Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 1330993, 16 pages
https://doi.org/10.1155/2022/1330993

Research Article

Reinforcement Learning-Based Service-Oriented Dynamic Multipath Routing in SDN

Kai-Cheng Chiu, Chien-Chang Liu, and Li-Der Chou

Department of Computer Science and Information Engineering, National Central University, Taoyuan 32001, Taiwan

Correspondence should be addressed to Li-Der Chou; cld@csie.ncu.edu.tw

Received 14 July 2021; Revised 8 December 2021; Accepted 14 December 2021; Published 31 January 2022

Academic Editor: Jianhui Lv

The increasing quality and various requirements of network services are guaranteed because of the advancement of the emerging network paradigm, software-defined networking (SDN), and benefits from the centralized and software-defined architecture. The SDN not only facilitates the configuration of the network policies for traffic engineering but also brings convenience for network state obtainment. The traffic of numerous services is transmitted within a network, whereas each service may demand different network metrics, such as low latency or low packet loss rate. Corresponding quality of service policies must be enforced to meet the requirements of different services, and the balance of link utilization is also indispensable. In this research, Reinforcement Discrete Learning-Based Service-Oriented Multipath Routing (RED-STAR) has been proposed to understand the policy of distributing an optimal path for each service. The RED-STAR takes the network state and service type as input values to dynamically select the path a service must be forwarded. Custom protocols are designed for network state obtainment, and a deep learning-based traffic classification model is also integrated to identify network services. With the differentiated reward scheme for every service type, the reinforcement learning model in RED-STAR gradually achieves high reward values in various scenarios. The experimental results show that RED-STAR can adopt the dynamic network environment, obtaining the highest average reward value of 1.8579 and the lowest average maximum bandwidth utilization of 0.3601 among all path distribution schemes in a real-case scenario.

1. Introduction

As the diversity of network services increases, users accordingly demand high quality of service (QoS) [1]. Each service may be pursued with different network metrics, such as less response time for voice over Internet protocol (VoIP) and low packet loss rate for file transmission. A traffic engineering scheme must forward the traffic of specific services to routes, whereas the traffic of various services is being transmitted within a network. Some routes have different attributes within a network; thus, the service traffic must be appropriately routed to the corresponding paths. Meanwhile, the utilization of every link must also be balanced. With the abovementioned issues, the main problem can be formulated as follows: given a set of services and network link states, an optimal path must be assigned for the traffic of each network service to meet its QoS requirements and the link utilization must be balanced as much as possible.

The emerging software-defined networking (SDN) paradigm will be a potential solution to dynamically assign paths and obtain network link states. The SDN with the global view of the network enables the rapid and dynamic deployment of network policies [2, 3], which are widely used in enterprise and wide area networks. The control plane within a traditional network device is separated from the data plane in SDN, and a logically centralized controller comes into being. The controller communicates with the network devices via an open-standard protocol, namely, OpenFlow, and the switches that support OpenFlow are known as OpenFlow Switches (OFSs) [4]. By adding flow rules via OpenFlow to an OFS, the OFS can execute the instructions designated by the controller. The flow rules are

in charge of either modifying the packet header fields or forwarding the traffic, which can be used to carry out policies to meet the QoS requirements [5].

In the environment of SDN to enforce policies, a corresponding traffic engineering algorithm or mechanism can be used in the controller [6, 7]. However, before we distribute the paths for each service, the service traffic must be first identified, which is known as a traffic classification (TC) task. The TC can be approximately categorized into three approaches [8, 9]: (a) port-based, (b) payload-based, and (c) machine learning-based. Traditional port-based methods identify packets by the well-known port numbers [10] assigned by the Internet Assigned Numbers Authority [11], which is an instant TC scheme but suffers from dynamic port number utilization [12, 13]. Payload-based approaches inspect the payload within a packet by predefined patterns, which can handle dynamic port numbers but are weak at processing encrypted traffic [14, 15]. Machine learning-based approaches exploit various algorithms to classify service traffic by taking either statistical information or packet bytes as input values [16–18]. In this research, a deep learning TC model constituted of the autoencoder and 1D convolutional neural network (CAPC) is used in the SDN controller. The data collection and processing methods, model construction and training, and performance evaluation of the TC model are presented in our previous work [19]. This study is the first to integrate the deep learning TC model within a network environment.

After service classification, an identified service can be assigned to an ideal route by the learning-based algorithm. This work aims to support the services with their required QoS and simultaneously balance the traffic load. The Reinforcement Discrete Learning Service-Oriented Multipath Routing (RED-STAR) mechanism is proposed to dynamically distribute routes in a network to every service and to tackle the problem. As a deep reinforcement learning (DRL) [20] method, RED-STAR considers the network metrics, that is, bandwidth utilization, link latency, and packet loss rate, as the environment state. The metrics are periodically measured and updated for the route distribution task. However, errors of the obtained measurements may occasionally occur because of software simulation or hardware defects. A metric regularization scheme is included in this work. The deep neural network (DNN) model in RED-STAR takes the regularized environment attributes as input values and generates the output via its inner neural network (NN) computation. Each output value of the DNN model represents the reward value of a route, also known as an action, and the action with the highest reward value is the best route in DNN's perception. The reward scheme is inconsistent for different genres of services because of the varying degrees of QoS. For example, VoIP services attach great importance to latency; thus, high latency results in a low reward. Text messages concern more on packet loss rate; thus, a high packet loss rate also leads to a low reward. The differentiated reward scheme prompts the RED-STAR to allocate the appropriate path to the corresponding service traffic. In addition, high unbalanced link utilization incurs a low reward to balance the utilization of links. The "discrete" learning in RED-STAR is a slight modification from a typical DRL scheme, which will be further discussed in the following sections.

The major contributions of this research are summarized as follows:

(1) A deep learning TC model is integrated within a network environment to classify the incoming packet encapsulated in packet-in messages, which is an innovative implementation.

(2) Custom protocols for network metrics obtainment are designed, and RED-STAR regularizes the measured metrics to provide the DRL model with stable input data.

(3) The reward scheme considers different QoS requirements of services and load balancing issues, and RED-STAR distributes routes to services relying on the custom reward scheme.

(4) The DRL mechanism is applied in the SDN, takes network metrics and service type as the environment, and considers routes in the network as the action set, which is a novel traffic engineering paradigm.

(5) The experiments are implemented with real service traffic (i.e., PCAP file traffic replayed by Bit-Twist [21]) instead of simulated traffic (e.g., randomized packet payload generated by iPerf [22]). The results show that the proposed method performs better than other route distribution schemes when considering load balancing and QoS requirements.

The remainder of this work is organized as follows. Related work and background are discussed in Section 2. The system architecture is illustrated in Section 3. The system workflow is elaborated in Section 4. The proposed RED-STAR route distribution is detailed in Section 5. Experimental results are demonstrated in Section 6. Finally, Section 7 concludes this work.

2. Related Work and Background

In this research, two main issues are targeted to be addressed for route distribution: (a) QoS guarantee of network services and (b) bandwidth utilization, offloading and balancing. Existing works regarding both topics are discussed in the following paragraphs, and the background and applications of DRL will also be investigated.

2.1. QoS Guarantee of Network Services. The traditional network architecture cannot thoroughly offer the QoS guarantee of each service, whereas the emergence of SDN enables the flexible flow rule addition and accelerates the deployment of QoS routing policies [23, 24]. The common strategy of QoS is to reserve bandwidth for specific services, which guarantees the least available bandwidth for each service. Oliveira et al. [25] used Resource Reservation Protocol and OpenFlow to set up a dedicated channel between a service requester and a service provider, with a static

threshold of bandwidth to guarantee file transfer time. Tomovic et al. [26] utilized the SDN mechanism to offer priority flow bandwidth guarantees, designed an algorithm for route calculation and bandwidth reservation, and compared the performance with the best-effort and shortest path routing and IntServ. However, the requirements of services are not limited to the minimum bandwidth guarantee but involve maximum latency and packet loss rate tolerance. Links may have different network metrics, wherefore a superior path distribution algorithm for service traffic is required. Tseng et al. [27] proposed a multiobjective genetic algorithm (GA) to dynamically forecast the resource utilization and energy consumption in the cloud data center. The GA forecasts the resource requirement of the next time slot according to the historical data in previous time slots. Li et al. [28] presented a novel service functions (SF) deployment management platform that allows users to dynamically deploy edge computing service applications with the lowest network latency and service deployment costs in edge computing network environments. Tseng et al. [29] proposed a gateway-based edge computing service model to reduce the latency of data transmission and the network bandwidth from and to the cloud. An on-demand computing resource allocation can be achieved by adjusting the task schedule of the edge gateway via lightweight virtualization technology.

2.2. Bandwidth Utilization Offloading and Balancing. Apart from QoS guarantees, traffic offloading and balancing are inevitable issues, which often occur in a multipath network environment [30]. The SDN controller with the global view of a network can observe the network state and dynamically formulate a strategy to optimize traffic forwarding. Traffic offloading is essential when congestion occurs, and the increase of throughput is the primary goal. Chiang et al. [31] proposed a traffic distribution method to offload the incoming traffic. They utilized Link Layer Discovery Protocol (LLDP) in finding disjoint paths and Dijkstra in finding the shortest path with minimum hop counts to increase the overall throughput of the multipath network. Yahya et al. [32] pointed out the defect of the current prevalent open shortest path (OSPF) algorithms, which are prone to selecting merely one single best path for traffic forwarding and likely incur traffic congestion. The authors have developed a depth-first search algorithm to select several best paths according to link utilization, and the group action feature of OFS is used to distribute traffic across multiple paths. Despite an uncongested network, traffic balancing is still desirable to prevent future congestion. Challa et al. [33] proposed a CentFlow routing algorithm to enhance the node and link utilization depending on the centrality measures and temporal node degree. Tseng et al. [34] integrated the hypervisor technique with container virtualization and constructed an integrated virtualization (IV) fog platform for deploying industrial applications based on the virtual network function. Tseng et al. [35] addressed the design pattern of the 5G micro operator and proposed a Decision Tree-Based Flow Redirection (DTBFR) mechanism to redirect the traffic flows to neighbor service nodes. The DTBFR mechanism allows different μOs to share network resources and speed up the development of edge computing in the future.

2.3. Reinforcement Learning. A typical reinforcement learning (RL) [36] scenario involves three essential elements: an environment, agent, and action set. The agent is the learning entity that receives the state from the environment in a sequence of discrete times, $t = 0, 1, 2, \ldots$, where s_t is the state obtained at time t. After receiving s_t, the agent will select an action, a_t, to be performed by its policy according to the gained information. The environment of s_t will be influenced by a_t, thereby transforming into s_{t+1}. A reward value, r_t, standing for the score of performing a_t in s_t, will accordingly be generated by the environment and given back to the agent. On the basis of r_t, the agent will determine the performance of the previous action and tune its inner algorithm, attempting to obtain high values under the following states.

Q-learning [37] is a representative RL paradigm that has a Q-function to estimate the expected reward value of performing an action under a state (i.e., Q value):

$$Q^{\pi}(s,a) = E_{s'}\left[r + \lambda Q^{\pi}\left(s', a'\right) \mid s,\ a\right], \tag{1}$$

where $Q^{\pi}(s,a)$ represents the Q value of an action. The policy π determines the action to be performed, and r is the reward value of performing a under s. After the state-action pair (s, a), a new state s' comes out. A discount factor λ is multiplied by $Q^{\pi}(s', a')$ to reduce the impact of events over time. The Q-function in Q-learning is implemented with a Q table, storing the expected reward values of each action under each state. Once the reward is obtained, the Q table updates its stored value as follows:

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s,a)\right), \tag{2}$$

where $\max_{a'} Q(s', a')$ is the maximum expected reward under the next state, which is multiplied by an adjustable discount factor γ. The addition of r and $\gamma \max_{a'} Q(s', a')$ subtracted from the original $Q(s,a)$ indicates the error of the Q value predicted by the Q-function. The difference is multiplied by a learning rate α and added to $Q(s,a)$ to update the Q-function.

Although traditional RL works well in simple tasks, it cannot handle high input dimensionality and it suffers from slow convergence. Combined with the emerging deep learning, which mitigates the abovementioned problems, DRL has appeared. The DRL has recently been applied in several fields, such as video gaming [20], self-driving systems [38], and even computer networking [39, 40]. Hossain et al. [41] raised the issue of situation-aware management to ensure application-driven QoS and utilized link delay and packet loss rate as QoS metrics. DRL-based intelligent routing decision-making is proposed to optimize routing paths, with delay and loss rate as the observation space and weighted delay and loss rate as the reward scheme. Lin et al.

[42] used an RL adaptive routing in a hierarchical SDN network. The customized reward function is calculated with delay, packet loss rate, and bandwidth multiplied by the corresponding weighted parameters. The parameters are tuneable and configured according to the requirements of services.

There are many RL methods that learn some weights and then employ conventional routing algorithms. Yu et al. [43] proposed a deep deterministic policy gradient routing optimization mechanism (DROM) for SDN to achieve a universal and customizable routing optimization. The DROM simplifies the network operation and maintenance by improving the network performance, such as delay and throughput, with a black-box optimization in continuous time. Sun et al. [44] built an intelligent network control architecture TIDE (time-relevant deep reinforcement learning for routing optimization) to realize the automatic routing strategy in SDN. An intact "collections-decision-adjustment" loop is proposed to perform an intelligent routing control of a transmitting network. Stampa et al. [45] designed a DRL agent that optimizes routing. The DRL agent adapts automatically to current traffic conditions and proposes tailored configurations that attempt to minimize the network delay. Pham et al. [46] exploited a DRL agent with convolutional neural networks in the context of knowledge-defined networking (KDN) to enhance the performance of QoS-aware routing. Guo et al. [47] proposed a DRL-based QoS-aware secure routing protocol (DQSP). While guaranteeing the QoS, the DQSP can extract knowledge from history traffic demands by interacting with the underlying network environment and dynamically optimize the routing policy. Rischke et al. [48] designed a classical tabular RL approach (QR-SDN) that directly represents the routing paths of individual flows in its state-action space. QR-SDN is the first RL SDN routing approach to enable multiple routing paths between a given source (ingress) switch and destination (egress) switch pair while preserving the flow integrity. Ibrar et al. [49] proposed an intelligent solution for improved performance of reliable and time-sensitive flows in hybrid SDN-based fog computing Internet of Things (IoT) systems (IHSF). IHSF solves several problems related to task offloading from IoT devices in a multihop hybrid SDN-F network context.

Based on the abovementioned related research, the feature of this study is to propose a reinforcement discrete learning-based service-oriented multipath routing to understand the policy of distributing an optimal path for each service. The RED-STAR takes the network state and service type as input values to dynamically select the path a service must be forwarded. The seven papers related to the motivations and problems to be solved in this study are compared. The comparison table is shown in Table 1. Compared with IHSF or other methods, our proposed method considers the type of traffic and selects the best routing path.

3. System Architecture

The overall system is an SDN paradigm, which can be regarded into two parts: data plane and control plane. The data plane is in charge of forwarding the traffic; the control plane is the primary site to deploy the custom modules, which are the core components of the architecture.

The details of both parts are illustrated in Figure 1 as a UML diagram. The data plane is composed of OFSs, forwarding the traffic between the server and the client according to the rules deployed in the flow table. The flow table stores the commands delivered by the controller, and OFSs forward the packets or modify the header fields according to the rules. The OFSs communicate with the control plane via OpenFlow channels, whether flow deployment or statistical report. As for the control plane, several custom modules constitute the controller, which is described as follows.

3.1. Controller. The controller object is responsible for maintaining the information of routers, links, and service objects within the network. A router stands for an OFS; a link is a path between two OFSs, and service is the traffic type being transmitted. In addition, the controller periodically requests the network metrics and regularizes and updates the obtained metrics. The controller also periodically reallocates the paths for services and trains the DRL agent to improve the path allocation. Regardless of the out-of-band communication or in-band communication between the switch and the controller, the method proposed in this study is applicable.

3.2. Router. A router object is an entity in the control plane representing an OFS. When an OFS is activated and notifies the controller, a router object will be instantiated. A router object collects the information of each port by sending the port request messages and explores the topology by sending LLDP messages. Moreover, whenever a packet unmatched to flow rules is sent to the controller, the router will normalize and classify the packet to a service type by the CAPC deep learning model. A router is also in charge of the communication between OFSs and the control plane, such as flow addition and port statistics requests.

3.3. Classifier. The classifier is a component of a router object that normalizes and classifies packets, and the classification result will be returned to the router. The normalization and training process are detailed in our previous work [19].

3.4. Link. A link object consists of three network metrics: bandwidth utilization, latency, and packet loss rate. The controller is responsible for maintaining and updating the links, and the metrics are regularized before being updated. The metrics are the main factors and state for the future DRL path distribution.

3.5. Service. A service object records the service type and its specific reward calculation policy. After the controller distributes a path for a service, the allocated path (last action) will be recorded. Once the network metrics are obtained, the metrics will be the input values of the reward calculation method to generate the reward for the last action.

Table 1: RL signal comparison between our proposed method and related research methods.

RL signals	State	Action	Reward
DROM [43]	The traffic matrix (TM) of the network	The weights of links in the network	The network operation and maintenance strategy
TIDE [44]	The traffic matrix (TM) of the network	The weights of links in the network	The QoS strategy
Stampa et al. [45]	The traffic matrix (TM) of the network	The weights of links in the network	The mean network delay
Pham et al. [46]	The traffic matrix (TM) of the network	The weights of links in the network	The mean of QoS metrics/the mean of qualified flows
DQSP [47]	The frequency of packet-in message, the occupancy rate of the flow table, and the channel occupancy rate	The weight of the node assigned as the next hop	The node packet loss rate, node forwarding delay, and flow table status
QR-SDN [48]	The currently selected path for each flow	Determine the path of flow(s)	The sum of latencies along the current paths of the flows
IHSF [49]	The path reliability, delay, bandwidth utilization, and the number of disturbed flows in case of link's failure	Determine the path of flow(s)	The path's reliability level, the minimum number of disturbed flows, maximum bandwidth utilization, and minimum delay
Proposed RED-STAR	The service type, current bandwidth utilization, packet loss rate, and the latency of each link	Determine the path of flow(s)	The QoS requirements of services and link utilization balancing

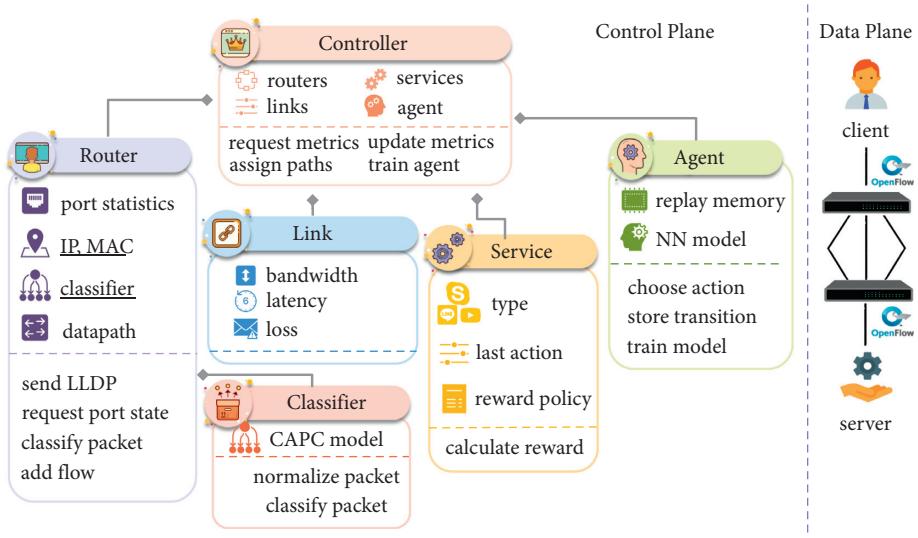

Figure 1: UML diagram of the overall system architecture.

3.6. Agent. The agent object belongs to the controller, having an experience replay memory to train the NN model. The NN model is used to select a path for service and trained by the transitions in the replay memory. After path allocation, the previous state, the selected path, reward, and the current state are saved as a transition into the replay memory.

4. System Workflow

A few procedures must be accomplished to model the QoS path distribution as an RL problem. This section details each procedure, including network state observation, path distribution, and the reward mechanism.

The typical framing of an RL scenario: an agent takes actions in an environment, which is interpreted as a reward and a representation of the state, which are fed back into the agent. The state of the environment includes the type of service, the current bandwidth utilization, the packet loss rate, and the delay of each link. The description of each procedure is shown in Figure 2, where the observation is to learn the changing environment of the network for the RL agent. The observation includes the service type, current bandwidth utilization, packet loss rate, and the latency of each link. After receiving a state as the input, the agent will select a path for service and add a flow to OFSs. Subsequently, the reward value is generated on the basis of the distribution and service type. The three kinds of data, state, action, and reward, will be stored in the replay memory for agent training.

4.1. Observation. A few steps must be completed to form a state, including topology discovery, metric measurement, and metric regularization.

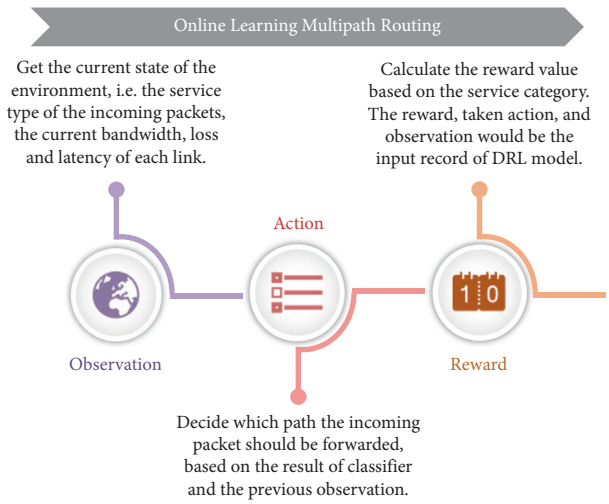

Figure 2: Progress of modeling an RL path distribution problem.

4.1.1. Link-Layer Topology Discovery. The task that must be accomplished first is topology construction to offer the paths (action set) for distribution. The LLDP is used to explore the link status of each port. An LLDP packet is crafted with a designated chassis ID, that is, the ID of an OFS, and a port number, thereafter sent out from the port of the OFS. The connected port on the other side will receive the packet, thereby encapsulating the packet in an OpenFlow packet-in message and forwarding back to the controller. The controller can construct the topology of the network. When discovering the link-layer topology, the topology is only a connectivity topology that gives the links between individual network nodes, but does not yet construct end-to-end paths. The entire process of topology discovery is shown in Figure 3, and the notations used in the figure are explained in Table 2.

4.1.2. Metric Measurement. The network metrics are measured periodically as the state of the DRL model. The measurement approaches of bandwidth utilization, latency, and packet loss rate are listed in the following order.

(1) Bandwidth Utilization Measurement. Every OFS keeps its accumulated transmission byte number up to date. Whenever receiving a port request message, the OFS will answer the port reply to the controller, containing the accumulated transmitted bytes. The controller can thereafter calculate the difference between the previous and the current values, thereby obtaining the bandwidth utilization at this time. The detailed process of the measurement is depicted in Figure 4, and the notations used in the figure can be referred to in Table 3.

(2) Link Latency Measurement. A custom protocol is designed to measure the latency. The packet format of the protocol complies with the Ethernet frame; the ether type of which is set as an arbitrary value (0×8787). The destination MAC address remains blank, and its source MAC address is filled in with the timestamp at that time. The length of the

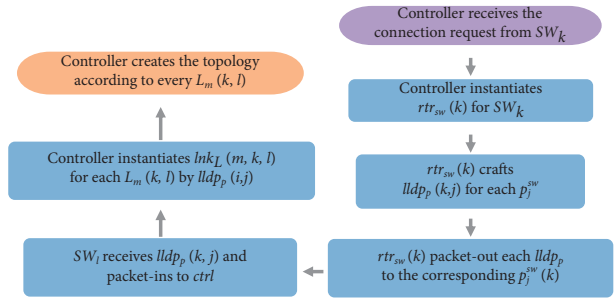

Figure 3: Flowchart of link-layer topology discovery.

custom probing packet is designed as 14 bytes, less than the one used in other research [41, 50]. Whenever an OFS receives a packet with a 0×8787 ether type, it sends the packet within a packet-in message to the controller. Afterward, the controller removes the timestamp from the packet and calculates the difference between the current and the timestamp in that packet. Finally, the outcome is subtracted from the latency between the OFSs and the controller, and the latency of a link is measured. The overall process of latency measurement is shown in Figure 5, and the notations used in the figure can be referred to in Table 4.

(3) Packet Loss Rate Measurement. Similar to the latency measurement, a custom packet format is used for the packet loss rate. The ether type of the packet is set as 0×7878, and the destination and source MAC addresses remain blank. Initially, the controller sends out a fixed number of probing messages to OFSs. Whenever an OFS receives the packet with a 0×7878 ether type, it drops the packet immediately. After a fixed period, the OFS delivers the number of 0×7878 packets it receives from the controller. The controller then calculates the difference between the received number and the original quantity of probing packets sent before. Therefore, the packet loss rate can be calculated. The entire process of packet loss rate measurement is illustrated in Figure 6, and the notations used in the figure can be referred to in Table 5.

4.1.3. Metric Regularization. Two aspects must be considered before the utilization of the obtained network metrics. The first aspect indicates that the same value of different metrics has different meanings, for example, latency is presented in milliseconds; bandwidth utilization and packet loss rate are presented in ratio, but 50% bandwidth utilization is definitely better than 50% packet loss rate. The second aspect indicates that some metrics occasionally go wrong.

For the first problem, a mechanism is required to dimension each metric into a similar scale range (0-1), where a larger value is better than a smaller one. In dimensioning a bandwidth utilization value, if the value is originally 0%, then the normalized value will be 1; if the value is originally 100%, then the normalized value will be 0:

$$bw_{\text{lnk}}^{\text{norm}}(m,k,l) = \left(-bw_{\text{lnk}}(m,k,l)\right) + 1, \tag{3}$$

Table 2: Notations for link-layer topology discovery.

Notation	Description
sw_i	The i^{th} OpenFlow switch
$id_{sw}(i)$	Chassis ID of sw_i
$rtr_{sw}(i)$	Instantiated router object of sw_i in controller
$P_j^{sw}(i)$	The j^{th} port of sw_i
$no_P(j,i)$	Port number of $P_j^{sw}(i)$
$L_m(k,l)$	The m^{th} link between sw_k and sw_l
$\text{lnk}_L(m,k,l)$	Instantiated object of $L_m(k,l)$ in $ctrl$
$lldp_P(i,j)$	LLDP packet generated for P_j^{sw}, composed of $id_{sw}(i)$ and the port number of $P_j^{sw}(i)$

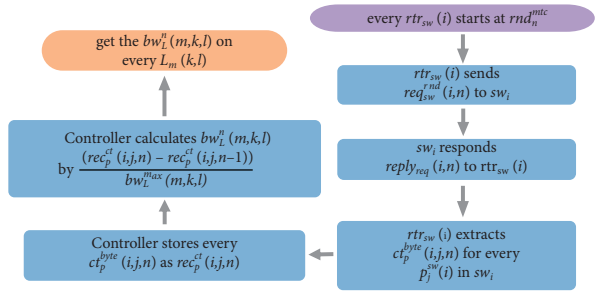

Figure 4: Flowchart of bandwidth utilization measurement.

Table 3: Notations for bandwidth utilization measurement.

Notation	Description
rnd_n^{mtc}	n^{th} round to pull metrics
$req_{sw}^{rnd}(i,n)$	Port statistics request message for sw_i in rnd_n^{mtc}
$\text{reply}_{req}(i,n)$	Port statistics reply message with respect to $req_{sw,rnd}(i,n)$
$ct_P^{\text{byte}}(i,j,n)$	Total byte count as the sum of tx and rx byte number of $P_j^{sw}(i)$ in rnd_n^{mtc}
$rec_P^{ct}(i,j,n)$	Record saved from $ct_P^{\text{byte}}(i,j,n)$
$bw_L^{\max}(m,k,l)$	Maximum bandwidth in bytes of $L_m(k,l)$
$bw_L^n(m,k,l)$	Bandwidth utilization in ratio on $L_m(k,l)$ in rnd_n^{mtc}

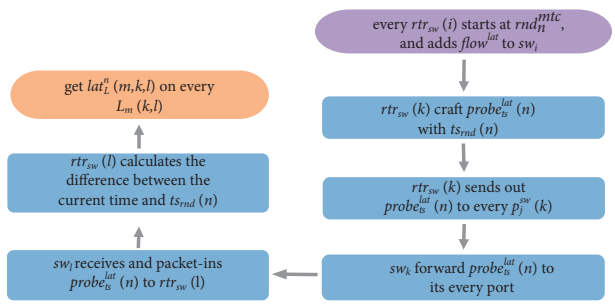

Figure 5: Flowchart of link latency measurement.

Table 4: Notations for link latency measurement.

Notation	Description
flow^{lat}	Flow with match field as ether_type = $0x8787$ and action field as packet_in to calculate the latency
$ts_{rnd}(n)$	Timestamp of rnd_n^{lat}
$\text{probe}_{ts}^{\text{lat}}(n)$	Probe packet crafted with $ts_{rnd}(n)$ for latency measurement
$\text{lat}_L^n(m,k,l)$	Latency in ms on $L_m(k,l)$ in rnd_n^{mtc}

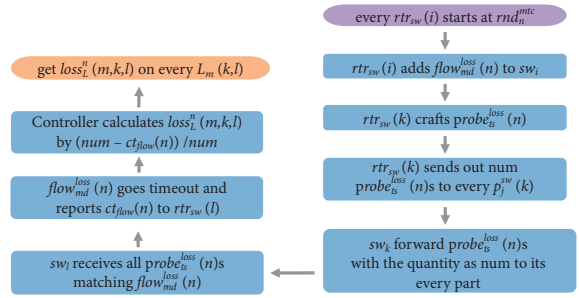

Figure 6: Flowchart of packet loss rate measurement.

Table 5: Notations for packet loss rate measurement.

Notation	Description
$\text{flow}_{rnd}^{\text{loss}}(n)$	Flow with match field as ether_type = 0x7878, action field as none, and hard time out as 1 second
$ct_{\text{flow}}(n)$	Packet count $\text{flow}_{rnd}^{\text{loss}}(n)$ match
$\text{probe}_{rnd}^{\text{loss}}(n)$	Probe packet used for loss measurement at rnd_n^{mtc} number of $\text{probe}_{rnd}^{\text{loss}}(n)$ once to send
num	Loss rate in ratio on $L_m(k,l)$ in rnd_n^{mtc}

where $bw_{\text{ln}k}(m,k,l)$ is the original bandwidth utilization and $bw_{\text{ln}k}^{\text{norm}}(m,k,l)$ is the normalized value. For latency normalization, if the original latency is 0 ms, then the value will be 1; if the original latency is 100 ms, then the value will be 0:

$$\text{lat}_{\text{ln}k}^{\text{norm}}(m,k,l) = -\left(\frac{\text{lat}_{\text{ln}k}(m,k,l)}{100}\right) + 1, \tag{4}$$

where $\text{lat}_{\text{ln}k}(m,k,l)$ is the original latency and $\text{lat}_{\text{ln}k}^{\text{norm}}(m,k,l)$ is the normalized value. Finally, if the original packet loss rate is 0%, then the normalized value will be 1; if the original packet loss rate is 10%, then the normalized value will be 0:

$$\text{loss}_{\text{ln}k}^{\text{norm}}(m,k,l) = (-10) \times \text{loss}_{\text{ln}\,k}(m,k,l) + 1, \tag{5}$$

where $\text{loss}_{\text{ln}k}(m,k,l)$ is the original loss rate and $\text{loss}_{\text{ln}k}^{\text{norm}}(m,k,l)$ denotes the normalized one.

The second problem is solved by the custom mechanism, which is determined by evaluating the difference between the new and the mean value. If the difference is greater than the standard deviation, then it will be determined as an anomaly value and be regularized as the mean value. The dynamic standard deviation can be obtained as follows:

$$\text{Var}(X) = E\left[x^2\right] - E[x]^2, \tag{6}$$

where $\text{Var}(X)$ is the variance and used to calculate the standard deviation.

4.2. Action Selection. With the gained input value in the observation, the DRL model can determine a path to forward the traffic of a service. The proposed model is following a complete path approach with a multipath capability. The proposed RED-STAR adopts an *ε-greedy* scheme, in which an ε probability and 1-ε probability are found to randomly select an action to be performed and to make the decision on the basis of the calculation result of the DRL model.

Once an OFS receives a packet unmatched to the installed flow, the packet will be sent to the controller within a *packet-in* message. Thereafter, the CAPC model is used to classify the service type that the packet belongs to. Then, a service object is instantiated and added to the service list of the controller. The agent within the controller subsequently allocates a path for each service by the *ε-greedy* approach, thereby training its NN model for good allocation. A forwarding flow of the corresponding service will be added to the OFS. A flow is composed of a matching field and an action field. The matching field is set as the IP address of the IP address and the port number of that service, and the action is set to the forwarding port according to the selected path.

ε in the *ε-greedy* approach is a variable, which is initially set to a value close to 1. With training, the ε value gradually decreases (7). Considering that the NN model is not robust at the beginning, the probability of ε is set to a relatively high value, also known as *exploration*. With time, the decision made by the NN model becomes better and ε becomes smaller. At present, we can rely more on the NN model to allocate a path for a service, which is known as *exploitation*.

$$\varepsilon \longleftarrow 0.995 \times \varepsilon \,|\, \varepsilon \in [0.01, 1]. \tag{7}$$

4.3. Reward Scheme. Two factors are involved in the reward scheme: QoS requirements of services and link utilization balancing. Each factor accounts for the reward value of 1; thus, the maximum reward value is 2.

4.3.1. QoS Reward. Each service attaches different importance to the metrics, where a differentiated reward policy is needed. After path allocation for services, the controller will request and receive the metrics for the next turn. Once the controller receives the new metrics, the rewards of services of the last path allocation will be calculated on the basis of their individual policy. A total of 16 applications will be classified into four main categories (Table 6). File transfer services

emphasize the packet quality with less corruption and loss, thereby tolerating high latency. Video streaming and VoIP services are sensitive to link latency, thereby allowing slight packet loss; thus, the latency weight of which must be set higher. Remote control services demand moderate response time and packet loss rate; thus, the weights of the two are set equally. The reward calculation for the four services is formulated as follows:

$$r_{svc}(i) = w_{svc}^{\text{lat}}(i) \times \text{lat} + b_{svc}^{\text{lat}}(i) + w_{svc}^{\text{loss}}(i) \times \text{loss} + b_{svc}^{\text{loss}}(i), \tag{8}$$

where $r_{svc}(i)$ is the reward value of the i^{th} service. The reward is the sum of the weighted latency and loss rate of the selected path for the service, latency, and loss base value. The latency weight $w_{svc}^{\text{lat}}(i)$ is set higher for latency-sensitive services (streaming and VoIP), and the latency base $b_{svc}^{\text{lat}}(i)$ is set lower. The loss weight $w_{svc}^{\text{loss}}(i)$ is set lower, and the loss base is set higher for the latency-sensitive services. The weight and base values are also set correspondingly on the basis of their QoS requirements. The actual weight and base values for services are depicted in Figure 7. Therefore, the latency and packet loss rate take half of the QoS reward (maximum of 0.5 for each).

4.3.2. Link Utilization Reward. An unbalanced path allocation results in a low reward to utilize the bandwidth of the network. The utilization of each link is gathered, and the utilizations of the most and least used link are removed for reward calculation. The large utilization value is subtracted by the small value, and a high difference leads to a low reward value, and vice versa:

$$r_{\text{lnk}}^{bw}(k,l) = -\left(bw_{\text{lnk}}^{\max}(k,l) - bw_{\text{lnk}}^{\min}(k,l)\right) + 1, \tag{9}$$

where $r_{\text{lnk}}^{bw}(k,l)$ is the reward value of utilization balancing. The difference between the maximum and minimum utilization is turned to negative and added by 1. Thereafter, the QoS reward and balancing reward are added as the final reward value of a path distribution:

$$r_{svc}(i) \longleftarrow r_{svc}(i) + r_{\text{lnk}}^{bw}(k,l). \tag{10}$$

5. DRL Route Distribution

In a general DRL case, s_t performed with a_t results in r_t and s_{t+1} and a transition (s_t, a_t, r_t, s_{t+1}) will be saved in the replay memory. The memory contains several transitions from which the agent arbitrarily selects n transitions to train its NN model. The proposed RED-STAR mechanism is a slight modification (Reinforcement Discrete learning (RED)) of a classic DRL model, that is, deep Q-network (DQN). The RED-STAR considers the actual reward r_t affected by s_t and a_t, without s_{t+1} (Figure 8).

The main idea of RED is that the route distribution of services does not influence each other directly. For example, the first distribution is targeted at the Skype VoIP service and the next distribution is for LINE VoIP service, whereas the two distributions have no correlation. The state s_t of Skype will not lead to s_{t+1} of LINE. Therefore, a transition stored in the replay memory is constituted of s_t, a_t, and r_t.

Table 6: Targeted applications and their categories.

File transfer	Video streaming	VoIP	Remote control
FTP	RTP video	LINE VoIP	RDP (Windows)
SFTP	RTSP video	Skype VoIP	VMware
OneDrive	UDP video	Zoom VoIP	XenServer
SCP	YouTube video	Join.me VoIP	NCU cloud

A typical DRL model involves two NN models: an NN model for action selection and an NN model for the calculation of targeted values. s_t of the transitions randomly selected from the replay memory is fed into the prediction NN model, and the output of which is the action to be performed in this round. s_{t+1} of the transitions is fed into the target NN model, and the output of which is the targeted value to be updated by the prediction model. The output of the prediction model is regarded as the estimated expected reward of performing a_t under s_t, whereas the output of the target model multiplied by a discount factor γ and added by r_t is considered as the practical expected reward (Figure 9). The prediction NN model trains and updates itself with the practical expected reward as targeted values.

Different from the traditional DRL operation, the output of the NN model in the RED-STAR mechanism stands for the actual reward value r_t for performing a_t under s_t, rather than the expected reward. During training, s_t is the input data for the NN model and r_t is set as the targeted value (Figure 10). The model updates its parameters to approximate the targeted value.

The structure of the RED-STAR NN model is depicted in Figure 11, which is a three-layer deep learning structure. The input layer at the top receives 28 features, including the one-hot encoded service type, the network metrics of all routes, and the route allocation state. The output layer contains neurons with the same number as the routes. The value of each neuron represents the reward value for the corresponding path. The mean square error (11) is set as the loss function of the NN model, which is the criterion to determine how "bad" the model is. The *Adam* [51] is a gradient descent method used to update the parameters of the NN model.

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2. \tag{11}$$

To date, the overall contour of the route distribution mechanism has been illustrated. The process of the mechanism can be briefly presented by three procedures: (a) the router objects request their port statistics from the OFSs, thereby obtaining the link states of the topology. The controller then updates the link states received by the router, which is the observation step described in Section 4.1. (b) The agent allocates routes to the services on the basis of its policy, and the controller calculates the reward values of services according to their reward policy. The tasks in this

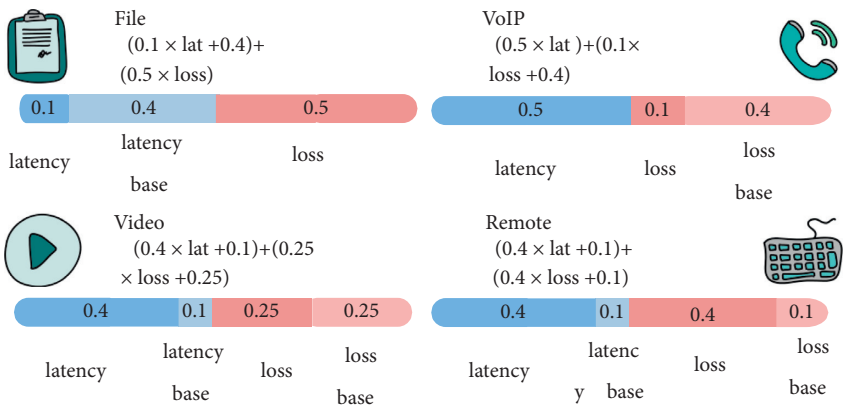

Figure 7: Reward calculation parameters for each service.

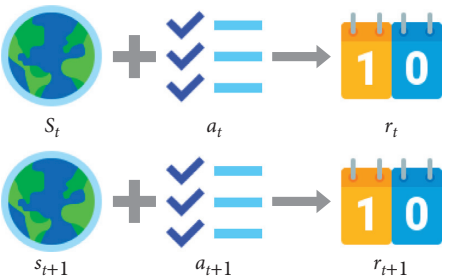

Figure 8: State-action interactions of reinforcement discrete learning.

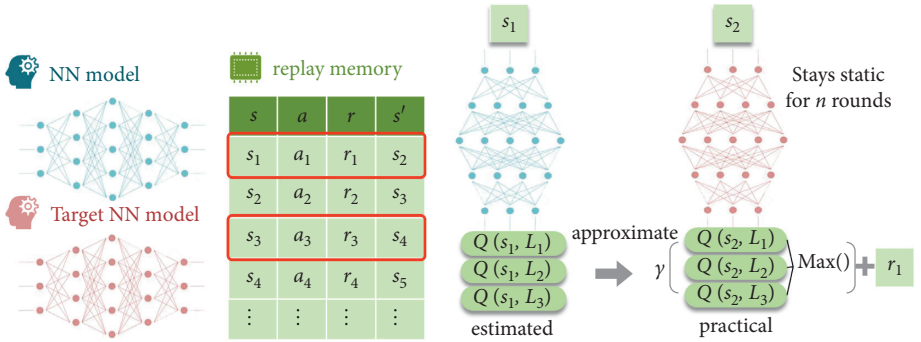

Figure 9: Operation of a traditional DRL model.

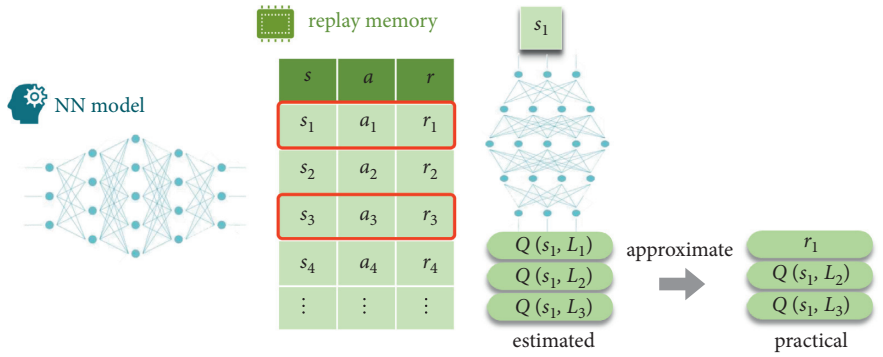

Figure 10: Operation of the RED-STAR model.

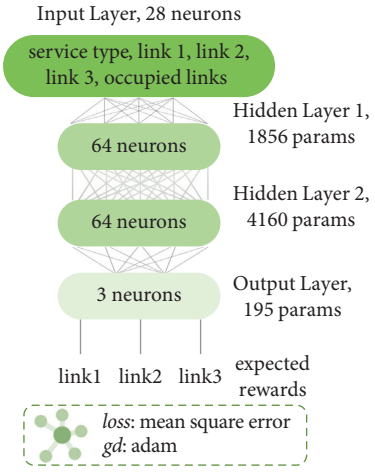

Figure 11: Model structure of the NN model of RED-STAR.

procedure are the action selection and reward obtainment steps in Sections 4.2 and 4.3. (c) The agent trains its NN model on the transitions selected from the replay memory to improve its selection, as the content presented in this section. After training, the overall process returns to step (a) and keeps executing the loop cycle.

6. Experiments and Evaluation

Several scenarios are simulated to evaluate the effectiveness and performance of the proposed RED-STAR mechanism. The environment settings, that is, hardware and software specifications, SDN network construction, and link attribute settings, are first introduced. Thereafter, the performance of the RED-STAR mechanism compared with the other two route distribution schemes (i.e., shortest path distribution (SPD) and DQN) in different scenarios is demonstrated.

6.1. Environment Settings. The specifications are listed in Table 7. An Ubuntu virtual machine is installed atop a VMware ESXi hypervisor, running Mininet [52] as the network simulator. The Bit-Twist traffic replay toolkit [21] is used for service traffic generation; thus, the traffic in the simulation is the actual PCAP files of certain services. TensorFlow and Keras are used for NN model construction and training.

The network environment topology and configuration are simply set (Figure 12), in which two hosts are in charge of traffic transmission, and the links are featured differently. The delay value and packet loss rate are set proportionally: a higher delay value is configured along with a lower packet loss rate (Link 3), and vice versa (Link 1). The maximum bandwidth values are all set to 100 Mbps. The above-mentioned configuration allows the scheme to determine the route distribution for all services in transmission on the basis of their QoS requirements.

Table 7: Hardware and software specification.

Hardware/software	Specification
CPU	AMD Opteron™ processor 4386
Hypervisor	VMware ESXi 6.7.0 13006603
Operating system	Ubuntu 16.04 LTS
Network simulator	Mininet 2.3.0d6
OpenFlow switch simulator	Open vSwitch 2.11.0
SDN controller	Ryu 4.32
Traffic generator	Bit-Twist 2.0
Machine learning engine	TensorFlow 1.14.0
Machine learning toolkits	Keras 2.1.6, scikit-learn 0.22.1
Data preprocessing	Pandas 1.0.0, NumPy 1.16.0

6.2. Reward Scheme. In this scenario, a LINE VoIP service is transmitted at 10 Mbps in the network. The VoIP is a latency-sensitive and loss-rate-tolerant service, from which we can directly identify that the first link must be the best route for LINE VoIP. A random path distribution scheme is first tested, and its obtained reward values are shown in Figure 13, in which the reward oscillates from 1.6 to 1.8.

The RED-STAR model is prone to arbitrary selection of a route at the beginning based on the ε-*greedy* policy, as ε gradually declines to rely on the NN model. The reward value gained by RED-STAR is shown in Figure 14(a). The NN model approximately converges after the 200th second and obtains high reward values. Initially, the model selects the third link, causing the expected reward of the third link to grow quicker than the others (Figure 14(b)). After the convergence, the model has been aware of that the first link is more suitable for the LINE VoIP service, thereby fixing its route allocation to select the first link more often and obtain higher rewards.

6.3. Composition of Different Services. The traffic of three services is replayed to the network simultaneously to evaluate the performance of each scheme, and the bandwidth consumption of every service is equally set to 10 Mbps in this scenario. The schemes deployed on the controller are in charge of distributing a route for each service. The rewards gained by the three schemes are shown in Figures 15(a)–15(c), where the value of SPD remains the same, and the two learning models obtain higher values with time. The RED-STAR obtains the highest average value for every service.

The DQN model performs worse than RED-STAR, and SPD does not improve with time. The average reward value of the three services of each scheme is presented in Figure 15(d), where RED-STAR converges faster than DQN with the greatest value of 1.8579. Apart from QoS, load balancing is one of the key factors of the reward, which can be separately discussed. In Figure 15(e), the maximum bandwidth utilization of RED-STAR decreases with time, reaching the bottom at around the 500th second, and keeps at a relatively low value of around 0.1. Therefore, RED-STAR has the lowest average utilization (0.1181), indicating that it uses the bandwidth resources in the most effective way.

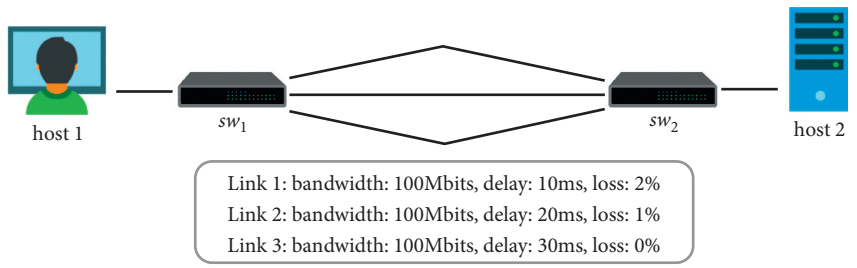

Figure 12: Simulated network topology configuration.

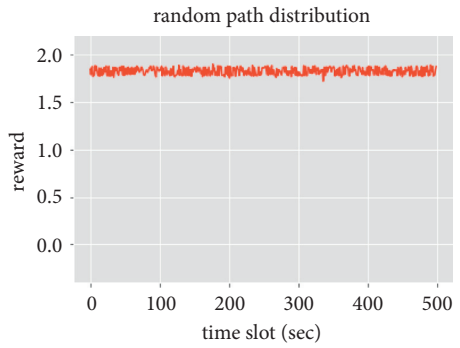

Figure 13: Obtained reward values of a random path distribution scheme.

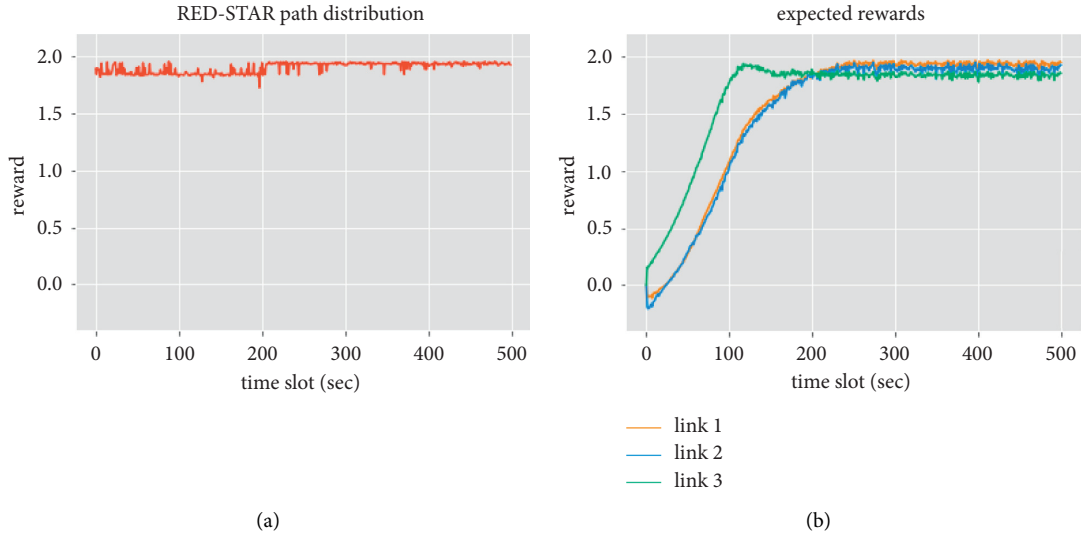

Figure 14: Obtained and expected reward values of the RED-STAR scheme. (a) Actual reward value of RED-STAR and (b) expected reward value of each path of RED-STAR.

6.4. Composition of Realistic Service Traffic. In the last scenario, every service has the same bandwidth consumption of 10 Mbps, which cannot reflect the network traffic in the real world. The bandwidth consumption of each service varies according to its category (e.g., VoIP with 1 Mbps, video with 20 Mbps, and file transfer with 40 Mbps). Six services are involved in this case, which is a more complex route distribution task. The reward gained by SPD (Figure 16(a)) remains at a stable reward value; the DQN model approximately converges after the 500th second (Figure 16(b)), obtaining higher values on all services than SPD; the RED-STAR also converges at the 600th second (Figure 16(c)), having average values on three services higher than DQN, and becomes steady after convergence. The average reward of six services in each scheme is presented in Figure 16(d). The SPD scheme keeps at approximately 1.6, whereas the two learning-based models grow steadily from 1.4 to 1.9. The RED-STAR and DQN achieve similar average rewards with 1.8579 and 1.8568, respectively, at the end. Both of them are able to adopt the realistic traffic scenario, of which the models neither have the knowledge of service bandwidth consumption nor take the consumption as the input data. Regarding load balancing, RED-STAR has a lower maximum utilization rate than DQN at the end (Figure 16(e)),

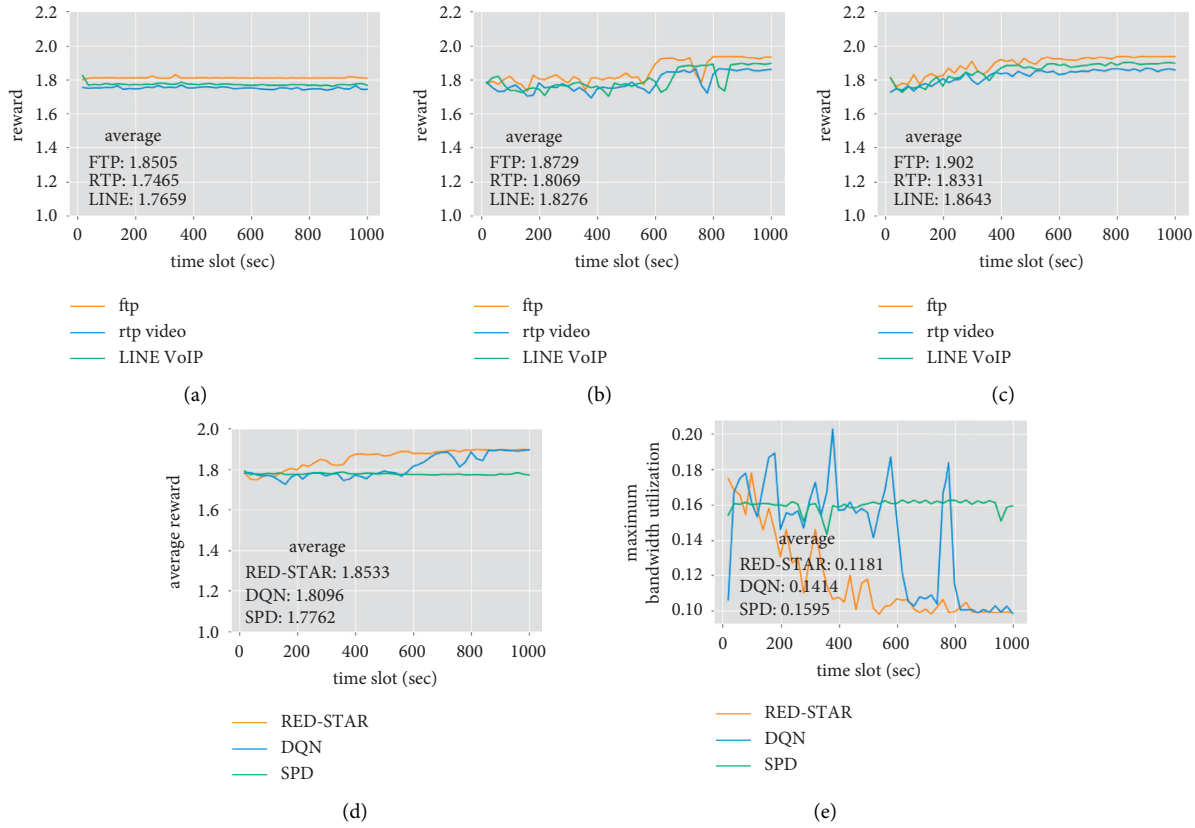

Figure 15: Performance of three path distribution schemes in the three-service scenario. (a) Reward gained by SPD; (b) reward gained by DQN; (c) reward gained by RED-STAR; (d) average reward of three services of each scheme; (e) maximum bandwidth utilization of each scheme.

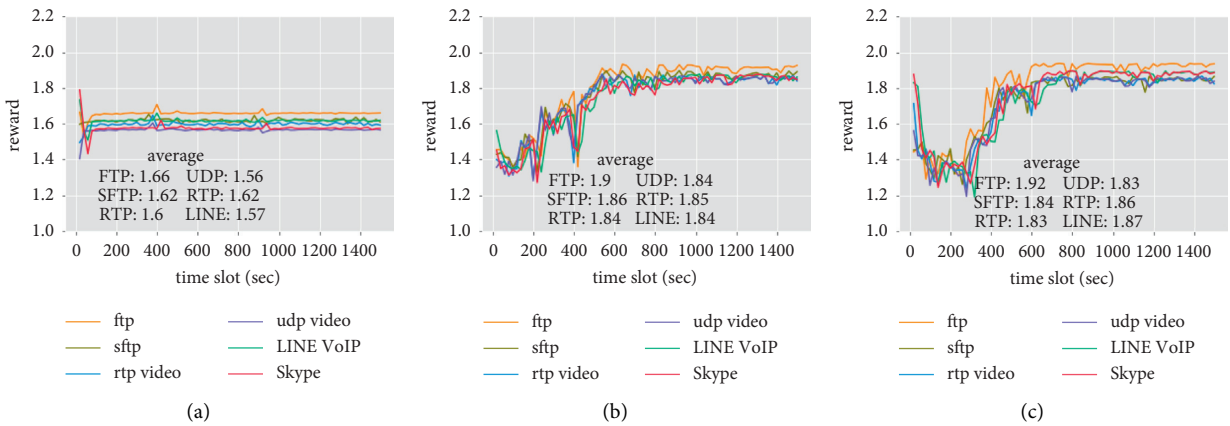

Figure 16: Continued.

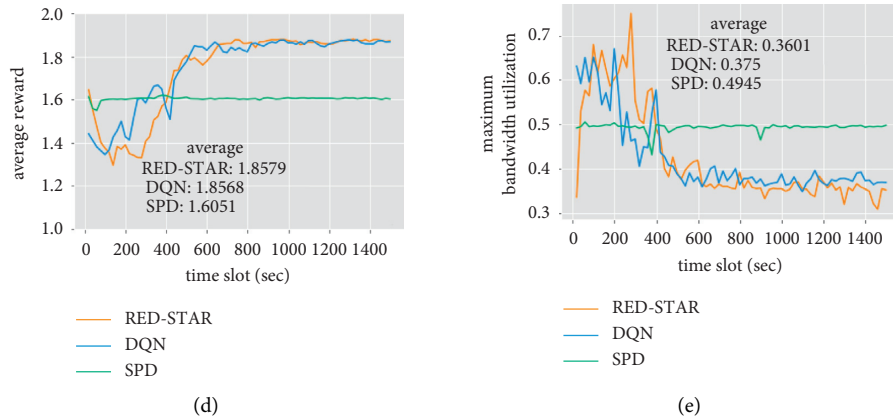

Figure 16: Performance of three path distribution schemes in the realistic six-service scenario (the average values are calculated from the 500th to the 1500th second after convergence). (a) Reward gained by SPD; (b) reward gained by DQN; (c) reward gained by RED-STAR; (d) average reward of three services of each scheme; (e) maximum bandwidth utilization of each scheme.

indicating that DQN gains more reward on the QoS requirements, thereby having a similar average reward as RED-STAR.

7. Conclusions

RED-STAR considers the QoS requirements of different services and balances the link utilization with the average reward value of 1.8533, which is greater than the other two schemes, and the lowest average maximum bandwidth utilization of 0.1181 in the three-service traffic scenario. Moreover, in the realistic six-service traffic scenario, RED-STAR still achieves the best average reward of 1.8579 and the lowest average maximum bandwidth utilization of 0.3601 among all schemes.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

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

Acknowledgments

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan, under Grants MOST 105-2221-E-008-071-MY3, MOST 108-2221-E-008-033-MY3, and MOST 110-2218-E415-001-MBK.

References

[1] Y. Chen, T. Farley, and N. Ye, "QoS requirements of network applications on the Internet," *Information—Knowledge—Systems Management*, vol. 4, no. 1, pp. 55–76, 2004.

[2] D. B. Rawat and S. R. Reddy, "Software defined networking architecture, security and energy efficiency: a survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 325–346, 2017.

[3] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[4] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: a survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[5] M. Rezaee and M. H. Yaghmaee Moghaddam, "SDN-based quality of service networking for wide area measurement system," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 5, pp. 3018–3028, 2020.

[6] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proceedings of the IEEE INFOCOM*, pp. 2211–2219, Turin, Italy, April 2013.

[7] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in SDN/OSPF hybrid network," in *Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols*, pp. 563–568, Raleigh, NC, USA, October 2014.

[8] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: data collection and traffic classification," in *Proceedings of the 2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pp. 1–5, Singapore, November 2016.

[9] J. Yan and J. Yuan, "A survey of traffic classification in software defined networks," in *Proceedings of the 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, pp. 200–206, Shenzhen, China, August 2018.

[10] J. Touch, E. Lear, A. Mankin, M. Kojo, K. Ono, and M. Stiemerling, "Service name and transport protocol port number registry," Internet Assigned Numbers Authority (IANA), https://www.iana.org/assignments/service-names-port-numbers.

[11] Iana. Internet Assigned Numbers Authority, Internet Assigned Numbers Authority, https://www.iana.org/.

[12] A. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proceedings of the International Conference on Passive and Active Network Measurement*, Boston, MA, USA, March 2005.

[13] A. Madhukar and C. Williamson, "A longitudinal study of P2P traffic classification," in *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pp. 179–188, Monterey, CA, USA, September 2006.

[14] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2014.

[15] D. Sanvito, D. Moro, and A. Capone, "Towards traffic classification offloading to stateful SDN data planes," in *Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–4, Bologna, Italy, July 2017.

[16] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for Internet of Things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[17] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: deep learning based encrypted network traffic classification in SDN home gateway," *IEEE Access*, vol. 6, pp. 55380–55391, 2018.

[18] P. Wang, X. Chen, F. Ye, and Z. Sun, "A survey of techniques for mobile service encrypted traffic classification using deep learning," *IEEE Access*, vol. 7, pp. 54024–54033, 2019.

[19] K.-C. Chiu, C.-C. Liu, and L.-D. Chou, "CAPC: packet-based network service classifier with convolutional autoencoder," *IEEE Access*, vol. 8, pp. 218081–218094, 2020.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, and D. Wierstra, "Playing Atari with deep reinforcement learning," 2013, https://arxiv.org/abs/1312.5602.

[21] A. Heng, "Bit-twist: libpcap-based Ethernet packet generator," 2006, https://bittwist.sourceforge.net/.

[22] C. Hsu and U. Kremer, "IPERF: a framework for automatic construction of performance prediction models," in *Proceedings of the Workshop on Profile and Feedback Directed Compilation*, pp. 1–10, Paris, France, October 1998.

[23] A. Binsahaq, T. R. Sheltami, and K. Salah, "A survey on autonomic provisioning and management of QoS in SDN networks," *IEEE Access*, vol. 7, pp. 73384–73435, 2019.

[24] K. Bouraqia, E. Sabir, M. Sadik, and L. Ladid, "Quality of experience for streaming services: measurements, challenges and insights," *IEEE Access*, vol. 8, pp. 13341–13361, 2020.

[25] A. T. Oliveira, B. J. C. A. Martins, M. F. Moreno, A. B. Vieira, A. T. A. Gomes, and A. Ziviani, "SDN-based architecture for providing QoS to high performance distributed applications," in *Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 602–607, Natal, Brazil, June 2018.

[26] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for QoS provisioning," in *Proceedings of the 2014 22nd Telecommunications Forum Telfor (TELFOR)*, pp. 111–114, Belgrade, Serbia, November 2014.

[27] F.-H. Tseng, X. Wang, L.-D. Chou, H.-C. Chao, and V. C. M. Leung, "Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1688–1699, 2018.

[28] D. C. Li, B.-H. Chen, C.-W. Tseng, and L.-D. Chou, "A novel genetic service function deployment management platform for edge computing," *Mobile Information Systems*, vol. 2020, Article ID 8830294, 22 pages, 2020.

[29] C.-W. Tseng, F.-H. Tseng, Y.-T. Yang, C.-C. Liu, and L.-D. Chou, "Task scheduling for edge computing with agile VNFs on-demand service model toward 5G and beyond," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–13, 2018.

[30] Z. Shu, J. Wan, J. Lin et al., "Traffic engineering in software-defined networking: measurement and management," *IEEE Access*, vol. 4, pp. 3246–3256, 2016.

[31] Y. Chiang, C. Ke, Y. Yu, Y. Chen, and C. Pan, "A multipath transmission scheme for the improvement of throughput over SDN," in *Proceedings of the 2017 International Conference on Applied System Innovation (ICASI)*, pp. 1247–1250, Sapporo, Japan, May 2017.

[32] W. Yahya, A. Basuki, W. Maulana, S. R. Akbar, and A. Bhawiyuga, "Improving end-to-end network throughput using multiple best paths routing in software defined networking," in *Proceedings of the 2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pp. 187–191, Bali, Indonesia, July 2018.

[33] R. Challa, S. Jeon, D. S. Kim, and H. Choo, "CentFlow: centrality-based flow balancing and traffic distribution for higher network utilization," *IEEE Access*, vol. 5, pp. 17045–17058, 2017.

[34] F.-H. Tseng, M.-S. Tsai, C.-W. Tseng, Y.-T. Yang, C.-C. Liu, and L.-D. Chou, "A lightweight autoscaling mechanism for fog computing in industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4529–4537, 2018.

[35] C.-W. Tseng, Y.-K. Huang, F.-H. Tseng, Y.-T. Yang, C.-C. Liu, and L.-D. Chou, "Micro operator design pattern in 5G SDN/NFV network," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 3471610, 14 pages, 2018.

[36] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, UK, 1998.

[37] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[38] J. Duan, S. Eben Li, Y. Guan, Q. Sun, and B. Cheng, "Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data," *IET Intelligent Transport Systems*, vol. 14, no. 5, pp. 297–305, 2020.

[39] N. C. Luong, D. T. Hoang, S. Gong et al., "Applications of deep reinforcement learning in communications and networking: a survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[40] M. Latah and L. Toker, "Artificial intelligence enabled software-defined networking: a comprehensive overview," *IET Networks*, vol. 8, no. 2, pp. 79–99, 2019.

[41] M. B. Hossain and J. Wei, "Reinforcement learning-driven QoS-aware intelligent routing for software-defined networks," in *Proceedings of the 2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 1–5, Ottawa, ON, Canada, November 2019.

[42] S. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "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)*, pp. 25–33, San Francisco, CA, USA, June 2016.

[43] C. Yu, J. Lan, Z. Guo, and Y. Hu, "DROM: optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, no. 18, pp. 54539–64533, 2018.

[44] P. Sun, Y. Hu, J. Lan, L. Tian, and M. Chen, "TIDE: time-relevant deep reinforcement learning for routing optimization," *Future Generation Computer Systems*, vol. 99, pp. 401–409, 2019.

[45] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntes-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," pp. 1–3, 2017, https://arxiv.org/abs/1709.07080.

[46] Q. T. A. Pham, Y. Hadjadj-Aoul, and A. Outtagarts, "Deep reinforcement learning based QoS-aware routing in knowledge-defined networking," in *Proceedings of the 14th EAI International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pp. 1–13, Ho Chi Minh City, Vietnam, December 2018.

[47] X. Guo, H. Lin, Z. Li, and M. Peng, "Deep-reinforcement-learning-based QoS-aware secure routing for SDN-IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6242–6251, 2020.

[48] J. Rischke, P. Sossalla, H. Salah, F. H. P. Fitzek, and M. Reisslein, "QR-SDN: towards reinforcement learning states, actions, and rewards for direct flow routing in software-defined networks," *IEEE Access*, vol. 8, pp. 174773–174791, 2020.

[49] M. Ibrar, L. Wang, G.-M. Muntean, J. Chen, N. Shah, and A. Akbar, "IHSF: an intelligent solution for improved performance of reliable and time-sensitive flows in hybrid SDN-based FC IoT systems," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3130–3142, 2021.

[50] K. Phemius and M. Bouet, "Monitoring latency with OpenFlow," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pp. 122–125, Zurich, Switzerland, October 2013.

[51] D. P. Kingma and J. L. Ba, "Adam: a method for stochastic optimization," in *Proceedings of the 3rd International Conference for Learning Representations*, pp. 1–41, San Diego, CA, USA, December 2015.

[52] Mininet Team, "Mininet: an instant virtual network on your laptop (or other PC)," 2018, https://mininet.org/.