# Intelligent Routing Based on Reinforcement Learning for Software-Defined Networking

Daniela M. Casas-Velasco, *Graduate Student Member, IEEE*,
Oscar Mauricio Caicedo Rendon, *Senior Member, IEEE*, and Nelson L. S. da Fonseca, *Senior Member, IEEE*

***Abstract*—Traditional routing protocols employ limited information to make routing decisions, which can lead to a slow adaptation to traffic variability, as well as restricted support to the Quality of Service (QoS) requirements of applications. This article introduces a novel approach for routing in Software-defined networking (SDN), called Reinforcement Learning and Software-Defined Networking Intelligent Routing (RSIR). RSIR adds a Knowledge Plane to SDN and defines a routing algorithm based on Reinforcement Learning (RL) that takes into account link-state information to make routing decisions. This algorithm capitalizes on the interaction with the environment, the intelligence provided by RL and the global view and control of the network furnished by SDN, to compute and install, in advance, optimal routes in the forwarding devices. RSIR was extensively evaluated by emulation using real traffic matrices. Results show RSIR outperforms the Dijkstra's algorithm in relation to the stretch, link throughput, packet loss, and delay when available bandwidth, delay, and loss are considered individually or jointly for the computation of optimal paths. The results demonstrate that RSIR is an attractive solution for intelligent routing in SDN.**

***Index Terms*—Reinforcement learning, routing, Software-defined networking, knowledge-defined networking.**

## I. Introduction

ROUTING determines the path taken by packets from a source to a destination node [1]. Although traditional Internet protocols, such as Open Shortest Path First (OSPF) [2] and Routing Information Protocol (RIP) [3], have successfully delivered best-effort traffic for several decades, the growth of traffic and the diversification of the Quality of Service (QoS) requirements of emerging Internet applications pose new challenges for the transport of the flows of packets generated by these applications [4]. Moreover, traditional routing protocols use limited information to make routing decisions, which can lead to a slow adaptation to dynamic traffic changes and restricted support to diverse QoS requirements [5].

Software-defined networking (SDN) and Machine Learning (ML) techniques have been envisioned to provide innovation for routing protocols [6], [7], [8]. Several solutions [9]–[16] have enhanced traditional routing protocols by leveraging SDN features, such as programmability, global view, logically centralized control, and decoupling of network control and packet forwarding [17]. However, these do not fully exploit knowledge about the network operation to accomplish intelligent routing. Other approaches have attempted to improve routing by using ML techniques [18]–[23]; however, distributed routing is typically assumed. This type of routing increases signaling overhead and can contribute to the formation of congestion. Several other solutions [24]–[30] employ both ML and SDN, but they depend on traditional routing protocols.

In this article, we introduce a novel approach for routing in SDN, called **R**einforcement Learning and **S**oftware-Defined Networking for **I**ntelligent **R**outing (RSIR). RSIR adds a Knowledge Plane and defines a routing algorithm based on Reinforcement Learning (RL) that takes into account link-state information to explore, learn, and exploit potential paths for intelligent routing, even during dynamic traffic changes. This algorithm capitalizes on interaction with the environment, the intelligence provided by RL, and the global view and control of the network furnished by SDN. It computes and installs, in advance, optimal routes in the routing tables of the switches on the Data Plane. RSIR was extensively validated using emulation and real traffic matrices. Results show that RSIR outperforms the Dijkstra's algorithm in relation to stretch, link throughput, delay, and packet loss produced when delay, loss, and available bandwidth are individually or jointly used as cost to compute optimal paths. Results evince that RSIR is a promising solution to replace traditional routing protocols in SDN.

The contributions of this article are:

- An architecture that employs RL for achieving efficient and intelligent routing in SDN;
- A proactive RL-based routing algorithm that considers link-state metrics to explore, learn, and exploit potential routes;
- A prototype of the proposed architecture.

The remainder of this article is organized as follows. Section II describes the related work. Section III details RSIR, and Section IV introduces the RSIR routing algorithm.

Manuscript received April 21, 2020; revised August 11, 2020 and October 22, 2020; accepted November 2, 2020. Date of publication November 10, 2020; date of current version March 11, 2021. The authors would like to thank CAPES, the Sao Paulo Research Foundation (FAPESP) under grant #19/03268-0 and 2015/24494-8. The associate editor coordinating the review of this article and approving it for publication was N. Zincir-Heywood. *(Corresponding author: Nelson L. S. da Fonseca.)*

Daniela M. Casas-Velasco and Nelson L. S. da Fonseca are with the Institute of Computing, University of Campinas, Campinas 13083-852, Brazil (e-mail: danielac@lrc.ic.unicamp.br; nfonseca@ic.unicamp.br).

Oscar Mauricio Caicedo Rendon is with the Department of Telematics, Universidad del Cauca, Popayán 190002, Colombia (e-mail: omcaicedo@unicauca.edu.co).

Digital Object Identifier 10.1109/TNSM.2020.3036911

TABLE I
ACRONYMS

| Acronym | Term | Acronym | Term | Acronym | Term |
|---|---|---|---|---|---|
| API | Application Programming Interface | NN | Neural Network | SDN | Software-Defined Networking |
| BGP | Border Gateway Protocol | OSPF | Open Shortest Path First | TCP | Transmission Control Protocol |
| IoV | Internet of Vehicles | QoS | Quality of Service | UDP | User Datagram Protocol |
| KDN | Knowledge-Defined Networking | RIP | Routing Information Protocol | VM | Virtual Machine |
| LLDP | Link Layer Discovery Protocol | RL | Reinforcement Learning | WSN | Wireless Sensor Network |
| ML | Machine Learning | RSIR | RL and SDN Intelligent Routing | | |
| NBI | Northbound Interface | SBI | Southbound Interface | | |

TABLE II
RELATED WORK

| Paper | Description | Control | SDN | ML | Performance metrics |
|---|---|---|---|---|---|
| [9] | An OSPF implementation in SDN, and conventional networks | C | ✓ | | Convergence time, throughput, packet loss ratio, jitter and delay |
| [10] | A comparative analysis of SDN and OSPF legacy networks | C | ✓ | | Convergence time |
| [11] | A comparison of routing performance when link failure occurs, considering the topology scale between SDN and conventional routing in traditional networks | C | ✓ | | Convergence time |
| [12] | NSAF: An application-aware routing in SDN employing application registration, network monitoring, violation detection, and routing control | C | ✓ | | Bandwidth, packet loss, delay, and jitter |
| [13] | SAQR: A QoS-aware routing in hybrid SDNs which adjusts the weights of metrics of a cost function using Dijkstra and genetic algorithms | C | ✓ | | Throughput, delay, and loss rate |
| [14] | An adaptive greedy flow routing algorithm for performance improvement in software-defined networks | C | ✓ | | Average delay, blocking probability, and running time |
| [15] | L2RM: A framework for load-balancing and route management for fat-tree SDN-based data center networks | C | ✓ | | Number of installed rules |
| [16] | Flow-aware routing and forwarding for SDN scalability in Wireless Data Centers | C | ✓ | | Number of flow setup requests |
| [18] | Q-PR: A routing algorithm that considers the message importance to reduce overhead with learning techniques in Wireless Sensor Networks | D | | ✓ | Delivery rate |
| [19] | FROMS: Routing mechanism in which nodes share local information without producing overhead | D | | ✓ | Delivery rate and network overhead |
| [20] | Appropriate input and output characterizations of network traffic for supervised neural networks in Wireless Sensor Networks (WSNs) | D | | ✓ | Signaling overhead, throughput and delay |
| [21] | Introduction of a proof-of-concept for applying the deep learning technique for performing intelligent traffic control in future networks | D | | ✓ | Signaling overhead, throughput and delay |
| [22] | ORuML: Optimized Routing in wireless networks by using ML techniques to predict the type of the source and destination nodes | D | | ✓ | Accuracy and Area under ROC curve |
| [23] | AQ-Routing: mobility, stability-aware adaptive routing protocol for data routing in MANET–IoT systems | D | | ✓ | Mean packet loss ratio and end-to-end delay |
| [24] | SDCoR: A Q-learning based approach for selecting a traditional routing algorithm in response to the Internet of Vehicles (IoV) environment changes | D | ✓ | ✓ | Delivery delay and delivery ratio |
| [25] | A supervised deep learning route computation for core networks over Software-defined routers | C | ✓ | ✓ | Delay, throughput, and signaling overhead |
| [26] | An ML-based meta-layer composed of multiple modules, each associated with a single source-destination pair for getting their routing path | C | ✓ | ✓ | Network delay |
| [27] | Distributed routing with SDN in which an agent learns the most important parameters to define the link costs for calculating routes in OSPF | D | ✓ | ✓ | Delay and jitter |
| [28] | QAR: QoS-aware adaptive routing with multi-layer hierarchies intended to minimize signaling delay in large SDNs | C | ✓ | ✓ | Delivery delay, packet loss,and hop count |
| [29] | A meta-heuristic with ML traffic prediction for energy efficient optical routing in mobile metro-core networks | C | ✓ | ✓ | Energy consumption and time overhead for routing calculation |
| [30] | An ML mechanism to accomplish the routing and wavelength assignment for an input traffic matrix in optical networks | C | ✓ | ✓ | Computational time |

C: Centralized and D: Distributed

Section V presents the RSIR prototype and the result of its evaluation. Section VI presents conclusions and future work. For the sake of readability, Table I presents the acronyms used in this article.

## II. RELATED WORK

This section brings up research on routing based on SDN, and ML as well on these two techniques. Table II briefly summarizes the description of the protocols reviewed, the type of

routing employed, how SDN and ML are considered, and the metrics evaluated. Next, we briefly describe these papers and point out their main shortcomings.

In SDN, instead of flooding routing updates over the entire network, information is sent to a centralized controller. Indeed, the controller is the only device informed about route changes; as a result, the signaling overhead and routing convergence time are less affected than in traditional routing [31]. The Control Plane provides a logically centralized control point that can make decisions based on collected information and deploy routing rules on the Data Plane. The work in [9]–[16] proposed routing solutions based on SDN features, such as programmability, global view, the decoupling of network traffic and control, and logically centralized control. However, these proposals use traditional distributed routing protocols that do not learn from all potential features affecting network congestion. Moreover, these proposals do not exploit intelligent solutions based on ML techniques.

The work in [18]–[23] employed ML techniques, such as RL and Neural Networks (NNs) to improve routing decisions. However, these proposals do not exploit SDN features. They deploy routing strategies in a distributed fashion in a way that routing nodes turn themselves into a learning entity that makes local routing decisions based on information learned from the environment. However, these routing solutions can generate signaling overhead and contribute to network congestion.

The work in [24]–[30] introduced routing strategies that encompass both ML and SDN. Network programmability provided by SDN allows the inclusion of ML techniques in the routing solutions. This type of solution is based on traditional routing protocols, either for deploying routing strategies or training ML models. Such dependence makes the proposed solutions prone to choose the same paths when similar congestion patterns occur.

RSIR aims at addressing the shortcomings of the proposals mentioned above. RSIR leverages the centralized, view, control, and programmability of SDN, as well as the cognitive capabilities of RL. RSIR proactively defines and installs routes based on decisions made by using link-state information with no dependence on traditional routing protocols. Moreover, it does not add any signaling overhead to the network operation.

## III. RSIR

This section presents an overview of RSIR, its architecture, and components.

### A. Overview

RSIR adds a Knowledge Plane, that employs RL, to SDN for achieving intelligent routing. The addition of a Knowledge Plane to a network was first proposed by Clark *et al.* [32] in 2003 and later revisited by Mestres *et al.* in 2017, when they proposed the concept of knowledge-defined networking (KDN) [33]. The Knowledge Plane is intended to furnish descriptions (recognize-explain), recommendations (recognize-explain-suggest), and automation (recognize-act) in support of decision-making processes such as routing. KDN involves ML [34], telemetry [35], network analytics [36], and a Knowledge Plane [32] to enable intelligent SDN. In addition to the Knowledge Plane, the KDN architecture includes the traditional SDN planes (i.e., Management, Application, Control, and Data) [6], [37], [38]. The SDN planes support network automated management and control, while the Knowledge Plane obtains and transforms information into knowledge by using ML techniques for improving network management and operation.

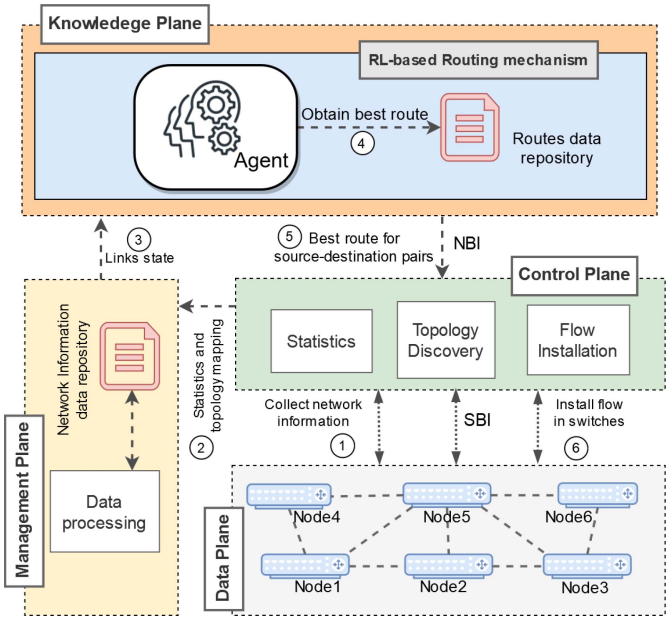



Fig. 1. RSIR architecture.

RSIR was designed for routing the network traffic automatically by using information about the network operation. RSIR uses RL to find the best route for all the source-destination pairs by employing just a few link-state metrics (i.e., available bandwidth, loss, and delay) as features of the RL process. RSIR takes advantage of SDN planes to obtain a centralized view and control. In this way, RSIR can adapt routes according to dynamic traffic changes. Moreover, RSIR does not rely on traditional routing protocols for calculating and deploying routing strategies as in [24]–[27], [29], [30], all of which employ both SDN and ML.

Overall, RSIR operates as described in Figure 1, and the explanation is detailed next. ① The Control Plane collects raw data about the network status by periodically querying the Data Plane. ② The Management Plane retrieves these data to calculate and store link-state information. ③ The Knowledge Plane recovers information from the Management Plane. ④ The RL-agent uses this information to explore-exploit all the possible routes for each pair of nodes. It learns and calculates the best route according to the state of the links. ⑤ The Knowledge Plane stores the information about the routes computed by the RL-agent. ⑥ The Control Plane retrieves the route information to install paths in the flow tables of switches before new traffic arrives (i.e., RSIR is proactive). In

this way, the Data Plane can forward packets without consulting the controller, eliminating the latency resulting from the queries sent to the controller on a flow basis. RSIR reacts fast to traffic changes, and it can make routing decisions based on only three link metrics.

### *B. Architecture and Components*

Figure 1 depicts the RSIR architecture intended to provide intelligent routing in SDN. We then provide details about the layers and components of RSIR.

*1) The Data Plane:* This plane includes the forwarding devices and the links connecting them. These devices perform a set of basic tasks, such as the forwarding of incoming packets to a specific port or the dropping of packets. The traffic moves according to the path information installed in the flow tables of the forwarding devices; these operate unaware of the rest of the network and rely on the Control, Management, and Knowledge Planes to populate and install their flow tables. The Data Plane realizes the RSIR routing strategies and periodically provides network information by responding to queries sent by the Control Plane.

*2) The Control Plane:* This plane builds up a global view of the Data Plane by gathering information (per port or flow) from the forwarding devices. It handles the correct installation of the routing rules in the table of these devices at the Data Plane. The Control Plane includes three modules: *Topology Discovery, Statistics, and Flow Installation modules*.

The *Topology Discovery module* sends *feature-request* messages to the forwarding devices (i.e., a switch) on the Data Plane, which answers back by sending *feature-reply* messages containing feature information, such as the id, number of ports, and the state of the ports. With the received information, this module infers the topology by relating the port of each switch to the ports of the neighboring switches, and the hosts connected to the ports at each switch. Topology information is available during execution as a collection of tuples (i.e., switch id, port id, neighbor switch id, neighbor port id) associated with the network links.

The *Statistics module* sends *request-state* messages to each forwarding device on the Data Plane at every $t$ seconds. This module receives the *request-stats* reply messages asynchronously. Finally, this module maintains statistical information during execution time for processing by the Management Plane, and periodically sends messages to keep the global view of the network updated.

The *Flow Installation module* operates proactively by populating the flow tables of switches ahead of time for all traffic matches; a Southbound Interface (SBI) such as OpenFlow can perform the installation. The installation of the paths (as flow entries) in the flow tables influences the forwarding of traffic in the Data Plane [39]. A misleading flow entry may cause the sending of traffic to highly utilized paths, and, as a result, the occurrence of network congestion may occur.

*3) The Management Plane:* This plane ensures the correct operation and performance of the network in the long term. It contains the *Data Processing* module and the *Network Information Data Repository*. The *Data Processing module* retrieves and uses raw data gathered by the Control Plane (via *Statistics* and *Topology Discovery modules*) to calculate the link available bandwidth, delay, and packet loss ratio. These metrics characterize the link-state for the selection of routes.

The *Network Information Data Repository* stores the metrics calculated by the *Data Processing module*. This repository contains a set of entries that represents the source-destination pairs and the corresponding tuples of metrics. An example of an entry in this repository would be as follows: (source $= n_1$, destination $= n_2$, av_bw $= 100$ Kbps, delay $= 1.3$ ms, loss $= 0.5\%$). The link-state information is an input to the Knowledge Plane.

*4) The Knowledge Plane:* This plane learns the network behavior and employs intelligence to the computation of paths. It interacts with the Management and Control Planes for retrieving link-state information and installing the computed routes. This Plane contains the *Route Data Repository* and the RL-agent. The *Route Data Repository* contains a set of entries with information about the path for all source-destination pairs. Each entry is a tuple of source, destination, and best path. An example of an entry in this repository would be as follows: $(src = n_8,\ dst = n_{14},\ route = [n_8, n_5, n_{16}, n_{14}])$.

The RL-agent fills the *Route Data Repository* by computing the most-rewarding route for each source-destination pair. In particular, the RL-agent uses a Q-learning algorithm to learn the routing policy. The action-state value function $Q_t(S_t, A_t)$ that estimates rewards for each action-state pair determines the routing policy. The RL-agent explores all possible action-states for a source to reach its destination while updating the value function and obtaining the corresponding reward. This reward indicates the impact of executing an action $A_t$ (i.e., choosing a neighbor node as next hop) at the state $S_t$ (i.e., source node) on the available link bandwidth, delay, and packet loss ratio. Section IV defines the operation of the RL-based routing.

The knowledge Plane can be deployed either on top of the Control Plane, as is the Application Plane in the standardized SDN [40], or separately [32], [33]. Both deployments are feasible, since the Knowledge Plane communicates with the Control Plane via Northbound Interfaces (NBIs). RSIR uses a separate Knowledge Plane to avoid overloading the Control Plane with RL tasks.

*5) Flow Handling Proactive Process:* This process involves a proactive path computation and the installation of flows (see Figure 2). The *Statistics module* gathers raw data about the network and passes them on to the *Data Processing module*. This latter module obtains link-state information by processing port information and stores it in the *Network Information Data Repository*. The Knowledge Plane then receives link-state information. The RL-agent uses this information to explore and learn the best routes for all pairs of nodes. The routing decision is made available in the *Route Data Repository*. The *Flow Installation module* of the Control Plane reads the routing decision from the *Route Data Repository* on the Knowledge Plane. For each pair of nodes, the *Flow Installation module* receives the location of the source-destination hosts from the *Topology Discovery module*. It then reads the

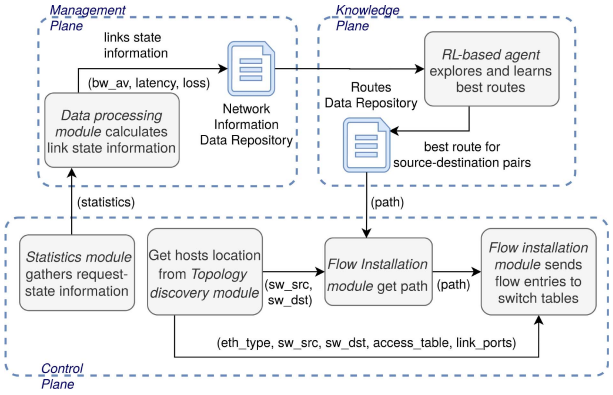



Fig. 2. Proactive flow handling process.

information about the chosen path and host locations to write the flow entries in the flow table of the corresponding switch.

*6) The Topology Change Handling Process:* This process is evoked whenever there is a change in the network topology. It involves three tasks: detection of topology changes, calculation of routes, and installation of routes. The *Topology Discovery module* performs the detection of topology changes (due to the addition or failure of either a node or a link) by periodically gathering information about the topology at each Change Detection Time ($t_{chad}$). The RL-agent uses a period, called Learning Time ($t_{lear}$), to calculate routes by running Algorithm 1 (see Section IV). The *Flow Installation module* uses another period (Installation Time ($t_{inst}$)), to update the forwarding tables for all switches on the Data Plane. The duration of $t_{inst}$ depends on the number of forwarding tables, and the number of flow entries to be updated. The total time required to recover from a topology change is, thus, $t_{chad} + t_{lear} + t_{inst}$.

## IV. RSIR Routing

The RL-agent uses the Q-learning technique to define the routes to be followed by flows. Q-learning is a model-free technique and, thus does not require knowledge about the underlying reward resulting from taking a specific action in a particular state [41]. The Q-learning technique (RL-agent) approximates the optimal action-value function $Q_t(S_t, A_t)$ named as Q-function by visiting all action-states and updating the value in consideration of a pre-defined number of independent episodes [42]. A learning episode comprises a sequence of steps taken in the transition from an initial state to a target state (i.e., a source and destination node).Each step consists of selecting and performing an action, changing the state (i.e., moving from one to another), and receiving a reward. The updated Q-function value is the underlying reward for the execution of action $A_t$ in the state $S_t$, which provides an optimal reward. Next, we provide details about the RL-agent, the RL-based routing algorithm, and its computational complexity.

### A. Reinforcement Learning Agent

*1) The Reward Function:* The reward function, defined in Equation (1), is used to find the best routing options on the basis of the three mentioned features gathered by the Knowledge Plane from the Management Plane. The reward is inversely proportional to the available link bandwidth $bwa_{link}$ and directly proportional to the link delay $d_{link}$ and the link packet loss ratio $l_{link}$. The values $\beta_1, \beta_2$ and, $\beta_3 \in [0, 1]$ are tunable parameters useful for providing a weight value for a specific metric in the calculation of the reward.

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

We normalized Equation (1) to avoid one of the link-state metrics having a greater influence than the others during the learning process of the RL-agent. Since the metrics considered in Equation (1) are in different units, the factors were normalized using the Min-Max technique [43]. This normalization technique involves re-scaling the range of the metrics to a range with values in an arbitrary interval: [*a*, *b*]. Equation (2) shows the normalized version of the reward function, where $\hat{bwa}$, $\hat{d}$, and $\hat{l}$ are the normalized values of the link available bandwidth, delay, and loss ratio, respectively.

$$\hat{R} = \beta_1 \cdot \frac{1}{\hat{bwa}} + \beta_2 \cdot \hat{d} + \beta_3 \cdot \hat{l} \tag{2}$$

Each normalized value (i.e., $\hat{bwa}$, $\hat{d}$, and $\hat{l}$) was obtained by employing Equation (3). In this equation, $x_i$ is a value to be normalized, and *X* represents the set of values used in the normalization.

$$\hat{x_i} = a + \frac{(x_i - min(X)) \cdot (b - a)}{max(X) - min(X)} \tag{3}$$

*2) The State Space (S):* Each state in the State Space corresponds to a switch on the Data Plane, and a transition from one state to another corresponds to a link connecting the two corresponding switches. Therefore, the topology of the State Space is a graph corresponding to the topology of the switches on the Data Plane. The cardinality of the set of states in the State Space is $|S = \{s_i\}| \equiv N$, where *N* is the number of switches on the Data Plane. The *Topology Discovery module* builds a topology map with nodes corresponding to states in the State Space and, then, passes that map to the RL-agent.

*3) The Action Space:* This corresponds to the set of all actions, *A*, that can be taken on the states of the State Space. For each state, $s_i \in S$, the RL-agent can take one of a set of actions; each action leads to the transition to another state $s_j \in S$. The neighboring states of a state $s_i$ correspond to the neighboring switches of that associated to the state $s_i$. Therefore, the number of potential actions when on a state, $s_i$, is the node degree of that state $dr(s_i)$ and the cardinality of *A* is $|A| \equiv \sum_{s_i \in S} dr(s_i)$.

*4) The Optimal Policy:* This policy is designed to minimize the reward value in the Q-learning routing process. In this way, the RL-agent learns to avoid links with a high delay and loss ratio as well as prioritizing links with large available bandwidth when choosing a path. The agent approximates the optimal Q-function $Q_{t+1}(S_t, A_t)$ by visiting all the pairs of action-states. It updates and stores the Q-value in the Q-table used to find the best path for a pair of nodes. The Q-value is a measure of the overall expected reward when the RL-agent is in a state $S_t$ and performs action $A_t$.

The RL-agent uses Equation (4) to update the Q-value. In Equation (4), the expression between square brackets represents the updated value which is the difference between the current estimate of the optimal Q-function $Q_t(S_t, A_t)$ for a pair of state-action $(S_t, A_t)$ and the new estimate. The updated Q-value ($Q_{t+1}$) depends on the previous value $Q_t$ that is influenced by the result of $(S_t, A_t, R_t, S_{t+1})$ and $\alpha$. $R_t$ is the reward at time $t$ and $\alpha \in [0, 1]$ being the learning rate that determines the weight of the newly gained information in relation to the previous one. A factor $\alpha = 0$ makes the RL-agent unable to learn from the latest $(S_t, A_t)$ pair, while a factor $\alpha = 1$ allows the agent to keep the learned information by considering the immediate reward $R_t$ for the pair $(S_t, A_t)$.

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha \times \left[ R_t + \min_{\mathrm{A}} Q_t(S_{t+1}, A) - Q_t(S_t, A_t) \right] \tag{4}$$

The Q-learning technique is both computationally and memory-efficient in a typical SDN routing scenario. However, if the state space and the action space are large, Q-learning can take time and require more data to find the optimal policy (i.e., converge).

*5) Exploration and Exploitation Method:* This method is used by Q-learning to find the value of the Q-function. In Q-learning, there is a trade-off between selecting the expected optimal action (exploitation) and selecting a different action in the hope it may yield a greater reward in the future (exploration) [44]. Our approach uses the $\varepsilon$-greedy exploration and exploitation method. $\varepsilon$-greedy takes an $\varepsilon \in [0, 1]$ as a tunable parameter which allows the agent to exploit with a probability $pr = \varepsilon$ and to explore with a probability $pr = 1 - \varepsilon$. Therefore, this parameter determines how much the RL-agent exploits and explores during the learning process.

The RL-agent uses Equation (5) to select the next action at a specific state. At each step, it generates a random value $x \in [0, 1]$. If $x < \varepsilon$, the agent exploits; otherwise, it explores.

$$\mathcal{A} = \begin{cases} \min_{\mathrm{A}} Q_t(S_t, A), & if\ x < \varepsilon \\ random\ action, & otherwise. \end{cases} \tag{5}$$

### *B. RL-Based Routing Algorithm*

The routing algorithm implements the learning process used to find the best paths for all the pairs of nodes on the Data Plane. Algorithm 1 receives as input the learning rate $\alpha$, the parameter $\varepsilon$ (see Equation (5)), the number of learning episodes, all pairs of source-destination nodes, and link-state information. The output is the set of best-rewarding routing paths for all pairs of nodes, given the state of the links. The path is formed by the state-action pairs with the lowest values in the Q-table.

Our routing algorithm finds the most-rewarding path for every pair of nodes in the network. For each pair (*src*, *dst*), the algorithm executes a Q-learning process by following the steps defined in Lines 1 to 13. The RL-agent initializes the Q-table $Q(\mathcal{S}, \mathcal{A})$ with zero (Line 2). The Q-learning process

**Algorithm 1:** Q-Learning Routing

**Input** :
Learning rate: $\alpha$
Exploration and exploitation parameter: $\varepsilon$
Number of learning episodes: *n*
All pair of nodes (*src*, *dst*): *Plist*
Network link-state

1 **foreach** *(src, dst)* $\in$ *Plist* **do**
2  Initialize *Q*: $Q(\mathcal{S}, \mathcal{A}) = 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
3  **for** *episode* $\leftarrow 1$ **to** *n* **do**
4  Start in state $S_t = src \in \mathcal{S}$;
5  **while** $S_{t+1}$ *is not dst* **do**
6  Select $A_t$ for $S_t$ with policy derived from Q using $\varepsilon$-greedy exploration and exploitation method;
7  $R_{t+1} \leftarrow R(S_t, A_t)$ `// Agent gets the reward from network link-state and observes new state` $S_{t+1}$;
8  $Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha \cdot \left[ R_t + \min_{\mathrm{A}} Q_t(S_{t+1}, A) - Q_t(S_t, A_t) \right]$ `// Update Q-function Eq. 4;`
9  $S_t \leftarrow S_{t+1}$ `// Move to the next state;`
10  **end**
11  **end**
12  Take Q table and find the path between *src* and *dst* with state-action pairs that achieved the lowest Q-values
13 **end**
14 Store the set of paths for all pair of nodes in the network in the *Routs Data Repository*

then begins with node *src* as the initial state (Line 4). The RL-agent goes through learning episodes to minimize the Q-value by using the reward function. The agent attributes low Q-values to paths formed by links with large available bandwidth, short delays, and small packet loss. Then, the algorithm goes through learning episodes (Line 3) until $S_t$ becomes the end state (i.e., node *dst*, Lines 5 to 9). First, the RL-agent then selects the next node from the Action Space (selects $A_t$ for $S_t$); the RL-agent chooses a node from the neighboring nodes of the current one as next-hop by using the $\varepsilon$-greedy exploration and exploitation method (Line 6). Next, the RL-agent uses the network link-state information and the state $S_t$ to calculate the reward associated with the action $A_t$, and observes the new state $S_{t+1}$. The reward is computed by using Equation (1) (Line 7). After that, and considering the learning rate, initial considerations, reward, and new state, the RL-agent tunes the values of the Q-function by using Equation (4) (Line 8). It then moves to the new state (Line 9), the episode ends, and a new episode starts.

Finally, after the RL-agent has completed a transition, it uses the resulting Q-table to compute the most-rewarding path between the *src* and *dst* nodes, based on the state-action pairs

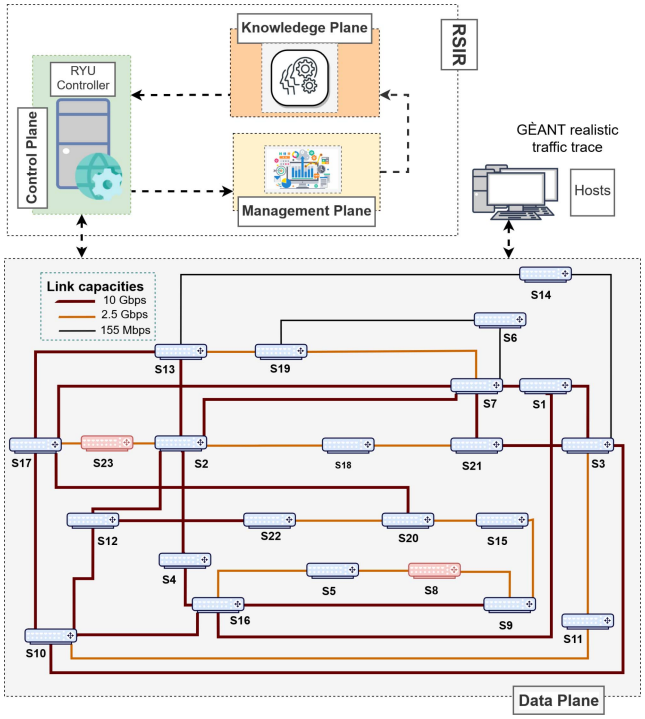



Fig. 3. Test environment.

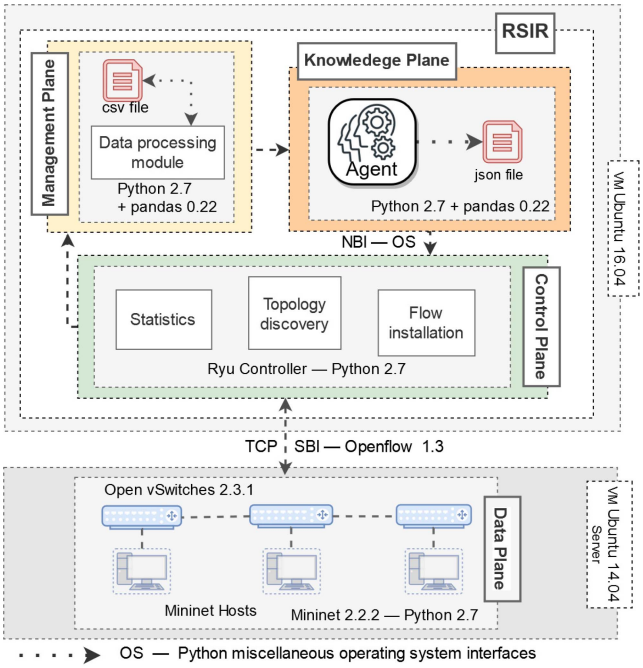



Fig. 4. Prototype of RSIR.

that achieved the lowest Q-values (Line 12). Once the RL-agent finds the best path for all pairs of source-destination nodes, it stores them in the *Route Data Repository* (Line 14). The *Flow Installation Module* then retrieves these best paths and installs them in the routing table of the switches.

The worst-case complexity of Algorithm 1 is derived next.

*Lemma 1:* In Q-learning algorithms with a state space topology with a linear upper action bound $b \in \aleph_0 \iff e \leq bn$ for all $n \in \aleph_0$, where $e$ is the cardinality of the action space and $n$ the cardinality of the state space, the worst-case complexity is $O(n^2)$

*Proof:* see Page 103 in [45] ■

*Theorem 1:* The worst case complexity of Algorithm 1 is $O(N^2)$.

*Proof:* $dr(s_i) \leq N-1$, for $N \in \aleph_0 \implies |A| \leq N^2$ ■

The worst-case complexity of the Dijkstra's algorithm is also $O(N^2)$ and its heap implementation $O(Nlog\ N)$ [46]. In Section V, it will be shown that RSIR routing has low overhead as well as convergence time.

## V. Evaluation

This section presents the evaluation of RSIR. Sections V-A and V-B show the test environment and the prototype of RSIR, respectively, while Sections V-C and V-D discuss the gathering of performance metrics and traffic generation, respectively. Section V-E presents the set up of the learning parameters, and Section V-F discusses the results.

### A. Test Environment

Fig. 3 presents the test environment used for the evaluation of RSIR. It includes a topology mirroring the GÉANT topology [47] which is the European data network for the research and educational community. GÉANT has 23 nodes and 37 links. In tests, link capacity distribution follows the 2004 GÉANT topology which has 50% of the links with 10 Gbps, 40% with 2.5 Gbps, and 1% with 155 Mbps.

We built and deployed the GÉANT topology in Mininet 2.2.2 [48] by using a Python script. As Mininet is limited to the resources of its host machine, we scaled the 10 Gpbs, 2.5 Gpbs, and 155 Mbps link capacities of GÉANT to 100 Mbps, 25 Mbps, and 1.55 Mbps, respectively. The traffic load was also scaled to the same proportion. In our deployment, each switch had a host that forwarded and received traffic. We deployed the Data Plane over an Ubuntu Server 14.04 Virtual Machine (VM) with 8 GB RAM and used Mininet with Open vSwitches 2.3.1 and hosts.

### B. Prototype

Fig. 4 presents the RSIR prototype. In the Control Plane, we used a Python-based Ryu controller [49]. The *Statistics, Topology Discovery and Flow Installation modules* were developed and deployed using the Ryu Application Program Interface (API), which allows querying and retrieving statistics and features from the switches. For the Management and Knowledge Planes, we developed the *Data Processing module* and the RL-agent by using Python 2.7 with Pandas 0.22 library [50]. The Pandas library provides high-performance, easy-to-use data structures, and data analysis tools for Python. We used Comma-Separated Values (CSV) and JavaScript Object Notation (JSON) files to store the link-state information provided by the Management Plane and the routing paths calculated by the RL-agent, respectively. The RSIR prototype, as well as all test scripts, are available in [51].

The Control, Management, and Knowledge Planes were run on an Ubuntu 16.04 VM with a Core i5-4690 processor and 10 GB RAM. The Data Plane ran on an Ubuntu Server 14.04 VM with a Core i5-4690 processor and 4 GB RAM. The VMs used for this prototype were hosted by an Ubuntu Desktop 16.04 with an Intel Core i5-4690 and 16 GB RAM. These VMs communicated using the Transmission Control Protocol (TCP). For the communication between the Control and the Data Plane, we used Openflow1.3 as the protocol between since it is the *de-facto* protocol used as SBI in SDN [15].

### C. Performance Metrics

According to Table II, the performance metrics most common used to evaluate routing proposals are throughput, loss ratio, and delay. In addition to these metrics, we compared the stretch of the paths given by RSIR with those given by the Dijkstra's algorithm using different edge weights. The stretch compares the length of an actual path to the theoretical shortest path [52]. For computing the shortest path, we developed an application that executes the Dijkstra's algorithm with equal edge weight values on the SDN controller.

We computed the link throughput and loss using the number of packets that passed through the switch port connected to the link. At each port, the SDN controller periodically samples the number of bytes transmitted and received. By comparing the retrieved values at two different instants, it is possible to discover the instantaneous throughput. After sending a *request-stats* to the Data Plane, at time $t_1$, a reply message is received containing the number of bytes received, $b_{t_1}$. Then, after a period *p*, another *request-stats* message is sent, and the number of bytes received $b_{t_2}$ is retrieved from its reply message. The expression $used\_bw = [(b_{t_2} - b_{t_1})/p]$ gives the instantaneous throughput, where *p* is the duration of the sampling interval.

The computation of the loss ratio uses the *request-stats* values in the reply messages; with the expression $loss = (btx_{t_1} - brx_{t_2})/btx_{t_1}$ giving the instantaneous loss ratio. After sending a *request-stats* to the Data Plane, at time $t_1$, a reply message is received containing the number of transmitted bytes, $btx_{t_1}$. Then, after a period *p*, another *request-stats* message is sent, with the reply message received containing the number of received bytes, $brx_{t_2}$.

We computed the instantaneous delay by following the process described in [53], which uses the messages of the Link Layer Discovery Protocol (LLDP) [54] and the OpenFlow messages [55]. The SDN controller, $c_0$, sends an LLDP message which goes through the path $c_0 - s_i - s_j - c_0$, and returns to $c_0$; with $s_i$ and $s_j$ being switches connected by the link $(s_i, s_j)$. The time elapsed between the transmission and reception of the LLDP message is the difference between the timestamp values in this message ($d_{lldp_{cij}}$). The time taken by the message to go from $c_0$ to the $s_i$ port ($c_0$-$s_i$) is estimated as half the time that elapsed between the transmission and reception of the OpenFlow *echo_request* and *echo_reply* messages sent by $c_0$ to $s_i$. A similar procedure is used to estimate the time elapsed by the message to go from $s_j$ to $c_0$. The instantaneous delay in the link $(s_i, s_j)$, $d_{si-sj}$, is then computed as $d_{si-sj} = d_{lldp_{cij}} - d_{c_0-s_i} - d_{c_0-s_j}$.

In a period of 10*s*, the *Statistics module* collects the statistics; the *Data Processing module* computes the metrics that characterize the link-state, the RL-agent computes the paths according to Algorithm 1; and the *Flow Installation Module* installs the computed paths in the flow tables of the switches. The definition of the duration of the interval followed the guidelines in [56].

The RL-based routing is compared to that suggested by the different variations of the Dijkstra's algorithm using the instantaneous delay ($\text{Dijkstra}_{\text{delay}}$), instantaneous loss ($\text{Dijkstra}_{\text{loss}}$), and link available bandwidth ($\text{Dijkstra}_{\text{bw}}$) as edge weights. Moreover, we compare the results of RSIR with the routing derived by the Dijkstra's algorithm that uses the value given by Equation (1) ($\text{Dijkstra}_{\text{comp}}$) as edge weight. Routing determined by these variations of the Dijkstra's algorithm was subject to the same traffic scenario used for RSIR and executed as an application on the SDN controller. In this way, these algorithms could be compared under the same conditions.

### D. Traffic Generation

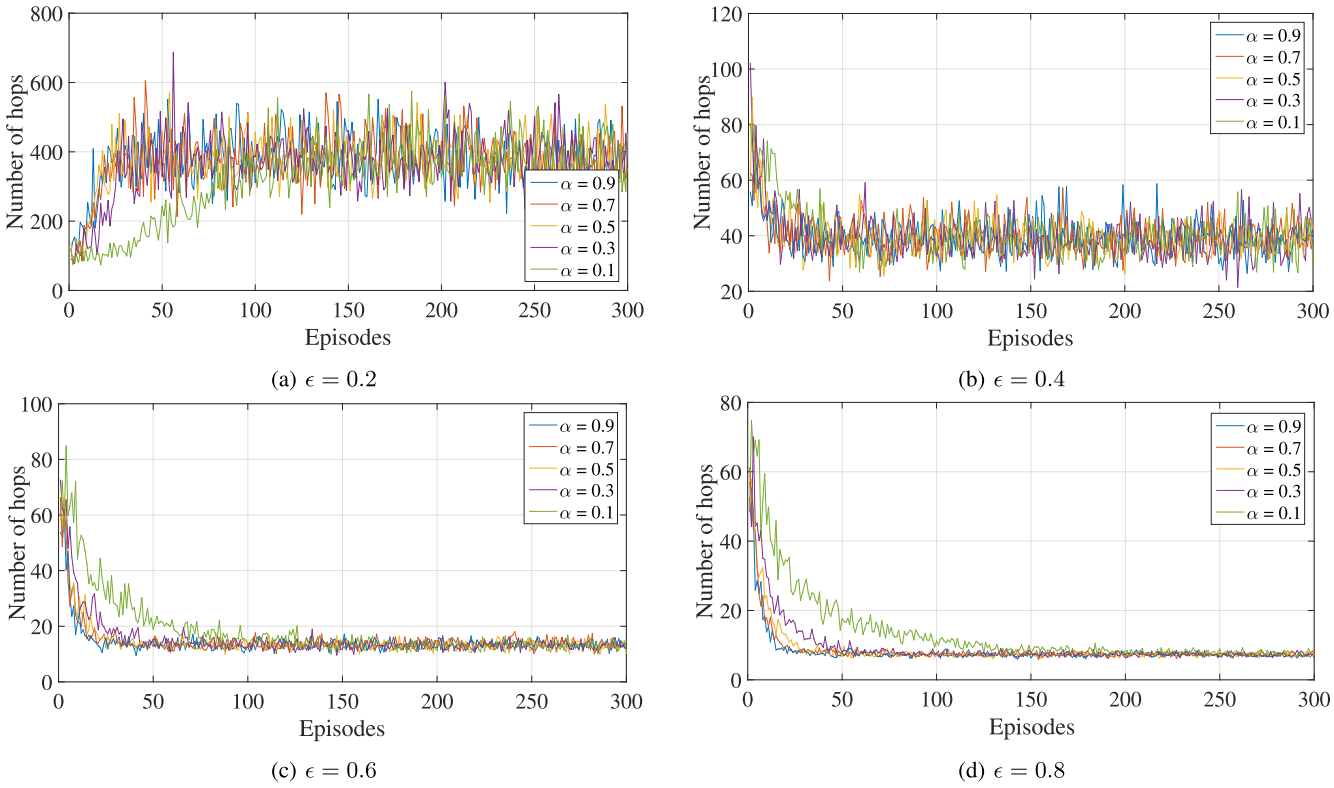
The tool *iperf3* [57] was used to generate traffic in the Mininet-based emulation; scripts to run clients and servers *iperf3* on the hosts were developed. User Datagram Protocol (UDP) traffic was generated since *iperf3* allows specifying a target transmission rate per connection. The experiment involved the complete execution of the *iperf3* scripts that generated traffic specified by a traffic matrix. We used sixteen publicly available intra-domain traffic matrices [58] for the GÉANT topology to generate the traffic between pairs of nodes. The matrix values offered the traffic of different pairs at different times of the day for 4 months in 2006. We tagged the employed traffic matrices with different hours of the day and defined the peak hours as having high traffic intensity (i.e., from 7:00h to 13:00h) since the traffic matrices are anonymous.

### E. Learning Parameters Setup

One important aspect when using an RL-solution is the determination of the values for the learning rate $\alpha$ and exploration $\epsilon$ (Equations 4 and 5). We employed the number of hops along a path between a pair of source and destination nodes as a parameter for comparison.

We ran the Q-learning algorithm (see Algorithm 1) to find the potential paths between a pair of nodes by varying $\alpha$ and $\epsilon$. The chosen target pair was h23 (switch 23) and h8 (switch 8) since this pair is one of those further apart (see red switches in Fig. 3). The shortest path found by the Dijkstra's algorithm had 5 hops when all the links had the same cost. For each $\epsilon$ value (i,e., 0.8, 0.6, 0.4, and 0.2), five $\alpha$ values (*i,e.,* 0.9, 0.7, 0.5, 0.3, and 0.1) were tested, with 300 episodes per test.

The evolution of the convergence to the shortest path as a function of $\alpha$ and $\epsilon$ needs to be assessed. Fig. 5 presents the number of hops of the paths found by the RL-agent for different values of $\alpha$ and $\epsilon$. Figs. 5(a) and 5(b) show that when the $\epsilon$ value was close to 0, the RL-agent explored random actions from the Action Space resulting in paths with many hops.




(a) $\epsilon = 0.2$ (b) $\epsilon = 0.4$

(c) $\epsilon = 0.6$ (d) $\epsilon = 0.8$

Fig. 5. Number of hops for different learning parameters $(\epsilon, \alpha)$.

When $\epsilon$ values were close to 1 (Figs. 5(c) and 5(d)), the RL-agent tended to exploit the knowledge previously gathered instead of randomly exploring the Action Space. For $\epsilon = 0.6$ and $\epsilon = 0.8$, the RL-agent exploited, with a probability of 0.6 and 0.8, the best-known action during the learning process. With a probability of exploitation value close to 1, the RL-agent tended to find shorter paths along the sequence of episodes, eventually finding the shortest path. For $\alpha = 0.9$ and $\alpha = 0.7$, the RL-agent retained the information learned from recent rewards. Based on these results, the most appropriate values for the RL-agent parameters of RSIR were $\alpha = 0.9$ and $\epsilon = 0.8$. With these values (see Fig. 5(d)), the number of hops of the path converged quickly to the shortest path.

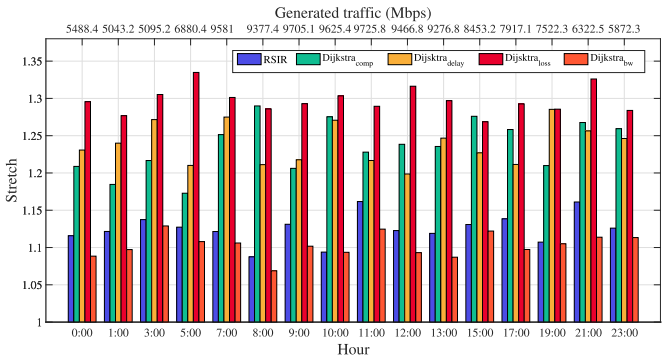



Fig. 6. Mean stretch throughout the day.

### *F. Results and Analysis*

In this section, we compare the mean stretch values average for all chosen paths, the mean link delay, loss ratio, and throughput averaged over all the links in the network, (i.e., the mean of a mean link measure. For instance if the target metric is delay, it is the mean of all mean link delay values) given by RSIR to those given by variations in the Dijkstra's algorithm, namely, $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, $\text{Dijkstra}_{\text{bw}}$, and $\text{Dijkstra}_{\text{comp}}$. For each hour of the day, we collected and plotted the mean values. The figures also show the traffic generated per hour.

Fig. 6 shows the mean stretch calculated for all the paths found by RSIR and the variations in the Dijkstra's algorithm. To compute the stretch, we compare the path length with the shortest path given by the Dijkstra's algorithm with equal edge weights, which gave the shortest path as that with the minimum number of hops. The results showed that the paths chosen by RSIR have smaller stretch values than those produced by $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, and $\text{Dijkstra}_{\text{comp}}$. Indeed, RSIR chooses a larger number of shorter paths than did the three variations of the Dijkstra's algorithm. RSIR indicated paths with stretch values 14%, 16%, and 15% stretch smaller than those obtained by the $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, and $\text{Dijkstra}_{\text{comp}}$ algorithms, respectively. Results also showed that RSIR indicated paths with stretch values slightly higher (<3%) than those produced by $\text{Dijkstra}_{\text{bw}}$.

Fig. 7 depicts the mean link delay for the results of the RSIR and those of the variations of the Dijkstra's algorithm. The mean link delay values produced by RSIR are, on average, 35%, 50%, and 8% and, at most, 70%, 85%, and 30% lower than those given by the $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, and $\text{Dijkstra}_{\text{bw}}$ algorithms, respectively. The mean delay

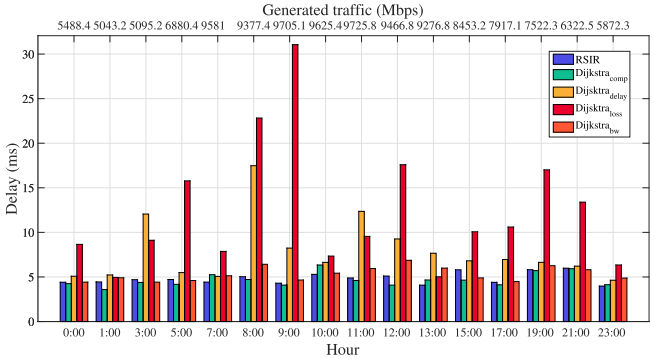



Fig. 7. Mean delay throughout of the day.

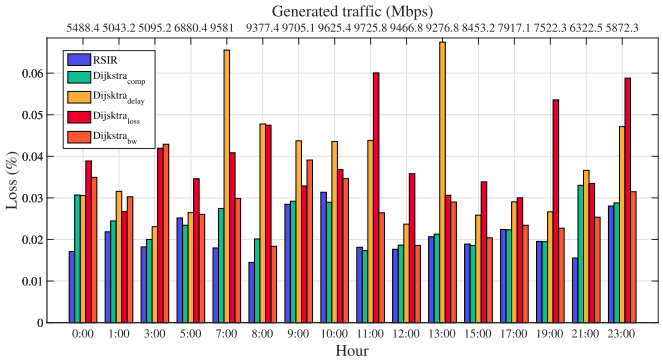



Fig. 8. Mean loss throughout the day.

values produced by RSIR are lower than those given by $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, and $\text{Dijkstra}_{\text{bw}}$ since RSIR usually chooses shorter and less congested paths than those chosen by the other three variations of the Dijkstra's algorithm.

Fig. 8 depicts the mean link loss. The mean loss values produced by RSIR are, on average, 30% and, at most, 65% lower than the loss ratio given by the four variations of the Dijkstra's algorithms. The mean loss values produced by RSIR are lower than those provided by $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, $\text{Dijkstra}_{\text{bw}}$, and $\text{Dijkstra}_{\text{comp}}$ since RSIR chooses shorter paths than do the Dijkstra variations most of the time. Fig. 7 and Fig. 8 also show that the mean link delay values produced by RSIR are slightly higher ($<3\%$) than those given by $\text{Dijkstra}_{\text{comp}}$, but the mean loss values produced by RSIR are, on average, 10% and, at most, 50% lower than the loss ratio given by $\text{Dijkstra}_{\text{comp}}$. The Dijkstra's algorithm variations select routes usually longer and use, more frequently, low capacity links, leading to traffic concentration and congestion on these links.

Fig. 9 shows the mean link throughput along the day. RSIR produced a broader distribution of the flows over the network and, therefore, used a higher number of paths less utilized than do the $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, $\text{Dijkstra}_{\text{bw}}$, and $\text{Dijkstra}_{\text{comp}}$. The link throughput was consequently lower than those produced by the four Dijkstra variations with, although the mean delay and loss were lower, as previously discussed. The link throughput was on average 4%, 21%, 3%, 17% and, at most, 12%, 27%, 15%, and 26% lower than those produced by the $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, $\text{Dijkstra}_{\text{bw}}$, and $\text{Dijkstra}_{\text{comp}}$ algorithms, respectively.

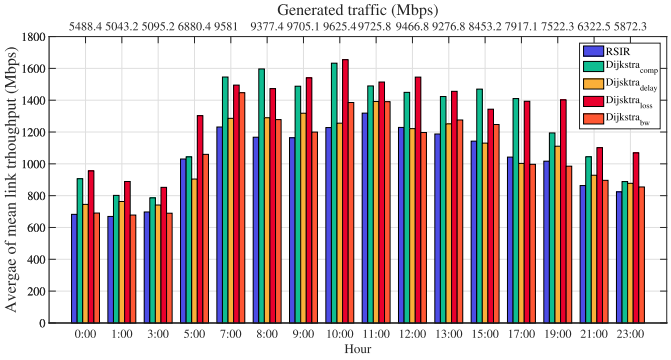



Fig. 9. Mean link throughput along the day.

The RL-agent computed all the routes for the GÉANT network in 7.49 s, while the centralized variations of Dijkstra's algorithm calculated all the routes for the same network in 5s; the difference (1.51 s) is about 30% of the time the Dijkstra's algorithm take but yet still a non-significant absolute value.

### G. Topology Change Analysis

We assessed the reaction time for topological changes by RSIR and compared this to those from the RIP and OSPF protocols. The topology change handling process of RSIR involved detecting the topology changes, calculating new routes, and installing them (Section III-B6). In the case study, we set $t_{chad}$ to 1 s. A lower value could have been set, but such value would increase the traffic intensity between the switches and the controller. The RL-agent computed all the routes for the GÉANT network in 7.49 s (the average execution time of Algorithm 1) and the *Flow Installation* module spent on average 1.6 s to update the flow entries (22) at each switch (23 in total). Thus, RSIR took on average $t_{chad}+t_{lear}+t_{inst} = 10.6s$ to handle a change in the network topology.

On the other hand, RIP would typically take 30 s to handle a topological change. The lower response time of RSIR was due to the adoption of a centralized controller with a global view of the network, as well as the adoption of a link-state routing approach. Moreover, the OSPF reaction to topological changes depended mainly on the duration of the following intervals: hello-interval, dead-interval, and spf-delay. The hello-interval defines how often Hello Packets are transmitted and usually takes 10 s (by default), ranging from 1 s to 255 s. These packets allow routers running OSPF to recognize their neighboring routers. The expiration of the dead-interval indicates that a neighboring router is down, and it takes by definition the time equivalent to four hello intervals. The spf-delay defines the time elapsed between the detection of a topological change and the beginning of the calculation of new routes. This value is, by default, 5 s. Considering a minimum hello-interval $= 1$ s and an spf-delay $= 1$ s, in the GÉANT network, OSPF reacts to topological changes in approximately 9 s (5 s + the average execution time of the Dijkstra's algorithm). This is approximately the same time as the RSIR takes.

## VI. Conclusion and Future Work

This article has introduced RSIR, an approach based on RL for intelligent and efficient routing in SDN. The RSIR

algorithm considers metrics related to the state of the network links and explores, learns, and exploits optimal routes, even under traffic fluctuation. Comparison of the results given by RSIR to those the Dijkstra's algorithm using different edge weights have shown that RSIR outperforms these algorithms. This is due to the fact that RSIR produces a larger number of shorter paths than do the other algorithms, and this avoids traffic concentration and congestion. The variations of the Dijkstra's algorithm, on the other hand, produce longer paths, which include low capacity links. The $\text{Dijkstra}_{\text{loss}}$ concentrates traffic on a smaller number of paths and consequently produces the largest mean delay and loss values. The minimization of the reward function (Equation (1)) allows the RL-based agent to learn, explore, and exploit the best paths for delivering the offered load.

For future work, we plan to evaluate the impact of RSIR on the throughput at the application layer as well as explore Deep Reinforcement Learning (DRL) to enhance RSIR decisions, especially for large networks. Moreover, we intend to take advantage of traffic predictions in the selection of paths.

## REFERENCES


[1] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. Boston, MA, USA: Pearson, 2012.

[2] R. A. Guerin, A. Orda, and D. Williams, "QoS routing mechanisms and OSPF extensions," in *Proc. IEEE Global Telecommun. Conf.*, vol. 3. Phoenix, AZ, USA, 1997, pp. 1903–1908.

[3] A. A. Ajani, B. J. Ojuolape, A. A. Ahmed, T. Aduragba, and M. Balogun, "Comparative performance evaluation of open shortest path first, OSPF and routing information protocol, RIP in network link failure and recovery cases," in *Proc. Int. Conf. Electro Technol. Nat. Develop. (NIGERCON)*, Owerri, Nigeria, 2017, pp. 280–288.

[4] J. S. Marcus. (2014). *The Economic Impact of Internet Traffic Growth on Network Operators*. [Online]. Available: https://ssrn.com/abstract=2531782

[5] B. Mao *et al.*, "A tensor based deep learning technique for intelligent packet routing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Singapore, 2017, pp. 1–6.

[6] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[7] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *J. Internet Serv. Appl.*, vol. 9, no. 1, p. 16, 2018.

[8] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proc. 16th ACM Workshop Hot Topics Netw.*, 2017, pp. 185–191.

[9] A. Rego, S. Sendra, J. M. Jimenez, and J. Lloret, "OSPF routing protocol performance in software defined networks," in *Proc. 4th Int. Conf. Softw. Defined Syst. (SDS)*, Valencia, Spain, 2017, pp. 131–136.

[10] A. A. Khan, M. Hussain, M. Zafrullah, and M. S. Zia, "A convergence time optimization paradigm for OSPF based networks through SDN SPF protocol computer communications and networks (CCN)/delay tolerant networks," in *Proc. Int. Conf. Future Netw. Distrib. Syst. (ICFNDS)*, 2017, p. 43.

[11] D. Gopi, S. Cheng, and R. Huck, "Comparative analysis of SDN and conventional networks using routing protocols," in *Proc. Int. Conf. Comput. Inf. Telecommun. Syst. (CITS)*, Dalian, China, 2017, pp. 108–112.

[12] J. Park, J. Hwang, and K. Yeom, "NSAF: An approach for ensuring application-aware routing based on network QoS of applications in SDN," *Mobile Inf. Syst.*, vol. 2019, Apr. 2019, Art. no. 3971598.

[13] C. Lin, K. Wang, and G. Deng, "A QoS-aware routing in SDN hybrid networks," *Procedia Comput. Sci.*, vol. 110, pp. 242–249, Jul. 2017.

[14] A. Shirmarz and A. Ghaffari, "An adaptive greedy flow routing algorithm for performance improvement in software-defined network," *Int. J. Numer. Model. Electron. Netw. Devices Fields*, vol. 33, no. 1, p. e2676, 2020.

[15] Y.-C. Wang and S.-Y. You, "An efficient route management framework for load balance and overhead reduction in SDN-based data center networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1422–1434, Dec. 2018.

[16] C.-C. Chuang, Y.-J. Yu, and A.-C. Pang, "Flow-aware routing and forwarding for SDN scalability in wireless data centers," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1676–1691, Dec. 2018.

[17] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," *J. Netw. Comput. Appl.*, vol. 67, pp. 1–25, May 2016.

[18] R. Arroyo-Valles, R. Alaiz-Rodriguez, A. Guerrero-Curieses, and J. Cid-Sueiro, "Q-probabilistic routing in wireless sensor networks," in *Proc. Int. Conf. Intell. Sens. Sens. Netw. Inf. (ISSNIP)*, Melbourne, VIC, Australia, 2007, pp. 1–6.

[19] A. Forster and A. L. Murphy, "FROMS: Feedback routing for optimizing multiple sinks in WSN with reinforcement learning," in *Proc. Int. Conf. Intell. Sens. Sens. Netw. Inf. (ISSNIP)*, Melbourne, VIC, Australia, 2007, pp. 371–376.

[20] N. Kato *et al.*, "The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 146–153, Jun. 2017.

[21] Z. M. Fadlullah *et al.*, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2432–2455, 4th Quart., 2017.

[22] S. Chaudhary and R. Johari, "ORuML: Optimized routing in wireless networks using machine learning," *Int. J. Commun. Syst.*, vol. 33, no. 11, p. e4394, 2020.

[23] A. Serhani, N. Naja, and A. Jamali, "AQ-routing: Mobility-, stability-aware adaptive routing protocol for data routing in MANET–IoT systems," *Clust. Comput.*, vol. 23, no. 1, pp. 13–27, 2020.

[24] C. Wang, L. Zhang, Z. Li, and C. Jiang, "SDCoR: Software defined cognitive routing for Internet of vehicles," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3513–3520, Oct. 2018.

[25] B. Mao *et al.*, "Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1946–1960, Nov. 2017.

[26] L. Yanjun, L. Xiaobo, and Y. Osamu, "Traffic engineering framework with machine learning based meta-layer in software-defined networks," in *Proc. 4th IEEE Int. Conf. Netw. Infrastruct. Digit. Content (IC-NIDC)*, Beijing, China, 2014, pp. 121–125.

[27] S. Sendra, A. Rego, J. Lloret, J. M. Jimenez, and O. Romero, "Including artificial intelligence in a routing protocol using software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC) Workshops*, Paris, France, 2017, pp. 670–674.

[28] S.-C. 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 *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*, San Francisco, CA, USA, 2016, pp. 25–33.

[29] R. Alvizu, S. Troia, G. Maier, and A. Pattavina, "Matheuristic with machine-learning-based prediction for software-defined mobile metro-core networks," *J. Opt. Commun. Netw.*, vol. 9, no. 9, pp. D19–D30, Sep. 2017.

[30] I. Martín *et al.*, "Machine learning-based routing and wavelength assignment in software-defined optical networks," *IEEE Trans. Netw. Services Manag.*, vol. 16, no. 3, pp. 871–883, Sep. 2019.

[31] G. B. Figueiredo, N. L. S. D. Fonseca, and J. A. S. Monteiro, "A minimum interference routing algorithm with reduced computational complexity," *Comput. Netw.*, vol. 50, no. 11, pp. 1710–1732, 2006. [Online]. Available: https://doi.org/10.1016/j.comnet.2005.06.015

[32] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the Internet," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2003, pp. 3–10.

[33] A. Mestres *et al.*, "Knowledge-defined networking," *SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 2–10, 2017.

[34] S. Ayoubi *et al.*, "Machine learning for cognitive network management," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 158–165, Jan. 2018.

[35] J. Hyun and J. W.-K. Hong, "Towards knowledge-defined networking using in-band network telemetry," in *Proc. Asia–Pac. Netw. Oper. Manag. Symp. (APNOMS)*, Taipei, Taiwan, 2017, pp. 54–57.

[36] S. Yan, A. Aguado, Y. Ou, R. Wang, R. Nejabati, and D. Simeonidou, "Multilayer network analytics with SDN-based monitoring framework," *J. Opt. Commun. Netw.*, vol. 9, no. 2, pp. A271–A279, Feb. 2017.

[37] A. I. Montoya-Munoz, D. M. Casas-Velasco, F. Estrada-Solano, A. Ordonez, and O. M. C. Rendon, "A YANG model for a vertical SDN management plane," in *Proc. Colombian Conf. Commun. Comput. (COLCOM)*, Cartagena, Colombia, 2017, pp. 1–6.

[38] F. Estrada-Solano, A. Ordonez, L. Z. Granville, and O. M. C. Rendon, "A framework for SDN integrated management based on a cim model and a vertical management plane," *Comput. Commun.*, vol. 102, pp. 150–164, Apr. 2017.
[39] S. Achleitner, N. Bartolini, T. He, T. La Porta, and D. Z. Tootaghaj, "Fast network configuration in software defined networking," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1249–1263, Dec. 2018.
[40] "SDN architecture 1.1, ONF TR-521," in *Architectural Framework: ONF*, ONF, Menlo Park, CA, USA, 2016.
[41] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE Access*, vol. 7, pp. 55916–55950, 2019.
[42] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
[43] L. Al Shalabi and Z. Shaaban, "Normalization as a preprocessing engine for data mining and the approach of preference matrix," in *Proc. Int. Conf. Depend. Comput. Syst. (DepCoS)*, Szklarska Poreba, Poland, 2006, pp. 207–214.
[44] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for Q-learning in random stochastic mazes," in *Proc. Symp. Comput. Intell. (SSCI)*, Athens, Greece, 2016, pp. 1–8.
[45] S. Koenig and R. G. Simmons, "Complexity analysis of real-time reinforcement learning," in *Proc. AAAI 11th Nat. Conf. Artif. Intell.*, 1993, pp. 99–107.
[46] D. Fan and P. Shi, "Improvement of Dijkstra's algorithm and its application in route planning," in *Proc. 7th Int. Conf. Fuzzy Syst. Knowl. Discov.*, vol. 4. Yantai, China, 2010, pp. 1901–1904.
[47] P. T. Kirstein, "European international academic networking: A 20 year perspective," in *Proc. TERENA Netw. Conf.*, 2004, pp. 1–18.
[48] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Proc. Colombian Conf. Commun. Comput. (COLCOM)*, Bogota, Colombia, 2014, pp. 1–6.
[49] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu, and E. C. Popovici, "A comparison between several software defined networking controllers," in *Proc. 12th Int. Conf. Telecommun. Mod. Satellite Cable Broadcast. Serv. (TELSIKS)*, Nis, Serbia, 2015, pp. 223–226.
[50] W. McKinney, *Data Structures for Statistical Computing in Python*, S. van der Walt and J. Millman, Eds. Austin, TX, USA: SciPy, 2010, pp. 51–56.
[51] D. M. Casas-Velasco, O. M. Caicedo, and N. L. S. Da Fonseca. (2020). *RSIR: Reinforcement Learning and SDN Intelligent Routing*. [Online]. Available: https://github.com/danielaCasasv/RSIR-Reinforcement-Learning-and-SDN-Intelligent-Routing
[52] C. Werle, S. Mies, and M. Zitterbart, "On benchmarking routing protocols," in *Proc. 17th IEEE Int. Conf. Netw. (ICON)*, Singapore, 2011, pp. 305–310.
[53] L. Liao and V. C. Leung, "LLDP based link latency monitoring in software defined networks," in *Proc. IEEE 12th Int. Conf. Netw. Serv. Manag. (CNSM)*, Montreal, QC, Canada, 2016, pp. 330–335.
[54] Z. Shu *et al.*, "Traffic engineering in software-defined networking: Measurement and management," *IEEE Access*, vol. 4, pp. 3246–3256, 2016.
[55] *Ryu Application API*, Ryu Project Team, Tokyo, Japan. Accessed: Jun. 16, 2016. [Online]. Available: https://ryu.readthedocs.io/en/latest/ryu_app_api.html
[56] E. F. Castillo, L. Z. Granville, A. Ordonez, and O. M. C. Rendon, "IPro: An approach for intelligent SDN monitoring," *Comput. Netw.*, vol. 170, Apr. 2020, Art. no. 107108.
[57] S. Srivastava, S. Anmulwar, A. M. Sapkal, T. Batra, A. K. Gupta, and V. Kumar, "Comparative study of various traffic generator tools," in *Proc. Recent Adv. Eng. Comput. Sci. (RAECS)*, Chandigarh, India, Mar. 2014, pp. 1–6.
[58] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, 2006.

**Daniela M. Casas-Velasco** (Graduate Student Member) received the bachelor's degree in electronics and telecommunications engineering from the University of Cauca, Popayán, Colombia, in 2017, and the M.Sc. degree in computing science from the University of Campinas, Campinas, Brazil, in 2020, where she is currently pursuing the Ph.D. degree in computer science. Her research interests include network management, SDN, wireless communications, machine learning, and related works to telecommunication.

**Oscar Mauricio Caicedo Rendon** (Senior Member, IEEE) received the bachelor's degree in electronics and telecommunications engineering and the M.Sc. degree in telematics engineering from the University of Cauca, Popayán, Colombia, in 2001 and 2006, respectively, and the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2015. He is currently a Full Professor with the Department of Telematics, University of Cauca, where he is a Member with the Telematics Engineering Group. His research interests include network and service management, network virtualization, and software-defined networking.

**Nelson L. S. da Fonseca** (Senior Member, IEEE) received the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1994. He is currently a Full Professor with the Institute of Computing, State University of Campinas, Campinas, Brazil. He has authored or coauthored more than 400 papers and supervised more than 70 graduate students. He is currently the Vice President with the Technical and Educational Activities of the IEEE Communications Society (ComSoc). He served as the ComSoc Vice President with the Publications and Member Relations, and the Director with the Conference Development, Latin America Region, and On-Line Services. He was a Recipient of the 2012 IEEE ComSoc Joseph LoCicero Award for Exemplary Service to Publications, the Medal of the Chancellor of the University of Pisa in 2007, and the Elsevier *Computer Network* Journal Editor of Year 2001 Award. He is the Former Editor-in-Chief of *IEEE Communications Surveys and Tutorials*.