Routing with ART: Adaptive Routing for P4 Switches With In-Network Decision Trees

Routing with ART: Adaptive Routing for P4 Switches With In-Network Decision Trees

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*—Recent advances in Machine Learning (ML) brought several advantages also within computer network management. For programmable data planes, however, it is more challenging to benefit from these advantages, given their limited resource capabilities colliding with the complexity of ML models. In this paper, we propose ART, an attempt to simplify ML-based solutions for routing, so that they can "fit", *i.e.*, be executed, on P4 switches. To provide such model simplification, ART relies on efficient knowledge distillation techniques, converting, in particular, Deep Reinforcement Learning (DRL) models into a simpler Decision Tree (DT). Our evaluation results validate the accuracy of the extracted model and the application of the model logic directly into switches with little impact, paving the way for a more reactive data plane programmability via machine learning integration.**

***Index Terms*—in-network machine learning, P4, reinforcement learning**

I. Introduction

Control plane and data plane programmability have improved the quality of several network management services. Classical problems (traffic routing, congestion control) and recent ones (traffic prediction/classification, anomaly detection) finds new ways of being deployed, leading to improved network performance and an easy way of controlling network devices for the operators. An established trend involves the use of Machine Learning (ML) and Deep Learning (DL) models to solve these problems, where in this paper we focus on routing approaches [1], [2]. The goal of these data-driven solutions is to mine past traffic conditions, acquiring information on what may be the optimal routing for current or future network flows [1], [3], [4]. In particular, several Reinforcement Learning-based approaches exist for routing [2], [5]–[8]. For example, QR-SDN [9] showcases the application of Q-learning to reduce network latency through optimizing multipath routing within a Software-Defined Networking (SDN) framework. Similarly, within SDN scenarios, RSIR [10] presents a knowledge plane that collaborates with the management plane of a centralized SDN controller to ascertain the shortest routing path and distribute the load based on link-state information representing the network's state. While these solutions have sound design, they rely on centralized (Software-Defined) controllers that inherently lack the agility to swiftly adapt to sudden traffic fluctuations.

To keep up with the traffic rate, recent solutions are designed to run computations and applications in network devices exploiting the programmability and flexibility of SDN architectures and data-planes, *e.g.*, Switch-ASIC, network interface cards (NICs), FPGA-based network devices, P4-enabled switches [11]. This paradigm is referred to in the literature as *In-Network Machine Learning*, when ML processes, either training or inference, are done entirely in the network. In-network applications can adjust to growing demands on cloud network infrastructure by potentially operating at a line rate and handling all incoming traffic or a specific subset thereof. Such a paradigm is promising since network devices are already deployed within the network, and any data used for inference is routed through them.

Current solutions are hindered by switch hardware limitations and thus limit the usage of programmable switches to either collect important features for the model [12] or the run of trained ML models for the inference task [13]. To the best of our knowledge, no available solution can run adaptive ML-based routing decisions directly inside the switches [14].

To cope with this challenge while meeting the hardware constraints, *e.g.,* of P4 switches, we propose a solution called *Adaptive Routing with Trees (ART)*. In ART, we apply Deep Reinforcement Learning (DRL) to routing. The algorithm learns how to route based on the current network utilization with the help of the P4 switch network telemetry capabilities. Measurements collected with P4 are then fed in the DRL model, to adapt routing tables dynamically.

The trained DRL is then distilled into a Decision Tree (DT) that can be easily installed in P4 in the form of flow rules. The distillation process follows a teacher-student approach and allows the conversion of a complex decision process (the one of the Deep Neural Network of DRL) into a lightweight and interpretable model, *i.e.*, the Decision Tree. In particular, every P4 switch runs a learning agent that feeds the DRL and decides the forwarding port for the incoming packet according to the IP destination and the current port utilization.

Our experimental results obtained over an emulated network in Mininet show that ART can effectively learn to route packets to avoid network congestion while minimizing the additional overhead. In turn, ART reduces packet delays and increases the throughput while minimizing packet losses.

The rest of the paper is structured as follows. Section II describes the ART's design and components, while Section III presents and discusses our experimental results. Finally, Section IV concludes the paper.

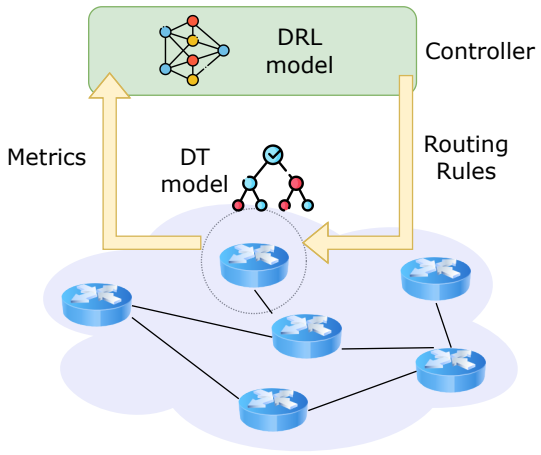


Fig. 1: ART's components. The P4-programmable switches exchange traffic metrics to the control plane where resides the DRL model. Such a model is then distilled into DTs that well fit the P4 packet processing (data plane).

II. System Design

A very recent trend is to automatically route data packets by using information about the network traffic. This process is often based on reinforcement learning (RL) to find the best route for all source-destination pairs by using only a few link-state metrics (*i.e.*, available bandwidth, loss, and delay) as features of the RL process. Despite the advantages of automation and adaptability to various network conditions, RL models are poorly suited for programmable switch architectures. To address these limitations without losing the advantages of RL-based routing, we have developed ART, represented in Fig. 1.

As visible from the figure, ART revolves around using three components: *(i)* the SDN controller, that interacts with the P4 switches, *(ii)* a Deep Reinforcement Learning (DRL) module, which implements the learning algorithm needed to route packets, *(iii)* a Decision Tree (DT) module that uses the trained DRL to build the routing rules which will be injected into the P4 code.

A. Switch-Controller Interaction

P4, a common networking programming language, enables the definition of a switch's data plane processing in a high-level structure and generates efficient code that can be executed on various hardware targets, including ASICs, FPGAs, and CPUs [11]. It provides a way to define how packets should be processed by a network device, including how they are parsed, matched against rules, and modified. However, programming in P4 is widely recognized as challenging due to the impossibility of having loop cycles, the restriction of if-else conditions to specific code areas, and the debugging process, which involves going through huge log files that record each received and forwarded packet. Furthermore, while highly used and adapted to different scenarios, P4 does not have a restart mechanism that automatically allows stopping the code, updating it, and rebooting it [15], [16]. Current solutions can hardly change routing rules at the varying network and traffic conditions [17]. To overcome it, in ART we decided to use the P4Runtime API [18] to enable a continuous interaction between the SDN controller and the P4-based switches. P4Runtime uses a *gRPC* connection between the controller and the switch to manipulate the P4 elements (*e.g.,* add, update, or delete table entries, read the values of counters) and even perform packet manipulation.

Data Plane. In ART, every time a packet is received, the switch parses it and extracts three values: the destination IP address, the packet's size, and the timestamp. These extracted fields are then forwarded to our controller via P4Runtime using the digest mechanism to avoid sending unnecessary packet information. The P4 switch can thus send the packet out to the port using the language's match-action paradigm. The controller rules the association IP prefix-next hop.

Control plane. The controller combines the information coming from the switches (*i.e.*, packet-related data and timestamp) and the network load to build the metrics used to take routing decisions. These values are then sent to our DRL module, which computes the reward function and decides which port to forward the packet to according to the link utilization and capacity (see Section II-B for details). The match-action rule is then transmitted to the P4Runtime API to populate a pre-configured table on the P4 code. When it is time to forward the packet, our P4-programmed switch uses the information stored in the table to choose the egress port. Finally, the timestamps received from switches are used to compute the packet latency, the metric that is then used as an indicator of the goodness of the routing decision.

As emerged from this discussion, the switch-controller interaction is key to route packets but may also introduce excessive overhead. Aware of the latency that this continuous interaction with a centralized controller may entail, we programmed the P4 switches to asynchronously digest every received packet to the controller, in order to parse packets at line rate. Moreover, the DT model used for routing is updated only every f received packets. We set f equal to $1,000$, as this value is a good trade-off between responsiveness and performance.

B. DRL module in ART

Despite the presence of an external SDN controller, we train a DRL model per switch, which leads to a decentralized architecture where each switch is an agent of the model. In other words, this turns our solution into a Multi-Agent Reinforcement Learning (MARL) setting [19]. Choosing a MARL strategy over a single-agent one offers significant advantages. First, a MARL strategy allows for exploring different decision-making approaches among individual switches, proving robustness in response to varying network conditions and traffic patterns. Second, a decentralized approach like MARL improves scalability and enhances the network's ability to handle complex scenarios, guaranteeing elevated performance while efficiently allocating resources [20].

Each agent (*i.e.,* switch) of our MARL environment chooses the forwarding port according to the reward output in a scenario where other agents also try to reach the same goal independently. To schematize this, each agent of our topology $i \in \mathcal{M}$ operates within a shared environment where, for each given state $s \in \mathcal{S}^i$, it performs an action $a \in \mathcal{A}^i$ that brings him to a new state $s' \in \mathcal{S}^i$ and computes the reward function $r \in \mathcal{R}^i$. In this context, the objective is to find the optimal policy π^i for the agent $i \in \mathcal{M}$ so that $\pi^i : \mathcal{S}^i \rightarrow \mathcal{A}^i$ maximizes the local reward $r \in \mathcal{R}^i$, defined as $R^i : S^i \times A_i \rightarrow \mathbb{R}$, and the state transitions with a probability function of $P : S^i \times A^i \times S^i \rightarrow [0, 1]$. At each time step t, given the state $s_t \in S$ and the corresponding 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, representing the learning model and corresponding to the chosen action a_t at the state s_t. It is important to remark that each agent has a local view of the environment, locally computing the reward and deciding the action, but then the agent's choice has a global impact on the MARL environment.

In ART, for each agent *i*, the action $a_t \in A^i$ that can be taken is a discrete number $[1, N] \in \mathbb{N}$ corresponding to the ports available for the switch. While the set of states S^i corresponds to the agent's input which is composed of three values: *(i)* the packet's destination, *(ii)* the packet's size and, *(iii)* the port's utilization. Our solution starts from executing the distributed DRL module, which invokes a customized network environment using the gymnasium library. The environment calls the P4Runtime API at startup, which connects to the corresponding switch. Then, for each switch, our API waits for the *digested* packets from the data-plane and forwards it to the DRL module that computes the reward according to three main factors: *(i)* the packet's latency, *(ii)* the links capacity, and *(iii)* the links utilization. In ART, we computed the reward function R^i considering two values: the contribution given by the packet's latency r_2^i and the one given by the chosen port's utilization r_1^i. In detail,

$$r_1^i = \frac{1}{1 + latency} \tag{1}$$

and

$$r_2^i = \frac{port\_utilization}{port\_capacity} \tag{2}$$

To summarize, the reward function for each agent *i* is:

$$R^i = r_1^i - \lambda * r_2^i \tag{3}$$

where the λ is the model's hyperparameter chosen according to the algorithm's performance.

According to the reward function, the DRL module chooses the forwarding port, which is communicated to the P4Runtime API to finally inject this information inside a P4 table used by the ingress pipeline to forward the packet to the chosen next hop. This operation is performed for each received packet in order to train the DRL module.

C. DT module in ART

After every f packets, the trained model is distilled into a DT module that predicts possible forwarding ports according to the same input space. This procedure helps build an interpretable DT model that will be used to construct and retrieve splitting criteria of the tree and inject them directly inside the P4 code in the form of match-action rules.

The choice of generating a DT model and injecting its rules inside the P4 code has two reasons. *(i)* First, it is well known that switches have a limited amount of available resources (*e.g.,* CPU, RAM) [21]. This means that adopting a DRL technique might be unfeasible and could require the use of external modules (*e.g.,* Data Processing Unit) to provide faster data processing capabilities or would require distillation of these techniques into look-up tables (LUT), as in [22]. In this latter case, however, the linear increase in LUT adoption can lead to an exponential increase in memory consumption, which could have a negative impact on the overall performance of the network. *(ii)* Second, with the injection of DT rules directly inside the ingress pipeline's tables, the P4 code does not constantly interact with the API at the reception of a packet, making the entire packet processing phase smoother. Moreover, this conversion is made possible since existing DRL-based routing systems can be viewed as rule-based decision-making systems, and DT models can represent this class of solutions.

To reach this goal, we use a *teacher-student* training methodology [23], where the DT model (*i.e.,* the student) reproduces the trained DRL (*i.e.,* the teacher) by considering the tuple $< action, state >$ generated by the model for each connected switch. In ART, for each episode of the trained DRL model and until the DRL model is not terminated, we predict an action based on the current state of the environment. This value is then applied to the topology, taking the action as input and returning the next state, the reward, and a boolean indicating whether the episode is terminated. For each episode's action, our agent interacts with the environment, transitioning from the current state to the next one. These sets of actions and states are then used to train the DT.

One specific challenge we faced in this conversion is that traditional DT algorithms often produce an excessively large number of branches to match the performance of Deep Neural Networks (DNNs). To address this issue, we utilize two key observations to reduce the number of branches to a manageable level for network operators. Firstly, it is common for appropriate policies to consistently recommend the same action for a significant portion of observed states. By leveraging the output data from the teacher DRL, the DT can effectively reduce the decision process and the number of branches. Secondly, different input-output pairs have varying impacts on the policy's performance. To address this, we employ a specialized resampling technique [24] that enables the teacher DRL to guide the DL in prioritizing actions that lead to the best outcomes.

In conclusion, in ART we limit the DT branches to the

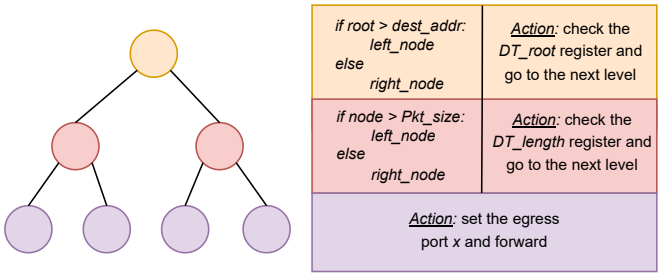


Fig. 2: Workflow of the injected DT model. The left side of the figure shows the level of the DT as just trained from the DRL. The right side shows the cascade of *if-else* conditions according to the splitting criteria and the related performed action.

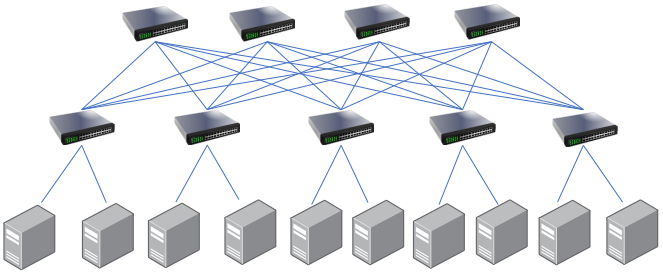


Fig. 3: Fat-tree network topology used in our evaluation.

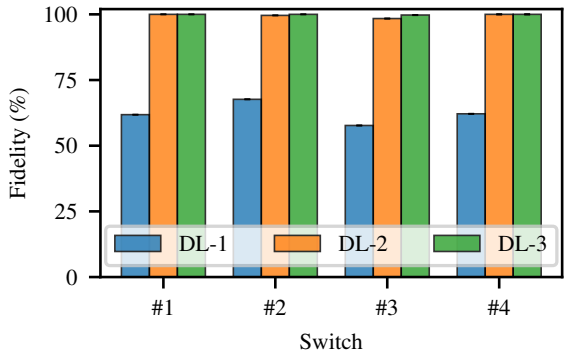


Fig. 4: Accuracy of the distilled DTs expressed as fidelity compared to original DRL models.

first two levels of depth. This choice is mainly motivated by implementation reasons and switch constraints, along with an evaluation of similar performance for other injected level depths. While at first thought, one might think that installing more rules would lead to a higher level of accuracy, in reality, this is not always true: injecting many rules and if-else conditions into P4 can significantly increase the complexity of the code, resulting in a slower processing time, thus degrading the overall performance [25].

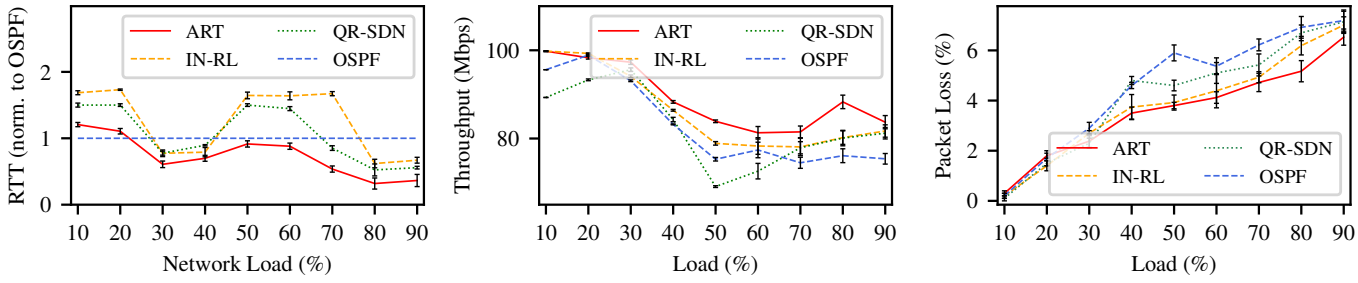
When constructing the DT, the splitting criteria rely significantly on two key values: the packet's destination address and its size. These values are injected into the ingress pipeline of the P4 code with the P4Runtime API, which facilitates the dynamic configuration and orchestration of network policies. This API stores these values into specific registers based on the corresponding DT level, which are then queried using a cascade of *if-else* conditions to find the appropriate forwarding port. An example of this process is illustrated in Fig. 2. The right side of the figure shows an example of the possible DT's splitting criteria. In the root layer of this example, the condition is to verify whether the root value, previously injected into a register by the P4Runtime API, is lower than the received packet's destination address. It is important to note that the DRL model encodes the root value of the studied topology so that each destination address corresponds to a single value. This encoded value is then received by the DT during its training phase. Our packet first undergoes a size check at the first level to verify if it is smaller than the value received from the API. It subsequently progresses to the second level, where it is set to be forwarded to the leaf-output egress port.

In conclusion, as mentioned earlier, to further make this approach feasible, the DT generation and injection process is performed every f received packets, allowing ART to contact the centralized controller with a low frequency.

III. Evaluation

To measure ART's performance, we deployed it on a simulated environment using Mininet, a network emulator specifically designed for software-defined networks (SDN). This powerful emulator can be used in conjunction with the Behavioral Model Version-2 (BMV2) software to deploy P4-programmable switches. To do so, Mininet allows network programmers to run and debug their code into packet-processing pipeline actions of the C++11 software switch as it would be on real hardware.

We evaluated our solution over a data-center topology composed of 10 servers and 9 switches arranged in a leaf-spine fashion where all the links have 100 Mbps bandwidth, as visible in Fig. 3. In all the experiments, we simulated congestion with *iperf3*, a networking tool used to measure performance according to different settings (*e.g.,* throughput, protocol). Adopting *iperf3* allowed us to reproduce increasing network loads by sending packets from each server to the others, thus verifying how the network behaves when congested and replicating the transmission as in [26]. At increasing network loads, we evaluated ART's performance by computing the RTT, throughput, and packet loss while also determining the confidence interval at a 95% level. Finally, we compared these results to relevant works: a traditional routing protocol such as OSPF, which forwards packets using a longest prefix match-action criteria; a centralized SDN solution, QR-SDN [9], that uses a tabular reinforcement learning (RL) model to perform routing across the network; and a prototyped in-network RL, referred to as IN-RL, where the RL model runs directly in the P4 switches by using C++ external functions, as described in [27].



(a) RTT (b) Throughput (c) Packet loss

Fig. 5: Performance comparison with benchmarks in terms of (a) RTT (normalized to OSPF for clarity), (b) throughput, and (c) packet loss, at varying network load in the topology. ART can jointly improve these three metrics.

A. *Evaluation Results*

To evaluate our solution, we started by generating increasing traffic using *iperf3* to quantify the routing strategy at different network loads. We start our evaluation process by measuring the fidelity between the trained DRL model and the distilled DT for the top four switches of the network (Fig. 4). The fidelity is computed as the number of times the actions of DT match the ones of the original DRL model: having a high fidelity ensures that the decision-making process of a more lightweight DT matches the learned behavior of a more complex model. We measure such a fidelity for three depth levels of the DT, namely, depth level 1, $DL-1$, depth level 2, $DL-2$, and depth level 3, $DL-3$, when the tree has three layers. As visible from the figure, while one single layer in the DT leads to an average 60% of fidelity, two levels of the DT already achieve a satisfying level of fidelity, on average 99.9%. Although we do not see fidelity degradation even with more deep trees, we limit the DT to the first *two* layers. This decision also helps to contain the complexity of the resulting P4 code, where each split decision from the DT is stored in specific registers. By limiting the DT injection to the first two levels, we ensure that the P4 code remains manageable and efficient, as it focuses on critical decision points without introducing extremely long if-else conditions, a potential bottleneck in scenarios where real-time decision-making is essential.

We then report the comparisons of our solution against other benchmarks in Fig. 5. In Fig. 5a, we computed the Round Trip Time (RTT) for all the aforementioned solutions at increasing network load and normalized the results to the traditional routing protocol OSPF. We can see that when the network load is low (from 10% to 20%), all the RL-based solutions perform worse than OSPF. This outcome is a clear consequence of the overhead introduced by any ML-driven approach, showing how the computational time of adopting RL models can heavily affect the forwarding decision, thus bringing higher RTTs. However, benefits can be clearly seen when the network starts to be more congested. When the network load increases, the RL-based solutions perform better than the traditional routing protocol. In particular, ART is able to achieve lower RTTs at all network loads despite a heavily congested network (from 30% of network load). Conversely, other benchmarks perform better than OSPF only when the network is either not too congested (from 30% to 40%) or heavily congested (from 70% to 90% of network load).

We then visualize the average traffic throughput that our network can offer at different network conditions (Fig. 5b). The figure shows how ART can achieve higher throughput than the other solutions despite the network's congestion level. The learned DTs of the switch can efficiently react to the upcoming network congestion and re-route packets over an unloaded path. It is important to remember that the throughput does not necessarily align with the evaluated RTT, as it is independent of the sent or received packets. Therefore, this result about throughput is particularly important as it demonstrates that ART can preserve the throughput of TCP flows. In general, we can also observe how the idea of in-network machine learning well fits routing problems, and IN-RL leads to higher throughput than traditional centralized solutions as QR-SDN.

Another important measure to consider when assessing a solution is packet delivery, which shows how well the network reacts to congestion. A significant number of packet losses results in poor transmission quality since the client might repeatedly send packets, which circumstance badly affects our host's performance and network resources. Fig. 5c shows the percentage of packet loss at increasing network loads when sending UDP packets. For this test, we select UDP as the transmission protocol because TCP already implements a re-transmission process on the sending host and we are interested in network-oriented decisions. It is visible from the figure that when our network is not heavily congested (from 10% to 30% of load), all solutions behave quite similarly, leading to a few losses. However, when the network load increases (from 50% to 90%), ART is able to deliver more packets than the alternatives. Having a reactive mechanism running directly in the switches allows ART's implementation to quickly react to congested scenarios, notably reducing overhead in telemetry and re-configuration.

IV. Conclusion

In this paper, we introduce ART, a decentralized machine learning (ML) solution utilizing the innovative Deep Reinforcement Learning (DRL) mechanism to enhance the routing process within the network. Notably, to fit the constraints introduced by P4 programmable switches, such a power and memory-consuming model is transformed into a simpler Decision Tree (DT). The routing strategy enables the consideration of current link loads, facilitating packet rerouting through less congested paths. In our experimental evaluations, we compare ART against three benchmarks: a traditional routing implementation, a centralized RL-based solution, and a hybrid solution where the RL model runs inside the switch. Preliminary results demonstrate that the absence of interaction with a centralized SDN controller, along with simple yet accurate DT, reduces RTT and packet loss, even under network congestion, while also achieving superior throughput.

Acknowledgments

This work was supported by NSF Award #2201536, and by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunication of the Future" (PE00000001 - "RESTART").

References

[1] 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.

[2] 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*, pp. 1–1, 2023.

[3] 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.

[4] 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.

[5] Z. Mammeri, "Reinforcement Learning Based Routing in Networks: Review and Classification of Approaches," *IEEE Access*, vol. 7, pp. 55 916–55 950, 2019.

[6] 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.

[7] 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.

[8] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. da Fonseca, "DRSIR: A Deep Reinforcement Learning Approach for Routing in Software-Defined Networking," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4807–4820, 2021.

[9] 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.

[10] 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.

[11] 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.

[12] F. Musumeci, A. C. Fidanci, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-Learning-Enabled DDoS Attacks Detection in P4 Programmable Networks," *Journal of Network and Systems Management*, vol. 30, pp. 1–27, 2022.

[13] Y.-S. Lu and K. C.-J. Lin, "Enabling Inference Inside Software Switches," in *20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2019, pp. 1–4.

[14] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "In-Network Machine Learning Using Programmable Network Devices: A Survey," *IEEE Communications Surveys & Tutorials*, 2023.

[15] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "NAIL: A Network Management Architecture for Deploying Intent into Programmable Switches," *IEEE Communications Magazine*, 2023.

[16] M. Hogan, S. Landau-Feibish, M. Tahmasbi Arashloo, J. Rexford, D. Walker, and R. Harrison, "Elastic switch programming with p4all," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, 2020, pp. 168–174.

[17] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE access*, vol. 9, pp. 87 094–87 155, 2021.

[18] P4Runtime Spec. Accessed: 2023-12-27. [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[19] 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.

[20] D. Monaco, A. Sacco, E. Alberti, G. Marchetto, and F. Esposito, "A collaborative and distributed learning-based solution to autonomously plan computer networks," in *2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2023, pp. 1–5.

[21] M. Blöcher, L. Wang, P. Eugster, and M. Schmidt, "Switches for hire: Resource scheduling for data center in-network computing," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 268–285.

[22] L. De Marinis, E. Paolini, R. A. Bakar, F. Cugini, and F. Paolucci, "Cascaded look up table distillation of p4 deep neural network switches," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 2111–2116.

[23] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting Deep Learning-Based Networking Systems," in *Proc. of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20)*, 2020, pp. 154–171.

[24] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable Reinforcement Learning via Policy Extraction," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[25] H. Stubbe, S. Gallenmüller, M. Simon, E. Hauser, D. Scholz, and G. Carle, "Keeping up to date with p4runtime: An analysis of data plane updates on p4 switches," in *International Federation for Information Processing (IFIP) Networking 2023 Conference (IFIP Networking 2023)*, 2023.

[26] 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.

[27] 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 *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 517–522.