2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN) | 979-8-3503-9804-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICIN56760.2023.10073505

A Collaborative and Distributed Learning-Based Solution to Autonomously Plan Computer Networks

Doriana Monaco * Alessio Sacco * Enrico Alberti * Guido Marchetto * Flavio Esposito ‡
* Department of Control and Computer Engineering, Politecnico di Torino, Italy
‡ Computer Science Department, Saint Louis University, USA

***Abstract*—The high programmability provided by Software Defined Networking (SDN) paradigm facilitated the integration of Machine Learning (ML) methods to design a new family of network management schemes. Among them, we can cite self-driving networks, where ML is used to analyze data and define strategies that are then translated into network configurations by the SDN controllers, making the networks autonomous and capable of auto-scaling decisions based on the network's needs. Despite their attractiveness, the centralized design of the majority of proposed solutions cannot keep up with the increasing size of the network. To this end, this paper investigates the use of a multi-agent reinforcement learning (MARL) model for auto-scaling decisions in an SDN environment. In particular, we study two possible alternatives for distributing operations: a collaborative one, where controllers share the same observations, and an individual one, where controllers make decisions according to their own logic and share only some basic information, such as the network topology. After an experimental campaign performed both on Mininet and GENI, results showed that both approaches can guarantee high throughput while minimizing the set of active resources.**

***Index Terms*—software-defined networking, distributed learning, reinforcement learning**

I. Introduction

The presence of new requirements, e.g., high reliability, zero packet loss, and real-time interaction, posed by data-intensive applications, e.g., augmented/virtual reality, industrial 4.0, or healthcare, exacerbates the need for more performant, scalable, resilient, and self-adapting networks [1], [2]. To support such applications, there is a need to rethink the design of both networks and applications, creating more intelligent and autonomous networks. Next-generation networks are envisioned as the answer to network operators and service providers to replace existing infrastructures and to introduce a new platform able to support new telecommunication businesses and services. They are considered key enablers for delivering new services that are available to any place, at any time, on any device.

To provide such (artificial) intelligence, there is an increasing interest in equipping networks with autonomous run-time decision-making capabilities incorporating distributed machine learning (ML) algorithms, to foster automation in network configurations, network management, and network resiliency. While AI/ML technologies continue to evolve at a rapid pace, moving from a paradigm of supervised learning towards distributed self-learning requires solving several challenges in the design and deployment of wide-scale networks. Among those challenges, the scalability of AI/ML models for managing the Next Generation Internet is particularly critical.

Recent studies have partially addressed some of these challenges by employing model-free approaches to efficiently manage network resources. In particular, Reinforcement Learning (RL) finds profitable applicability given its ability to fit the network dynamics well without any prior knowledge [3]–[6]. With such auto-scaling solutions, networks can deactivate idle resources that may increase unnecessary (energy) costs and provide redundant facilities to face workload peaks or unexpected failures. However, current solutions fail to manage large-scale networks as they are limited by the single-controller architecture. In this paper, we argue that a multi-agent approach is needed to overcome limitations. At the same time, distributed controllers raise new challenges to be addressed, for example, controllers must cooperate by sharing their information in order to make the best decision.

In this paper, we propose and evaluate two different ways of distributing information among controllers. A first *collaborative distributed* approach, in which controllers share their local network information to achieve a common global view of the network state to make the finest decisions; and a second *individually distributed* approach, in which controllers autonomously manage their own network area and only share topology information or its changes.

In addition to evaluating different control plane architectures, we also study the impact that different data plane technologies can have. In particular, we implemented a network based on P4-enabled switches and OpenFlow-enabled switches, where the network is emulated over Mininet and then replicated over a real-world testbed as GENI. Regarding the data-plane level, we experienced how P4 implementation is more efficient, while performance with OpenFlow are fluctuating. Regarding the control-plane level, we have observed that the *individually distributed* approach, despite its simplicity and less overhead, leads to similar results, thus being a valid solution in a real deployment.

II. Related work

Since SDN centralized controller architectures are limited in terms of scalability, reliability, and availability, interest has grown in physically-distributed control plane architectures, which can face big dynamic networks such as data centers and WANs. Onix [7], ONOS [8], and HyperFlow [9] are examples of multi-controller architectures that operate on a

global view of the network through the distribution of their state, but differ for the data structures they employ and their replication mechanisms. While in Onix and ONOS, controllers are aware only of a portion of the managed network and later share with others information to construct a global view, in Hyperflow the controllers have the same global view of the network and run with the illusion to have control over all of it. Opposed to them, there are logically distributed architectures, such as the one proposed by DISCO [10], where each controller takes decisions based on its local view, distributing only topology information that is necessary for basic services, e.g. routing. Having a per domain structure, DISCO provides a distributed control plane that can manage large and heterogeneous networks such as WANs.

In this paper, two different controller designs are compared: collaborative and individually distributed architecture. The former, similarly to Onix and ONOS, deploys multiple controllers that share their network state through Raft [11] to achieve a consistent view of the entire network, later used for the RL decision-making process. In the latter, similarly to DISCO, controllers feed the model with their local network state and are equipped with a TCP channel for inter-controller communications in order to trade topological information for routing.

At the same time, literature about automating network operations is rich [12]. An RL approach to automate networking tasks has been proposed with DeepConf [13]. In their auto-scaling use case, the model learns the best links to activate at each step based on a double state input composed of a traffic matrix and a network topology matrix. Some studies applied RL to optimize power-consumption by consolidating traffic in the minimum amount of links, rewarding power saving and flow completion time [14] or using a deep learning model to predict the traffic and activate the sleep/wake up model for some switches and links based on current and predicted traffic [15]. The main difference between our work and other solutions relies on the controller design, since we employ multiple controllers and manage every network area with a different model instance. Although Multi-Agent Reinforcement Learning (MARL) has been used in other domains, e.g., routing [16] or VM consolidation [17], the usage of possible distributed architecture in auto-scaling networks is mostly unexplored.

III. Multi-Agent Controllers Design

With this paper, we attempt to move towards a fully automated network, often referred to as *self-driving* network. Self-driving networks can measure, analyze, and control themselves in an automated manner, reacting to environmental changes, e.g., traffic volume, while adjusting and optimizing themselves as needed [18]. In particular, in this paper, we propose an autonomous network planning framework that can dynamically scale up and down network resources as traffic volume changes to optimize resource utilization while guaranteeing high applicative throughput. The basic idea of our solution is to activate supporting switches that are typically unused, but that can be leveraged to deal successfully with congestion. On the contrary, to consolidate resources, they can be deactivated in order to save computing power.

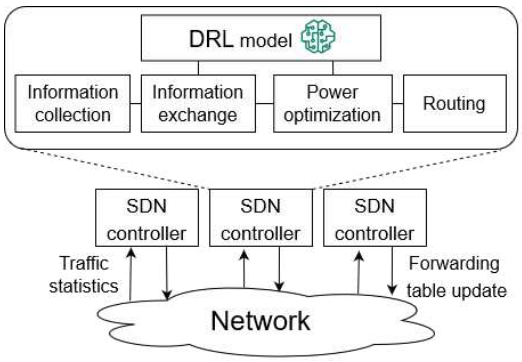


Fig. 1: System architecture. The SDN controllers receive in input the network statistics and, using the DRL model, select the best set of network resources to deploy. According to this decision, routing tables should be appropriately updated.

We sketch the main operations of our solution in Figure 1. First, the multiple SDN controllers deployed collect and maintain traffic statistics from local switches and calculate the throughput of flows. Based on the chosen design, specific information are exchanged with other controllers and the network state is built up. The RL model generates the best action to take based on the input received and gets rewarded. The action list may change the status of supporting switches. In these circumstances, the virtual topology view is modified and re-routing is performed, preferring less congested paths.

Our solution employs a Deep Reinforcement Learning (DRL) approach where the agent observes the state of the environment, our network, and generates an action that will alter the environment. Every action is rewarded with a scalar value to learn the best policy to actuate. The DRL agent receives the inputs, selects the best action, uses the reward value to evaluate the goodness of the chosen action, and proceeds with this process continuously. We deploy the Deep Q-Learning algorithm that, making use of two neural networks (*main* and *target* network) with different weights, increases stability during training. Differently from vanilla Q-Learning, the neural network in this implementation maps input states to pairs of (action, Q-value), where the *Q-value* is the maximum estimated return for taking action into a given state, but equivalently updates the neural network weights conforming to Bellman Equation. During the training phase, we adopt the *Epsilon-Greedy* policy in order to balance exploration and exploitation. The model initially explores the environment by taking randomized actions with a certain probability that decreases as the training goes on, in order to later exploit and evaluate the temporarily learned policy. To optimize the learning process, the RL agent periodically performs *experience replay*. The learning phase is separated from acting and relies upon randomly sampled batches of recorded data. The neural network output is mapped into a distribution of probabilities by a *Softmax* layer, and the action with the highest probability is then chosen. As in any other RL-based algorithm, we need

to define the main variables of the framework: state, action, and reward. In our solution, we have the following mappings:
State space: The agent state space is based on the load of links. The number of links considered as input depends on the specific distribution framework selected (see Section III-A).
Action space: For each supporting switch, the model defines if such a switch has to be powered on or off.
Reward: The purpose of the model is to find the best resource allocation that maximizes the network-aware reward, where our objective is to detect the minimal set of active resources that can satisfy traffic demands. With this in mind, we construct the reward function considering the network performance and a penalization for resource overprovisioning to discourage a scenario where supporting switches are active but unhelpful to traffic. The reward equation results as the average network throughput deducted by the power associated with each activated supporting switch.

A. Two Multi-Agent Network Management Approaches

We now define the two possible alternatives that we designed to manage multiple sub-networks.
Collaborative distributed. The first design choice we propose deploys multiple controllers training DRL agents that learn how to produce collaborative behaviors.In particular, the state space we use is the global information on links utilization plus the state of supporting switches, i.e., whether they are active or not. In such a manner, agents can see the effect of their joint action in the network. Actions about the planning of network resources of the SDN controller, instead, regard the single sub-network. In other words, the model is logically centralized, but it runs in a multi-agent setting for reliability and scalability reasons. To gain a consistent network-wide view, metrics of local networks are aggregated and then distributed through Raft to all the controllers. At this point, the model input is built and sent to RL agents that will act as discussed previously.
Individually distributed. Opposed to the aforementioned design, we also present a fully distributed solution. The leading idea is to train agents independently of each other to make a comparison between the learned behaviors and reached performance of the two study cases. When controllers are instantiated, they have knowledge of their local network only, which they will autonomously manage all along. To guarantee essential services, e.g., routing, they are equipped with a TCP channel to share local information about hosts and topology that they will 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 nor model parameters. The main difference compared to the collaborative version resides in the state space: The input of each model is determined by local link utilization rather than global network utilization.

IV. Evaluation

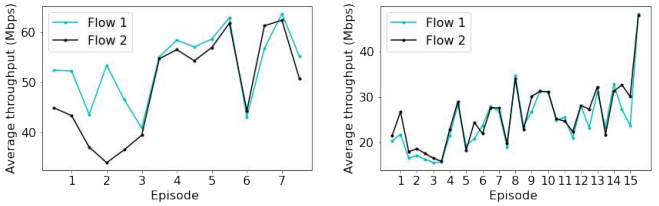
In this section, we analyze and compare the two designs proposed. We first deployed the network over Mininet [19], a network emulator that allows reproducing arbitrary virtual networks. We test performance when switches are programmed with the two most used languages: first P4-enabled and then OpenFlow (OF)-enabled switches. The network topology used contains 13 nodes, where 3 of them are supporting switches that can be activated/deactivated. We also employ 3 SDN controllers to manage different sub-networks. Half of the clients send iPerf3 traffic with custom scripts that, for each training episode, alternate highly rated traffic and weakly rated traffic.

Fig. 2 presents the average throughput for two concurrent flows during the training phase of the agents in the two management designs, and for both P4 and OpenFlow protocols. For each setting, we stop the monitoring when the reward converges. As training proceeds, the exploration probability decreases in favor of exploitation, and as our model learns the best policy, throughput is encouraged. This increasing trend highlights the fact that our program learns to maximize the designed reward function, and it reaches better network performance than a random policy. We can observe how, in the case of collaboration (Fig. 2a), the throughput is more stable. Moreover, in the case of P4 the throughput is notably higher than OpenFlow.

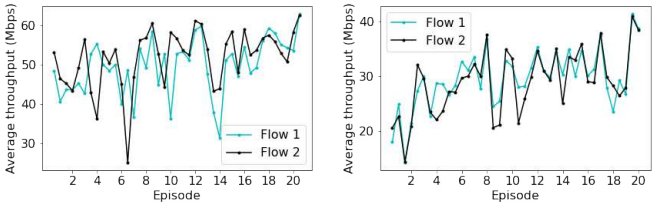
We further investigate these differences and analyze the impact that the data plane technologies can have. Fig. 3 provides a closer look at the throughput achieved with P4 and OpenFlow protocols during the test phase of the RL model. When using P4, the maximum throughput (as our network setup permits) is reached. All OpenFlow-based implementations face a lower and unstable throughput. This poor performance may be due to the higher number of rules configured in OpenFlow switches, which affect configuration and look-up time. If P4-based switches let us program one rule per destination address, this could not be possible with OpenFlow.

Since our model should also detect when resources are needless, we simulated congestion in the sub-network 2, followed by less intensive traffic. We show in Fig. 4 the throughput after the model has been trained. It can be observed that the corresponding controller powers it off when the traffic decreases (around 30 seconds), and it also autonomously decide to power on the supporting switch to sustain the increment in traffic (around 60 seconds). As far as concerns the collaborative solution, even though simulated flows traverse the whole network and all controllers are aware of the congestion, our model can identify the best spot where additional resources overcome congestion. It is clear from the figure that other agents keep supporting switches powered off. Whereas, in the individually distributed design, controllers don't know about congestion in other sub-networks and learned to act independently for the sake of their local network.

From the graphs it appears how, despite the conceptual difference between the two implementations, the experienced behavior produces similar results, with a slightly higher throughput for the collaborative framework. This result is particularly important since it can suggest that an individual distributed setting, which requires less exchange of packets among controllers, can still obtain acceptable performance.

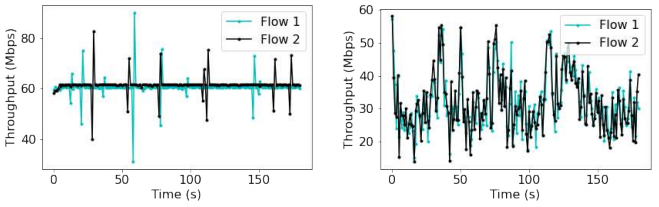


(a) Collaborative distributed (P4) (b) Collaborative distributed (OF)

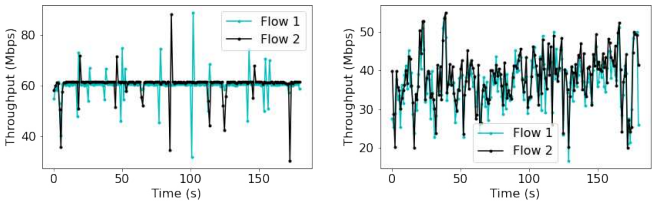


(c) Individually distributed (P4) (d) Individually distributed (OF)

Fig. 2: Throughput of flows during training. While differences between collaborative or individual is negligible, the throughput differs for P4 or OpenFlow (OF).



(a) Collaborative distributed (P4) (b) Collaborative distributed (OF)

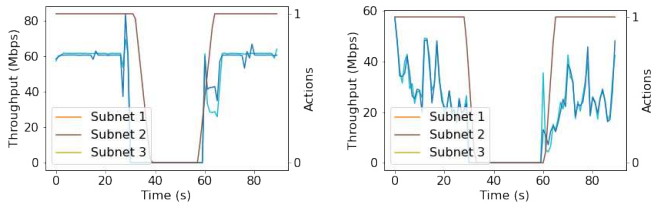


(c) Individually distributed (P4) (d) Individually distributed (OF)

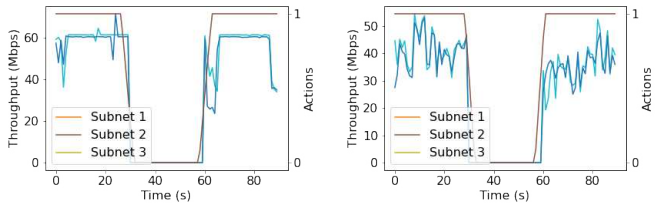
Fig. 3: Throughput of flows during testing. OpenFlow metrics are more jumpy.

Regarding the switch languages then, with the P4 implementation, the achieved throughput is the maximum allowed by our network setup. As OpenFlow networks reach lower throughput in respect of P4, we can also observe a more stable and higher throughput in our collaborative distributed solution.

Lastly, to understand the convenience of an auto-scaling system, we compare our model performance against a minimal setting and a network with always active resources that implement ECMP routing. Fig. 5 demonstrates that our independent version of logic distribution can achieve pretty similar performance to an *always on* case where switches are always active. However, in addition to this, our reactive logic can also reduce energy consumption powering off resources when they are not needed.

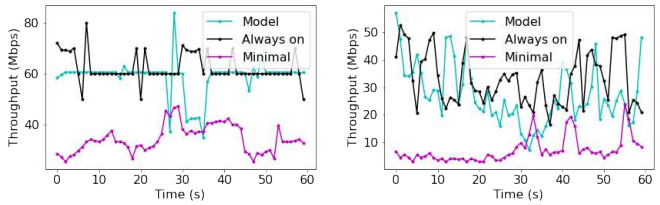


(a) Collaborative distributed (P4) (b) Collaborative distributed (OF)

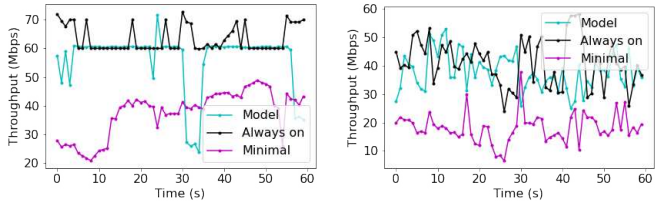


(c) Individually distributed (P4) (d) Individually distributed (OF)

Fig. 4: Evolution of throughput (left side) and actions (right side) in the three network regions.



(a) Collaborative distributed (P4) (b) Collaborative distributed (OF)



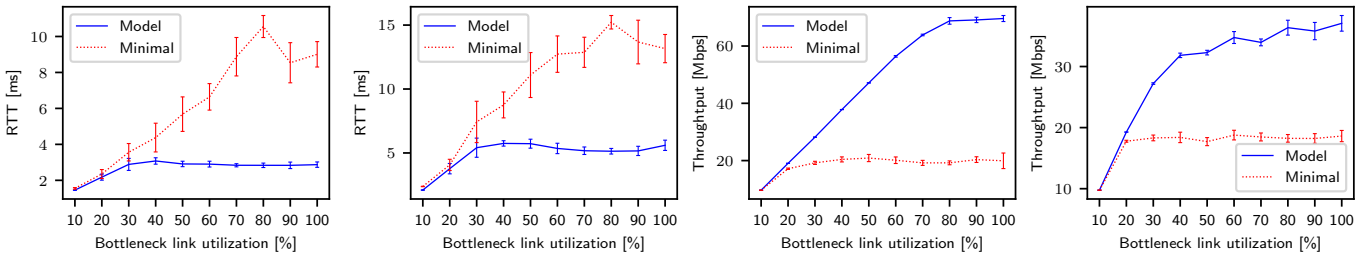
(c) Individually distributed (P4) (d) Individually distributed (OF)

Fig. 5: Throughput evolution for a minimal setting, always-on configuration, and our model.

A. GENI evaluation

To establish the practicality of our approach and understand how our solution performs over geographically distributed physical nodes with real cross-traffic and real packet schedulers, we deploy our solution on the GENI testbed[1]. Experiments are performed on a topology consisting of 9 nodes (referred to as *small*), and a larger composed of 18 nodes (referred to as *large*). Switches run OpenFlow rules. We compare the auto-scaling solution driven by the RL model against a minimal scenario where RL is not employed. In this case, the hosts used the shortest path configured by the Ryu controller. The second version requires the RL model, which can activate the path with the supportive switches.

[1]https://www.geni.net/



(a) small topology (b) large topology (c) small topology (d) large topology

Fig. 6: **(a) - (b) RTT evolution** when increasing bottleneck traffic. **(c) - (d) throughput evolution** over the GENI testbed. The auto-scaling schema can jointly reduce RTT while maximizing throughput.

Figure 6a-b shows the average RTT of 2 concurrent communications when increasing the bottleneck link utilization in the (a) small and (b) larger topology. It can be seen that when the bottleneck utilization starts increasing (more than 30% in small topology), the reactive planning schema is able to maintain the RTT stable and limited to around 2.5 ms. The most significant difference happens with high utilization, as our algorithm can find adequate paths to accommodate the incoming traffic. In the absence of such a mechanism, the RTT would reach 12 ms, which value can undermine the overall application transmissions. Similarly, for large topology, the RTT is kept limited. In Figure 6c-d we report the throughput of transmissions in the same conditions as before. We can clearly observe that, despite an increment in network utilization, our RL-based model can preserve the throughput in both circumstances. In the small topology (Figure 6c), the throughput can reach 70 Mbps, while in the large network, the throughput is up to 37 Mpbs due to the more clients transmitting at the same time.

V. Conclusion

In this paper, we addressed the need of managing virtual networks with multiple controllers in an automated manner, presenting two possible designs. In a *collaborative* framework, the models running on the controllers operate on a global network view, while in an *individual* framework, the controllers autonomously orchestrate their network area. In addition to diverse control-plane settings, we evaluated diverse data-plane languages, experiencing how tests with P4-enabled switches are more stable. Both logic distribution approaches are capable of deploying appropriate reaction changes based on the traffic load and seem to achieve similar performance. This outcome will be further investigated in the future with more realistic traffic conditions and larger topologies.

Acknowledgment

This work has been partially supported by the European Commission under the NGI Atlantic initiative, and by the National Science Foundation awards # 1908574 and # 2201536.

References

[1] A. Aijaz, M. Dohler, A. H. Aghvami, V. Friderikos, and M. Frodigh, "Realizing the tactile internet: Haptic communications over next generation 5g cellular networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 82–89, 2016.

[2] A. V. Ventrella, F. Esposito, A. Sacco, M. Flocco, G. Marchetto, and S. Gururajan, "Apron: An architecture for adaptive task planning of internet of things in challenged edge networks," in *IEEE 8th International Conference on Cloud Networking (CloudNet)*. IEEE, 2019, pp. 1–6.

[3] D. Zeng, L. Gu, S. Pan, J. Cai, and S. Guo, "Resource management at the network edge: A deep reinforcement learning approach," *IEEE Network*, vol. 33, no. 3, pp. 26–33, 2019.

[4] A. Sacco, F. Esposito, and G. Marchetto, "Supporting sustainable virtual network mutations with mystique," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2714–2727, 2021.

[5] D. Lee, J.-H. Yoo, and J. W.-K. Hong, "Deep Q-Networks Based Auto-Scaling for Service Function Chaining," in *International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–9.

[6] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin, "Network planning with deep reinforcement learning," in *Proceedings of the 2021 ACM SIGCOMM Conference*. ACM, 2021, pp. 258–271.

[7] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu *et al.*, "Onix: A distributed control platform for large-scale production networks," in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[8] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[9] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the Internet network management conference on Research on enterprise networking (INM/WREN 10)*, vol. 3. USENIX Association, 2010, pp. 10–5555.

[10] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed Multi-Domain SDN Controllers," in *2014 IEEE network operations and management symposium (NOMS)*. IEEE, 2014, pp. 1–4.

[11] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference (ATC 14)*, 2014, pp. 305–319.

[12] A. Sacco, F. Esposito, P. Okorie, and G. Marchetto, "LiveMicro: An Edge Computing System for Collaborative Telepathology," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[13] S. Salman, C. Streiffer, H. Chen, T. Benson, and A. Kadav, "DeepConf: Automating Data Center Network Topologies Management with Machine Learning," in *Proceedings of the 2018 Workshop on Network Meets AI & ML (NetAI'18)*. New York, NY, USA: Association for Computing Machinery, 2018, p. 8–14.

[14] P. Sun, Z. Guo, S. Liu, J. Lan, J. Wang, and Y. Hu, "Smartfct: Improving power-efficiency for data center networks with deep reinforcement learning," *Computer Networks*, vol. 179, p. 107255, 2020.

[15] X. Chen, X. Wang, B. Yi, Q. He, and M. Huang, "Deep learning-based traffic prediction for energy efficiency optimization in software-defined networking," *IEEE Systems Journal*, vol. 15, no. 4, pp. 5583–5594, 2021.

[16] X. You, X. Li, Y. Xu, H. Feng, J. Zhao, and H. Yan, "Toward packet routing with fully distributed multiagent deep reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020.

[17] K. Haghshenas, A. Pahlevan, M. Zapater, S. Mohammadi, and D. Atienza, "Magnetic: Multi-agent machine learning-based approach for energy efficient dynamic consolidation in data centers," *IEEE Transactions on Services Computing*, vol. 15, no. 1, pp. 30–44, 2022.

[18] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *arXiv preprint arXiv:1710.11583*, 2017.

[19] R. L. S. de Oliveira *et al.*, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014, pp. 1–6.