RLVNA: a Platform for Experimenting with Virtual Networks Adaptations over Public Testbeds

RLVNA: a Platform for Experimenting with Virtual Networks Adaptations over Public Testbeds

Antonino Angi * 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*—Network emulators and simulation environments traditionally support computer networking and distributed system research. The continued use of multiple approaches highlights both the value and inadequacy of each approach. To this end, several large-scale virtual networks testbeds, such as GENI and CloudLab, have emerged, allowing testing of a networked system in controlled yet realistic environments, focusing in particular on facilitating the test of network management schema in Software-Defined Network (SDN) scenarios. Nevertheless, setting up those experiments first and integrating machine learning models later in these deployments is challenging. In this paper, we propose designing and implementing a web-based platform that integrates Reinforcement Learning (RL)-based models with a virtual network experiment using resources acquired within a real-world testbed, e.g., GENI. Users are able to reserve the network resources (links, switches, and hosts) and configure them through our intuitive interface with little effort. The RL algorithm is then launched to learn how to steer traffic dynamically and according to diverse traffic network conditions. Such a model can be easily customized by the user, while our architecture enables fast reprogramming of the Open Virtual Switches via the SDN controller instantiated. We experimented with trace-based traffic to validate this user-friendly platform and evaluated how centralized and decentralized RL algorithms can effectively lead to self-driving networks. While in this paper, the system focuses on the deployment of experiments for virtual network adaptation, the platform can be easily extended to other network management mechanisms and machine learning algorithms.**

I. Introduction

Next-generation networks are envisioned as the solution for network operators and service providers who wish to upgrade their existing infrastructures and introduce a versatile platform that can support a wide range of telecommunication businesses and services. These networks are considered key enablers for delivering new data-intensive applications, e.g., augmented/virtual reality, industrial 4.0, or healthcare [1], [2]. The presence of new requirements, such as high reliability, zero packet loss, and real-time interaction, posed by these new services exacerbates the need for more performant, scalable, resilient, and self-adapting networks. Moreover, as the number of nodes increases, management becomes increasingly complex and impossible for a single person or team to manually configure and handle what is happening in real time [3], [4]. To support such applications, there is a need to rethink the design of both networks and applications, creating more intelligent and autonomous networks.

It is thus appearing that 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 [5]. 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 (above all) deployment of wide-scale networks. Among those challenges, two of them are particularly relevant to the requirements of the next-generation networks: the scalability of AI/ML models for network management, and the robustness of learning solutions in practical deployments. In particular, while the use of Reinforcement Learning (RL) can lead to automated networks [6]–[8], the entire design and experimentation process is still tedious. One common challenge these solutions face is the difficulty of testing them on real networks with actual traffic. While most solutions are evaluated using simulation tools like Mininet [9] or ns-3 [10], it is important to conduct experiments on real-world testbeds like GENI [11], FABRIC (which is still under development) [12], Chameleon Cloud [13], and CloudLab [14], which provide platforms for conducting replicable experiments. While these platforms are accessible in multiple countries and allow node reservation for research purposes, they often lack intuitive user-friendly interfaces for conducting large-scale ML-based network planning experiments. Configuring the desired scenario can be tedious, forcing the research to deal with low-level details of virtualized and real hardware, and efficient resource allocation is not always immediately attainable.

Researchers have initiated the development of systems aimed at facilitating the allocation of resources on such testbeds. For example, SAM [15] is a framework that can semantically represent and process data in a way that is understandable by machines for managing resources of federated IoT testbeds. SAM manages the resources' lifecyle and the federation of devices, focusing in particular on the federation of unmanned vehicles (UxVs). The problem of resources' allocation in networking testbeds is then studied in [16], where the authors provide an algorithm that optimizes the utility of the NITOS testbed, a collection of heterogeneous resources for wired and wireless networks [17]. While valuable attempts to facilitate network experimentation, users still need to learn intricate syntax to interact with the allocated resources and it is challenging to run ML-based (and RL-based in particular) algorithms to solve typical networking problems.

Trying to fill these gaps, in this paper, we present RLVNA,

a novel system that aims at simplifying the testing and run of this class of algorithms even for people with little experience in DevOps, testbeds, and infrastructures. Our focus is on providing the enabling technologies to project and deploy RL models, leaving room for the programmers to play with specific parameters, e.g., the number of steps in the RL process and number of neurons in the neural network (most RL algorithms make use of it). To validate our system, we conceived a decentralized RL-based algorithm (that for simplicity we call RLVNA) that extends an existing auto-scaling approach [6], which we implemented and tested first on Mininet and then in our developed platform on GENI [11]. This use case demonstrates the advantages of this type of proactive algorithms for network adaptations and the effectiveness of the overall system for SDN experimentation.

The rest of the paper is structured as follows. Section II describes the main functionalities proposed by RLVNA, while Section III presents the experimental results. Finally, Section IV concludes the paper.

II. System Design

In RLVNA, we designed a user-friendly platform that allows independent network experiments on a wide range of testbeds. At startup, the user is only required to choose the topology of interest and set up the customizing settings for the ML algorithm, while the platform automatically handles the IP address assignation and resource allocation. This strategy allows even less experienced users to manipulate their network.

A. Solution Architecture

Fig. 1 provides an overview of our solution, highlighting the three key components that make up our platform: the front-end interface, the decision logic application, and the testbed manager. These elements work together seamlessly to provide a comprehensive network experimentation environment that is user-friendly and efficient. Our *front-end interface* provides a web interface where the user can specify the solutions' topology and parameters. The *decision logic application (DLA)* is the core of our solution and provides the functionalities for the recourse management schema. It includes our RL model that is distributed among the number of controllers specified by the user. Finally, this information is passed to the testbed manager. The last component, the *testbed manager (TM)*, receives the topology and model information in input and dispatches the proper commands. Having a separate component with respect to the network management logic allows separating the concerns: the DLA handles all the network planning model and logic, and the TM deals with potentially different platforms, e.g., GENI, Cloudlab, and the diverse commands needed to turn up the topology desired for the experiments. This component manages the different commands that various testbeds may possibly have, and once the user request is received, it dispatches the opportune command.

During our experimental campaign, we mainly focused on deploying a topology over a large-scale testbed as GENI and testing the efficacy of an auto-scaling solution in the two fashions: centralized and distributed.

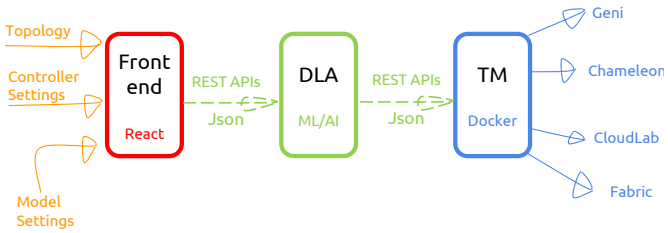


Fig. 1: RLVNA overview. The chosen architecture provides portability and adaptability to different testbeds.

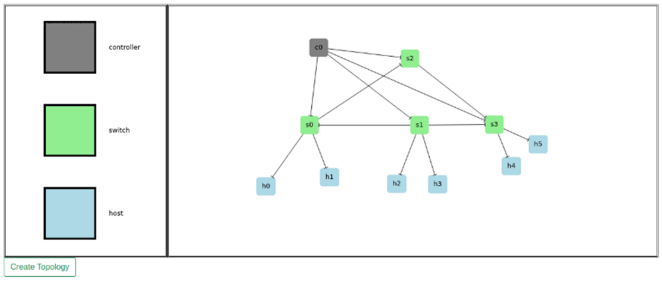
The last and most crucial work is the development of a generalized RLVNA algorithm. The RL algorithm we designed has two central cores. The first core is the presence of a Ryu controller [18] for configuring and querying the switches. The second core is the ML model, which, once received the measurements from the Ryu controller, evaluates whether activating or deactivating the support switches along with their paths is better. The algorithm's purpose is clearly to scale up and down, reducing the congestion in the network when it occurs while simultaneously avoiding energy consumption when not needed.

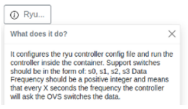
B. Web Interface for the User

The main functionality offered by the project's frontend includes the creation of custom topologies and the ability to run machine learning algorithms for network management within them. While the frontend is designed to be intuitive for the end user, we also include text boxes for additional information and explanations of the settings.

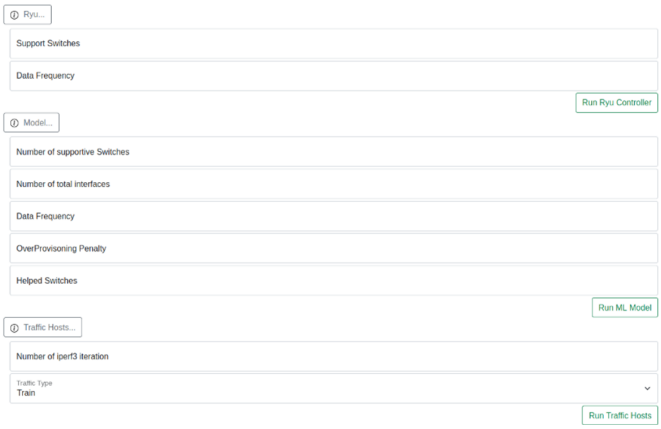
We report in Fig. 2a the web page presented to the user during the network and model deployment. Given that the main purpose of the frontend is to allow users to request resources, customize the reinforcement learning algorithm's settings, and test their algorithms in custom topologies using our platform, we can observe the two main sections that compose this web page: Topology definition at the top, and Machine Learning settings customization at the bottom of the page. The topology definition section features a large box where users can draw the topology they wish to test, while the second half of the page is dedicated to the reinforcement learning algorithm settings. These settings are designed to align with the parameters expected by the RLVNA algorithm we implemented. The necessary steps for running an experiment using our platform are as follows: (i) define and draw the custom topology, selecting among types of nodes available, i.e., switch, hosts, and SDN controller, (ii) configure and run the SDN controller, (iii) configure the ML model, (iv) demand for resource reservation, (v) in case of positive response, obtaining the ssh command to directly log inside the nodes.

By clicking on the "Create Topology" button, the drawn topology is parsed and converted into JSON format, then sent to the Testbed Manager via a REST API. The response type (affirmative or negative) and the response time clearly depend on the underlying testbed, but we assume it may take a few minutes. When such a response is received, we display a simple modal popup containing descriptive text and





(a)



(b)

Fig. 2: Screenshots of RLVNA's web platform

the specification file, e.g., RSpec and Manifest files, reporting the node descriptions, MAC and IP addresses, and physical locations used in the topology.

These topology customizations are closely linked to the parameters of the reinforcement learning algorithm, which means that different machine learning algorithms can be tested very easily in the settings. As shown in Fig. 2b, each subpart (SDN controller, RL model, ssh) is accompanied by an information box that provides details on the expected parameters and the purpose of the subsection. For instance, the Ryu controller configuration requires the list of supported switches and the frequency at which the Ryu controller should request switch statistics. On the other hand, the ML model is highly customizable and needs the number of supportive switches, the data frequency, the overprovisioning penalty, and the list of helped switches. To enable researchers to run various experiments, we provide ssh access to all nodes, including the controller, switches, and hosts.

C. Testbed Manager

Our platform can easily be used even by researchers with a strong background in machine learning but no prior networking knowledge, because the resource deployment and continuous monitoring is carried out by our testbed manager (TM) rather than the user. In particular, we designed the TM of our system to perform the necessary operations for reserving and configuring nodes in different testbed environments. This involves interacting with the platform, e.g., GENI, Cloudlab, to select the aggregate, slice, node types, and networking configurations, such as IP addresses. The TM configures each node, which can be a controller, switch, or host, and then returns a status string indicating whether the operation was successful or not. This process enables researchers to test their ML-based solutions in general, and RL-based in particular, in the selected SDN environment without the burden of handling specific commands.

D. Reinforcement Learning for Virtual Network Adaptations

In this study, we focus on a specific use case where RL finds great applicability: virtual network adaptations [6]–[8]. We thus want to validate the effectiveness of the RLVNA algorithm for optimizing network performance and energy consumption in a possible variety of networks. To achieve this goal, we have dockerized the RLVNA algorithm from the backend and inserted it into a compact and isolated container in the controller nodes. In our setting, all switches in the network will send their statistics to the controller with a particular frequency, and the controller will implement the RLVNA algorithm to optimize the network.

Our proposed solution employs a Deep Reinforcement Learning (DRL) approach where the agent observes the network's state and generates an action that alters the environment. Each action is rewarded with a scalar value to learn the best policy to actuate. The DRL agent selects the best action, uses the reward value to evaluate the chosen action's goodness, and continuously repeats this process. To increase stability during training, we use the *Deep Q-Learning* algorithm, which employs two neural networks with different weights to map input states to pairs of (action, Q-value). During training, we use the Epsilon-Greedy policy to balance exploration and exploitation.The *state space* is based on link load, and *action space* determines whether supporting switches should be powered on or off. The *reward function* considers network performance and penalizes over-provisioning, promoting resource allocation that maximizes network-aware rewards. The average network throughput minus the power consumed by each activated supporting switch constitutes the reward equation. The next step is obtaining the shortest path between one host and the others. We created two data structures, one that contains the shortest path considering the support switch and another data structure that contains the shortest path without considering these switches. It evaluates a reward function and, depending on the value decides the action to send to the controller. We defined two possible actions: support switch ON or support switch OFF. This process is repeated for all the supportive switches existing in the topology, and the action that the ML sends to the controller is an array with the combination of ON/OFF of all the support switches.

When an action is received by the Ryu controller, it triggers a series of operations. If the action involves enabling a "support switch", the first step is to identify the neighboring switches that require flow modification. In addition, the hosts

that can benefit from a new path through the target switch are determined using the shortest path algorithm. Half of these hosts are then directed to use the new path, while the other half continue using the old path. The use of the "support switch" results in an improvement in network performance, which will be discussed in the evaluation section. At the same time, if the action involves disabling a "support switch", the target switches become redundant, and their energy consumption can be reduced. In this case, the first step is to locate the neighbors of the target switch along with their associated hosts. Half of the previously enabled hosts are then redirected to use a path without the target switch, while the remaining half continue using the same path without modification.

To deal with possibly large networks, we modified this instance and defined a distributed framework based on Raft [19], a consensus algorithm that allows a group of machines to be resilient even if some of them fail. We adopted Raft as its capacity to enable technology to reach consensus in multi-controller topologies. In this setting, we set controllers to have a consistent network-wide view by exchanging metrics of local networks through Raft to all the controllers.

III. Results

The main goal of our experiments is two-fold: validating the architecture provided via the web interface, and assessing the feasibility and validity of adopting a centralized or distributed approach for network management decisions. The tests run both over a Mininet and GENI testbed, due to its easiness in deploying SDN architectures.

GENI-based platform. The GENI testbed offers a wide range of aggregate spread across the US, allowing for flexible and customized resource reservation. However, while being extensively used, GENI has many limitations, and collecting statistics and reserving resources proved to be challenging. Even in well-equipped aggregates, node reservation often fails, and configuring simple topologies with four switches could take minutes or hours. This limitation affects the data collection process and the number of configurations we could set up. Our proposed solution moves towards an easier to use testbed. To effectively operate with the GENI environment, in RLVNA our testbed manager (TM) uses the `geni-lib` library for choosing the aggregate, defining the node type, and their networking configuration, such as the IP addresses. The resource reservation is indeed the most significant operation the backend executes. In particular, it deletes the previous allocation in the slice, parses the frontend request, creates the Rspec request, stores the manifest response, configures the machines, and stores ssh command for future use.

In Fig. 3 we show the response time of our web server in receiving user requests and reserving appropriate demanded resources. It must be noted that this latency is largely affected by the time spent by GENI in reserving the resources. We can observe how our TM component can effectively dispatch the requests and generate opportune responses for the operations even when the number of resources to handle is considerable and the user asks for a topology composed of 60 nodes.

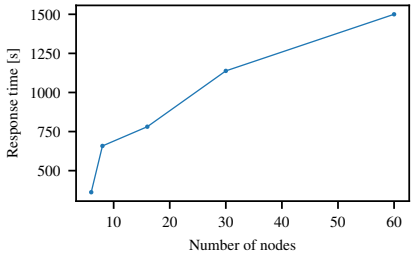
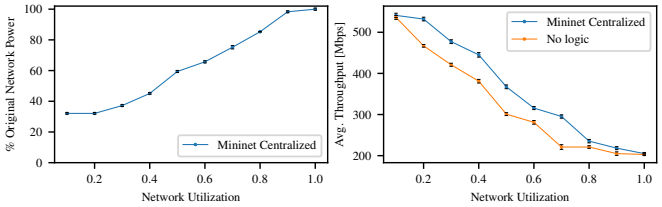


Fig. 3: Service response time of our web server for a GENI backend when the size of requested topology increases.



(a) (b)

Fig. 4: **Mininet experiments.** Performance evaluation in terms of (a) power saving and (b) throughput. RLVNA is an effective auto-scaling solution in a local environment.

A. Planning schema performance on Mininet

To better understand the results obtained with GENI, we start by running the RLVNA algorithm over the local testbed of Mininet. During our experiments, we used *iperf3* to generate traffic to measure throughput and *netperf* to measure latency, while *tcpdump* was used for analyzing the packets of the communication. *iperf3* is also used to generate background traffic to face different network utilization in the experiments. The SDN controller is a Ryu node [18] that interacts with the Docker container that carries the RL algorithm. To evaluate our platform, we considered a randomly generated topology composed of 13 switches and 18 hosts, where the links have a bandwidth of 1Gbps and we simply refer to this topology as "large topology". In this simulation, the clients send traffic at a varying throughput, starting with 10Mbps and scaling up to a total of 1Gbps each, to simulate congestion by reaching a specific network utilization quota, defined as the average of the load over all links weighted by their capacity. In other words, this is the sum of the link flows over the entire network divided by the sum of link capacities. For each setting we repeat the experiment for twenty iterations, averaging the values for a 2-minutes communication.

After setting up our simulation environment, we focused on the impact of adopting RLVNA compared to an implementation with no logic at varying of the network load in Mininet emulator (Fig. 4). The results show that, when the network utilization is not excessive, an auto-scaling approach can decrease the power by deactivating unused resources. For example, when the network is utilized at 50%, with RLVNA we can save about 50% of the power in a solution with no logic (Fig. 4a). At the same time, when the network is congested, our solution can create more links, re-direct the traffic, and preserve some throughput. Compared to a network

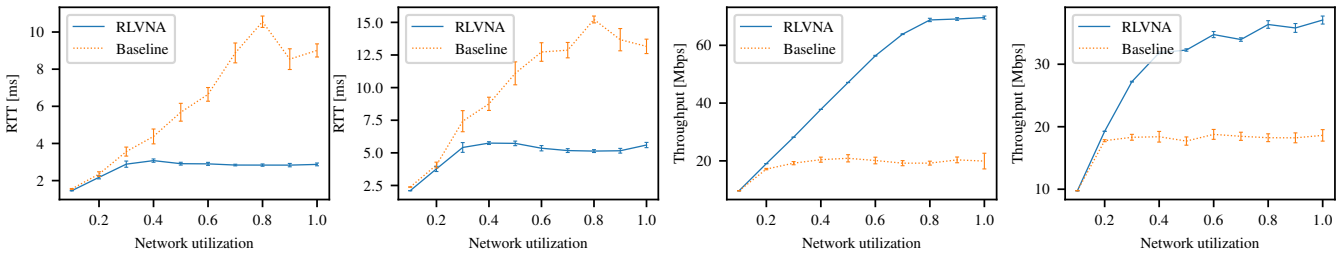


Fig. 5: **GENI experiments.** RTTs measurement for (a) the small topology (b) and for the large topology. Throughput measurement for (c) the small topology (d) and for the large topology.

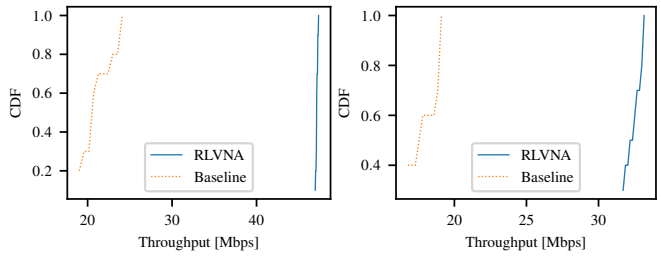


Fig. 6: CDF measurement. Experiments at 50Mbps in (a) the small topology and (b) the large topology.

in the absence of auto-scaling logic, we can observe that our solution stably achieves higher throughput (Fig. 4b).

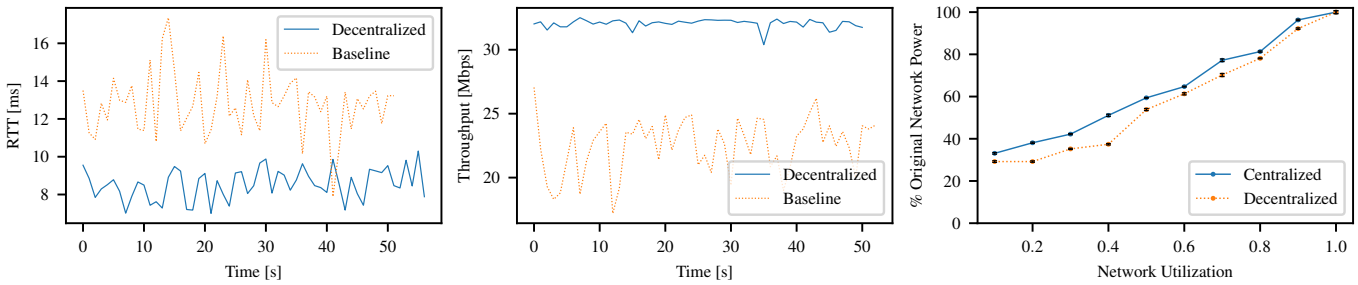
B. Planning schema performance on GENI

We then experimented over a real-world testbed as GENI, where the links are at 100 Mbps. With the methodology described before, we considered the performance at varying of the link load to show the benefits of the auto-scaling solution and compared the configuration where the RLVNA model is running and when is not. The need to experiment in a genuine environment, rather than an emulated one, was a key driving force behind our decision to undertake this project. Although Mininet is a valuable tool that allows experimenting with networks, it still remains an emulator, and results may be misleading. For this reason, we computed the performance metrics that RLVNA achieved in the GENI testbed, focusing on the Round Trip Time (RTT) and throughput, as shown in Fig. 5. To also study how the algorithm and our implementation can deal with diverse topology, we utilize a network comprising of 5 switches and 10 hosts, called "small topology", in addition to the large topology described earlier.

Fig. 5a shows that in both topologies, RLVNA and a baseline approach (in which the controller has no auto-scaling logic) exhibit similar performance when the network is lightly congested ($0.1 - 0.3$ of utilization). However, as the network congestion increases (from 0.4), the benefits brought by RLVNA become more visible, resulting in lower RTT across all tested network loads. Fig. 5b exhibits a similar trend, but due to the larger topology size, RLVNA starts to show lower RTT at a lower bottleneck level of 30%. The same behavior is visible when computing the throughput, as reported in Fig. 5c and Fig. 5d. In this scenario, the small and the large topology perform better with RLVNA at already low induced congestion ($0.2 - 0.4$) and achieve a higher throughput when compared to a baseline implementation even at the highest network loads (from 0.5). For completeness, we also present in Fig. 6 the Cumulative Distribution Function (CDF) of the throughput when sending at 50 Mbps. As depicted in Fig. 6a, our solution demonstrates good results in a small topology, maintaining the throughput up to 47 Mbps, while the baseline schema can only reach a maximum of 25 Mbps. Similar results are observed in the large topology, as illustrated in Fig. 6b. However, our implementation reaches a maximum of 33 Mbps in this case, while the baseline implementation is only able to achieve up to 20 Mbps.

In the previous section, we showed how our solutions can create almost any request for the network topology. However, we limit the number of nodes to 60, as this is an intrinsic limit of available resources on a single site of GENI aggregate (Fig. 3). This result motivates our hypothesis that to manage and test large-scale networks, we need to create a multi-site architecture where the controllers collaborate toward an efficient global management schema. As such, we studied how to deploy more (larger) networks over GENI, proposing a distributed learning phase design based on Raft. To simulate this scenario, we deployed a topology composed of 20 switches controlled by 3 SDN controllers, each managing a different subnetwork. The three controllers share information about the RL model used for the auto-scaling and adopt the leader-followers approach typical of Raft. One controller is elected as leader and maintains the global state, while the others maintain a replica of this state, following its evolution. By introducing our RL model, we can observe a simultaneous increase in throughput and a reduction in transmission delay. This can be seen in Fig. 7a, which shows that the RTT of a decentralized solution is significantly lower in RLVNA compared to an implementation without an ML logic throughout the entire 60-second tracking period. Additionally, Fig. 7b demonstrates that RLVNA achieves higher throughput than an implementation lacking RL support, indicating an optimal behavior.

As a last aspect, we considered the sustainability of our approach in a large-scale network environment to analyze how auto-scaling solutions can improve network management efficiency. In particular, we considered the differences between centralized and decentralized models for increasing network utilization. We report in Fig. 7c the energy efficiency computed by the RL agent for the two alternative approaches. When the



(a) (b) (c)

Fig. 7: **GENI experiments.** (a) RTT and (b) throughput evolution over time for the RLVNA model. (c) Comparison of the RLVNA setting of centralized and decentralized.

network utilization is close to 1.0 (100% of links utilized), then all links and switches must stay active, preventing the savage of power. With lower utilization, traffic can be concentrated over a reduced number of links, and the unused ones can be switched off. First, this outcome is similar to the one obtained in the Mininet environment (Fig. 4). Second, the decentralized approach can further reduce the percentage of original power.

The results confirm: (i) the benefits of a learning-based management scheme that can scale network resources up and down, (ii) how distributing the logic allows to control a larger network. A simple distributing approach, such as our Raft-based one, can be implemented to manage a possibly large network sharing the decisions' logic. In conclusion, the implementation of our solution through GENI showed that the initial results obtained over Mininet were accurate and reliable. The overhead introduced by the consensus algorithm is negligible and the advantages of the solution are clear.

IV. Conclusion

In this paper, we present a versatile platform that enables users of all levels of experience to reserve resources and conduct experiments with machine learning algorithms on real testbeds. Along with a validation of this platform, the paper studies the differences in performance between a Mininet and real-world experimentation, and between a centralized and decentralized algorithm. While our initial tests were conducted on the GENI testbed, our long-term plan is to expand our solution to a multi-cluster version. We believe that upcoming platforms like FABRIC [12] will play a crucial role in advancing this experimentation to new frontiers. Moreover, although more advanced methods to train a global model are available, e.g., Federated Learning and Split Learning, the results supports the simplicity of our decentralized approach.

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. Sacco, F. Esposito, and G. Marchetto, "Restoring Application Traffic of Latency-Sensitive Networked Systems Using Adversarial Autoencoders," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2521–2535, 2022.

[3] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "Orchestran: Network Automation Through Orchestrated Intelligence in the Open RAN," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 270–279.

[4] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Partially oblivious congestion control for the internet via reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1644–1659, 2022.

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

[6] A. Sacco, F. Matteo, Flocco andEsposito, 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.

[7] 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.

[8] 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.

[9] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *IEEE Colombian Conference on Communications and Computing (COLCOM '14)*. IEEE, 2014, pp. 1–6.

[10] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[11] Geni, Exploring Networks of the Future. Accessed: 2023-3-15. [Online]. Available: https://www.geni.net/

[12] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "Fabric: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.

[13] J. Mambretti, J. Chen, and F. Yeh, "Next generation clouds, the chameleon cloud testbed, and software defined networking (sdn)," in *International Conference on Cloud Computing Research and Innovation (ICCCRI '15)*. IEEE, 2015, pp. 73–79.

[14] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb *et al.*, "The design and operation of cloudlab." in *USENIX Annual Technical Conference (ATC)*, 2019, pp. 1–14.

[15] M. Avgeris, N. Kalatzis, D. Dechouniotis, I. Roussaki, and S. Papavassiliou, "Semantic resource management of federated iot testbeds," in *Ad-hoc, Mobile, and Wireless Networks: 16th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW 2017)*. Springer, 2017, pp. 25–38.

[16] D. Stavropoulos, V. Miliotis, T. Korakis, and L. Tassiulas, "Matching theory application for efficient allocation of indivisible testbed resources," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–7.

[17] Network implementation testbed using open source platforms. Accessed: 2023-4-24. [Online]. Available: https://nitlab.inf.uth.gr/NITlab/nitos

[18] Ryu controller. Accessed: 2023-3-15. [Online]. Available: https://ryu-sdn.org/

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