Routing based on Reinforcement Learning for Software-Defined Networking

Student: Daniela Maria Casas Velasco[1]
Advisor: Nelson Luis Saldanha da Fonseca[1]
Co-advisor: Oscar Mauricio Caicedo Rendón[2]

[1]Institute of Computing – Universidade Estadual de Campinas – Brazil
[2]Department of Telematics – Universidad del Cauca – Colombia

danielac@lrc.ic.unicamp.br, omcaicedo@unicauca.edu.co, nfonseca@ic.unicamp.br

Abstract. *Traditional routing protocols employ limited information to make routing decisions, leading to slow adaptation to traffic variability and restricted support to applications quality of service requirements. This paper introduces the work developed in the MSc. thesis entitled "Routing based on Reinforcement Learning for Software-Defined Networking", which defines routing approaches based on (deep) reinforcement learning. The results show that our solutions surpass routing algorithms based on Dijkstra as well as they are practical and feasible solutions for routing in Software-Defined Networking.*

Resumo. *Os protocolos de roteamento tradicionais empregam informações limitadas para tomar decisões de roteamento, levando a uma adaptação lenta à variabilidade do tráfego e ao suporte restrito aos requisitos de qualidade de serviço das aplicações. Este artigo apresenta o trabalho desenvolvido na tese de mestrado intitulada "Roteamento baseado em Aprendizado por Reforço para Redes Definidas por Software", que define duas abordagens de roteamento baseadas em aprendizagem (profunda) por reforço. Os resultados mostram que nossas soluções superam o desempenho dos algoritmos de roteamento baseados em Dijkstra bem como são soluções práticas e viáveis para roteamento em Redes Definidas por Software.*

1. Introduction

Routing is a fundamental network mechanism that determines the path taken by packets from a source to a destination node. In the Internet, traditional routing protocols have successfully delivered best-effort traffic for the past decades. However, such protocols use limited information to make routing decisions, leading to slow adaptation to dynamic traffic changes. Software-Defined Networking (SDN) provides a solution to this problem by unveiling new capabilities in routing [Gopi et al. 2017]. SDN employs a centralized view of the network which facilitates the use of intelligent mechanisms such as those based on Machine Learning (ML).

The thesis proposes two approaches that use Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) techniques for routing in SDN networks called **R**einforcement Learning and **S**oftware-Defined Networking for **I**ntelligent **R**outing

(RSIR) and Deep RSIR (DRSIR), the approaches add a Knowledge Plane based on RL in the SDN architecture. By considering metrics related to the state of the links and paths, RSIR explores, learns, and exploits potential paths for intelligent routing even under dynamic traffic changes. RSIR leverages the interaction with the environment, and the intelligence provided by RL, as well as the global view and control of the network provided by SDN, to compute and proactively install, in advance, optimal routes in the routing tables of the switches located on the Data Plane. DRSIR is an extension of RSIR based on DRL that enhances the performance even further, especially when considering large scenarios (*i.e.,* settings with larger number of state and action spaces). The performance of RSIR and DRSIR was extensively evaluated by emulation considering the stretch, delay, packet loss, and network throughput as performance metrics. Our solutions were compared to centralized Dijkstra-based routing when link-state metrics (delay, loss, and available bandwidth) are considered individually or jointly. Results show that RSIR and DRSIR outperform Dijkstra-based routing. Results evince that both RSIR and DRSIR are promising solutions to replace traditional routing protocols in SDN.

The contributions of this thesis are: *i)* an architecture and its prototype [Casas-Velasco et al. 2021] that uses RL/DRL for achieving efficient and intelligent routing in SDN, *ii)* a proactive RL-based routing algorithm for RSIR that considers link/path state metrics to explore, learn, and exploit potential routes, *iii)* a proactive DRL-based routing algorithm for DRSIR that considers path-state metrics to calculate potential routes and enhance the performance of RSIR when considering larger number of states and actions into the routing scenario, and *iv)* a prototype of a centralized version of the Dijkstra-based routing in SDN. The novelty of these contributions is that, unlike most existing works, RSIR and DRSIR allow the establishment of multiple optimization targets to find an optimal routing policy by leveraging the centralized view, control, and programmability of SDN, as well as the cognitive capabilities of RL/DRL. They proactively define and install routing paths based on decisions made by using information on the state of the network with no dependence on traditional routing protocols. Moreover, they do not add any signaling overhead to the network operation. These contributions are quite relevant for cognitive self-driving networks due to employment of intelligence in network decision-making processes.

2. Related Work

SDN and ML techniques have been envisioned to provide innovation to routing protocols. The work in [Gopi et al. 2017, Shirmarz and Ghaffari 2020] enhanced traditional routing protocols by leveraging SDN features. However, they do not fully exploit knowledge about the network operation to accomplish intelligent routing. Other approaches attempted to improve routing by using ML techniques [Chaudhary and Johari 2020, Serhani et al. 2020]; but distributed routing is typically assumed, which increases signaling overhead and can contribute to congestion formation. The solutions in [Mao et al. 2017, Martín et al. 2019] employ both ML and SDN, but they require building labeled data which implies high computational complexity and makes the routing solutions dependent on traditional routing protocols to build such datasets. The work in [Wang et al. 2018, Yu et al. 2018] employed RL and DRL techniques to optimize either the selection of routing algorithms or the link cost for further calculation of paths, but these works focus only on delay optimization.

3. RSIR: Routing in Software-Defined Networks based on Reinforcement Learning

RSIR follows the concept of Knowledge-Defined Networking (KDN) by adding a Knowledge Plane to SDN. In the Knowledge Plane, information is transformed into knowledge by using ML techniques for improving decision-making processes as routing. The KDN architecture also includes the traditional SDN planes (*i.e.,* Management, Application, Control, and Data) aimed at supporting network automated management and control. RSIR was designed for automatic routing by using network-state information. RSIR employs RL to find the best route for all the source-destination pairs by employing network-state metrics as features for the RL process. We conceived RSIR with two types of metrics, link and path metrics. RSIR$_{\text{links}}$ uses link-state metrics (*i.e.,* link available bandwidth, link loss, and link delay) and RSIR$_{\text{paths}}$ uses paths-state metrics (*i.e.,* path available bandwidth, path loss, and path delay).

Overall, RSIR operates as shown in Figure 1 as follows: *i)* the Control Plane collects raw data about the network state by periodically querying the Data Plane, *ii)* the Management Plane retrieves these data to calculate and store network-state information, *iii)* the Knowledge Plane recovers information from the Management Plane, *iv)* the RL-agent uses this information to explore all the possible routes for each pair of nodes, *v)* the routes computed by the RL-agent are stored, *vi)* the Control Plane retrieves route information to install paths in the flow tables of the switches proactively. In this way, the Data Plane can forward packets without consulting the controller, eliminating the latency resulting from the queries sent to the controller.

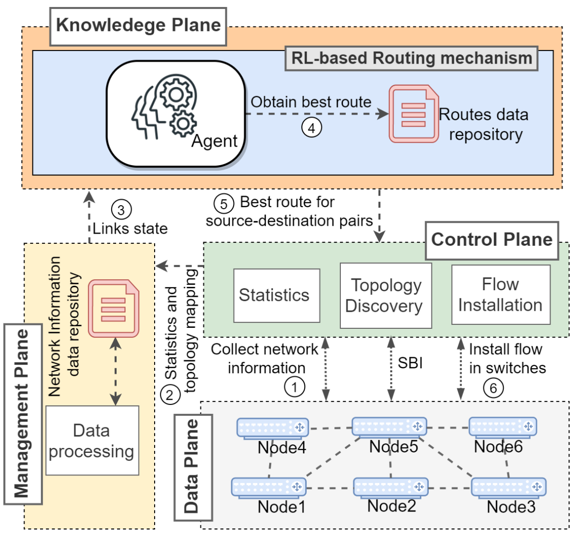
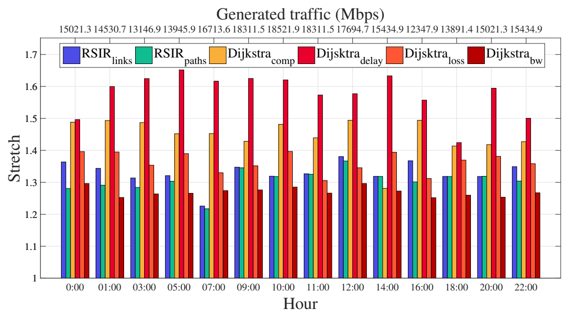


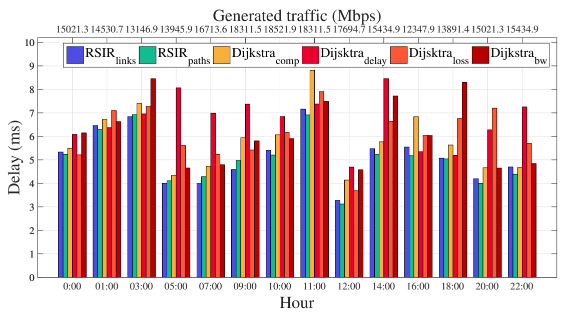
Figure 1. RSIR architecture

The RL-agents find a routing policy to avoid links/paths with large delay and loss ratio and prioritize links/paths with large available bandwidth. In RL, an agent iteratively interacts with the environment; at each step, the agent receives a representation of the state of the environment, selects an action, receives a numerical reward, and moves to a new state. The RSIR agent uses the Q-learning technique [Sutton and Barto 2018], which is model-free and value-based method that defines a value function called Q-function to update the quality value of an action on a state. The agent visits all action-state pairs to approximate the optimal Q-function; such a function defines the routing policy.

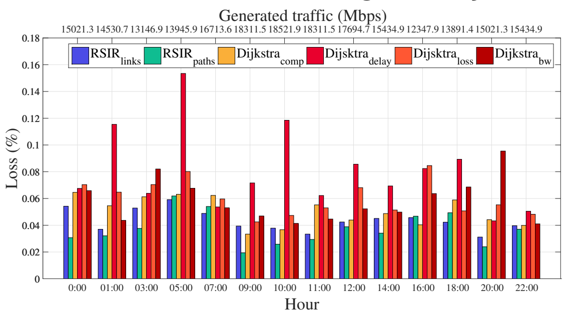
To establish multiple optimization objectives to find a routing policy, the Reward function considers the available bandwidth, loss, and delay metrics that characterize either the link or path state. An SDN controller collects network statistics and the Management Plane further computes link loss, delay and throughput. In particular, the link throughput and loss are computed by using the number of packets that passed through the switch port connected to the link. Then, the available link bandwidth is computed as the difference be-



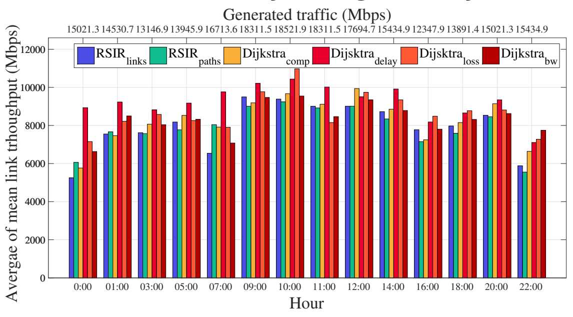
(a) **Mean stretch along the day**



(b) **Mean delay along the day**



(c) **Mean loss along the day**



(d) **Mean link throughput along the day**

Figure 2. Performance metrics results of RSIR$_{links}$ for a 32-node topology

tween the link capacity and the instantaneous throughput of the link. The computation of the instantaneous delay is conducted following the process described in [Liao and Leung 2016], which uses messages of the Link Layer Discovery Protocol (LLDP) and OpenFlow protocol. The Management Plane uses the computed link-state metrics to further compute path-state metrics by using the metrics of each link composing a path.

We evaluated RSIR considering different topologies. In these topologies, each switch has a host that forwards and receives traffic. We deployed the Data Plane using Mininet 2.2.1 and used the tool *iperf3* for generating traffic regarding traffic matrices at different times of the day. We compared the performance of RSIR to different variations of the Dijkstra algorithm using different edge weight such as instantaneous delay (Dijkstra$_{delay}$), instantaneous loss (Dijkstra$_{loss}$), and link available bandwidth (Dijkstra$_{bw}$) as well as a value computed by using these three metrics (Dijkstra$_{comp}$). The routing functions based on these variations of Dijkstra's algorithm were subject to the same traffic scenario and are executed as an application on the SDN controller. The performance metrics used in the comparison were instantaneous link throughput, loss ratio, and delay. Moreover, we evaluated the stretch of the paths given by RSIR with those given by the Dijkstra algorithm. The stretch compares the length of a path to the theoretical shortest path. An evaluation of the learning parameter values defined for the RL-based agent is reported in the thesis and omitted in this paper due to space limitation.

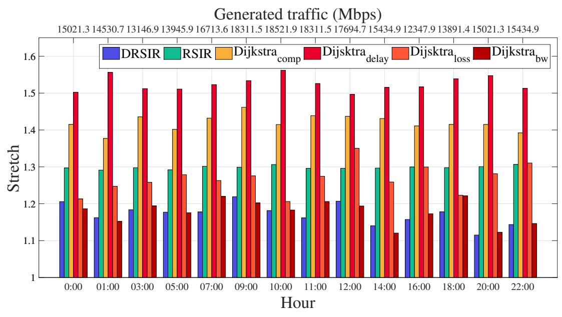
Next, for the sake of brevity we only present the results of RSIR$_{links}$ for a 32-node topology. Figure 2a presents the mean stretch calculated for all the paths found by RSIR$_{links}$ and the variations of the Dijkstra's algorithm. The mean stretch values show that RSIR$_{links}$ chooses a higher number of shorter paths than do Dijkstra$_{delay}$, Dijkstra$_{loss}$, and Dijkstra$_{comp}$. RSIR$_{links}$ indicates paths with 15%, 3%, and 8% stretch values smaller than those obtained by Dijkstra$_{\text{delay}}$, Dijkstra$_{\text{loss}}$, and Dijkstra$_{\text{comp}}$ algorithms, respectively. Results also show that RSIR$_{\text{links}}$ chooses paths with stretch values slightly higher ($< 4\%$) than those produced by Dijkstra$_{\text{bw}}$. Figures 2b and 2c demonstrate that RSIR$_{\text{links}}$ indicates paths that produce lower mean delay and mean loss ratio than those produced by the four Dijkstra's variations. The mean delay given by RSIR$_{\text{links}}$ are 21%, 16%, 15% and 11% lower than those given by the Dijkstra$_{\text{delay}}$, Dijkstra$_{\text{loss}}$, Dijkstra$_{\text{bw}}$, and Dijkstra$_{\text{comp}}$, respectively. Also, the mean loss ratio values produced by RSIR$_{\text{links}}$ are on average 25% and at most 55% lower than those produced by the four Dijkstra's variations. The values of mean link throughput presented in Figure 2d show that the distribution of traffic by RSIR$_{\text{links}}$ uses a higher number of paths less utilized than do all the four Dijkstra's variations. The link throughput is at most 33%, 19%, 24% and 17% lower than those produced by Dijkstra$_{\text{delay}}$, Dijkstra$_{\text{loss}}$, Dijkstra$_{\text{bw}}$, and Dijkstra$_{\text{comp}}$, respectively.

RSIR$_{\text{paths}}$ overperformed RSIR$_{\text{links}}$. Results showed that RSIR$_{\text{links}}$ produced values of mean delay slightly higher ($< 2\%$) and mean loss ratio values 27% higher than those produced by RSIR$_{\text{paths}}$. By providing the RL-agent with path-state metrics, the learning process is simplified. The agent directly explores and exploits the path options as routing decisions instead of extracting information from the link-state to determine paths. Consequently, the employment of path-state metrics contributes to the achievement of better performance than RSIR$_{\text{links}}$ since decisions based on local (link) information may be not aligned with the state of other links on a path to the destination node.

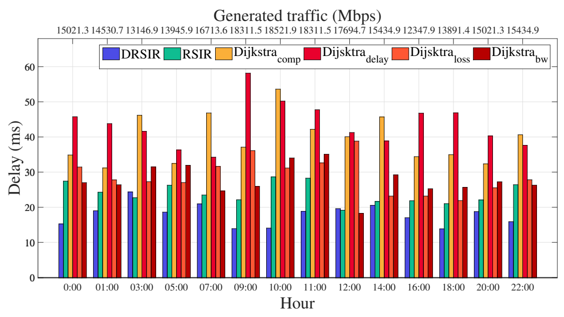
4. DRSIR: Routing in Software-Defined Networks based on Deep Reinforcement Learning

When the size of the State and Action Spaces increases, the RL algorithms may present limited performance due to the increase in the number of learning episodes necessary for the RL-based agent to converge when having to iteratively experience more state-action pairs to achieve reliable estimations. Furthermore, RL algorithms could require huge storage space to store the experience information. To address such a problem, DRL is employed since it leverages the function approximators employed in Deep Learning (DL) to improve the learning speed and the performance of RL algorithms. DRSIR is a DRL-based solution for routing in SDN that aims at providing intelligent routing regarding a network-state defined by path-state metrics since it showed to be more advantageous when making routing decisions. The DRSIR architecture differentiates from the RSIR one by the employment of DRL at the Knowledge Plane. The DRL-agent uses the Deep Q-Network (DQN) technique [Mnih et al. 2015] composed of two independent Neural Networks (NNs) called Online and Target Networks and a *Replay Memory* database. The Online and Target NNs have the same structure. The former estimates the Q-values on the current state, while the latter outputs the Q-values on the next state. For setting learning parameters, we used the convergence of a minimized reward in episodic training as a parameter for comparison.

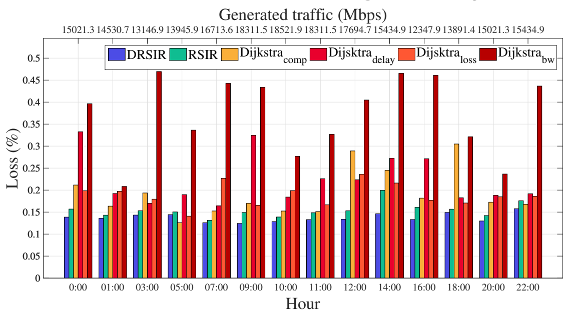
We evaluated DRSIR for three different topologies, 23-nodes, 32-nodes, and 48-nodes to assess its performance in scenarios with different number of state-actions pairs by using the same performance metrics used to evaluate RSIR. Several preliminary tests were conducted to set the more suitable learning parameter values. The results of such tests were not included in this paper for the sake of brevity but are available in the thesis. DRSIR was compared with RSIR$_{\text{paths}}$ and the variations of Dijkstra's algorithm. For the



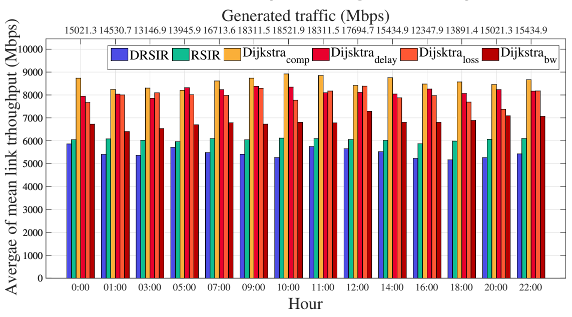
(a) Mean stretch along the day



(b) Mean delay along the day



(c) Mean loss along the day



(d) Mean link throughput along the day

Figure 3. Performance metrics results of DRSIR in 48-nodes topology

sake of brevity, we show results only for a 48-nodes topology. Figure 3 shows results for stretch, link delay, loss ratio, and link throughput obtained by DRSIR, RSIR with path-state metrics, and the Dijkstra variations. Figure 3a depicts that DRSIR produces a higher number of shorter paths than $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, and $\text{Dijkstra}_{\text{comp}}$. DRSIR produces paths that have 23%, 7%, and 17% stretch values smaller than those obtained by $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, and $\text{Dijkstra}_{\text{comp}}$ algorithms, respectively. DRSIR obtains paths with stretch values slightly higher ($< 1\%$) than those produced by $\text{Dijkstra}_{\text{bw}}$. Figures 3b and 3c show the mean delay and mean loss ratio. The mean delay values of DRSIR are on average, 32%, 31%, 19% and 36% smaller than those produced by $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, $\text{Dijkstra}_{\text{bw}}$, and $\text{Dijkstra}_{\text{comp}}$, respectively. Results of mean loss ratio observed with DRSIR are on average 36% and at most 57% lower than those produced by the four Dijkstra's variations. Figure 3d shows the mean link throughput values along the day. The mean link throughput produced by DRSIR is at most 36%, 34%, 25% and 40% lower than those produced by $\text{Dijkstra}_{\text{delay}}$, $\text{Dijkstra}_{\text{loss}}$, $\text{Dijkstra}_{\text{bw}}$, and $\text{Dijkstra}_{\text{comp}}$, respectively. DRSIR indicates paths with links less utilized than do the Dijkstra's variations.

When comparing DRSIR and $\text{RSIR}_{\text{paths}}$, results showed that DRSIR indicates paths with stretch values lower (9% and at most 14%) than the paths indicated by $\text{RSIR}_{\text{paths}}$. The mean delay and mean loss ratio values obtained by DRSIR were 24% and 10% lower than those produced by $\text{RSIR}_{\text{paths}}$, respectively. Moreover, DRSIR and $\text{RSIR}_{\text{paths}}$ indicate paths that produced a similar distribution of flows over the network. The mean link throughput values produced by DRSIR are slightly lower than those produced by $\text{RSIR}_{\text{paths}}$ (on average 10%). DRSIR selects paths slightly shorter and less congested than $\text{RSIR}_{\text{paths}}$. The paths selected by DRSIR produce lower mean delay and loss values than those selected by $\text{RSIR}_{\text{paths}}$. The agent of DRSIR computed all the routes for the 48-nodes topology in $\approx 7.2s$, while the centralized variations of Dijkstra spent $\approx 6s$, which is a non-significant difference.

RIP and OSPF are wildly used routing protocols in networks. We assess the reaction time to topological changes of RSIR and DRSIR and compare them to those of the RIP and OSPF. The topology change handling process of RSIR and DRSIR involves times detecting the topology changes t_{chad}, calculating new routes t_{lear}, and installing them t_{inst}. For RSIR and DRSIR, the setup of t_{chad} is $1s$. The RL-agent of RSIR (using path-state metrics) computes all routes for the 48-nodes topology in $t_{lear} = 4.7s$. The *Flow Installation* module spends on average $t_{inst} = 1.6s$ for updating the flow entries; thus, RSIR takes on average $t_{chad}+t_{lear}+t_{inst} = 8.1s$ to handle change in the topology. In DRSIR, the agent computes all routes in $t_{lear} = 7,2s$ and takes on average $9.8s$ to handle a topology change. DRSIR obtained a t_{lear} higher than the achieved by RSIR because the DRL-agent spends more time training the NNs. However, superior performance offsets this cost. In comparison with RIP that would typically take $30s$ to handle a topological change, DRSIR presents a slower response time due to the centralized controller global view of the network, the generalization provided by the DRL-agent, and the adoption of a network-state routing approach. The OSPF reaction to topological changes when considering a minimum hello-interval = $1s$ and an spf-delay = $1s$ is approximate $9s$ ($5s$ + the average execution time of the Dijkstra's algorithm), which is slightly lower than the time DRSIR takes.

5. Scientific production

- Casas-Velasco, D. M., Villota-Jacome, W. F., da Fonseca, N. L., & Rendon, O. M. C. Delay Estimation in Fogs Based on Software-Defined Networking. In 2019 IEEE GLOBECOM (pp. 1-6). Qualis A1. H-index 96.
- D. M. Casas-Velasco, O. M. C. Rendon and N. L. S. da Fonseca, "Intelligent Routing Based on Reinforcement Learning for Software-Defined Networking," in IEEE Transactions on Network and Service Management, vol. 18, no. 1, pp. 870-881, March 2021, doi: 10.1109/TNSM.2020.3036911. Qualis A1. IF 3.878.
- DRSIR: A Deep Reinforcement Learning Approach for Routing in Software-Defined Networking. Submission to IEEE Transactions on Network and Service Management, 2021. Qualis A1. IF 3.878.

6. Conclusion

In this thesis, we introduced routing approaches for SDN named RSIR and DRSIR based on RL and DRL, respectively. Results evince several conclusions. In RL-based algorithms, the optimization targets can be flexibly adjusted by changing the reward function. The reward functions and the action/state spaces of both RSIR and DRSIR allowed achieving routing policies that outperformed the Dijkstra algorithm. In DRSIR, the DRL-agent benefits from DL to handle large state/action spaces improving the performance of RSIR. DRSIR indicates paths shorter and less congested most of the time than those indicated by Dijkstra and $\text{RSIR}_{\text{paths}}$. As a consequence, the mean delay and loss produced by DRSIR are lower. RSIR and DRSIR do not need labeled datasets for training the agents since they learn by interacting with the environment. The acquisition of such datasets is costly and makes the routing solutions dependent on traditional routing protocols, which use limited information for routing. RSIR and DRSIR are promising solutions for routing in SDN. Moreover, our architecture prototype using RL/DRL for routing offers developers a basis to create network management applications for optimizing and automating network tasks. For future work, we intend to explore multi-level RL schemes to face centralized control challenges such as scalability on larger-scale networks. We also intend to improve the setting up of the learning parameters in order to turn the agent self-configurable.

References

Casas-Velasco, D. M., Caicedo, O. M., and Da Fonseca, N. L. S. (2021). Routing based on (deep) reinforcement learning for software-defined networking. In *https://github.com/danielaCasasv/DRSIR-DRL-routing-approach-for-SDN*.

Chaudhary, S. and Johari, R. (2020). Oruml: Optimized routing in wireless networks using machine learning. *International Journal of Communication Systems*.

Gopi, D., Cheng, S., and Huck, R. (2017). Comparative analysis of sdn and conventional networks using routing protocols. In *CITS*, pages 108–112. IEEE.

Liao, L. and Leung, V. C. (2016). Lldp based link latency monitoring in software defined networks. In *CNSM*, pages 330–335. IEEE.

Mao, B., Fadlullah, Z. M., Tang, F., Kato, N., Akashi, O., Inoue, T., and Mizutani, K. (2017). Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning. *IEEE Transactions on Computers*, 66(11):1946–1960.

Martín, I., Troia, S., Hernández, J. A., Rodríguez, A., Musumeci, F., Maier, G., Alvizu, R., and de Dios, Ó. G. (2019). Machine learning-based routing and wavelength assignment in software-defined optical networks. *IEEE Transactions on Network and Service Management*, 16(3):871–883.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Serhani, A., Naja, N., and Jamali, A. (2020). Aq-routing: mobility-, stability-aware adaptive routing protocol for data routing in manet–iot systems. *Cluster Computing*, 23(1):13–27.

Shirmarz, A. and Ghaffari, A. (2020). An adaptive greedy flow routing algorithm for performance improvement in software-defined network. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 33(1).

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction- second edition*, volume 1. Cambridge, Massachusetts.

Wang, C., Zhang, L., Li, Z., and Jiang, C. (2018). Sdcor: Software defined cognitive routing for internet of vehicles. *IEEE Internet of Things Journal*, 5(5):3513–3520.

Yu, C., Lan, J., Guo, Z., and Hu, Y. (2018). Drom: Optimizing the routing in software-defined networks with deep reinforcement learning. *IEEE Access*, 6:64533–64539.