ROAR: Routing Packets in P4 Switches With Multi-Agent Decisions Logic

ROAR: Routing Packets in P4 Switches With Multi-Agent Decisions Logic

Antonino Angi * Alessio Sacco * Flavio Esposito † Guido Marchetto *
* Department of Control and Computer Engineering, Politecnico di Torino, Italy
† Computer Science Department, Saint Louis University, USA

***Abstract*—The soaring complexity of networks has led to more complex methods to efficiently manage and orchestrate the multitude of network environments. Recent advances in machine learning (ML) have opened new opportunities for network management automation, exploiting existing advances in software-defined infrastructures. Advanced routing strategies have been proposed to accommodate the traffic demand of interactive systems, where the common architecture is composed of a data-driven network management schema collecting network data that feed a reinforcement learning (RL) algorithm. However, the overhead introduced by the SDN controller and its operations can be mitigated if the networking architecture is redesigned. In this paper, we propose ROAR, a novel architectural solution that implements Deep Reinforcement Learning (DRL) inside P4 programmable switches to perform adaptive routing policies based on network conditions and traffic patterns. The network devices act independently in a multi-agent reinforcement learning (MARL) framework but are able to learn cooperative behaviors to reduce the queuing time of transmitting packets. Experimental results show that for an increasing amount of traffic in the network, there is both a throughput and delay improvement in the transmission compared to traditional approaches.**

***Index Terms*—deep reinforcement learning, P4, routing**

I. Introduction

In recent years, there has been a rapid increase in the number of brand-new applications, which not only places more and more demands on communication technologies, e.g., 5G and 6G, but also poses significant difficulties for the Internet. Each application, in particular, has distinct yet strict requirements for latency, jitter, throughput, and packet loss rate. We can observe that as networks continue to evolve and become more complex, the need for efficient routing mechanisms becomes increasingly crucial.

One dictating trend is applying Machine Learning (ML) and Deep Learning (DL) to routing with the aim of leveraging information about past traffic conditions to learn good routing configurations for future conditions [1]. The flexibility provided by SDN enables a fast reactive and proactive network management, in which the SDN controller can easily observe the network and react to changes in traffic demand and evolutions [2], [3]. One class of ML is Reinforcement Learning (RL), which well fits routing problems given its automatic learning improvements to achieve the optimal policy [4]. In recent years many research works integrated RL and Deep Reinforcement Learning (DRL) into SDN networks for routing optimizations [5], [6].

However, these approaches based on centralized controllers are inherently too slow to respond to fine-grained traffic changes, as in short traffic bursts. Moreover, even when the software control plane is local on the switches, the ability to select new routes is often limited and not fast enough [7].

To overcome these limitations, recently programmable data planes have gained popularity, and different works have developed mechanisms that, operating entirely in the data plane, enable real-time adaptation [8], [9]. These solutions can deliver considerable performance benefits over more static mechanisms and centralized approaches using fine-grained performance information on hardware timescales. However, these techniques are limited to trivial performance-aware policies that are unable to learn during the execution and, consequently, adapt to multiple scenarios. Moreover, as the complexity of the network grows, determining the optimal routing policies to avoid congestion and improve performance has become increasingly essential but also challenging.

To automate routing decisions directly in the network device, we designed a Reinforcement learning for Autonomous Routers (ROAR). In our solution, we uses network programmability in general, and P4 [10] programmable switches in particular, to perform distributed routing decisions via Deep Reinforcement Learning (DRL). As such, every switch of the network is an agent of the DRL system, which uses the algorithm to decide the forwarding port for the incoming packet according to two main factors: the next hop to the destination, known for the topology of interest, and the port's outgoing queue, to be minimized. Since the switches constantly learn from the environment, the model is periodically trained to consider the impact of different routing decisions on network performance and select the best route based on learned policies. Designing a DRL model with P4 is known to be challenging since the architecture does not support loops, complex arithmetical operations, or if-else conditions in action blocks, which are essential for the DRL algorithm. To overcome this limitation, we modify the P4-16 compiler in order to adapt to a C++ external module, which is the fundamental intermediary between the P4 application and the DRL algorithm. We evaluated our solution on an emulated network over Mininet, showing that when the network starts being congested, the benefits of ROAR can be observed in the increment of throughput and delay reduction.

The rest of the paper is structured as follows. Section II presents literature about RL-based routing. In Section III, we describe ROAR's components. Section IV presents the experimental results, and finally, Section V concludes the paper.

II. Related work

With the increasing number of connected devices and more demanding applications, e.g., Tactile Internet, meta-verse, the volume of data traffic flowing through networks has grown exponentially. This has created a pressing need for efficient routing mechanisms to optimize network performance, reduce latency, and ensure reliable data transmission, bringing researchers to propose different automated and reactive approaches, usually combining ML techniques, such as RL and DRL, to perform traffic prediction for load balancing and routing [7]. One example is QR-SDN [11], where the authors use tabular RL (Q-learning) techniques to reduce latency across the network by optimizing multipath routing. An SDN centralized controller routes packets using a flow-preserving strategy that aims to minimize the latency of transmissions. Another interesting study is RSIR [12] that implements a knowledge plane, in connection with the management plane, to store the data, which are then used by the SDN centralized controller to find the routing shortest path and balancing the load, comparing their results with the classic Dijkstra algorithm. As the authors suggest, adopting a centralized controller with a global view brought a low response time in case of topology changes. This solution works well in limited scenarios, but in large network scenarios and with an increasing number of flow numbers, we assist in a downgrade of throughput and overall delay due to the constant interaction with a centralized controller that cannot ensure reliable packet transmissions. Over the last few years, Deep Reinforcement Learning (DRL) methods have also been quickly adopted in many fields of networking, from the Internet of Things (IoT) to concurrent multipath transfer data scheduling, mobile-edge computing, and heterogeneous networks, focusing especially on routing optimization and congestion control [5]. An example of DRL-based approach is [13], where the authors propose DROM, a deep policy gradient routing optimization mechanism that uses neural networks to improve the network's overall performance. However, this work requires human intervention for the network strategy customization and maintenance that affects the rewarding function [7]. Routing optimization mechanisms were also essential in heterogeneous networks where various devices with different capabilities, such as cell phones, laptops, and tablets, are connected, and efficient resource utilization was vital to avoid delays and maximize the throughput. An example of an application in a heterogeneous network is SmartCC [14], a multipath congestion control approach based on DRL where an asynchronous RL framework learns a set of congestion rules and adapts the congestion windows accordingly. Several attempts to alleviate network congestion and balance network load utilizes DRL methods running without [15] or with an SDN controller, such as DRLS [16] and IQoR-LSE [17], where great focus is brought to the collection of network measurements.

While with the advent of programmable switches there was an attempt to move part of the computation inside network devices, e.g., [18], [19], to the best of our knowledge there is no current routing solution having a data-driven DRL routing algorithm running inside the device. As shown in Fig. 1, differently from other centralized solutions that face the issue of data collection, we propose a routing optimization mechanism aiming at providing a general solution adaptable to any network scenario and any P4-compatible switch.

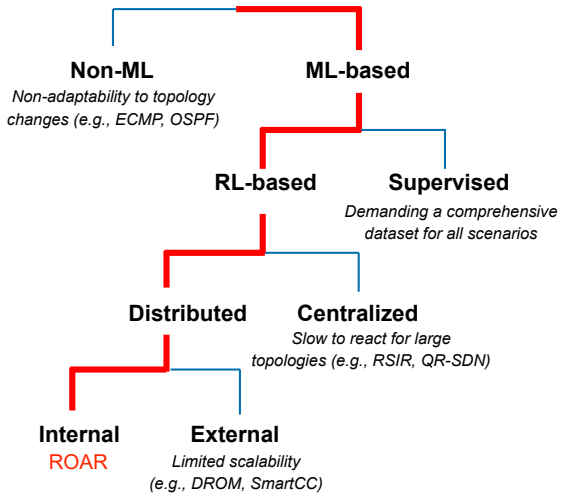

Fig. 1: Design space for adaptive routing techniques highlighting difference with state-of-the-art.

III. System Architecture and Components

In this section, we describe the design of our solution, as well as the principles behind this definition. As represented in Fig. 2, we can observe that ROAR revolves around using P4 switches to control the forwarding plane directly inside the network device, adopting forwarding decisions according to the outcome of a deep reinforcement learning approach. Every switch of our network is composed of three main blocks: (i) the P4 application, which constitutes the logic of the network device, (ii) the Deep Reinforcement Learning (DRL) module, that runs the learning algorithm to route the packets, (iii) the Inter-process communication (IPC) module, which is composed of the high-level modules and data structures needed to connect the P4 application to the DRL module.

A. DRL Module in ROAR

In ROAR, every switch of the network is an agent of the DRL, making the solution a Multi-Agent Reinforcement Learning (MARL) [20] scenario that applies forwarding decisions according to the result of the DRL model in an environment where other agents are also trying to reach the same goal. We consider a system of M agents (routers) within a shared environment without a centralized controller responsible for gathering rewards or making decisions on behalf of the agents. In this setup, the collection of agents is represented as $\mathcal{M}_t$, where M denotes the cardinality of the set, and every agent has the capability to communicate with all other agents. In particular, the composition of the agent set can possibly vary over time (e.g., failures) and is defined as $\mathcal{M}_t$ at a given time $t \in \mathbb{M}$.

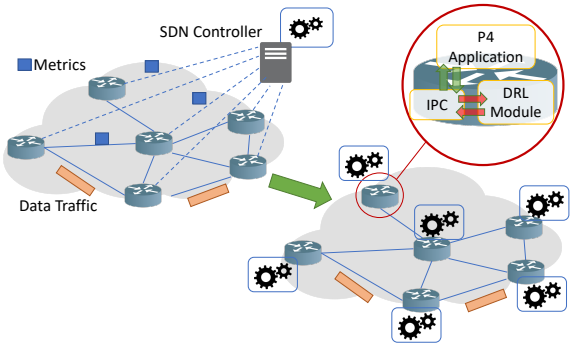

Fig. 2: Overview of ROAR and its differences compared to a centralized solution. The solution is based on the control of packet forwarding planes directly in networking devices, also showing the blocks composing every network switch.

Our time-varying MARL process is defined as a tuple $\langle \{S^i\}_{i\in\mathcal{M}}, \{A^i\}_{i\in\mathcal{M}}, P, \{R^i\}_{i\in\mathcal{M}}, \{\mathcal{M}_t\}_{t\geq 0}\rangle$, where S^i denotes the local state space of agent i in $\mathcal{M}_t$, and A^i is the action set that agent i can execute. Besides, $A = \prod_{i=1}^{M} A^i$ is the joint action space of all agents, also referred to as the global action profile. We then proceed by defining the local reward function of agent i, denoted as $R^i : S \times A \to \mathbb{R}$, and the state transition probability function $P : S \times A \times S \to [0, 1]$. In this setup, we assume that the states and actions have a global impact but are locally observable, as well as the rewards, which are only observed locally. At each time step t, given the state $s_t \in S$ and the joint actions of the agents $a_t = (a_t^1, \ldots, a_t^M) \in A$, each agent receives an individual reward r_{t+1}^i. This reward is given by an equation that captures the incentive that the learning model wants to model and is determined by $R^i_{(s_t,a_t)}$. Additionally, the system transitions to a new state $s_{t+1} \in S$ with a probability of $P(s_{t+1}|s_t, a_t)$.

Our model is considered fully decentralized and *individual* as each agent receives rewards locally and performs actions independently. Opposed to an SDN centralized scenario, where the controller has a global view of the network, we design a fully distributed solution. The leading idea is to train agents independently of each other to simplify the coordination process among routers and reduce the overhead caused by continuous updates. On the contrary, routers only exchange information about the known topology to consolidate in a virtual global network view. In this scenario, agents are independent of each other, considering that they do not share network state information or model parameters.

In each ROAR's agent i, the *action* A^i is a discrete number ranging from 1 to N, where N is the number of ports the switch uses. The *state* S^i, instead, is composed of three elements: (i) current destination, which is the current packet's destination IP address, (ii) future destinations, which is a list of L next packet's destination IP addresses that follow the same route as the current one, (iii) action history, which is a list of the last k actions adopted for the current packet's destination. Every time a given action for a certain state has been performed, e.g., a packet has been forwarded towards a specific port, the reward function is evaluated to update the expected cumulative reward (Q-value) for that state-action couple. Thus, it takes into account two main factors: (i) queuing time, which is the time every packet has spent in the output queue, and (ii) the distance of the chosen next hop from the final destination. While we want to minimize the time spent by the packets in the outgoing queues, the distance of the next hop from the final destination is the only information the agent knows about the global topology. The *reward* function R^i also considers two indicators: delivered, σ_1, set to 1 if correctly routed, 0 otherwise, and dropped, σ_2, set to 1 if the packet has been dropped, 0 otherwise. Their value is always set to 0 in the case of spine switches, as they are not directly connected to any destination host.

Summarizing, the reward function for each agent i is:

$$R^i = \lambda_1 * \sigma_1 - \lambda_2 * q - \lambda_3 * \sigma_2 - \lambda_4 * \sigma_3 * d \qquad (1)$$

where: (i) the λ values are the model's hyper-parameters set during the training to check the performance of the algorithm, (ii) q is the time that the packet has spent in the queue before being sent, (iii) σ_3 is a parameter indicating whether the switch is a spine one and is multiplied by d, i.e., the distance to the destination switch.

Our Neural Network. In ROAR, we determined the number of input features for our NN, i.e., the state space, by computing their mean reward and selecting the list length for which they achieve the highest value. After different settings and evaluations, we set as input the last two taken decisions, i.e., the length for the "future destination" and "action history" state. Being categorical features, we need to convert them into numerical ones using encoding techniques. For this work, we used the One-Hot-Encoding method, which uses *dummy* variables to perform categorical encoding and performs better than other techniques according to the precision-recall curve (PR-AUC) metric [21].

These encoding features are now ready to be the inputs of our NN model. We evaluated different NN structures on leaf and backbone switches, finding that an architecture composed of 3 hidden layers of 128, 64, and 32 neurons can achieve the best performance. It is important to notice that the DRL algorithm is not applied every time a switch receives a packet. The overhead would be too high and it would take tens of milliseconds to make a routing decision which would be intolerable at line rate for modern switches. The resulting trade-off is to adopt a static routing guided by the DRL algorithm by means of periodical updates. It has been tested that updating the single route towards a specific destination every $10,000$ packets maximize the performance of our network.

B. P4-based Actions

P4 is a networking programming language that allows defining the data plane processing of a switch in a high-level structure and generates efficient code that can run on different hardware targets, including ASICs, FPGAs, and CPUs. It provides a way to define how packets should be processed through a network device, including how they are parsed,

matched against rules, and modified. To do so, P4 is composed of three main blocks: a parser, a match-action pipeline, and a deparser. The parser is designed as a finite-state machine that analyzes and extracts headers. For example, a packet may begin with an Ethernet header, followed by an IPv4 or IPv6 header; the parser extracts all of these headers and passes them to the next step, the match-action pipeline. This stage is composed of checksum verification algorithms, ingress, and egress pipelines, which are composed of structures (e.g., tables, registers, counters) that P4 uses to customize switch behavior and implement routing strategies based on policy. The deparser is the final stage, defining how outgoing packets are constructed from a set of header fields. In the following, we mostly focus on the match-action pipeline's ingress and egress while considering that the parser recognizes and extracts the Ethernet, IP, UDP, TCP, and ICMP headers.

When a ROAR's router receives a packet, it performs two main operations: (i) inserts the current packet's IP destination address inside the "future destinations" data structure, accessed by the IPC module, (ii) chooses the output port given by the DRL module. The egress performs two other main operations: (i) forwards the packet according to a FIFO criteria and removes its destination IP address from the "future destinations" data structure, (ii) interacts with the IPC module to determinate the reward for that specific forwarding action according to the time spent by the packet in the outgoing queue. Due to P4 limitations that prevent implementing any machine-learning method inside the default P4 compiler, we had to modify it and enable *"extern"* instances, which allow implementing external methods outside the P4 program [22]. The P4 program can reference the extern object and pass inputs to it, but the implementation details of how the object works are hidden from the P4 program. This makes the network's control and data plane separation easier, allowing the P4 program to focus on packet processing logic while leaving the low-level details of interacting with hardware to the extern objects.

C. IPC module

In ROAR, we use an IPC module deployed inside each switch to act as an intermediary between our P4-based network application and the DRL algorithm. The IPC block serves to bridge the two modules by providing a means for the P4 application to communicate the packet counters to DRL, and for the DRL to communicate the chosen actions, i.e., next hop, to the P4 forwarding plane. As mentioned previously, our IPC module includes functionality for preprocessing and transforming data to make it suitable for the DRL algorithm and, in particular, for the NN method, as well as for monitoring and collecting feedback on the performance of the system for the reward function. More in detail, our IPC module uses sockets abstraction written in C++ to establish a communication channel and exchange data with the DRL module.

IV. Evaluation

This section describes our experimental settings and results obtained over a virtual testbed like Mininet, focusing on a comparison between ROAR, a traditional routing protocol and a centralized SDN solution.

A. Evaluation settings

To validate ROAR's benefits, we used Mininet a network emulator that allows reproducing virtual networks and use them as a testbed for simulation purposes. Being specifically designed for software-defined networking (SDN), it supports P4-compatible switches via the Behavioral Model Version 2 (BMV2), which allows compiling a P4 program into packet-processing actions of C++11 software switches. The topology used for testing is composed of 10 servers connected to their switches which are consequently connected to other 4 switches, in a leaf-spine fashion, with all the links of the topology with 100 Mbps bandwidth. In all the performed tests, we are using iperf3, a tool to perform measurements on the network according to different bandwidths, protocols, and buffer setups. Each server of the network sends packets to any other server, varying the number of receiving servers (from 1 to 9) and replicating the workload as described in [23]. For each network load, we then computed the average of the obtained results (*i.e.,* RTT, FPS, throughput, and packet loss) and drew the two-tailed confidence interval at 95%. For the context of this work, we compare our results against two alternatives: a traditional routing protocol implementation as OSPF, which uses the longest prefix match tables to perform routing across the network and does not depend on the current network load; and a centralized SDN solution, QR-SDN [11], that routes packets according to the output of a tabular RL algorithm.

B. Random Traffic Generation

To study the performance of our solution, we started by generating traffic using the iperf3 tool that helped us control the level of congestion of our network and compare the results with QR-SDN and the traditional routing implementation, as OSPF (Fig. 3). We can see from Fig. 3a that when the load is low (10% to 20%), the network is not congested, and QR-SDN shows a lower Round Trip Time (RTT), while ROAR and OSPF perform similarly. This might be caused to the fact that ROAR uses DRL to choose the optimal route and, for every $10,000$ packet, the IPC module interacts with the DRL one to retrieve information about the best port to forward the packet. This interaction can indeed impact on the network performance, decreasing the overall throughput. However, when the network load starts to increase (20% to 40%), a significant difference is visible with OSPF, while QR-SDN still achieves a lower RTT than ROAR. When our network is highly congested (from 50% to 90% of network load), we can clearly identify the benefits brought by ROAR, where the RL method accurately chooses the best forwarding port to avoid congestion and decrease the RTT, while for QR-SDN the interaction with an SDN controller impacts the overall performance of the network. A similar behavior is visible when evaluating the throughput, as in Fig. 3b. The figure shows that when the network is not congested and our load is low (10% to 20%), ROAR performs as the traditional routing

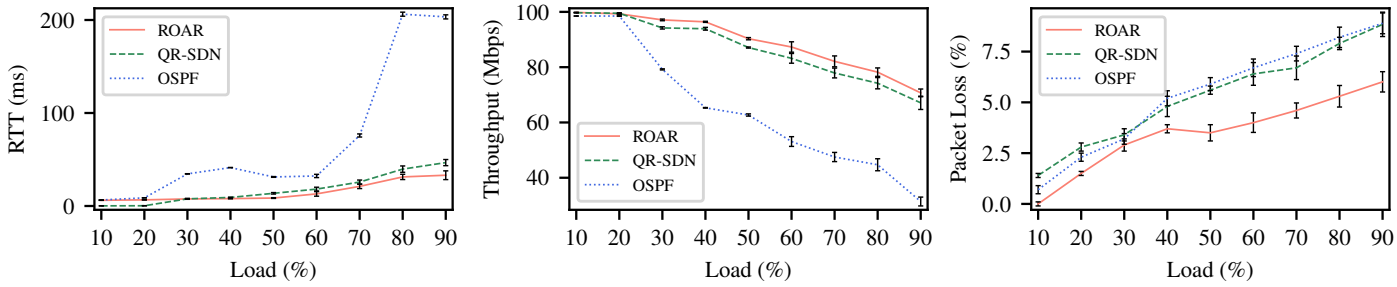

Fig. 3: Comparison of ROAR, OSPF and QR-SDN, measuring the (a) RTT evolution, (b) throughput, and (c) packet loss at increasing network load.

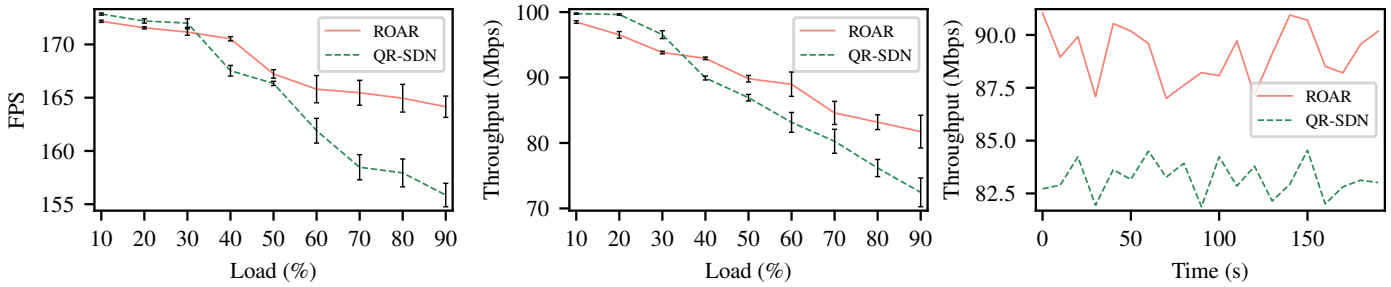

Fig. 4: Comparison of ROAR and QR-SDN with realistic traffic workloads, measuring the (a) FPS evolution and (b) throughput at increasing network load. (c) Throughput in 200 seconds when the network load is at 60%.

implementation, while QR-SDN achieves higher throughput. However, a different behavior can be seen when our network load increases. While QR-SDN achieves better throughput than OSPF, ROAR allows our network to handle congestion better and perform better than the compared solutions.

Packet delivery is another relevant metric when evaluating a solution, as it is an indicator of how well the network responds to the implemented strategy. Dealing with a high number of packet loss means a low transmission quality with the need to send packets again, negatively impacting the performance of our model and the network resource utilization. For this reason, we report in Fig. 3c the packet loss of when transmitting UDP packets and network runs ROAR and compared it to QR-SDN and OSPF. For this experiment, we chose UDP as the default transport protocol to validate the only effect of routing over losses, since for TCP we could also have the TCP congestion control protocol running on the host to interfere. As visible from the figure, our solution can constantly reduce the number of packets lost compared to alternatives, due to the faster reaction when congestion starts appearing. By promptly reacting and adapting the next hop, ROAR is able to reduce the number of losses. This result is particularly important for delay-sensitive transmission, where keeping the number of retransmissions to the minimum is crucial.

C. Trace-based Evaluation

To evaluate our solution in the context of a realistic workload scenario, we used captures taken from publicly available datasets [24] and replayed them in our network using the well-known *tcprelay* tool. The file, representing traffic

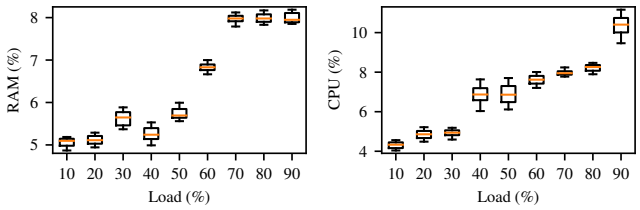

Fig. 5: (a) RAM and (b) CPU consumption as a percentage of available resources of a reference Intel Tofino switch.

collected from a data center, has been analyzed to extract and replicate the flows in our topology by opportunely adapting IP addresses. We reported in Fig. 4 the results of our evaluation.

We start in Fig. 4a by counting the flows per second (fps) that the network can handle while running the file capture. When our network is not congested, and the network load is low (10% to 30%), QR-SDN is able to send a higher number of fps than our solution. However, when the network starts being more congested and the network load increases (from 40%), it is notable that ROAR performs better, being able to send a high number of fps even at the highest network load. As previous experiments showed, when the network state (expressed as load) is more prominent, the RL module can differentiate actions and select a more appropriate path.

A similar behavior is visible when evaluating the throughput in both our solution and QR-SDN (Fig. 4b). While QR-SDN achieves higher throughput at lower network loads, ROAR is able to outperform when the network is fully congested. Finally, we considered the first 200 seconds of execution at a

fixed network load of 60% and reported the result in Fig. 4c. It is visible from the figure that for the entire evaluation period, ROAR achieves better throughput than QR-SDN, with results coherent with the ones reported before. These results are extremely important to assess the validity of local performance-aware routing not only with synthetic traffic, but also with more realistic traffic patterns.

D. Can ROAR run over real switches?

Aware of the impact on resource consumption that a DRL model might cause when implemented on physical switches (e.g., FPGA, Tofino), we computed the RAM and CPU usage of ROAR at a varying network load. To do so, we took as reference the X308P-48Y-T programmable switch [25], combined with its embedded Data Processing Unit (DPU), proportioning the results to its computing power. We reported the results in Fig. 5. In Fig. 5a, we can see how our solution consumes a low amount of RAM when the network load is low and the network is congested. This amount increases only up to 2% at the highest load (70%-90%). The same behavior is visible in Fig. 5b, where the CPU consumption only increases at high network loads. These figures prove that, despite the DRL module, the IPC module interaction, and the NN algorithm, ROAR requires small hardware resources, making it suitable to deploy on real programmable switches.

V. Conclusion

In this paper, we propose ROAR, a distributed ML-based solution that uses the novel Deep Reinforcement Learning (DRL) mechanism to optimize the routing process of the network while doing all the computation directly inside the P4 programmable switches. This approach allows to take into consideration the current link load and re-route packets over less congested paths. In the experimental results, we compared our solution to a traditional routing implementation and a centralized SDN solution. It has been proven that the absence of an interaction with a centralized SDN controller reduces RTT even when the network is congested, also achieving higher throughput, especially at higher network loads.

Acknowledgment

This work has been partially supported by NSF awards 2133407 and 2201536.

References

[1] A. Sacco, F. Esposito, and G. Marchetto, "Resource Inference for Sustainable and Responsive Task Offloading in Challenged Edge Networks," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 3, pp. 1114–1127, 2021.

[2] Y.-J. Wu, P.-C. Hwang, W.-S. Hwang, and M.-H. Cheng, "Artificial intelligence enabled routing in software defined networking," *Applied Sciences*, vol. 10, no. 18, p. 6564, 2020.

[3] A. Sacco, F. Esposito, and G. Marchetto, "RoPE: An Architecture for Adaptive Data-Driven Routing Prediction at the Edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 986–999, 2020.

[4] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[5] T. Fu, C. Wang, and N. Cheng, "Deep-learning-based joint optimization of renewable energy storage and routing in vehicular energy network," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6229–6241, 2020.

[6] C. Liu, M. Xu, Y. Yang, and N. Geng, "DRL-OR: Deep reinforcement learning-based online routing for multi-type service requirements," in *IEEE INFOCOM - IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[7] R. Amin, E. Rojas, A. Aqdus, S. Ramzan, D. Casillas-Perez, and J. M. Arco, "A survey on machine learning techniques for routing optimization in sdn," *IEEE Access*, vol. 9, pp. 104 582–104 611, 2021.

[8] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, 2020, pp. 701–721.

[9] L. Yu, J. Sonchack, and V. Liu, "Mantis: Reactive programmable switches," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM '20)*. ACM New York, NY, USA, 2020, pp. 296–309.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[11] J. Rischke, P. Sossalla, H. Salah, F. H. Fitzek, and M. Reisslein, "QR-SDN: Towards Reinforcement Learning States, Actions, and Rewards for Direct Flow Routing in Software-Defined Networks," *IEEE Access*, vol. 8, pp. 174 773–174 791, 2020.

[12] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. da Fonseca, "Intelligent Routing Based on Reinforcement Learning for Software-Defined Networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2020.

[13] C. Yu, J. Lan, Z. Guo, and Y. Hu, "Drom: Optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 64 533–64 539, 2018.

[14] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu, "Smartcc: A reinforcement learning approach for multipath tcp congestion control in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 11, pp. 2621–2633, 2019.

[15] S. S. Bhavanasi, L. Pappone, and F. Esposito, "Dealing with changes: Resilient routing via graph neural networks and multi-agent deep reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 2283–2294, 2023.

[16] L. Zhao, J. Wang, J. Liu, and N. Kato, "Routing for Crowd Management in Smart Cities: A Deep Reinforcement Learning Perspective," *IEEE Communications Magazine*, vol. 57, no. 4, pp. 88–93, 2019.

[17] B. Dai, Y. Cao, Z. Wu, and Y. Xu, "IQoR-LSE: An Intelligent QoS On-Demand Routing Algorithm With Link State Estimation," *IEEE Systems Journal*, vol. 16, no. 4, pp. 5821–5830, 2022.

[18] C. Yu, W. Quan, D. Gao, Y. Zhang, K. Liu, W. Wu, H. Zhang, and X. Shen, "Reliable cybertwin-driven concurrent multipath transfer with deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 8, no. 22, pp. 16 207–16 218, 2021.

[19] A. Sapio, M. Canini *et al.*, "Scaling Distributed Machine Learning with In-Network Aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*, 2021, pp. 785–808.

[20] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in multi-agent systems and applications-1*, pp. 183–221, 2010.

[21] C. Seger, "An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing," 2018.

[22] J. S. da Silva, F.-R. Boyer, L.-O. Chiquette, and J. P. Langlois, "Extern objects in p4: an rohc header compression scheme case study," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 517–522.

[23] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.

[24] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 267–280.

[25] "48x25gb+8x100gb, intel tofino p4 programmable bare metal switch: Asterfusion," July 2022. [Online]. Available: https://cloudswit.ch/product/48x25gb8x100gb-intel-tofino-p4-programmable-bare-metal-switch-asterfusion/