing node as $J_i = (c_{fi}, m_{fi}, s_{fi})$. Due to different dimensions, normalization is necessary for both T_i and J_i. Let us denote the normalized versions as T_i' and J_i' using the formula (2). The distance between them can then be expressed as:

$$d(T', J') = \sqrt{\sum_{i=1}^{n} (T_i' - J_i')^2}, (n = 3) \tag{15}$$

Calculate the set of distances Z_i' between all computing nodes in D and the computational task. The minimum distance D_{min} indicates that the resource requirements of computing tasks are closest to the performance of this computing node. Therefore, this node is the best computing node that matches the resource requirements of the computational task.

4. RSCR

This chapter provides an overview of RSCR and introduces the functions of its major modules.

4.1. Overview

As early as 2003, Clark et al. [33] proposed the addition of a knowledge plane to achieve intelligence in networks. Subsequently, Mestres et al. [34] built on this concept and introduced the idea of a Knowledge-Defined Network (KDN). RSCR, on the foundation of SD-CFN, incorporates a knowledge plane where RL is integrated to achieve intelligent computing routing.

RSCR utilizes available bandwidth, delay, and packet loss rate as features for the RL process. Combined with the optimal computing node identified in Section 3, it computes the route from the computing request source to the optimal computing node. RSCR, leveraging the global view of SDN, can dynamically adjust routes based on changing traffic patterns. Furthermore, unlike traditional routing that relies on protocols such as RIP, OSPF, and BGP [35], RSCR, as described in paper [36–38], integrates with SDN using new routing protocols. In SD-CFN, timely and reliable communication is also crucial. Reference [39] evaluates the timeliness of drone-assisted networks, and we can introduce protocols centered on the Age of Information (AoI) for further research in the future. Regarding SD-CFN architecture communication security, we can try to introduce blockchain technology to ensure data security and privacy. Wang et al. [40] proposed a blockchain-assisted distributed access control scheme for drone computing networks, which can enable drones to autonomously manage their identities, attributes, and access policies. Referring to this solution, SD-CFN can be provided with higher efficiency and fault tolerance, making reliable computing services possible.

As shown in Figure 1, a detailed explanation of RSCR is provided below: (1) The Computing Resource Perception Unit periodically perceives the computing resources of the computing devices on the computing resource end in the data plane. Computing resources include CPU, memory, and storage resources. It is important to note that the computing resource end is composed of a Kubernetes cluster responsible for providing computing resource data. (2) The Network Resource Perception Unit periodically queries the data plane to collect raw data on the state of the network topology. (3) The Management Plane retrieves computing resource data to segment computing nodes. (4) The Management Plane retrieves network resource data to calculate and store link state information. (5) The Knowledge Plane retrieves information about the overall computing-network resources from the management plane. (6) RL Agent explores all possible routes between each source-destination node pair and computes the optimal route based on link states. (7) The Knowledge Plane stores all routing information. (8) The Control Plane retrieves routing information before the computing task arrives and proactively installs the optimal path between the computing request source and the optimal computing node in the form of flow tables on the switches.

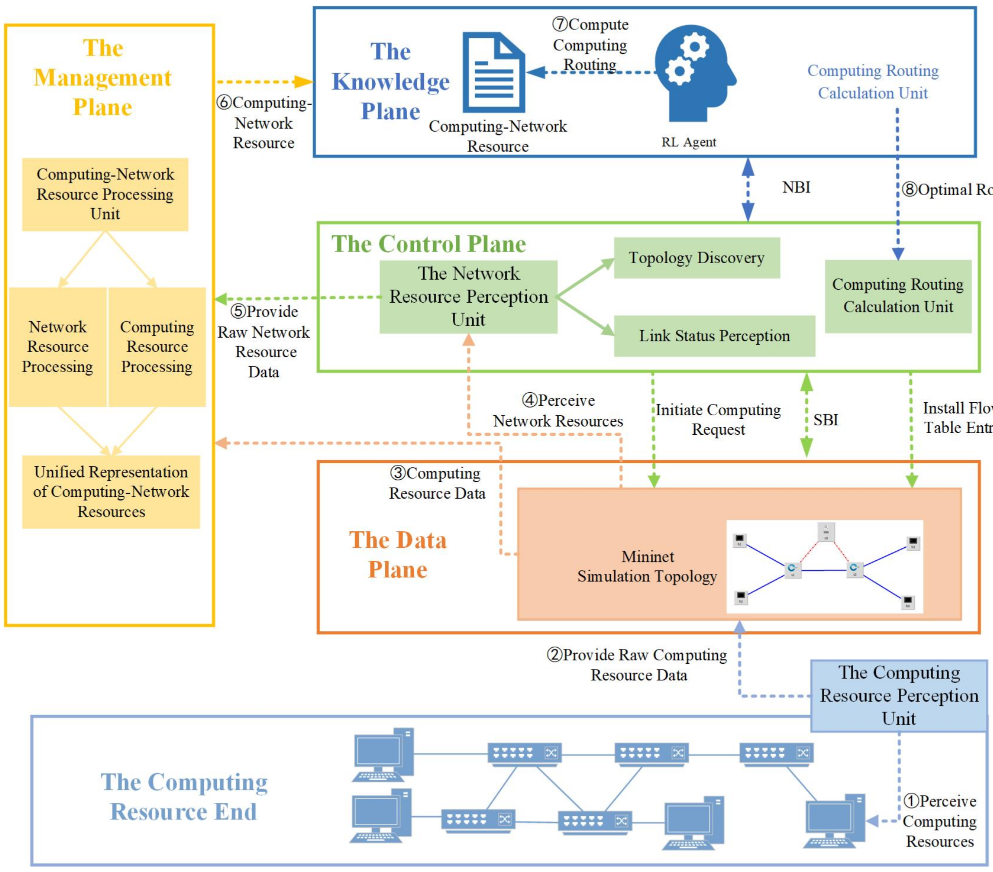

The Management Plane
Computing-Network Resource Processing Unit
Network Resource Processing
Computing Resource Processing
Unified Representation of Computing-Network Resources
⑥Computing-Network Resource
The Knowledge Plane
Computing-Network Resource
⑦Compute Computing Routing
RL Agent
Computing Routing Calculation Unit
NBI
⑧Optimal Route
The Control Plane
The Network Resource Perception Unit
Topology Discovery
Link Status Perception
Computing Routing Calculation Unit
⑤Provide Raw Network Resource Data
④Perceive Network Resources
Initiate Computing Request
SBI
Install Flow Table Entry
③Computing Resource Data
The Data Plane
Mininet Simulation Topology
②Provide Raw Computing Resource Data
The Computing Resource Perception Unit
The Computing Resource End
①Perceive Computing Resources

Figure 1. RSCR Architecture.

4.2. Architecture

(1) *The Data Plane:* The data plane is a collection of all infrastructure, including end devices, forwarding devices, and the links connecting them. Responsible for the actual processing and forwarding of data packets, it utilizes the programmability and real-time response of flow table rules. It executes instructions from the control plane, implements RSCR routing policies, and responds to the periodic transmission of computing resources

and network resource information by the computing resource perception unit and network resource awareness unit.

(2) *The Control Plane:* The control plane is a core component of the architecture, responsible for formulating network policies and dynamic management, constructing a global view of the data plane. Separated from the data plane, it uses a centralized controller to manage and adjust flow table rules of network devices in real-time, responding to the requirements of different applications and services to optimize network performance, security, and availability. This plane mainly consists of two modules: the network resource awareness unit and the computing route deployment unit.

(3) *The Management Plane:* The management plane is a crucial component of the SDN architecture, responsible for centralized network management and control. It provides network administrators with an abstract way to define network policies, configure devices, monitor network performance, and apply intelligent decision-making, achieving flexibility, automation, and intelligent management of the network. The management plane collaborates with the data plane, where the computing unit processes collected raw data on computing resources and network resources to segment computing nodes and calculate available bandwidth, latency, and packet loss rates. These data describe the computational capabilities and network status of computing nodes.

(4) *The Knowledge Plane:* The knowledge plane is a core component of RSCR, designed to integrate and manage widely distributed knowledge sources to support intelligent decision-making and reasoning. It provides functions for the storage, retrieval, inference, and application of knowledge, enabling the system to automatically analyze and understand information for better meeting user needs and supporting intelligent decision-making. This plane, through the management plane, processes well-handled computing-network resource data and computing task requests proposed by the data plane to calculate and install the optimal route. The deployment of the knowledge plane is highly flexible; it can be deployed on top of the control plane, similar to the application plane in standardized SDN [41], or deployed separately [34]. RSCR opts for a separate deployment of the knowledge plane to avoid overloading the control plane.

(5) *The Computing Resource Perception Unit:* This unit is responsible for perceiving the computing resources of each computing node in the computing resource end and sending the perceived raw data to the computing-network resource processing unit of the management plane. The computing resource perception process is illustrated in Figure 2, involving three main components: Client, Master, and Cluster. The Master is in charge of the entire Kubernetes cluster, composed mainly of API-Server, kube-scheduler, kube-controller-manager, and cloud-controller-manager. Additionally, the Metrics-Server is deployed on the Master. Nodes receive instructions from the Master to carry out tasks. It is noteworthy that the Metrics-Server is not part of the API-server; it is independently deployed based on the Aggregator plugin mechanism, providing services uniformly to the API-server. By accessing the /api/metrics.k8s.io/v1beta1 interface exposed by Metrics-Server, periodic data can be obtained. These data are collected from the kubelet's Summary API and include both cAdvisor's monitoring data and kubelet's summary information. Through processing the collected information, extracting computing resources is performed to achieve computing resource awareness.

(6) Network Resource Perception Unit: This unit is divided into two functions: Topology Discovery and Link Status Perception. The Topology Discovery module sends request messages to forwarding devices (switches) in the data plane, and the forwarding devices respond by sending feature information such as ID, port count, and port status. Based on the received messages, this module associates each port of each switch with neighboring switch ports and hosts connected to each switch port, thereby inferring the network topology. The Link Status Perception module is mainly responsible for perceiving fundamental data on link status in terms of computing delay (d_{link}), instantaneous throughput (bwu_{link}), and packet loss rate (l_{link}).

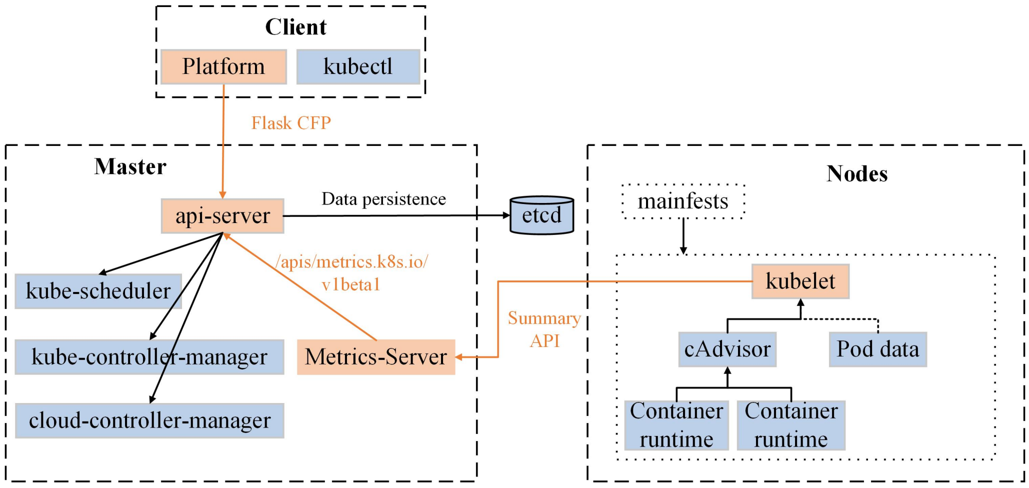


Figure 2. Computing Resource Perception Process.

(7) Computing-Network Resource Processing Unit: This unit mainly consists of two modules, namely Network Resource Processing and Computing Resource Processing. Network Resource Processing is responsible for processing the collected raw network data, calculating available bandwidth (bwa_{link}), delay (d_{link}), and packet loss (l_{link}). The controller periodically sends Port-Stats-Request messages to specified switches at each port. It retrieves the sent byte count (bt_1), received byte count (bt_2), and port lifetime (time difference between sending and receiving data) δt from the Port-Stats-Reply messages from the switches. With this information, it can calculate the instantaneous throughput $bwu_{link} = \frac{b_{t2} - b_{t1}}{\Delta t}$. Therefore, the available bandwidth of the link (bwa_{link}) is represented as the difference between link capacity (cap_{link}) and instantaneous throughput can be represented as $bwa_{link} = cap_{link} - bwu_{link}$. The link's instantaneous packet loss rate (l_{link}) can be represented as $l_{link} = \frac{b_{t1} - b_{t2}}{b_{t1}}$.

The calculation of link latency follows the method described in reference [42], which adheres to the Link Layer Discovery Protocol (LLDP) [43] and the OpenFlow protocol [44]. The SDN controller c_0 sends LLDP messages, and the messages traverse the path $c_0 - s_i - s_j - c_0$, where s_i, s_j represent the switches connected by the link (s_i, s_j). The time difference between the LLDP message transmission and reception, denoted as $d_{lldp_{cij}}$, estimates the time it takes for the message from c_0 to port s_i. This estimation considers half of the time for the transmission and reception of OpenFlow echo_request and echo_reply messages sent from c_0 to s_i. A similar process is used to estimate the time the message takes from s_j to c_0. Therefore, the instantaneous latency of the link (s_i, s_j), denoted as d_{link}, can be represented as $d_{link} = d_{s_i - s_j} = d_{lldp_{cij}} - d_{c_0 - s_i} - d_{s_j - c_0}$.

Computing-Network Resource Processing involves segmenting compute nodes based on their computing capabilities according to the DCRMM algorithm, using the collected compute resource data.

(8) Computing Routing Calculation Unit: Analyzing the current network link status and compute distribution based on computing-network resource data, coupled with the compute resource requirements of computing tasks, this unit calculates the optimal routing. The specific calculation method will be detailed in Section 5.

(9) Computing Routing Deployment Unit: This unit proactively writes flow table entries into the switches in the network through SDN applications, guiding the processing and forwarding of traffic. This process is typically accomplished using the Southbound Interface (SBI). By installing flow table entries on switches, the SDN controller can precisely define how to handle specific types of traffic, including routing, filtering, forwarding, or modification. The installation of flow table entries affects the traffic forwarding in the data plane [45]. Effective installation of flow table entries can optimize network performance and implement various network policies, but incorrect installation may lead to network failures or performance issues.

5. SD-CFN Routing

SD-CFN architecture's routing adopts the Q-Learning algorithm. Q-learning is a classic reinforcement learning algorithm used to solve Markov Decision Process (MDP) problems. Its core idea is to learn a state-action value function $Q_t(S_t, A_t)$, which evaluates the long-term expected return for taking a specific action in a given state. This algorithm is a model-free algorithm, so it does not require knowledge of the potential returns for taking specific actions in specific states [46]. For the state-action value function $Q_t(S_t, A_t)$, its update rule can be expressed as:

$$Q_t(S_t, A_t) = (1-\alpha)Q_t(S_t, A_t) + \alpha(r + \gamma max_{A_{t+1}} Q_{t+1}(S_{t+1}, A_{t+1})) \tag{16}$$

Here, $Q_t(S_t, A_t)$ is the estimated value (Q-value) of selecting action A_t in state S_t, α is the learning rate, r is the immediate reward obtained after selecting action A_t in state S_t, γ is the discount factor, and $max_{A_{t+1}} Q_{t+1}(S_{t+1}, A_{t+1})$ represents the maximum estimated value after selecting the optimal action A_{t+1} in the next state S_{t+1}.

In this paper, the learning of the agent involves a series of steps transitioning from the initial state to the target state (the computing task requester and the target computing node). Each step includes selecting and executing actions, changing the state (transitioning from one state to another), and receiving rewards. The updated Q-function values are based on the fundamental rewards obtained by taking action in the state, providing the optimal reward. Next, we will provide a detailed introduction to the RL agent and the RL routing algorithm.

5.1. Reinforcement Learning Agent

(1) State Space (S): Each state in the state space corresponds to a forwarding device in the data plane, specifically referring to SDN switches in this context. State transitions correspond to links connecting two switches. Therefore, the size of the state space is equal to the number of switches in the network topology.

(2) Action Space (A): It corresponds to the set of all actions taken for all states within the state space A. For a given state $s_i \in S$, the reinforcement learning agent can take any action, and each action leads to a transition from state s_i to one of its neighboring states. The neighboring states of state s_i correspond to the adjacent switches related to the state s_i. Therefore, the number of actions for state s_i is equal to its degree $dr(s_i)$, which is the number of switches connected to that state. The size of the action space A can be defined as $|A| \equiv \sum_{s_i \in S} dr(s_i)$.

(3) Reward Function: The reward is inversely proportional to the available bandwidth bwa_{link}, and directly proportional to the delay d_{link} and packet loss rate l_{link}. $\beta_1, \beta_2, \beta_3 \in [0,1]$ are tunable parameters, allowing for different weights to be set for different features.

$$R = \beta_1 \frac{1}{bwa_{link}} + \beta_2 d_{link} + \beta_3 l_{link} \tag{17}$$

Due to the different dimensions of bandwidth bwa_{link}, latency d_{link}, and packet loss rate l_{link}, it is necessary to normalize using Equation (2). Equation (18) provides the normalized reward function:

$$R' = \beta_1 \frac{1}{bwa'_{link}} + \beta_2 d'_{link} + \beta_3 l'_{link} \tag{18}$$

(4) Optimal Policy: The goal of the optimal policy is to minimize the reward values in the Q-Learning routing process. In this way, the agent can choose paths with high available bandwidth, low latency, and a small packet loss rate during the routing process. The intelligent agent approximates the optimal Q-function by accessing all action-state pairs. It updates and stores Q-values in the Q-table, which is used to find the optimal path from the computing request source to the target computing node. The Q-value is a measure of the overall expected reward when the reinforcement learning agent is in

state S_t and takes action A_t. The reinforcement learning agent updates the Q-value using Equation (19). In Equation (19), the updated Q-value (Q_{t+1}) depends on the previous value Q_t, and it is influenced by the result (S_t, A_t, R_t, S_{t+1}) and α. R_t is the reward at time t, and α determines the weight relationship between the newly acquired information and previous information. $\alpha = 0$ would make the reinforcement learning agent unable to learn from the latest (S_t, A_t) pair, while $\alpha = 1$ allows the agent to retain learned information by considering the immediate reward Rt for the (S_t, A_t) pair.

$$Q_{t+1}(S_{t+1}, A_{t+1}) = Q_t(S_t, A_t) + \alpha[R_t + min_A Q_t(S_{t+1}, A) - Q_t(S_t, A_t)] \tag{19}$$

(5) Exploration and Exploitation Strategy: In Q-Learning, there is a trade-off between choosing the expected optimal action (exploitation) and choosing different actions in the hope of obtaining greater rewards in the future (exploration) [47]. This paper adopts the ϵ-greedy exploration and exploitation method. ϵ-greedy uses $\epsilon \in [0,1]$ as a tunable parameter, allowing the intelligent agent to exploit with probability $pr = \epsilon$ and explore with probability $pr = 1 - \epsilon$. Therefore, this parameter determines the degree to which the reinforcement learning agent explores and exploits during the learning process. The agent uses Equation (20) to choose the next action for a specific state. At each step, it generates a random value $x \in [0,1]$. If $x < \epsilon$, the agent exploits; otherwise, the agent explores.

$$\mathcal{A} = \begin{cases} min_A Q_t(S_t, A), x < \epsilon \\ random\ action, otherwise \end{cases} \tag{20}$$

5.2. Routing Algorithm

This routing algorithm implements a learning process for finding the optimal paths for all node pairs in the data plane. The input data to Algorithm 1 include the learning rate α, parameter ϵ (see Equation (20)), the number of learning rounds n, connection of all nodes $Node\_list$, link status between all nodes $Link\_status$, and the pair of the computing resource requester and the target computing node (T_{src}, T_{dst}). The output is the set of optimal reward routes for all node pairs given the link state. This path is formed by state-action pairs with the lowest values in the Q-table. Finally, based on the distribution of node computing power, the optimal path from the computing request source to the target computing node is determined as the final result of computing routing. This path is then installed in the switches as flow tables, facilitating the scheduling of computing resources through the network.

The RSCR routing algorithm can find the optimal path for all pairs of nodes in the given topology. For each pair of nodes (src, dst) in the Nodelist, routing learning can be performed following the steps in Algorithm 1, lines 1–17. First, initialize the Q-table to 0 (line 2), and then, the algorithm goes through episodes (line 3) until the state S_t becomes the final state dst (lines 5–9). Starting with src as the initial state (line 4), assuming the next state is not the target state dst (line 5), the agent selects the action A_t based on Equation (20) (line 6). The agent calculates the reward value according to Equation (18) (line 7), then updates the state using Equation (19) (line 8), and moves to the new state (line 9) until the episode concludes.

After the RL agent completes learning, the optimal path between src and dst can be calculated based on the Q-table, specifically the path with the minimum Q-value (line 12). If (src, dst) is the same as (T_{src}, T_{dst}), then this node pair corresponds to the computing resource requester and target computing node pair we are looking for, and the corresponding path is the optimal computing resource path (lines 14–15). Finally, the knowledge plane stores routing information for all node pairs in $Paths$, and the optimal computing resource path is specially marked. Subsequently, the flow installation module retrieves these paths and installs them in the routing table of switches.

Algorithm 1: SD-CFN Routing: RSCR

```
Data: Learning rate: $\alpha$
      Exploration and exploitation parameter: $\epsilon$
      Learning episodes: $n$
      Connection of all nodes $(src, dst)$: $Node\_list$
      Link status between all nodes: $Link\_status$
      Node pairs of Initiating and the optimal node of the computing resource
      request: $(T_{src}, T_{dst})$
Result: The optimal path set of the given nodes: $Paths$
1  while $(src, dst) \in Node\_list$ do
2      Initialization $Q$: $Q(S, A) = 0, \forall s \in S, \forall a \in A$;
3      for $episode \leftarrow 1$ to $n$ do
4          Start in state $S_t = src \in S$;
5          while $S_{t+1}$ is not $dst$ do
6              Select $A_t$ for $S_t$ with policy derived from Q using $\epsilon$-greedy exploration
                 and exploitation method;
7              $R_{t+1} \leftarrow R(S_t, A_t)$;
8              $Q_{t+1}(S_{t+1}, A_{t+1}) = Q_t(S_t, A_t) + \alpha[R_t + min_A Q_t(S_{t+1}, A) - Q_t(S_t, A_t)]$;
9              $S_t \leftarrow S_{t+1}$
10         end
11     end
12     Find the path with the minimum Q-value in the Q-table for the pair $(src, dst)$.
13 end
14 if $(src, dst) = (T_{src}, T_{dst})$ then
15     $(T_{src}, T_{dst})$ is the optimal path from the computing resource requester to the
         target computing node
16 The knowledge Plane stores all routing information $Paths$, and additionally marks
     and stores the path $(T_{src}, T_{dst})$ for the computing resource requester and the
     target computing node.
17 return $Result$;
```

6. Results and Analysis

This section introduces the evaluation of RSCR. Section 6.1 describes the testing environment, Section 6.2 discusses signal generation, Section 6.3 introduces traffic generation, and Section 6.4 analyzes the experimental results.

6.1. Test Environment

Figure 3 shows the test topology of RSCR, namely the GÉANT topology. This topology is an international network in Europe used for education and research. GÉANT consists of 23 switches and 37 links, with 50% of the links at 10 Gbps, 40% at 2.5 Gbps, and 10% at 155 Mbps. We deployed the mentioned GÉANT topology in Mininet 2.3.1, with each switch connected to a host responsible for forwarding and receiving traffic. As Mininet's virtual hosts cannot effectively provide real and valid computing resource information, the computing resource data were based on real-time data from a Kubernetes cluster.

The data plane of RSCR is composed of the GÉANT topology. In the control plane, a Ryu controller is deployed, and topology discovery, link-state perception, and flow table installation are all developed based on this controller. The management plane involves data processing using Python 3.9 with Pandas 1.5.3 and Numpy 1.24.3 libraries. The knowledge plane, developed using Python 3.9 and RL, is responsible for computing algorithmic routing decisions. We store link-state information in the CSV format and the computed routing results in the JSON format. The experimental environment is deployed on a system with an 11th Gen Intel(R) Core(TM) i7-1165G7 processor and 16GB RAM running Ubuntu Desktop 20.04.

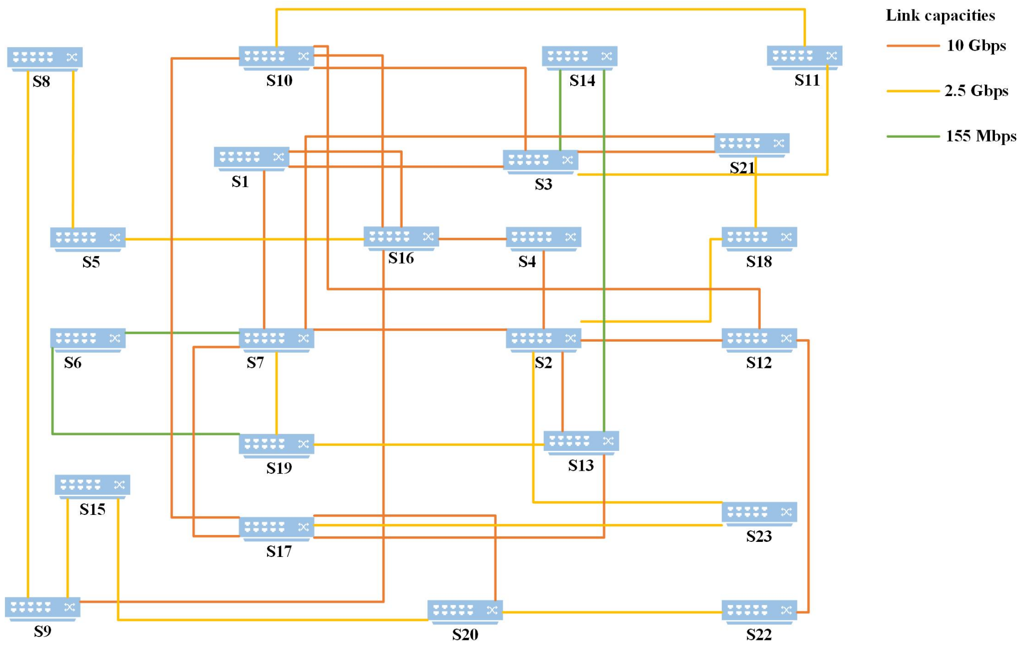


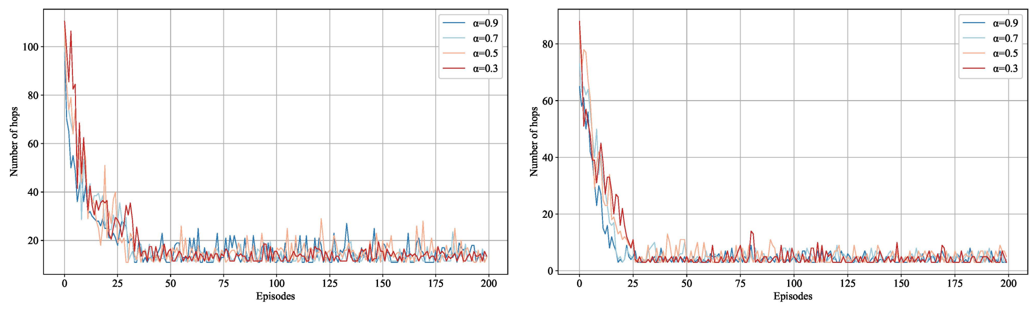
Figure 3. GÉANT topology.

6.2. Traffic Generation

iperf is a commonly used traffic generation tool in Mininet, and, accordingly, we have developed an iperf script to run on both the host and client sides. Following reference [48], the authors computed a traffic matrix for approximately 4 months at 15-minute intervals to conform to the XML format of TOTEM, providing the traffic matrix. We employ this traffic matrix to introduce background traffic to the GÉANT topology.

6.3. Parameters Setup

In RL algorithms, determining the appropriate values for the learning rate (α) and exploration rate (ϵ) is a crucial issue. To address this, we use the number of hops in the path between the source and destination addresses as a criterion. Initially, we run the RSCR routing algorithm, continuously adjusting α and ϵ to explore potential paths between Switch 2 and Switch 22. When link costs are equal, the shortest path found by Dijkstra's algorithm is three hops. For each $\epsilon(0.8, 0.6)$, we test four α values $(0.9, 0.7, 0.5, 0.3)$ for 200 iterations each time. Figure 4 illustrates the number of hops between Switch 2 and Switch 22 under different parameters. As observed in the figure, reducing ϵ leads the RL agent to explore random actions in the action space, increasing the number of hops. Conversely, the agent tends to leverage previous experiences when ϵ is high. For $\alpha = 0.9$ and $\alpha = 0.7$, it is evident that the convergence speed of $\alpha = 0.9$ is superior to that of $\alpha = 0.7$. Based on these results, the most suitable values for the parameters of the RL agent in RSCR are $\alpha = 0.9$ and $\epsilon = 0.8$.



(**a**) $\epsilon = 0.6$ (**b**) $\epsilon = 0.8$

Figure 4. Number of Hops for Different Learning Parameters.

6.4. Results and Analysis

Figure 5 illustrates the impact of DCRMM, MDA, and MSA on the stability of computing nodes. We conducted tests on the average decrease rates in CPU, Memory (Mem), and Storage for all nodes after executing 10 tasks. The average decrease rate is defined by Equation (21), where $Node_{pre}$ represents the average utilization before task execution, and $Node_{now}$ represents the average utilization after task execution. The average CPU decrease rate of DCRMM is 1.60% lower than MSA and 1.45% lower than MDA. The average Mem decrease rate is 2.43% lower than MSA and 0.14% lower than MDA. For Storage, the average decrease rate is 15.80% lower than MSA and 3.48% lower than MDA. This indicates that DCRMM can reduce the variability of computational resources on nodes, enhancing stability among nodes.

$$S = Node_{pre} - Node_{now} \tag{21}$$

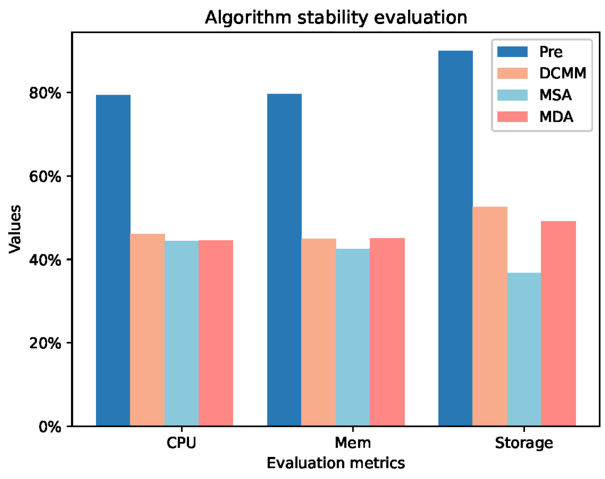


Figure 5. Algorithm Stability Assessment.

Figure 6 depicts the relationship between node utilization and the number of tasks for DCRMM, MDA, and MSA. As shown in Equation (22), node utilization (U) is represented as the ratio of selected nodes ($Node_s$) to the total number of nodes ($Node_t$). From Figure 6, it is evident that with an increase in the number of tasks, the node utilization of DCRMM, MDA, and MSA all experiences a corresponding improvement. However, it is clear from the figure that the node utilization of DCRMM is significantly better than that of MDA and MSA, both at low and high task numbers. This is attributed to the fact that DCRMM can simultaneously consider static and dynamic metrics, enhancing the precision of node-task matching.

$$U = \frac{Node_s}{Node_t} \tag{22}$$

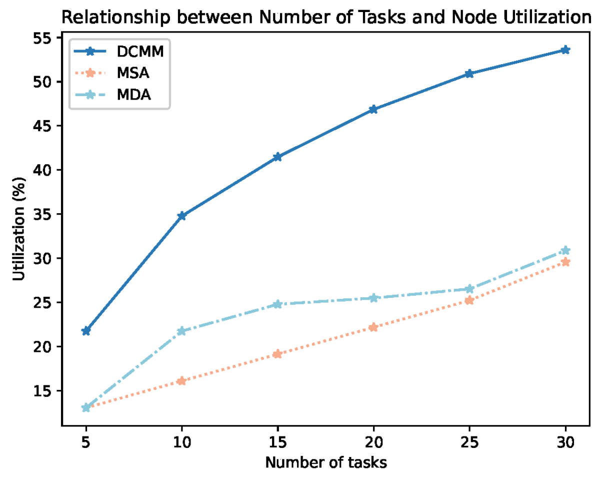


Figure 6. Relationship Between Node Utilization and Number of Tasks.

Figure 7 represents the relationship between the matching accuracy of selected nodes by DCRMM, MDA, and MSA and the number of tasks. As indicated in Equation (23), the matching accuracy (A) is represented as the ratio of nodes meeting the requirements of tasks ($Node_m$) to the selected nodes ($Node_s$). According to the figure, the matching accuracy of DCRMM can be maintained at 100%, while MDA has a matching accuracy of only below 60%, and MSA is below 40%. Moreover, with an increase in the number of tasks, both MDA and MSA show a decline in matching accuracy to varying degrees. This is because the DCRMM algorithm considers both dynamic and static metrics, performs node matching by calculating Euclidean distance, and prioritizes the removal of nodes that do not meet the requirements before computing the Euclidean distance. This ensures that all nodes in the filtered pool can meet the computational requirements of the tasks.

$$A = \frac{Node_m}{Node_s} \tag{23}$$

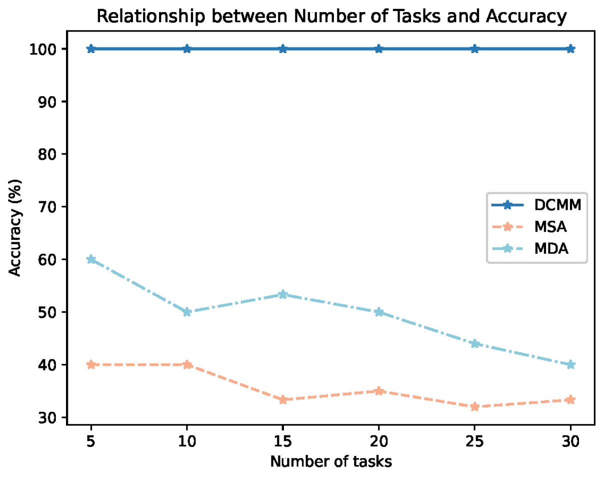


Figure 7. Relationship Between Node Accuracy and Number of Tasks.

Figure 8 illustrates the average link delay of the RSCR. According to Figure 8, it can be observed that the average delay of the RSCR is comparable to that of the OSPF algorithm. However, when compared to the OSPF_delay algorithm, the average delay of the RSCR algorithm is lower by 63%. This is because the RSCR exhibits a preference for paths with fewer hops and lower congestion levels in route selection.

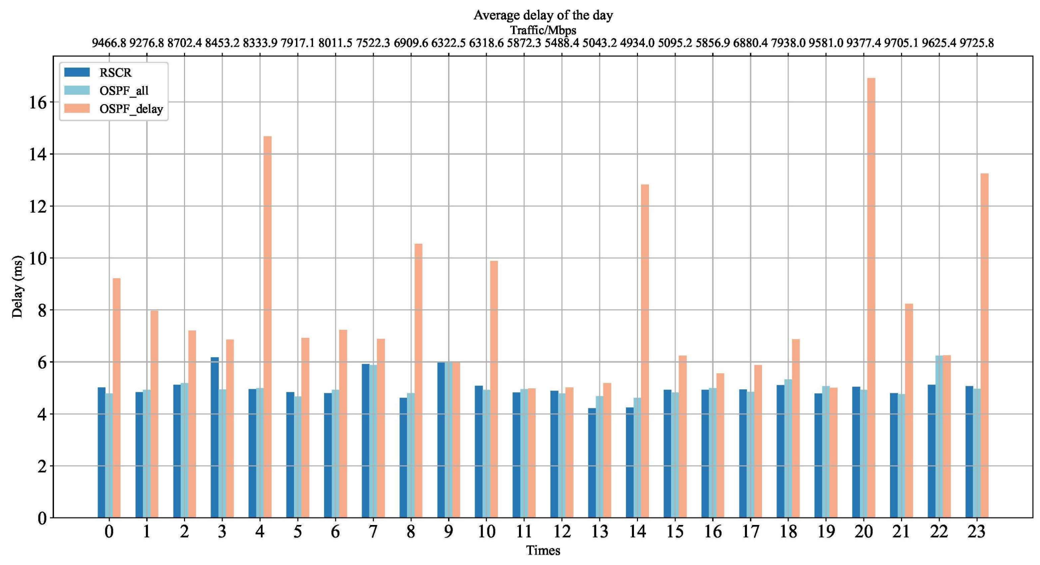


Figure 8. Average Delay Comparison Chart.

Figure 9 depicts the average packet loss rate of the RSCR algorithm. According to Figure 9, it is evident that the packet loss rate of RSCR is 15% and 25% lower than OSPF_all and OSPF_loss, respectively. In some scenarios, the packet loss rate of OSPF_all is more than twice that of the RSCR algorithm. This is attributed to the RSCR algorithm's capability to select relatively shorter paths, thereby reducing packet loss.

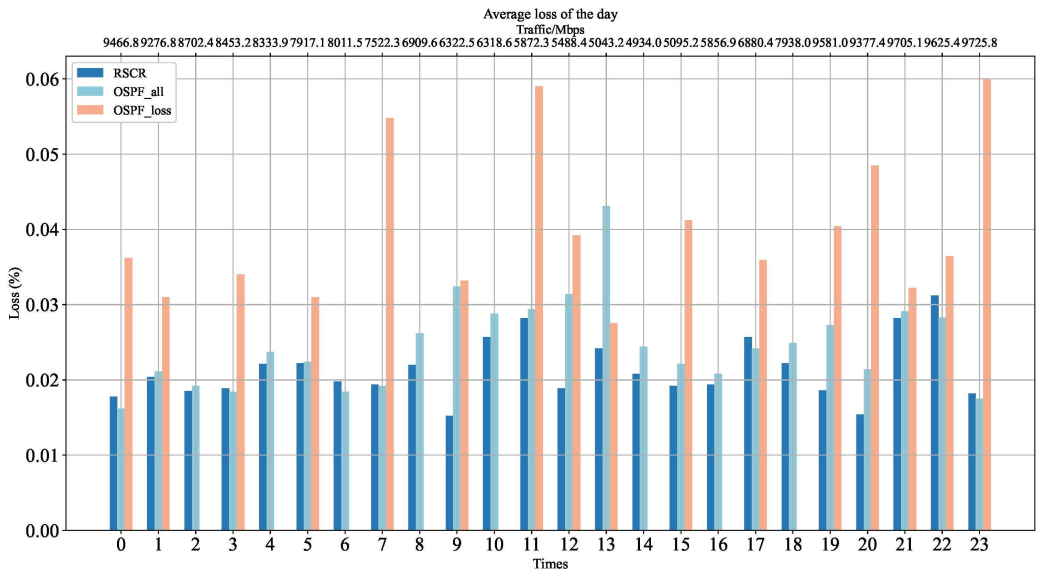


Figure 9. Average Packet Loss Rate Comparison Chart.

Figure 10 illustrates the average link throughput of RSCR throughout the day. The throughput of RSCR is lower than OSPF_all and OSPF_bw, primarily because RSCR selects paths that are less preferred by the latter two, resulting in a larger denominator when calculating the average link throughput. This also explains why the average delay and average packet loss rate of RSCR outperform other algorithms, while the throughput remains at a lower level.

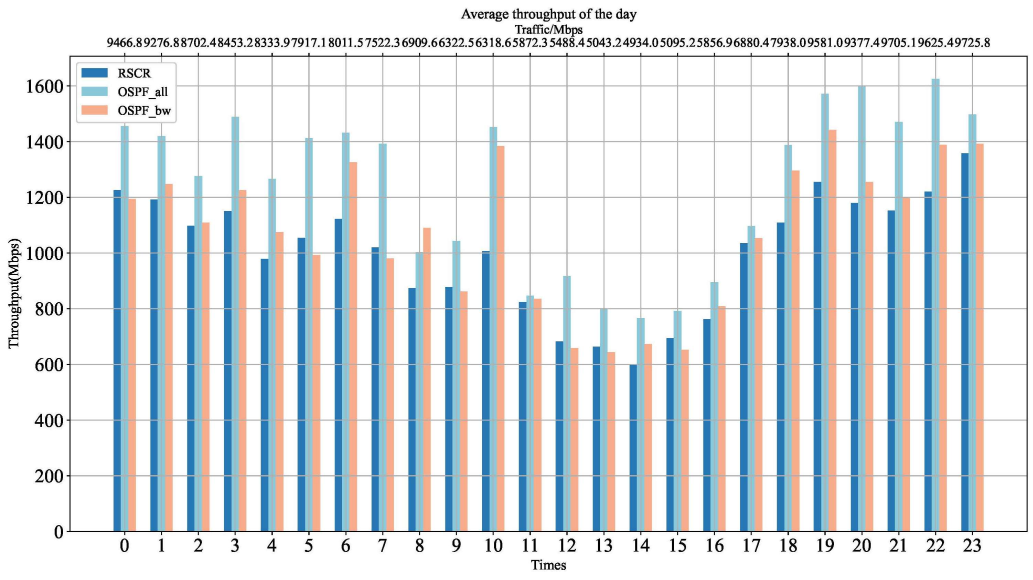


Figure 10. Average Throughput Comparison Chart.

7. Conclusions and Future Work

This paper proposes a unified modeling and measurement method for the dynamic and static performance indicators of computing nodes, named DCRMM. The paper provides a detailed explanation of how to select the optimal computing nodes based on measurement results. Experimental results indicate that compared to computing node measurement methods that only consider dynamic or static indicators, DCRMM demonstrates superior node stability, node utilization, and node matching accuracy, enabling more effective utilization of node resources. Additionally, the paper discusses a computing power routing algorithm, RSCR, based on the SD-CFN architecture. This algorithm employs a Q-Learning mechanism to effectively compute paths between the computing power requester and the computing power provider. We validate the algorithm on the GÉANT topology, and the experiments show that RSCR can ensure a significant reduction in packet loss while maintaining a delay close to that of the OSPF algorithm. Furthermore, it increases link utilization, allowing traffic to be distributed more widely across the entire network topology.

Based on this, we hope that future work can explore the use of transfer learning algorithms for more experimentation on dynamic topologies. This would enable the algorithm to make new decisions quickly in situations where the topology is constantly changing.

Author Contributions: Conceptualization, X.G. and S.R.; formal analysis, software, writing, X.G. and S.R.; supervision, C.W.; investigation and methodology, J.W.; validation and visualization, X.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The code of this article is open source at https://github.com/ryan1016/RSCR/ (will be available online in March 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Tian, L.; Yang, M.Z.; Wang, S.G. An overview of compute first networking. *Int. J. Web Grid Serv.* **2021**, *17*, 81–97. [CrossRef]
2. Tang, X.Y.; Cao, C.; Wang, Y.X.; Zhang, S.; Liu, Y.; Li, M.X.; He, T. Computing Power Network: The Architecture of Convergence of Computing and Networking towards 6G Requirement. *China Commun.* **2021**, *18*, 175–185. [CrossRef]
3. Yu, C.D.; Xia, G.M.; Wang, Z.H.; Soc, I.C. Trust Evaluation of Computing Power Network Based on Improved Particle Swarm Neural Network. In Proceedings of the 17th IEEE International Conference on Mobility, Sensing and Networking (MSN), Exeter, UK, 13–15 December 2021; pp. 718–725. [CrossRef]
4. Institute, C.U.R. China Unicom Computing First Network White Paper. Report, 2019. Available online: http://www.bomeimedia.com/China-unicom/white_paper/20191101-06.pdf (accessed on 5 December 2023).
5. Król, M.; Mastorakis, S.; Oran, D.; Kutscher, D. Compute first networking: Distributed computing meets icn. In Proceedings of the 6th ACM Conference on Information-Centric Networking, Macao, China, 24–26 September 2019; pp. 67–77.
6. Xu, Q.; Zhang, Y.; Wu, K.; Wang, J.; Lu, K. Evaluating and boosting reinforcement learning for intra-domain routing. In Proceedings of the 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), Monterey, CA, USA, 4–7 November 2019; pp. 265–273.
7. Moy, J. OSPF Version 2. Report 2070-1721, 1997. Available online: https://www.rfc-editor.org/rfc/rfc2178.html (accessed on 5 December 2023).
8. Bernard, F. Internet traffic engineering by optimizing OSPF weights. In Proceedings of the IEEE INFOCOM 2000, Conference on Computer Communications, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064), Tel Aviv, Israel, 26–30 March 2000.
9. Chen, Y.R.; Rezapour, A.; Tzeng, W.G.; Tsai, S.C. RL-Routing: An SDN Routing Algorithm Based on Deep Reinforcement Learning. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 3185–3199. [CrossRef]
10. Oliveira, C.A.; Pardalos, P.M. *Mathematical Aspects of Network Routing Optimization*; Springer: Berlin/Heidelberg, Germany, 2011.
11. Nayak, P.; Swetha, G.; Gupta, S.; Madhavi, K. Routing in wireless sensor networks using machine learning techniques: Challenges and opportunities. *Measurement* **2021**, *178*, 108974. [CrossRef]
12. Jacob, D.I.J.; Darney, D.P.E. Artificial bee colony optimization algorithm for enhancing routing in wireless networks. *J. Artif. Intell. Capsul. Netw.* **2021**, *3*, 62–71. [CrossRef]
13. Jiang, W. Graph-based deep learning for communication networks: A survey. *Comput. Commun.* **2022**, *185*, 40–54. [CrossRef]
14. Xin, L.; Song, W.; Cao, Z.; Zhang, J. Step-wise deep learning models for solving routing problems. *IEEE Trans. Ind. Informatics* **2020**, *17*, 4861–4871. [CrossRef]
15. Guo, Y.; Luo, H.; Wang, Z.; Yin, X.; Wu, J. Routing optimization with path cardinality constraints in a hybrid SDN. *Comput. Commun.* **2021**, *165*, 112–121. [CrossRef]
16. Bagaa, M.; Dutra, D.L.C.; Taleb, T.; Samdanis, K. On SDN-driven network optimization and QoS aware routing using multiple paths. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 4700–4714. [CrossRef]
17. Tu, Z.; Zhou, H.; Li, K.; Li, G.; Shen, Q. A routing optimization method for software-defined SGIN based on deep reinforcement learning. In Proceedings of the 2019 IEEE Globecom Workshops (GC Wkshps), Waikoloa, HI, USA, 9–13 December 2019; pp. 1–6.
18. He, Q.; Wang, Y.; Wang, X.; Xu, W.; Li, F.; Yang, K.; Ma, L. Routing optimization with deep reinforcement learning in knowledge defined networking. *IEEE Trans. Mob. Comput.* **2023**, *23*, 1444–1455. [CrossRef]
19. Phan, T.; Feld, S.; Linnhoff-Popien, C. Artificial intelligence—The new revolutionary evolution. *Digit. Welt* **2020**, *4*, 7–8. [CrossRef]
20. Boyan, J.; Littman, M. Packet routing in dynamically changing networks: A reinforcement learning approach. In Proceedings of the Advances in Neural Information Processing Systems, Denver, CO, USA, 29 November–2 December 1993; Volume 6.
21. Lin, S.C.; Akyildiz, I.F.; Wang, P.; Luo, M. QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach. In Proceedings of the 2016 IEEE International Conference on Services Computing (SCC), San Francisco, CA, USA, 27 June–2 July 2016; pp. 25–33.
22. Hassas Yeganeh, S.; Ganjali, Y. Kandoo: A framework for efficient and scalable offloading of control applications. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, Helsinki Finland, 13 August 2012; pp. 19–24.
23. McCauley, J.; Panda, A.; Casado, M.; Koponen, T.; Shenker, S. Extending SDN to large-scale networks. *Open Netw. Summit* **2013**, 1–2.
24. Rischke, J.; Sossalla, P.; Salah, H.; Fitzek, F.H.; Reisslein, M. QR-SDN: Towards reinforcement learning states, actions, and rewards for direct flow routing in software-defined networks. *IEEE Access* **2020**, *8*, 174773–174791. [CrossRef]
25. Hassen, H.; Meherzi, S.; Jemaa, Z.B. ϵ-QLMR: ϵ-greedy based Q-Learning algorithm for Multipath Routing in SDN networks. In Proceedings of the 2023 International Wireless Communications and Mobile Computing (IWCMC), Marrakesh, Morocco, 19–23 June 2023; pp. 234–239.
26. Casas-Velasco, D.M.; Rendon, O.M.C.; da Fonseca, N.L. Intelligent routing based on reinforcement learning for software-defined networking. *IEEE Trans. Netw. Serv. Manag.* **2020**, *18*, 870–881. [CrossRef]
27. Casas-Velasco, D.M.; Rendon, O.M.C.; Fonseca, N.L.S.d. DRSIR: A Deep Reinforcement Learning Approach for Routing in Software-Defined Networking. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 4807–4820. [CrossRef]
28. Yu, C.; Lan, J.; Guo, Z.; Hu, Y. DROM: Optimizing the routing in software-defined networks with deep reinforcement learning. *IEEE Access* **2018**, *6*, 64533–64539. [CrossRef]

29. Zhao, C.; Ye, M.; Xue, X.; Lv, J.; Jiang, Q.; Wang, Y. DRL-M4MR: An intelligent multicast routing approach based on DQN deep reinforcement learning in SDN. *Phys. Commun.* **2022**, *55*, 101919. [CrossRef]
30. Zhu, Y.; Tian, D.; Yan, F. Effectiveness of entropy weight method in decision-making. *Math. Probl. Eng.* **2020**, *2020*, 1–5. [CrossRef]
31. Taheriyoun, M.; Karamouz, M.; Baghvand, A. Development of an entropy-based fuzzy eutrophication index for reservoir water quality evaluation. *J. Environ. Health Sci. Eng.* **2010**, *7*, 1–14.
32. Carbó-Dorca, R.; Besalú, E. Geometry of n-dimensional Euclidean space Gaussian enfoldments. *J. Math. Chem.* **2011**, *49*, 2244–2249. [CrossRef]
33. Clark, D.D.; Partridge, C.; Ramming, J.C.; Wroclawski, J.T. A knowledge plane for the internet. In Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Karlsruhe, Germany, 25–29 August 2003; pp. 3–10.
34. Mestres, A.; Rodriguez-Natal, A.; Carner, J.; Barlet-Ros, P.; Alarcón, E.; Solé, M.; Muntés-Mulero, V.; Meyer, D.; Barkai, S.; Hibbett, M.J. Knowledge-defined networking. *ACM SIGCOMM Comput. Commun. Rev.* **2017**, *47*, 2–10. [CrossRef]
35. Black, U.D. *IP Routing Protocols: RIP, OSPF, BGP, PNNI, and Cisco Routing Protocols*; Prentice Hall Professional: Hoboken, NJ, USA, 2000.
36. Aldabbas, H. Efficient bandwidth allocation in SDN-based peer-to-peer data streaming using machine learning algorithm. *J. Supercomput.* **2023**, *79*, 6802–6824. [CrossRef]
37. Martín, I.; Troia, S.; Hernández, J.A.; Rodríguez, A.; Musumeci, F.; Maier, G.; Alvizu, R.; de Dios, O.G. Machine learning-based routing and wavelength assignment in software-defined optical networks. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 871–883. [CrossRef]
38. Samadi, R.; Nazari, A.; Seitz, J. Intelligent energy-aware routing protocol in mobile IoT networks based on SDN. *IEEE Trans. Green Commun. Netw.* **2023**, *7*, 2093–2103. [CrossRef]
39. Feng, H.; Wang, J.; Fang, Z.; Chen, J.; Do, D.T. Evaluating AoI-Centric HARQ Protocols for UAV Networks. *IEEE Trans. Commun.* **2023**, *72*, 288–301. [CrossRef]
40. Wang, J.; Jiao, Z.; Chen, J.; Hou, X.; Yang, T.; Lan, D. Blockchain-Aided Secure Access Control for UAV Computing Networks. *IEEE Trans. Netw. Sci. Eng.* **2023**, 1–14. [CrossRef]
41. ONF. SDN Architecture-Issue 1.1-ONF TR-521. 2019. Available online: https://opennetworking.org/wp-content/uploads/2014/10/TR-521_SDN_Architecture_issue_1.1.pdf (accessed on 5 December 2023).
42. Liao, L.; Leung, V.C. LLDP based link latency monitoring in software defined networks. In Proceedings of the 2016 12th International Conference on Network and Service Management (CNSM), Montreal, QC, Canada, 31 October–4 November 2016; pp. 330–335.
43. Shu, Z.; Wan, J.; Lin, J.; Wang, S.; Li, D.; Rho, S.; Yang, C. Traffic engineering in software-defined networking: Measurement and management. *IEEE Access* **2016**, *4*, 3246–3256. [CrossRef]
44. Ryu application API. Website, 2023. Available online: https://ryu.readthedocs.io/en/latest/ryu_app_api.html (accessed on 5 December 2023).
45. Achleitner, S.; Bartolini, N.; He, T.; La Porta, T.; Tootaghaj, D.Z. Fast network configuration in software defined networking. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1249–1263. [CrossRef]
46. Mammeri, Z. Reinforcement learning based routing in networks: Review and classification of approaches. *IEEE Access* **2019**, *7*, 55916–55950. [CrossRef]
47. Tijsma, A.D.; Drugan, M.M.; Wiering, M.A. Comparing exploration strategies for Q-learning in random stochastic mazes. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece, 6–9 December 2016; pp. 1–8.
48. Uhlig, S.; Quoitin, B.; Lepropre, J.; Balon, S. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Comput. Commun. Rev.* **2006**, *36*, 83–86. [CrossRef]