QoS-aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach

Shih-Chun Lin*, Ian F. Akyildiz*, Pu Wang†, and Min Luo‡
*BWN Lab, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332
†Department of Electrical Engineering and Computer Science, Wichita State University, Wichita, KS 67260
‡ Shannon Lab, Huawei Technologies Co., Ltd. Santa Clara
Email: slin88@ece.gatech.edu; ian@ece.gatech.edu; pu.wang@wichita.edu; min.ch.luo@huawei.com

***Abstract*—Software-defined networks (SDNs) have been recognized as the next-generation networking paradigm that decouples the data forwarding from the centralized control. To realize the merits of dedicated QoS provisioning and fast route (re-)configuration services over the decoupled SDNs, various QoS requirements in packet delay, loss, and throughput should be supported by an efficient transportation with respect to each specific application. In this paper, a QoS-aware adaptive routing (QAR) is proposed in the designed multi-layer hierarchical SDNs. Specifically, the distributed hierarchical control plane architecture is employed to minimize signaling delay in large SDNs via three-levels design of controllers, i.e., the *super, domain (or master), and slave controllers*. Furthermore, QAR algorithm is proposed with the aid of reinforcement learning and QoS-aware reward function, achieving a time-efficient, adaptive, QoS-provisioning packet forwarding. Simulation results confirm that QAR outperforms the existing learning solution and provides fast convergence with QoS provisioning, facilitating the practical implementations in large-scale software service-defined networks.**

***Index Terms*—QoS-awareness, adaptive routing, reinforcement learning, distributed hierarchical architecture, control plane design, software-defined networks (SDNs).**

I. Introduction

Software-defined networks (SDNs) have been recognized as the next-generation networking paradigm with the promise to dramatically improve network resource utilization, simplify network management, reduce operating cost, and promote innovation and evolution [1]–[3]. However, despite of their "advertised" promising features, a reliable end-to-end transportation upon SDNs is hard to design due to the requirements of various QoS provisioning and fast route (re-)configuration. In particular, aiming at upholding a great variety of applications, SDNs should fulfill various QoS requirements such as in packet delay, packet loss, and throughput. Moreover, as users and multi-tenancy applications greatly increase as well as network topology and traffic statistic change over time, SDNs should also provide a fast and adaptive data transportation in order to react such events in a real-time manner. Therefore, it becomes a great challenge and an urgent need to support a time-efficient and QoS-aware routing in large-scale SDNs.

While the specification of OpenFlow [4] requires the logically centralized control plane in SDNs, single controller scheme faces several crucial issues such as network scalability, single-point failure, and frequent reporting control tasks, per-flow data supervision. Recent work [5]–[7] focuses on distributed multi-controller platforms. In ONIX [5], a horizontally flat structure of all controllers is proposed and a general management API is implemented to connect multi-controllers. In [8], a scalable clustering approach and a self-learning adaptive mechanism are proposed to design and implement SDN controller clusters. In Kandoo [6], a completely hierarchical model is proposed, including a logically centralized root controller and local controllers, and local controllers can directly offload the applications that do not need network-wide information to the underlying switches. In Xbar [7], a recursive hierarchy design is proposed among multiple controllers that a lower-level controller is recognized as a switch by its upper controller. Facing these distributed SDN systems, an adaptive routing that exploits the system advantages and fulfills QoS requirements in a timely manner is still unexplored.

In this paper, we propose a QoS-aware adaptive routing (QAR) in our multi-layer hierarchical SDNs. First, inspired by the work of Kandoo [6] and Xbar [7], a distributed hierarchical control plane architecture is introduced that combines the advantages of both work and is complied with OpenFlow 1.2+. Specifically, the three hierarchical levels of distributed controllers, including the *super, domain (or master), and slave controllers*, and the switch subnets (i.e., clustering) are proposed. Exploiting such a novel architecture, the control loads are shared and the signaling delay can be largely reduced. Furthermore, with the aid of reinforcement learning [9], QAR algorithm is proposed through the examination of action policy, quality function, long-term revenue, and system model with reward function. Specifically, the softmax action selection policy, state-action-reward-state-action (SARSA) [10] method for quality update, and Markov decision process (MDP) with QoS-aware reward function are introduced to realize an efficient, adaptive, QoS-provisioning routing.

Inherited from reinforcement learning, QAR enjoys the four crucial features as follows:

- QAR has fast adaptation to the current network and traffic

978-1-5090-2628-9/16 $31.00 © 2016 IEEE
DOI 10.1109/SCC.2016.12

states for the time-varying multi-tenancy environment as well as network topology.
- QAR well distributes the traffic loads from QoS-aware reward, avoiding congestions as in shortest-path algorithms.
- QAR has great scalability due to the scalable learning, easily including new devices in the next learning iteration.
- QAR supports customized requirements from its tunable parameters in and generic design of the reward function.

Performance evaluation confirms that QAR outperforms the conventional Q-learning [9] approach with great time-convergence and QoS provisioning in real backbone network.

To the best of our knowledge, this work is the first to provide a QoS-aware adaptive routing with preferred fast-convergence, through reinforcement learning, in multi-layer hierarchical SDNs. The rest of the paper is organized as follows. Section II introduces the system model and Section III presents the proposed multi-layer hierarchical control plane. Section IV further provides the proposed QAR in the designated control plane architecture. Section V introduces the performance evaluation and Section VI concludes the paper.

II. System Model

We first examine the fundamental operation and network topology of SDNs. We then focus on the control plane design, particularly in the number of controllers.

A. Network Topology

As shown in Figure 1, while current network architecture vertically integrates closed and proprietary switches jeopardizing fast innovation, SDN brings a paradigm shift with horizontal and open interfaces that provide great design flexibility. A typical SDN generally consists of multiple OpenFlow enabled switches (i.e. OF switches), which constitutes data plane, and the centralized SDN controller(s) [4]. In particular, OF switches are software-configurable devices with multi-flow tables that can identify traffic flows. The traffic identification is realized through attribute analysis over seven-layers OSI [11] in terms of source/destination IP address, traffic type, subscriber profile, etc., and can be further enhanced via wildcard usage. Moreover, each OF switch needs to send the control purpose traffic, such as the route setup requests for new flows and real-time network congestion status, to the SDN controller. Based on the continuously received control messages, the controller optimizes the best routes for data flows according to dynamically changing traffic patterns and flow QoS requirements and sets up the routing tables of OF switches along the optimal path via certain protocols (e.g., OpenFlow), thus enabling highly efficient data transmissions and superior link utilization [12], which is already demonstrated in practical SDNs [13], [14].

Despite the promising performance of SDN, its effectiveness and scalability depends on the design of *resilience routing algorithm* that not only supports any kinds of QoS requirements from various applications, but fast adapts to time-varying requirements, traffic statistics, and network topology, serving as the focus of this paper. Note that, in the remainder of the paper, we refer to switches as OF switches to simplify the readability.

B. Single Controller versus Multiple Controllers

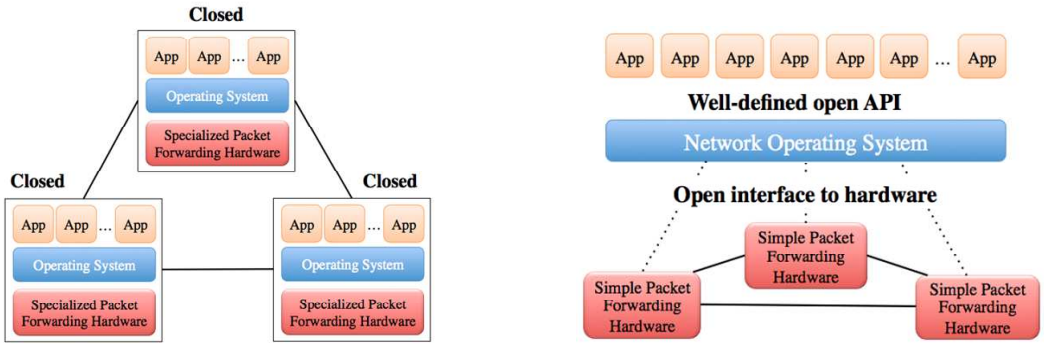
Both single- and multi-controllers infrastructures could be the solution candidates for the practical implementation of centralized control plane. More specifically, in single-controller SDNs, a controller manages the entire data planes by computing optimal flow routes with the aid of its high computation capability and global visibility of traffic and network states [15]. This scenario simplifies the management of complex flows as well as the corresponding customization, and allows network operators have direct and effective control over the network due to the only required configuration of a controller. However, these single-controller SDNs, having the computation limitation by a controller, usually face the scalability issue when network size or the number of flows increases. Moreover, the reliability is another critical issue due to a single point failure of the controller.

On the other hand, multi-controller SDNs are favored whenever a controller's capability is not enough for entire network or multiple controllers are more cost-effective in terms of performance and infrastructure cost. In this multi-controller model, several controllers jointly mange the data plane, and a centralized interface is required to provide *virtualization* for underlying switches that still see a single control unit from their perspectives. Thus, facing the popularity of multi-controller model in large-scale networks, such as data center clusters, cloud and inter-domain scenarios, we introduce a multi-layer hierarchical (distributed) architecture in the following, which enables fast signaling to resolve the design complexity and information inconsistency.

III. Multi-layer Hierarchical Control Plane Architecture

Based on the architecture designs of Kandoo [6] and Xbar [7], we propose a multi-layer hierarchical control architecture that combines the advantages of these two existing solutions and is complied with OpenFlow protocol [4]. The details are explained as follows.

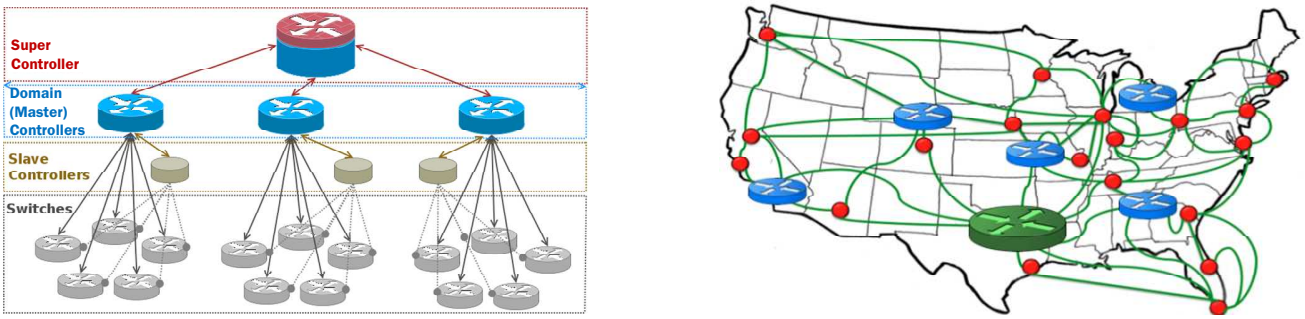
The proposed architecture consists of a recursive hierarchical control plane with three levels of controllers as shown in Figure 2a. While switches take charge of data forwarding and information collection of network status, the slave controllers provide read-only access to switches and receive port-status messages from them. These slave controllers not only serve as the message dispatchers as in [8] that ease the bottleneck of excessive control messages in the I/O frontend, but also can provide some simple control functions, such as traffic admission control, flow or congestion control, to share control workloads with domain controllers. Moreover, the domain (or master) controllers, having full accesses to switches, receive asynchronous messages (e.g., Packet-IN [4]) for flow-setup requests and are capable to modify switches' states by sending control messages. Finally, the only one super controller, connecting to domain controllers, also has



(a) Current closed architecture.

(b) SDN architecture.

Fig. 1: A paradigm shift from closed architecture to SDN architecture.



(a) Distributed hierarchical architecture.

(b) A deployment example in a real Sprint GIP network [16] over North America.

Fig. 2: Multi-layer hierarchical control plane.

full accessibility to switches and regulates the entire network functionalities. Note that, the interaction between super controller and domain controllers fulfills global flow setup and responds to every control action, including actions for switches' or controllers' failures, migrations, load-balancing, etc. Furthermore, the slave controllers only aim to offload control messages for various applications and do not require network-wide information, largely saving the signaling overheads. Towards this, the logically centralized control plane with global visibility is established by a physically distributed system. Thus, all applications running upon SDNs are unaware of the underlying distributed architecture.

A deployment example in a real backbone system of Sprint GIP network [16] is illustrates in Figure 2b, where a red spot denote a slave controller with a group of switches underneath, a blue device denotes a domain controller serving a switch subnet and possibly more than one slave controller, and the green device denotes the super controller to supervise the entire system. One advantage of the design is that this architecture decouples the failure recovery from path computation. In particular, the super controller designates a domain controller whenever a failure occurs, and the domain controller computes the corresponding recovery path to resolve the failure. What is more important, to ensure reliability and robustness upon controllers' failure, switches at the edge of a subnet can establish communication with more than one domain controller. In that way, it allows easy control handover between switch subnets with respect to their domain controllers. Furthermore, with full accessibility to switches, the super controller and domain controllers have the ability to identify the cause of switch asynchronization and enable/disable the warning notification.

In spite of these considerable advantages, to realize an efficient adaptive routing upon the proposed architecture, the hierarchical structure is exploited to greatly minimize the signaling delay between controllers and switches. The idea comes from (i) the signaling-load distribution and (ii) parallel path computation. Specifically, each domain controller is in charge of the signaling within its own switch subnet in such a way when the first packet of a flow arrives into a specific subnet, the optimal route is solely computed by the corresponding domain controller. Only when the destination switch of arriving flow is outside the source switch's subnet, the packet is then forwarded to the super controller and multiple subnets (and domain controllers) will be involved in the path computation. In particular, the super controller exclusively calculates the subnet-path, i.e., the set of subnets that the packet flow will go through to reach its destination, with the aid of its global visibility. Once the subnet-path is calculated, the super controller forwards the control messages to the involved domain controllers in order to active their own

path computations within the respective subnet. As the optimal paths in different subnets can be computed in parallel, the flow does not need to wait whenever it enters the next subnet along the calculated subnet-path, thus minimizing the entire signaling delay.

IV. QoS-aware Adaptive Routing (QAR) in Multi-layer Hierarchical SDNs

Based on the designated distributed hierarchical control plane, we propose an QoS-aware adaptive routing (QAR) with the aid of reinforcement learning technique. In the following, we first introduce the learning framework, then provide our design of QoS-aware reward functions, and finally propose the QAR algorithm.

A. Reinforcement Learning Framework

Reinforcement learning [9], belonging to a field of machine learning, captures the problem that an agent/decision maker tries to learn the behavior of dynamic system through the interactions with the system. Specifically, at each iteration, the agent receives the current state and the reward from the dynamic system, and then performs the respective action according to its pat experience in order to increase the long-term revenue via state transitions. The state and the reward are the two values that the agent will receive from the system, whereas the action is the only input that the system will receive from the agent. Different from the supervised learning techniques with an external knowledge supervisor, in reinforcement learning, the agent must discover the best action that maximizes the reward itself. This reward value indicates the success of agent's action decisions, and the agent learns which actions to be selected to provide the highest accumulated reward over time, i.e., the long-term revenue. While agent's actions affect not only the immediate reward but also the subsequent one, the key feature for the reinforcement learning is to perform incentive solution searching with regards of the system reward.

To realize this searching in an efficient way, the design challenge comes from the balance between action exploration and action exploitation, where such a trade-off is well studied in [17]. In particular, the agent has to exploit the past actions with great rewards, and to explore the system for better unknown actions at the same time. It means both the exploitation and exploration needs to be pursued conjointly for the optimal system performance. As there are many sophisticated algorithms that detail this joint consideration into their trial-and-error designs, the reinforcement learning only characterizes the interaction procedures instead of providing another learning methods. In other words, any learning algorithm can be seen and transformed into a reinforcement learning. In the following, based on the reinforcement learning technique, we provide several design ingredients that are used by our adaptive, time-efficient, and QoS-aware routing.

B. Markov Decision Problem: States, Actions, and Rewards

In addition to an agent/decision maker and the dynamic system, there are four main ingredients for the design of reinforcement learning: (i) the action policy, (ii) the quality function, (iii) the long-term revenue, and (iv) the system model with reward functions. Specifically, the action policy is the decision rules that will be taken by the agent. It is the mapping from the perceived system states to the corresponding actions, and guides the behavior of a reinforcement learning. Moreover, the quality function characterizes the quality of each state-action pair that indicates the differences between current state and steady state. Furthermore, the long-term revenue indicates the total rewards an agent can expect to accumulate over time with respect to each system state. Whereas the reward is given after each agent's current action, this revenue shows the long-term desirability of states regarding the future states that will be followed and their respective rewards. Last, the system model mimics the behavior of the real environment system, and gives a good reward prediction of the next state and quality from the current ones.

Towards this, Markov decision processes (MDPs) provide a mathematical framework for the system modeling with respect to the reinforcement learning. In particular, a MDP is denoted by the quadruple (S, A, P, R), where S denotes the finite state set, A denotes the finite action set, P denotes the set of state transition probabilities, and R denotes the reward set. Moreover, the reward prediction function should be designed according to the problem of interest. In the following, we first examine the details of action policy and quality function. The reward function design for our routing problem is provided later in Section IV-C.

1) Action Selection Policy: The policy specifies an agent's action selection and maps the state to the action. It balances the trade-off between action exploitation and exploration to maximize the quality value, as the agent aims to *explore* the state space in the beginning. Three policies are widely-used, i.e., the greedy, the ϵ-greedy, and the softmax [18]. For the greedy policy, the agent takes the action with the highest quality at every step. It simply exploits the current agent's knowledge base of state and quality function, and does not explore unknown states for possibly higher quality. This makes the policy undesirable when the quality function is non-stationary, which changes over time. Moreover, ϵ-greedy balances the current knowledge exploitation with action exploration that follows the greedy policy with probability $1-\epsilon$ and takes a random action with probability ϵ. It serves as an effective policy when there are a great variety of possible actions. However, the drawback of ϵ is that when it explores it chooses equally among all actions. This implies that it is as likely to choose the worst-appearing action as it is to choose the next-to-best action, which is unsatisfactory when the worst actions are very bad.

Towards this, we consider softmax and adopt it later in our designated routing. Regarding softmax, the probability $\pi_t(s, a)$ of choosing an action a_t with the current state s_t follows

$$\pi_t(s_t, a_t) = \frac{\exp\left(Q_t(s_t, a_t)/\tau_n\right)}{\sum_{b=1}^{n} \exp\left(Q_t(s_t, b_t)/\tau_n\right)}, \tag{1}$$

where n is the number of possible actions, $Q_t(s_t, a_t)$ denotes

the corresponding quality function, and τ_n is a parameter called temperature. This time-varying temperature controls the trade-off between exploration and exploitation. High temperature causes all actions to be equally probable (i.e., exploration), while low temperature favors the action with the maximum quality (i.e., exploitation of current knowledge base) that skews the policy toward a greedy one. In this way, the temperature parameter is annealed over the training phase that has greater exploration in the beginning and greater exploitation near the end. It means τ_n remains a high value in highly dynamic environments while decreases towards a low value in static environments where the convergence is assured.

To achieve the learning convergence in finite time, the temperature is thus set as a linear function over time as

$$\tau_n = -\frac{(\tau_0 - \tau_T)n}{T} + \tau_0, \quad n \leq T, \tag{2}$$

where T denotes the time to reach the convergence, and τ_0 and τ_T are the initial temperature and last temperatures at time T, respectively. It implies that $\tau_n \neq 0$ for all time and $\tau_n = \tau_T \approx 0$ for $n \leq T$ until any system-parameter change.

2) State-Action Quality Function: In addition to estimate the quality solely by the possible next system state, the agent can set up its quality function based on both the state and action. Specifically, the quality function $Q(s, a)$ is introduced that shows the quality for taking action a at the current state s. Thus, when the agent needs to choose an action for current state s, it simply calculates $Q(s, a)$ for each possible action a and chooses the next action according to these quality values. In the following, two methods for setting quality function are provided as the conventional Q-learning [9] and SARSA [10], and SARSA is adopted later in our designated routing. First, the well-known Q-learning, being an off-policy reinforcement learning, updates the Q function as follows

$$\begin{aligned} Q_{t+1}(s_t, a_t) := & Q_t(s_t, a_t) \\ & + \alpha \left[R_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right] \end{aligned} \tag{3}$$

where $\gamma \in [0, 1)$ is the discount factor that determines the importance of future rewards, $\alpha \in [0, 1)$ is the learning rate that determines the override extent of the newly acquired information to the old one, and R_t is the reward at time t. In Eq. (3), the agent updates the quality based on the maximum possible quality value among its actions. Specifically, the agent chooses and takes action a_t for the current state s_t via action selection policy, observes R_t and state s_{t+1}, and the Q function can be updated accordingly.

On the other hand, regarding the on-policy reinforcement learning of SARSA, the quality function is updated by

$$\begin{aligned} Q_{t+1}(s_t, a_t) := & Q_t(s_t, a_t) \\ & + \alpha \left[R_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right]. \end{aligned} \tag{4}$$

In this time, the agent updates Q function strictly on the knowledge base from the experience. Specifically in Eq. (4), different from Q-learning, the agent uses the action and the state at time $t+1$ to update quality value. Therefore, the only difference between these two methods is the way they set up for the future reward. In particular, whereas Q-learning utilizes the highest quality function at state s_{t+1} regarding all possible actions, SARSA adopts the quality function at state s_{t+1} with action a_{t+1}. General speaking, Q-learning simply assumes an optimal policy will be followed in the future; SARSA utilizes the policy that the agent indeed follows in the future. It means that the agent with SARSA can explicitly adopt the future reward that is really obtained, rather than assuming the optimal action with highest reward will be taken.

TABLE I: QoS Requirements With Respect to Traffic Types and Applications.

Traffic Type	Application	QoS-awareness
Elastic	Telnet connection; FTP session	Delay, losses
	Simple web page (HTTP)	Delay
	Heavy web page (HTTP)	Throughput
	STMP/POP3/IMAP	Losses
	FTP data connection	Throughput
	Data with Telnet	Losses
Inelastic	Real-time multimedia	Delay, throughput, jitter
	Control message	Delay

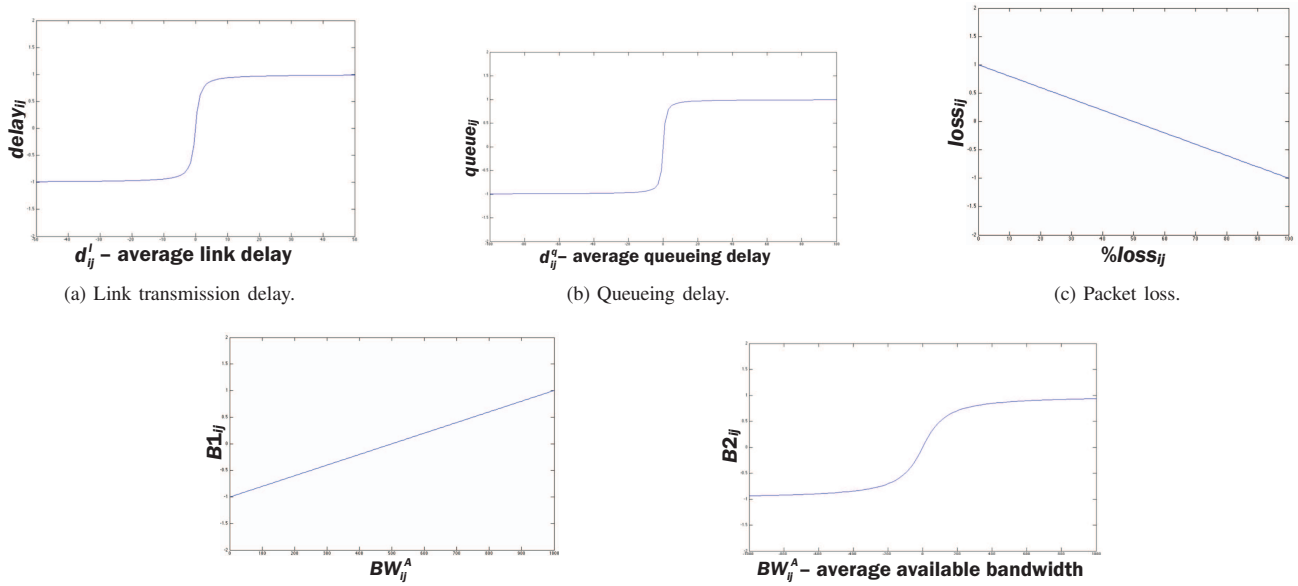
C. QoS-aware Reward Design

In this section, we propose QoS-aware reward functions that suits our design of QoS-aware routing. Specifically, based on reinforcement learning, the agent finds the routing path with the maximum QoS-aware reward with regard of traffic types and users' applications. In particular, TABLE I summarizes various QoS requirements of widely-applied traffic and applications. For example, real-time traffic, inelastic to adapt packet transmission rates, has great QoS-awareness among others.

Towards this, a QoS-aware reward function is proposed as

$$\begin{aligned} R_t := & R(i \to j|_{s_t, a_t}) \\ = & -g(a_t) + \beta_1(\theta_1 delay_{ij} + \theta_2 queue_j) + \beta_2 loss_j \\ & + \beta_3(\phi_1 B1_{ij} + \phi_2 B2_{ij}), \end{aligned} \tag{5}$$

which shows that the system at state s_t, receiving action a_t, forwards packets from node i to node j. In Eq. (5), $g(\cdot)$ denotes the cost to take action a_t that reveals the action impact to switch operations, and $\beta_1, \beta_2, \beta_3, \theta_1, \theta_2, \phi_1, \phi_2 \in [0, 1)$ are the tuneable weights, determined by the QoS requirements of flow. Aiming at QoS provisioning, the cost g is set to a constant value over actions, and the QoS-aware functions in



(a) Link transmission delay.

(b) Queueing delay.

(c) Packet loss.

(d) Available bandwidth in next node.

(e) Difference of available bandwidth in next node and average bandwidth.

Fig. 3: QoS-aware reward functions.

Figure 3 are defined as follows:

$$delay_{ij}=\frac{2}{\pi}\arctan(d_{ij}^{l}-\frac{\sum_{k=1}^{A(i)}d_{ik}^{l}}{A(i)}); \quad (6a)$$

$$queue_{ij}=\frac{2}{\pi}\arctan(d_{ij}^{q}-\frac{\sum_{k=1}^{N}d_{ik}^{q}}{N}); \quad (6b)$$

$$loss_{ij}=1-2\%\text{loss}_{ij}; \quad (6c)$$

$$B1_{ij}=\frac{2BW_{ij}^{A}}{BW_{ij}^{T}}-1; \quad (6d)$$

$$B2_{ij}=\frac{2}{\pi}\arctan(0.01(BW_{ij}^{A}-\frac{\sum_{k=1}^{N}BW_{ik}^{A}}{N})), \quad (6e)$$

where d_{ij}^{l} and d_{j}^{q} are the link transmission delay and packet queueing delay from node i to node j, respectively, $A(i)$ is the number of node i's neighbors and N is the number of switches in the considered switch subnet, and $\%loss_{ij}$, BW_{ij}^{A}, and BW_{ij}^{T} characterizes the packet loss, available bandwidth, and total bandwidth of link i-j, respectively. Eq. (6a) considers the link delay of link i-j comparing to other possible next hops, Eq. (6b) considers the queueing delay with respect to the average delay over the subnet, and Eq. (6c) characterizes the loss rate. Note that the different comparisons in Eq. (6a) and Eq. (6b), i.e., the neighbors $A(i)$ and the switches N in a subnet, respectively, indeed provides the thorough consideration of packet latency over the hierarchical structure and switch subnets. Eq. (6d) and Eq. (6e) indicate the available bandwidth of link i-j and that with respect to the average link bandwidth over the subnet, respectively. From Figure 3, all these QoS functions have the values within $[-1, 1]$, where the value closes to one means the link selection is preferred by the respective parameter and otherwise the value closes to negative one as the penalty.

D. QoS-aware Adaptive Routing (QAR)

Inspired by our previous study of adaptive computation framework for routing paths [19], we propose a QoS-aware adaptive routing (QAR) in **Algorithm 1** with a flow diagram in Figure 4 through reinforcement learning and designated QoS-aware rewards in multi-layer hierarchical SDNs. Specifically, as mentioned in Section III, this distributed SDN system consists of various switch subnets, and each subnet has a domain controller, one or many slave controllers, and many underlying switches. Moreover, domain controller takes charge of path calculation for each incoming flow, while slave controllers gather the network state and send the information updates to the domain controller shown in Figure 4. Therefore, QAR determines the forwarding path inside each subnet for the respective domain controller and the global forwarding direction among subnets for the super controller. The procedures are explained in detail as follows.

First, when a new flow arrives to a switch, the switch forwards the first packet of the flow to the domain controller and requests the forwarding path. The domain controller then updates the current network state regarding the latest information from slave controller(s), exploits the proposed reinforcement learning in **Algorithm 2** to select a feasible path with respect to QoS requirements of the flow, and modifies the forwarding tables of switches along the selected path. Furthermore, if the domain controller realizes that the destination switch does not belong to its subnet, the first packet will also be sent to the super controller. The super controller

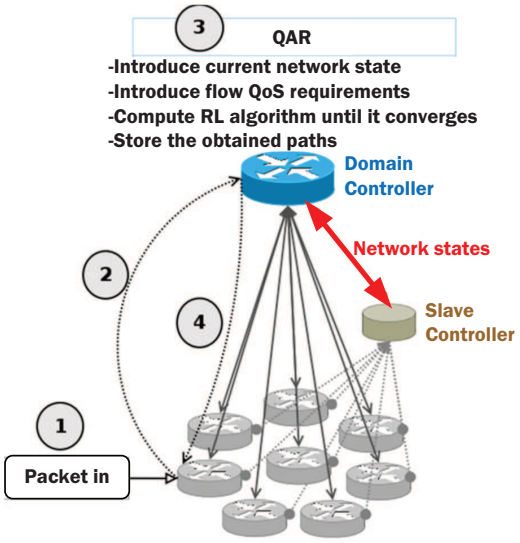


Fig. 4: Flow diagram of QAR.

then performs **Algorithm 2** to find the forwarding direction among subnets (i.e., subnet-path), and send the corresponding notifications to the domain controllers of involved subnets. In this way, the path searching of mater controller and several domain controllers can be executed simultaneously and thus save much computation time to facilitate time-efficient QAR.

Note that the computation loads, distributed among three levels of controllers, are largely reduced through this hierarchical load-sharing. Moreover, being complied with OpenFlow, if there exists the matching entry of the incoming flow, the switch will not send the packet to domain controller but simply forwards the packet following the existing matching. Thus, QAR successfully provides a QoS-aware, time-efficient, adaptive routing, particularly effective for large-scale SDNs. Finally, such a better routing function with the proper consideration of QoS requirements and changes of network status offers one of the best candidate for an innovative service-enabled, virtualized, and generalized network function that could be easily integrated with other network planning and management systems.

Algorithm 1: QoS-aware Adaptive Routing (QAR)

1 New flow f arrives to a switch in the subnet.
2 Switch forwards the first packet to domain controller E_f.
3 **if** $Dest(f)$ *is not in the same subnet* **then**
4 | Super controller executes **Algorithm 2**;
5 | Domain controllers along the subnet-path executes **Algorithm 2**;
6 **else**
7 | Domain controller E_f executes **Algorithm 2**;
8 **end**
9 Rest packets of flow are forwarded following the established flow tables in switches.

Algorithm 2: Reinforcement Learning

1 At source i (i.e., either switch or domain controller.)
2 **Initialize** $Q_0(s_0, a_0) = 0$ and R_0 from Eq. (5).
3 At time t:
4 **Choose** next-hop via a_t acc. to softmax in Eq. (1).
5 **Observe** R_t and s_{t+1}.
6 **Update** Q_{t+1} function acc. to Eq. (4).
7 Continue from step 4 to choose next-hop at time $t+1$.

V. Performance Evaluation

We evaluate the proposed QAR and compare it with the conventional learning approach over a real Sprint GIP network [16] with multi-layer hierarchical architecture as shown in Figure 2b. In particular, this network has 25 deployed nodes as switches and 53 links, and the actual link delay profiles can be obtained in [16]. In the following, we first evaluate the impact of step-size parameters in QAR, and then compare its performance with the Q-learning scheme [20].

A. Impacts of Step-Size Parameters in QAR

Based on the reinforcement learning, the performance of QAR is governed by the selection of step-size parameters (α,γ) as shown in Eq. (4). In particular, α adjusts the error that is included in the Q updating; $\gamma \in [0, 1)$ has zero value if the routing only considers the current reward and acts as greedy algorithm, and has the value close to one if the routing considers the long-term revenue. Figure 5 provides the number of hops of a suitable path via QAR with respect to (α,γ). It shows QAR converges only when $\gamma \in [0.7, 0.99]$, which implies the recent action indeed affects system performance more than the future ones. Moreover, with these large γ values, QAR performs better with few fluctuations when $\alpha \in [0.5, 1)$. Thus, these evaluations provide the preferred ranges of the step-size parameters with regards of routing performance, facilitating the practical implementation of QAR.

B. Performance Comparison of QAR and Q-Learning [9]

We examine the time evolution of QAR without and with QoS provisioning, and compare QAR with the conventional Q-learning [9]. First, Figure 6 shows the QAR results with respect to different time-to-live (TTL) values that denote the maximum allowable number of searching iterations in each episode (TTL is set as 100 in Section V-A). In particular, without QoS provisioning, the weights $\beta_1, \beta_2, \beta_3$ in Eq. (5) are set to zero, and Figure 6a shows that the greater the TTL is the faster the QAR converges. However, there exists a trade-off between algorithm convergence and end-to-end delay. Specifically, while an increasing TTL brings better convergence, it also increases the computation time of each searching episode and thus increases end-to-end delay.

Figure 6b further shows the results with QoS provisioning, where $\beta_1 = \beta_2 = \beta_3 = \theta_1 = \phi_1 = 1$ and $\theta_2 = \phi_2 = 0.5$. It indicates that the longer convergent time is required when considering QoS requirements. In particular, while QAR does

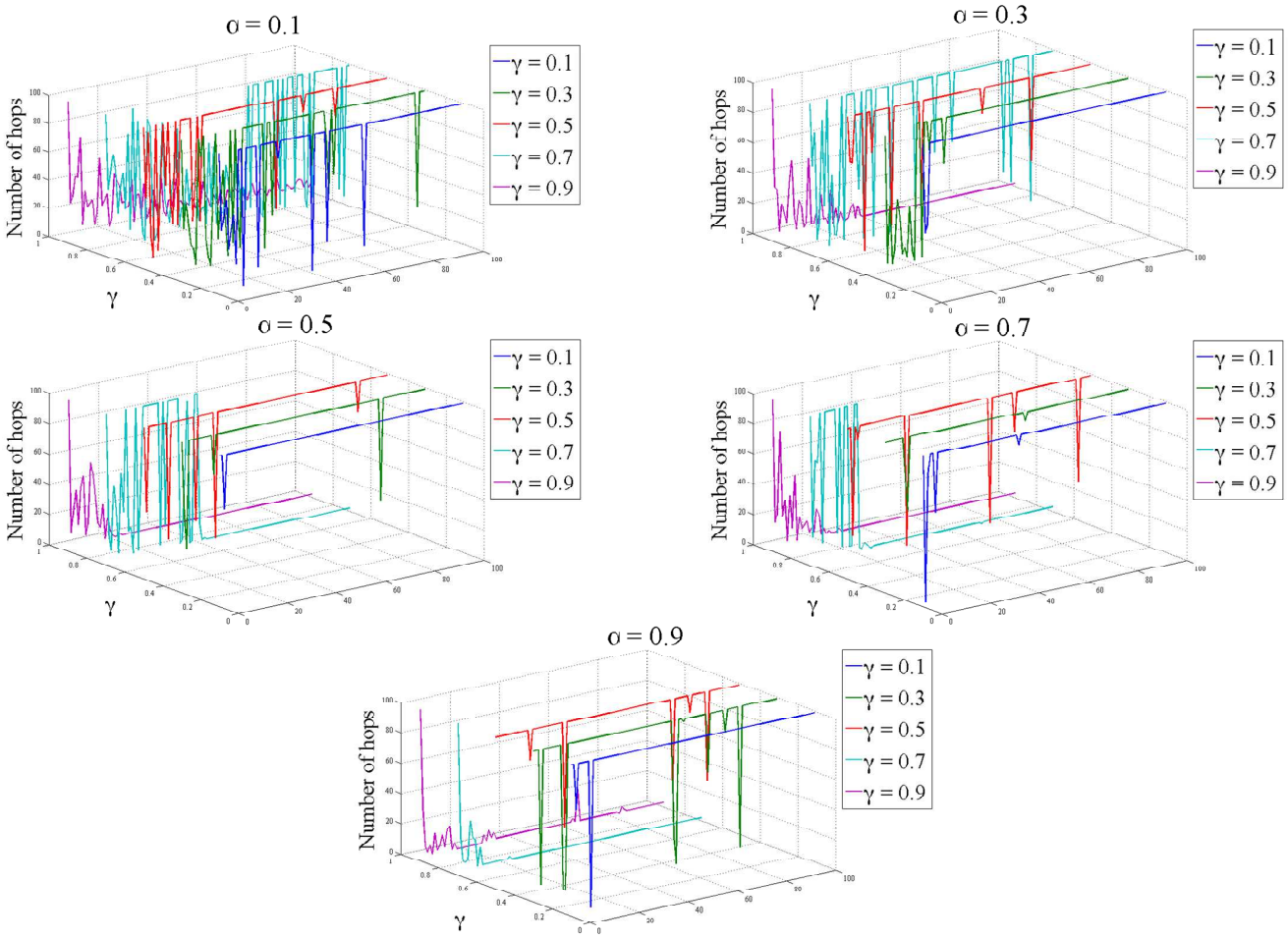
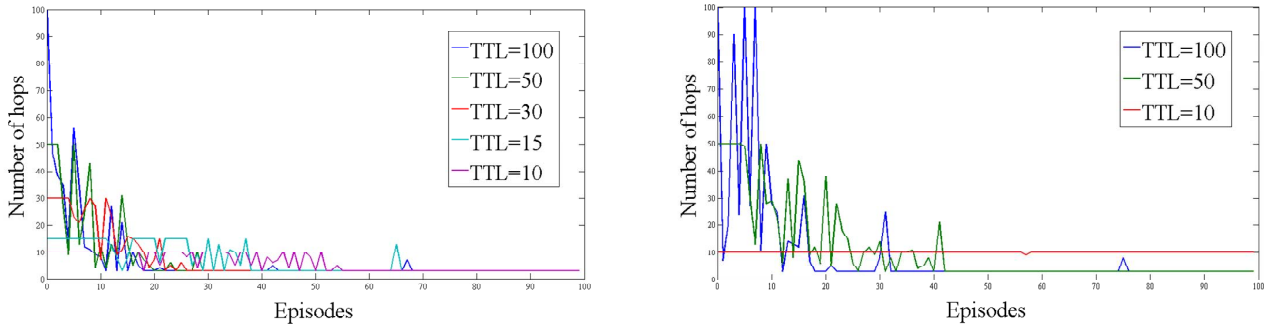


Fig. 5: Number of hops with respect to different step-size parameters (α,γ) of QAR.

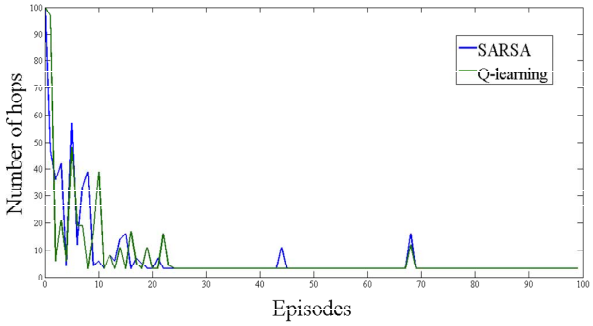


(a) Without QoS provisioning.

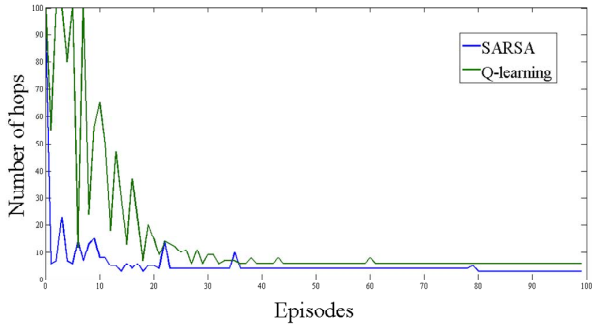
(b) With QoS provisioning.

Fig. 6: Time evolution of QAR.

not converge when TTL= 10, it requires double episodes to converge when TTL= 50. For TTL= 100, QAR takes almost the same episode to converge with or without QoS provisioning. Note that the fluctuations after the graph converges means that even finding a suitable forwarding path, QAR still tries to find the better solution with greater reward. Next, Figure 7 shows the comparison between QAR and Q-learning. It indicates that both approaches have similar convergent performance without QoS provisioning, but QAR outperforms with QoS provisioning. Therefore, this paper provides an effective and efficient routing algorithm upon multi-layer hierarchical architecture for large-scale SDNs.



(a) Without QoS provisioning.



(b) With QoS provisioning.

Fig. 7: Comparison between QAR and conventional Q-learning [9].

VI. Conclusion

In this paper, QAR is proposed via reinforcement learning in multi-layer hierarchical SDNs. This distributed hierarchical control plane is first introduced to minimize the signaling delay, serving as a realistic SDN architecture. QAR is then proposed to enable adaptive, time-efficient, and QoS-aware packet forwarding upon the proposed architecture. Performance evaluation confirms that QAR outperforms the conventional Q-learning approach with fast convergence when considering QoS provisioning. We have presented a novel design to facilitate on-line, QoS-aware routing in practical large-scale SDN and software service-defined network implementations.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, , and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[2] I. F. Akyildiz, P. Wang, and S. C. Lin, "Softair: A software defined networking architecture for 5g wireless systems," *Computer Networks*, vol. 85, pp. 1–18, 2015.

[3] I. F. Akyildiz, S. C. Lin, and P. Wang, "Wireless software-defined networks (w-sdns) and network function virtualization (nfv) for 5g cellular systems: An overview and qualitative evaluation," *Computer Networks*, vol. 93, pp. 66–79, 2015.

[4] ONF, "OpenFlow Switch Specification," version 1.4.0. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *OSDI*, 2010.

[6] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *ACM HotSDN*, Aug. 2012, pp. 19–24.

[7] J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker, "Extending sdn to large-scale networks," in *ONS*, 2013.

[8] M. Luo, Q. Li, M. Bo, K. Lin, X. Wu, C. Li, S. Lu, and W. Chou, "Design and implementation of a scalable sdn-of controller cluster," in *the 4th International Conference on Smart Systems, Devices and Technologies (SMART): INFOCOMP 2015*, Jun. 2015.

[9] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press, 1988.

[10] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," Tech. Rep., 1994.

[11] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. Addison-Wesley, 2008.

[12] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks (Elesvier) Journal*, vol. 71, pp. 1–30, Oct. 2014.

[13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM*, Aug. 2013, pp. 3–14.

[14] C. Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *ACM SIGCOMM*, Aug. 2013, pp. 15–26.

[15] S. C. Lin, P. Wang, and M. Luo, "Control traffic balancing in software defined networks," *Computer Networks*, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615002571

[16] Sprint, Overland Park, KS, "Sprint IP network performance," 2011, online available: http://www.sprint.net/performance.

[17] M. N. Katehakis and A. F. Veinott, *The Multi-Armed Bandit Problem: Decomposition and Computation*. Mathematics of OR, 1987.

[18] B. F. Lo and I. F. Akyildiz, "Reinforcement learning for cooperative sensing gain in cognitive radio ad hoc networks," *Wireless Networks, Springer*, vol. 19, no. 6, pp. 1237–1250, 2013.

[19] M. Luo, Y. Zeng, J. Li, and W. Chou, "An adaptive multi-path computation framework for centrally controlled networks," *Computer Networks (Elesvier) Journal*, vol. 83, pp. 30–44, 2015.

[20] C. Watkins and P. Dayan, *Q-Learning*. Machine Learning, 1992, vol. 8.