Supporting Sustainable Virtual Network Mutations With Mystique

Alessio Sacco, *Graduate Student Member, IEEE*, Matteo Flocco, Flavio Esposito, *Member, IEEE*, and Guido Marchetto, *Member, IEEE*

***Abstract*—The abiding attempt of automation has also permeated the networks, with the ability to measure, analyze, and control themselves in an automated manner, by reacting to changes in the environment (e.g., demand). When provided with these features, networks are often labeled as "self-driving" or "autonomous". In this regard, the provision and orchestration of physical or virtual resources are crucial for both Quality of Service (QoS) guarantees and cost management in the edge/cloud computing environment. To effectively manage the lifecycle of these resources, an auto-scaling mechanism is essential. However, traditional threshold-based and recent Machine Learning (ML)-based policies are often unable to address the soaring complexity of networks due to their centralized approach. By relying on multi-agent reinforcement learning, we propose Mystique, a solution that learns from the load on links to establish the minimal set of active network resources. As traffic demands ebb and flow, our adaptive and self-driving solution can scale up and down and also react to failures in a fully automated, flexible, and efficient manner. Our results demonstrate that the presented solution can reduce network energy consumption while providing an adequate service level, outperforming other benchmark auto-scaling approaches.**

***Index Terms*—SDN, reinforcement learning, auto-scaling, network management.**

I. Introduction

As USERS and traffic demands grow, the need to optimize our communication networks magnifies, denoting the evidence that networks dictate our technological world. Recent advantages in artificial intelligence (AI) and machine learning (ML) are paving the path to autonomous networks: networks that measure, analyze and control themselves autonomously [1]. Network automation has been desired in the last years, since it is almost impossible for human operators to render real-time network management [2]–[4].

Our focus in this article is on network reliability and network elasticity, i.e., the subproblem of autonomous networks that deals with the ability to auto-scale resources up and down, in harmony with changes in the environment, e.g., traffic demand. The advantages brought by the auto-scaling techniques are multiple. They reduce the cost of resource management, by deactivating resources that may increase unnecessary (energy) costs. At the same time, the network can provide redundant facilities to reroute traffic when workload peaks to unexpected levels.

As networks are becoming more programmable and virtualized, their complexity also increases, with the consequence that exploiting the offered programmability to guarantee high availability is a non-trivial task. Traditional threshold-based and recent ML-based auto-scaling policies are often unable to address the high complexity of networks and consequently to satisfy carrier-grade requirements such as reliability and stability. Furthermore, state-of-the-art solutions hardly combine these features altogether, such as [5] whose primary goal is the energy efficiency, or [6], which automatically scales Virtual Network Function instances via an ML classifier. Although reinforcement learning is emerging as a valuable technique to solve many networking problems, as in [7], [8], there is no solution incorporating network information to automatically and efficiently orchestrate network resources in a decentralized manner.

Our Contribution: In this article, we propose Mystique, a network management schema that, using Multi-Agent Reinforcement Learning (MARL), auto-scales to accommodate the traffic demand and reacts to possible failures. On the one hand, Mystique unburdens network nodes that are overcongested with traffic, to preserve the high bandwidth and high availability of the applications. On the other hand, it leverages healing strategies [9] to repair failing nodes and links.

Each MARL agent, a process running within a network controller, can learn an auto-scaling policy from experience, without any *a priori* knowledge or human intervention. By continuously monitoring the state of the network, the agent can make sharp decisions on how to optimize network performance and users' experience, exploiting SDN to promptly change the configuration. Moreover, the distributed nature of MARL makes it possible to exploit a (possibly) large number of SDN switches spread across the topology as probes. The system automatically re-balances both existing and new flows across nodes, while the agents communicate among them to obtain information about the other sub-network.

At the same time, it is well-known that from the operator's point of view, Quality of Experience (QoE) is an important

Manuscript received November 20, 2020; revised February 11, 2021; accepted February 12, 2021. Date of publication February 16, 2021; date of current version September 9, 2021. This work has been partially supported by NSF under Award Numbers CNS1647084, CNS1836906, and CNS1908574. The associate editor coordinating the review of this article and approving it for publication was M. Tornatore. *(Corresponding author: Alessio Sacco.)*

Alessio Sacco and Guido Marchetto are with the DAUIN, Politecnico di Torino, 10129 Turin, Italy (e-mail: alessio_sacco@polito.it; guido.marchetto@polito.it).

Matteo Flocco and Flavio Esposito are with the Department of Computer Science, Saint Louis University, St. Louis, MO 63103 USA (e-mail: matteo.flocco@slu.edu; flavio.esposito@slu.edu).

Digital Object Identifier 10.1109/TNSM.2021.3059647

aspect in keeping customers satisfied, and thus decreasing churn [10]. To this end, the decisions taken by Mystique aim to maximize overall QoE across multiple users and achieve a desired level of QoE fairness, while reducing the energy costs for active links and nodes.

Results validate our decentralized control plane, showing how Mystique can promptly adapt and modify its behavior to handle variations in workloads. Compared to other benchmark solutions, our algorithm can jointly improve the user satisfaction and more wisely utilize the network resources.

The remainder of the article is organized as follows. We present the context and the main functionalities offered by Mystique in Section II. The model considered and the formal problem that we attempt to solve are outlined in Section III. We then present the general approach in Section IV and further improvements for a feasible algorithm in Section V. The experimental results are shown in Sections VI and VII, and Section IX concludes the article.

II. System Design

In this section, we first identify the softwarized infrastructure and the advantages of auto-scaling solutions, like Mystique, in this scenario. Then, we present the mechanisms underpinning the overall system with particular focus on the offered features. In particular, we start analyzing a single entity, and we continue highlighting the elements employed to realize the parallelization of the model.

A. Edge Network Scenario

Although our proposed model is general enough to suit several network deployments, in this manuscript we target edge networks built from software-defined networking (SDN) architectures, due to their flexibility and possible customization. While the model is general and does not require specific functionalities offered by the underlying technology, to achieve automatic responses to network traffic, softwarized or virtualized networks are needed.

By edge network, we refer to a network located on the periphery of a centralized network. It sits entirely between the services and the endpoint devices using them, as well as between all the edge servers themselves.

In this context, network operations can take advantage of data-driven, machine-learning-based models to achieve more high-level goals and a holistic view of the underlying network. Our solution constitutes an attempt 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 changes in the environment, e.g., demand, while adjusting and optimizing themselves as needed [1]. This idea has been around in a variety of shapes, such as self-organized networks [11], cognitive networking [12], knowledge-defined networks [13] and data-driven networking [14], and lastly, self-driving networks [15].

Fig. 1 represents a common scenario that needs auto-scaling components. At first, the traffic demand increases reaching a non-tolerable level of congestion, and the system decides the consequent creation of resources to satisfy the traffic demand

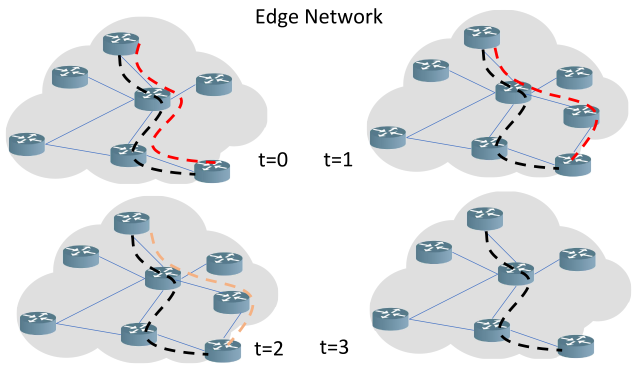


Fig. 1. The goal of our Mystique system is to learn via reinforcement learning how to adapt to network demand fluctuations by creating new virtual switches and split traffic (from t = 0 to t = 1), and to scale-down nodes and network resources when demands decrease (from t = 2 to t = 3).

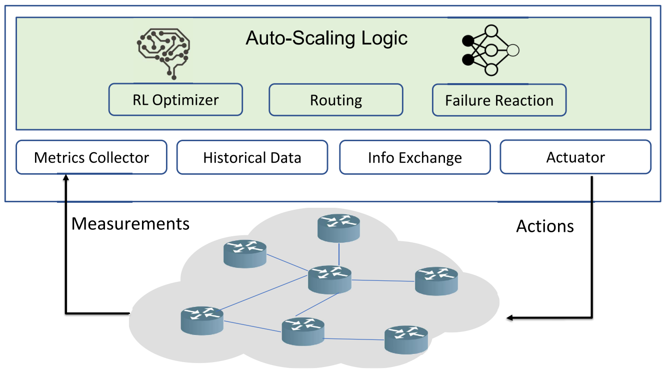


Fig. 2. System overview. The Software-Defined Network controller receives as input traffic statistics and outputs new flow routes and power on/off commands.

alongside re-routing for the interested flows. In a second time, the exigency of additional resources vanishes gradually as the traffic decreases, and the system reacts by removing the unnecessary devices. This simple example illustrates the feature and benefits of auto-scaling networks.

In this regard, recent advances in machine learning (ML), e.g., deep learning, and networking, e.g., SDN, programmable data planes, and edge computing, have fostered the development of these networks. However, a desirable and still missing feature is represented by the distributed detection of congestion with no centralized congestion recognition and control. Furthermore, to improve system performance, bottlenecks need to be identified, and efforts should be invested in alleviating these bottlenecks.

B. System Components

The main functions of Mystique are to auto-scale according to the traffic demand and react to failures when they occur. We developed and implemented these features in a system depicted in Fig. 2. In this context, the controller monitors the state of each switch in its sub-network to detect if one of the following events occurs: the switch is overloaded (congestion), the switch in under-utilized and can be deleted (cost-saving), the switch fails and the connectivity can be no longer guaranteed (failure). However, the network implements the control plane with several distributed controllers. Each of them controls a subset of switches and communicates with the other controllers, via the **Info exchange** process, to obtain a consistent network view. For any change in the controlled network region, e.g., new link, the controller notifies its peers. They also exchange the information required for computing the QoE for connected users.

The reinforcement learning (RL) module selects the best action, i.e., active network resources, but interacts with other processes to collect the information required for the decision and to notify about the outcome. In fact, we avail multiple processes to better separate concerns, but they cooperate to achieve our stated goal. The main functionalities are summarized thereafter.

Routing: Each agent dynamically creates and destroys virtual switches and virtual links in response to network fault or substantial network traffic changes. This means that, in these events, the agent is also responsible for re-steering the traffic and deciding what flows to move in response to these actions.

At the beginning, the route for each flow is selected by the controller based on the shortest path algorithm. In the case of multiple available paths between source and destination, a load balancing strategy is applied, i.e., flows are equally distributed among the multiple paths. In the following, we separate the events to face during the execution with the aim of a more clear presentation.

In the case of a *link or node failure*, the same resource is re-created. For a link failure, a new edge is created connecting the same source node and destination node. The neighbor of the switch modifies the forwarding rules reflecting the new port ids. For a switch failure, a new one is generated with the same links that the faulty switch had. This implies that all the flows previously installed on the old switch are moved to the new one, and the neighbor nodes that were connected need to be re-instructed with the new ports.

When a *new link or switch is created*, the topology is analyzed, and the flows that can take advantage of the new path(s) are identified. Among them, a subset of flows, i.e., in order to select half of the identified traffic, is transferred to the new alternative path. However, we remark that the number of flows to move is a consequence of a load balancing strategy that attempts to equally redistribute flows.

Finally, when *a link or switch is deactivated* due to energy saving considerations, all the flows traversing the deactivated item are considered, and a new path for each of them is set via the shortest path strategy.

Failure Reaction: We desire to react accordingly to the degree of the issue and take a proportional action. For this reason, the utilization of the switch (and connected links) is handled by the RL model, while a separate module manages the failure detection and reaction. Inspired by previous work [16], [17], we consider 5 possible faults that can take place in our scenario: (1) communication with the controller ended, (2) timeout of the response expired, (3) port fault, (4) flows of a particular host have blocked unexpectedly by the switch, (5) unexpected behavior of the switch. As the controller continuously monitors the state of the switches, it can replace the switch in case of (1)-(2) (three consecutive timeout expirations)-(5). Instead, in case of (3)-(4), the link originating from the fault port is re-created.

The mechanism of fault reaction is in addition to the fast failover provided by new versions of OpenFlow [18]. In this procedure, it is possible to install rules whose forwarding behavior depends on the local state of the switch. Hence, it allows fast failure recovery, as long as the SDN controller is able to anticipate every possible failure and precompute appropriate backup paths. However, OpenFlow fast failover is just to react to link failures, and no other events are taken into account, for example, switch failure. Even though this is equivalent to a failure of all the adjacent links, we argue that the controller can benefit from our model and adapt the routing and the application logic dynamically, as the network evolves. The fast failover is orthogonal to a reactive solution, as our model is. For this reason, both strategies are utilized for an improved fault reaction.

RL Optimizer: The optimizer's role is to find the network subset that satisfies current traffic conditions while avoiding the waste of resources. As input, it receives the topology, the power model of switches, and the current traffic conditions. These measurements are collected by the switches and reported periodically to the network controller, where resides the **metric collector** component. The collected data then feed the model on the agent, that outputs the best decision for the network itself, exploiting **historical data** to learn the goodness of a particular action upon the occurrence of similar conditions. When the decision is made, the **actuator** receives the output consisting of the set of active components and performs the appropriate commands. Moreover, the actuator also pushes the new routes into the network.

III. Auto-Scaling System Model

The ultimate goal of Mystique is to deploy the network resources in order to balance the management costs and goodness in application performance. In this section, we first describe the variables used in the following to specify the model; then, we formalize our problem. We summarize our notation in Table I.

A. Network Model and Preliminary Definitions

We define our model in two phases: the overseeing phase, the healing phase. In the overseeing phase, a network controller monitors the system to check for failures or congestion conditions. Specifically, it communicates with the nodes of the network to obtain information about the load on links attached. Given the context of an SDN topology, a network node commonly denotes a switch. However, our model is independent of the layer the device operates and can work when the nodes act as either routers or switches.

When a link (and implicitly its connecting nodes) becomes or is going to be too congested, the controller intervenes to mitigate the congestion. In the healing phase, we determine the quantity of new resources to create, given the intolerable network congestion. Hence, enough nodes and links are created, and traffic is redirected to them to ensure that the fairness index is close to 1 in the network system.

TABLE I
TABLE OF NOTATIONS IN THE MODEL FORMALIZATION

Notation	Description
G	Graph describing the topology
V	Set of vertices (network resources)
E	Set of edges
R_u	Set of vertices connected to vertex u
$c(u, v)$	Capacity of link (u, v)
s_i	Source of flow i
d_i	Destination of flow i
$f_i(u, v)$	Throughput of flow i over the link (u, v)
f_i	End-to-end throughput of flow i
$\phi_{u,v}$	Binary variable stating whether link (u, v) is powered ON
τ_u	Binary variable stating whether switch u is powered ON
$Z_i(u, v)$	Binary variable stating whether link (u, v) is used by flow i
$cl(u, v)$	Power cost for link (u, v)
$cs(u)$	Power cost for switch u
$L_{u,v}$	Load on the link (u, v)
B	Mapping function from QoS parameters to QoE value
F	QoE fairness index
C	Network power consumption
r	Reward function
P	Metrics collection interval
I	Learning step

Aside from the congestion detection, another goal of our system is to react to the failure of a network resource (node or link). In such a case, the response is similar, with the creation of one or more replacements. Moreover, a re-routing process occurs to notify the nodes of the existence of the new resources. After the actuation, a failure and congestion avoidance phase starts again, and the controller will continue to monitor the system to guarantee that traffic is returned to normal operation.

We formalize such phases using the following notation. Let the network be modeled by means of a graph $G = (V, E)$, where *V* is the set of vertices (nodes of the network), and *E* is the set of edges, standing for the links. Similarly to the multi-commodity flow problem, we assume that in the system there are *k* commodities $K_1, K_2, \ldots, K_k$, defined by $K_i = (s_i, d_i)$, where s_i and d_i are the commodity *i* source and destination. The flow of commodity *i* has an end-to-end throughput defined as f_i. Moreover, the throughput of flow *i* along the link (*u, v*) is $f_i(u, v)$ where each link of the network has a fixed capacity *c*(*u, v*).

Let $L_{u,v}$ be the load on the link (*u, v*), computed as follows:

$$L_{u,v} = \frac{\sum_i^k f_i(u, v)}{c(u, v)}, \tag{1}$$

considering that $f_i(u, v) = 0$ when flow *i* does not traverse the link (*u, v*).

QoE: As other user-centric services and application management [19], we focus on the quality of experience (QoE) perceived by the end-user. It is generally accepted that the degree or annoyance of a user of a networked service depends, in a non-trivial and often non-linear way, on the network's QoS [10]. Furthermore, the QoE is service-specific and is often different given the same network conditions, i.e., the way in which QoS can be mapped to QoE depends on the service offered.

For this reason, we define a general mapping function *B* to compute the QoE value y_i for the user *i*, given the QoS parameters in the set x_i:

$$B : x_i \mapsto y_i = B(x_i) \in [L; H], \tag{2}$$

where *L* is the lower bound and *H* the upper bound of the QoE value. The function *B* is required to map QoS parameters into the QoE domain [*L*; *H*] and map QoS to QoE uniquely. Thus, it does not need to be monotonic. We leave the choice of the mapping function out of the scope of this article, since the QoE models are often derived by subjective user studies. In our formalization we only require the value $y_i = B(x_i)$ to be normalized in [0; 1]. If *B* does not naturally return values in the range [0; 1], this can be achieved normalizing the QoE values $y_i^* = \frac{y_i - L}{H - L}$.

QoE Fairness: Recent results [20] have argued for a more informative notion of fairness, not limited to flows as classically studied in TCP, but more holistically over a set of metrics defining the QoE. We adopt the same definition of QoE fairness index *F* as follows:

$$F = 1 - \frac{\sigma}{\sigma_{max}} = 1 - \frac{2\sigma}{H - L}, \tag{3}$$

where σ is the standard deviation of the QoE values and quantifies the dispersion of the users' QoE in a system. The fairness index *F* is a linear transformation of the standard deviation σ of y_i to [0; 1], where $F = 1$ indicates perfect fairness and hold for minimum standard deviation ($\sigma = 0$). Conversely, $F = 0$ denotes the minimum fairness and is found when the standard deviation is at its maximum. The observed σ is normalized with the maximal standard deviation σ_{max} and specifies the degree of unfairness. Further, the maximum standard deviation is $\sigma_{max} = \frac{1}{2}(H - L)$, and we can observe that in our case where $y_i \in [0; 1]$, $\sigma_{max} = 0.5$. In conclusion, a system is absolutely QoE fair when all users receive the same QoE value.

This definition of fairness appears to be intuitive, i.e., high value if fair and low value if unfair, and compared to the most frequently used metric of Jain's fairness index [21] provides some additional properties. First, the index *F* is independent of QoE level (QoE level independence), and second, only depends on the absolute value of the deviation from the mean value, not whether it is positive or negative (deviation symmetric). The Jain's fairness index, instead, is sensitive to the used rating scale. These features ensure our model to be general enough and valid for multi-applications. We can also note that, as the *F* index is defined, QoE fairness says nothing about how good the system is and thus needs to be considered together with the achieved QoE in system design.

B. Optimizing Quality of Experience, Costs, and Fairness

Based on the aforementioned network model, we now describe the problem as an optimization problem aiming to simultaneously reduce management costs, alleviate congestion effects providing an adequate service, and ensure fairness among users.

First, we formalize the power consumption of network topology as:

$$C = \sum_{(u,v)\in E} cl(u,v)\phi_{u,v} + \sum_{u\in V} cs(u)\tau_u, \tag{4}$$

where $\phi_{u,v}$ is a binary decision indicating the power status of link (*u, v*), i.e., $\phi_{u,v} = 0$ refers to power off and $\phi_{u,v} = 1$ to power on. The same for the power status of a switch, where τ_u is a binary variable indicating if switch *u* is powered on. Besides, *cl*(*u, v*) and *cs*(*u*) are the power cost for link (*u, v*) and switch *u* respectively.

To optimize the power consumption, we act on these binary variables for every link and switch, and we constrain traffic to only active links and switches. We can now present the overall problem as follows:

$$\underset{X,Y}{\text{maximize}} \quad \mu\frac{1}{k}\sum_{i=1}^{k} B(x_i) + \lambda F - \omega C_{norm} \tag{5}$$

$$\text{s.t.} \quad \sum_{i=1}^{k} f_i(u,v) \le c(u,v)\phi_{u,v}; \forall(u,v) \in E \tag{6}$$

$$\tau_u \le \sum_{r\in R_u} \phi_{u,r}; \forall u \in V \tag{7}$$

$$\phi_{u,r} \le \tau_u; \forall u \in V, \forall r \in R_u \tag{8}$$

$$\sum_{r\in R_u} Z_i(u,r) \le 2; \forall u \in V, \forall r \in R_u \tag{9}$$

$$\phi_{u,v} = \phi_{v,u}; \forall(u,v) \in E, \tag{10}$$

where C_{norm} is the normalized cost, i.e., $0 \le C_{norm} \le 1$, to make it comparable to the other variables. Line (5) specifies the objective function, which attempts to maximize the average QoE and the fairness index F perceived by the end-user while reducing the total network power consumption. μ, λ, and ω are three parameters that balance the importance of power consumption with respect to the QoE and fairness index. By tuning these coefficients, the model can be tailored to specific requirements.

The constraints from (6) to (10) include some requirements to satisfy. In particular, (6) ensures that deactivated links have no traffic. Each flow is indeed restricted to the links powered on, i.e., for which $\phi_{u,v} = 1$. Therefore, for all links (*u, v*) used by commodity *i*, $f_i(u,v) = 0$ when $\phi_{u,v} = 0$. The objective of cost minimization enforces also the opposite: links with no traffic can be turned off. This line also imposes capacity constraints, as the total flow along each link must not exceed the link capacity. Further, (7) and (8) set a correlation between the link and the switch decision variable. Specifically, (7) imposes that when all links connected to a switch are off, the switch can be powered off. Similarly, when a switch is powered off, all the links connected to such a switch are also powered off, as stated by (8). Although splitting the flow across multiple links in the topology might reduce power by improving link utilization overall, it is known that this may cause undesirable packet reordering effects that negatively impact TCP performance [22]. For this reason, we prevent flow from getting split in the above problem by enforcing (9). This constraint ensures that the switch receives flow *i* from the incoming link and forward it to only one outgoing link, and the flow uses a total of at most two links attached to the switch *u*. Finally, in (10) we define that the link power status is bidirectional, and there is no concept of half-on Ethernet link. Thus, the full power cost for an Ethernet link is incurred when traffic flows in either direction.

Our optimal solution must select which resources to turn on and off, while satisfying the traffic constraints. The presence of binary variables makes the stated problem a mixed-integer linear program, which is NP-complete. Given the computation complexity, which can hardly scale to networks with a large number of nodes, in the following we attempt to solve the problem via a reinforcement learning approach.

IV. The Mystique Solution

The learning process at the basis of the system is built upon the reinforcement learning concepts for a continuous acquisition of knowledge of the network. In the following, we define the main elements which characterize our reinforcement learning problem. Since our MARL model can be viewed as an extension of centralized model, in the following, we describe the procedure as in a single agent variant in order to clearly describe the learning process. Further variations coming from the decentralization are explained later in Section V.

A. Reinforcement Learning Problem

Reinforcement learning is a well-known on-line technique that approximates the conventional optimal control technique known as dynamic programming [23]. The external world is commonly modeled as a discrete-time, finite-state, Markov Decision Process (MDP). The agent interacts with the external world and performs actions, where each action is associated with a reward. The objective of reinforcement learning is to maximize the long-term discounted reward per action.

A discounted Markov decision process, such as an RL problem, is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \lambda)$. Here S is the set of all states, which can be countable or uncountable, A is the set of all actions, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the Markov transition kernel, $R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}(\mathcal{S})$ denotes the distribution of the immediate reward, and $\lambda \in (0, 1)$ is the discount factor. Specifically, upon taking any action $a \in \mathcal{A}$ at any state $s \in \mathcal{S}$, $P(\cdot|\ s, a)$ is the probability distribution of the next state and $R(\cdot|\ s, a)$ defines the distribution of the immediate reward. Moreover, a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ of the MDP maps any state $s \in \mathcal{S}$ to a probability of distribution $\pi(\cdot|s)$ over $\mathcal{A}$.

In our solution, each reinforcement learning agent uses the one-step *Q-learning* algorithm [24]. In this context, the learned decision policy depends on the state-action value function *Q*, which estimates long-term discounted rewards for each state-action pair. Given a current state *x* and the possible available actions a_i, the agent selects each action with a *softmax* policy, which consists of a softmax function [25] that converts output to a distribution of probabilities. This means that it affects a probability for each possible action, given by the Boltzmann distribution:

$$p(a_i|s) = \frac{e^{Q(s,a_i)/T}}{\sum_{b\in actions} e^{Q(s,b)/T}}, \tag{11}$$

where T is the parameter that adjusts the randomness of action decisions. Thus, the agent selects the action, receives an immediate reward r, and moves to the next state s'.

At each time step, the agent updates the Q-value function $Q(s, a)$ by recursively discounting future utilities and weighting them by a positive learning rate α:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma V\big(s'\big) - Q(s,a)\big), \quad (12)$$

where γ is the discount factor and $V(s')$ is the Value function, which measure how good a certain state is, in terms of expected cumulative reward, and is given by:

$$V\big(s'\big) = \max_{b \in actions} Q\big(s', b\big). \quad (13)$$

In other words, the value function is the maximum reward that is attainable in the state following the current one, i.e., the reward for taking the optimal action thereafter. It is relevant to note that $Q(s, a)$ is updated only when a new action is taken, and that selecting actions stochastically as in (11) ensures that each action will be evaluated repeatedly.

As the agent explores the state space, its estimation of the Q-value function improves progressively, and eventually, each $V(s')$ approaches $\mathbb{E}[\sum_n = 1^1 \gamma^{n-1} r_{t+n}]$, where r_t denotes the reward received at time t upon the action chosen at time t–1. It has been shown that this Q-learning algorithm converges to an optimal decision policy for a finite MDP [24].

The RL agent receives the inputs and selects the best action, then updates the state of the table and proceed with this process continuously.

B. Reinforcement Learning States

Decisions taken by the agent should consider the network status and react properly to events that occur. Hence, the state of our RL model is composed of the load on links, $L_{u,v}$, for each link in the network. This metric is employed in the learning process to choose the best action and evaluate the performed action.

C. Actions

The RL algorithm defines the mapping between the inputs of the reaction logic, i.e., the network state, and the actions to be performed to address the issues coming from the ongoing traffic. The action selection is the key to the proposed algorithm to be effective. As in the other RL approaches, the control decisions are learned from experience, eliminating the burden of a more rigorous model. In line with the optimization problem defined, the RL agent computes a scaling action $a_t^{u,v} \in \mathcal{A} = \{1, 0\}$, for the link $(u, v) \in E$ at time step t. As mentioned above, $a_t^{u,v} = 1$ represents the link in the state on, and $a_t^{u,v} = 0$ represents the link in the state off. When the decision for the link differs from the current link state, the controller sends specific commands to change the state and actuate the output of the RL process.

In conformity with the cost optimization and the considerations previously explained (Section III-B), when all the links of a switch are down, the switch can be turned down as well. For this reason, in spite of the fact that the actions only refer to links, they also impact the switch state.

D. Reward

In accordance with the reinforcement learning approach, the agent finds the best resource allocation that maximizes the network-aware reward. In fact, the *reward* of the RL formulation specifies the appropriateness of the action taken in a particular state. The utility function instead specifies the objective of our algorithm by looking at the environment response. The aim is to find the decision policy of resource allocation with the maximum utility function for the network agent. Since the utility function is the real objective that we try to optimize, we directly use it as a *reward* for the learning process. Thus, we define a reward function, r, similar to the optimization function of the aforementioned formal problem:

$$r = \mu \frac{1}{k} \sum_{i=1}^{k} B(x_i) + \lambda F - \omega C_{norm}, \quad (14)$$

where μ, λ and ω are coefficients that control the importance of the average $B(x)$, which defines the QoE and the customer satisfaction, F, for the fairness of the QoE, and C_{norm}, that assess the network cost management. The function aims to minimize the over-provisioned resources while also minimizing the switches load in order to improve QoS parameters.

E. Challenges in Implementing the Model

The presented model has two main challenges to solve: *(i)* when the number of nodes and links rises conspicuously, the state space may become intractable by the framework, *(ii)* in these circumstances, not only the state space is difficult to handle, but also the action space. In fact, the decision of which switches and links to turn on/off depends on the network size.

To address these limitations that can arise for large topology, we apply three strategies:

(i) Firstly, the policies are stored in a tabular form and are approximated by means of *function approximators* [26]. A function approximator is composed of a manageable number of parameters, often referred to as policy parameters, θ. As a consequence, we represent the policy as $\pi_\theta(s, a)$. To represent the policy, many forms of function approximators can be used, for instance, linear combinations of state and action space features. Recently, deep neural networks (DNNs) [27] have been used with success as function approximators in large-scale RL tasks [28]. One of the advantages of DNNs adoption is that no hand-crafted features are needed. Inspired by these successes, we make use of a neural network to represent the policy in our RL formulation.

(ii) Secondly, the switches and links to activate/deactivate are limited to a smaller subset, in order to reduce the action set. Specifically, we define $V^a \subseteq V$, as the set of nodes that can be activated or deactivated. It is conventional in many networks, indeed, that some nodes are essential for the network functionalities and must always stay alive, e.g., switches attached to end-hosts. Hence, without loss of generality, V^a includes nodes that are helpful yet nonessential. The dimensionality of V^a largely depends on the topology and the degree of redundancy of the network. Furthermore, since our actions are per link, the model only considers the

links connected to nodes in V^a. It is relevant to note that only the action set is limited, whereas the state space still includes all the links.

(iii) Lastly, the state space can be further reduced via multiple controllers managing a different subset of the network. In such a case, the controllers exchange information about the underlying topology, in order to have a global view of the network. However, the information about the managed links is not exchanged. This approach allows to scale over large topologies and shorten the training time, since each controller has a model with a reduced scale. We define this strategy as *multi-agent Mystique*, and in the following, we detail its properties and study its effectiveness.

V. Achieving a Practicable Framework

As mentioned earlier, we address the complexity coming from large action and state spaces by approximating the Q-table via neural networks, reducing the total available actions, and distributing the information across more agents. In the following, we detail the techniques used and specify the impact on the RL problem. Finally, we outline the overall algorithm at the basis of Mystique.

A. Deep Q-Learning

In order to deal with the large state and action spaces, deep Q-learning uses neural networks, parameterized by θ, to approximate the Q-function. Hence, the Q-values are denoted as $Q(s,\ a;\ \theta)$, and the neural network is referred to as Q-network. In our system, the Q-learning process consists of two parts: *(i)* the Q-value approximation for the action selection, *(ii)* the Q-network update, where the loss between predicted Q-values and target Q-values is used to update the Q-network parameters θ, according to the gradient method:

$$\theta \leftarrow \theta + \alpha\big(target_Q - Q(s,a;\theta)\big)\nabla_\theta Q(s,a;\theta). \quad (15)$$

The "$target_Q$" is a target value calculated as follows:

$$target_Q = r + \lambda \max_{b \in actions} Q\big(s', b; \theta\big). \quad (16)$$

Moreover, to prevent the Q-function from diverging due to dynamical changes in the target [29], a separate network is introduced, the target network. It is a copy of the Q-function and is used to calculate the target value. This approach is usually denoted as Deep Q Network (DQN), and in our experiments, we examined its performance using a periodic update, i.e., every *I* seconds, of the target network with the current Q-function.

The use of the target network hinders fast learning due to this delay, but we utilize the proposed Experience Replay (ER) [30] to increase the reuse of data. ER is a heuristic that temporarily stores to memory a record of state transitions during a certain number of steps and randomly selects a data point from memory for learning. By doing so, the correlations between samples are reduced, and sample efficiency is increased.

B. Categorize Network Nodes

Due to the network setup and design, some nodes are very unlikely to be deactivated, as no other paths are available or the cost to create an alternative path is too high. For this reason, these nodes should not be considered throughout the reasoning of which links to preserve or dismiss. Thus, we define a new set $V^a \subseteq V$, which includes only the nodes of interest, and the decisions are limited to this new set. In such a way, we are able to drastically reduce the action set and speed up the learning process.

We define also a new edge set, $E^a \subseteq E$, which contains the links connecting the nodes in V^a. The action set therefore results being $a_t^{u,v} \in \mathcal{A} = \{1, 0\}$, for the link $(u, v) \in E^a$ at time step *t*.

Notwithstanding that the action is limited, the decisions take into account the entire set *E*, to optimize the process by considering the overall situation.

C. Multi-Agent Reinforcement Learning Framework

To scale-up and distribute the burden among separate agents, we leverage multiple controllers, where each of them is responsible for a sub-network. Along with improvements in terms of scalability, distributing the burdens among more agents brings resiliency to the event of failure of one controller. When one agent fails, the switches under its control are temporarily migrated under the supervision of a near controller. As such, our solutions can handle both failures at the data-plane (switches) and at the control-plane (controllers).

More formally, in a multi-agent setting, each agent maintains an individual policy π_i for the specific state space $\mathcal{S}_i$ and action space $\mathcal{A}_i$. The state space is, thus, limited to the sub-network managed, and includes a diverse set of switches and links. In the same way, the action set reflects the diverse set and is restricted to only the managed links.

Despite the difference in the RL model, the agents interact among them to obtain information about the global topology, as the routing decisions should consider a global view. A model is trained for each node, but they communicates possible metrics needed to compute the user QoE.

D. Mystique Overall Algorithm

In the light of all these elements, we propose a decentralized RL-based procedure to address the congestion problem in the network.

In Algorithm 1 we summarize the main steps underpinning our self-learning process aboard of each controller. For one thing, we initialize the Q-values table, *Q*(*s, a*) and other hyper-parameters. Next the continuous learning procedure starts (line 5). With a period *P*, the controller gets information about the congestion of the links, represented by the value $L_{u,v}$. Hence, the agent observes the current state, *s*, and verifies the absence of failures. In case of link or node failure the proper reaction is enacted. Either way, an action *a* is chosen for that state based on one of the action selection policies explained previously. Taking the action may involve creation or removal of links or nodes. In these circumstances, re-routing must take place, and the controller instructs the switches with

Algorithm 1 Resource Optimization Using RL

```
1: Let $P$ be metrics collection interval, and $I$ the learning step
2: Let $Q$ be the network approximating the Q-values
3: Let $TN$ be the target network
4: Initialize $Q(s,a)=0$, $T_1=0$, $T_2=0$
5: for each episode do
6:     if $t-T_1>P$ then
7:         Collect the state $s$ at time $t$
8:         Verify the presence of failures, and in case react
9:         Choose $a$ using policy derived from $Q$
10:        Take action $a$ by activating/deactivating selected resources
11:        Adjust routing accordingly
12:        Observe $r$, $s'$ and update the $Q$ network
13:        if $t-T_2>I$ then
14:            $TN \leftarrow Q$
15:            $T_2 \leftarrow t$
16:        $T_1 \leftarrow t$
17:        Notify updates to other controllers
```

the new flow rules to engineer the traffic. Once the adjustment is completed, the agent observes the reward, r, as well as the new state, s', and updates Q-value for the state s using r and the maximum reward possible for s'. The updating is done according to the formula and parameters described in (12). Afterwards, it is checked if I seconds are elapsed since last update, if so the Q network is copied to the target network *TN*. Finally, the agent set the state to the new state, and repeats the process until a terminal state is reached.

VI. Experimental Settings

In this section, we first describe the testbed configuration and the settings common throughout our experimental campaign. Further, we identify a representative network scenario for the evaluation and the algorithms to compare Mystique against.

A. Implementation

To demonstrate the practicality of our approach, we have implemented the proposed scheme in an emulated scenario. Mystique needs to receive the current utilization and to operate on the flow paths decisions. These network capabilities can be achieved via NetFlow [31] and SNMP for the traffic data, while source routing and policy-based routing allow the path control. The network switches can be implemented via different technologies, such as Quagga, FRRouting, OVS, Bind, P4. However, due to the ease of use of SDN in prototyping, we use OVS switches [32] featuring OpenFlow, combined with an SDN-controller to obtain the above requisites. OpenFlow, providing a flow table abstraction, is used to push the computed set of application-level routes to each switch. Besides the flow installation component, it provides the flow-specific counters that can be accessed by external entities via open API, and enables the port power control.

Every switch communicates with an SDN controller, which is implemented with Ryu [33]. Ryu is a component-based software-defined networking framework that provides network visibility and control atop a network of OpenFlow switches. In particular, the Ryu controller communicates with the switches to collect the metrics, and pushes the adjusted flow routes. The controller behavior can be customized via the REST APIs that the controller is equipped with. Finally, in the centralized version, the number of Ryu controllers is limited to one, whereas in the distributed version, the number can increase to guarantee fault tolerance of the controller agent.

B. Experimental Setup

We deployed several environments to assess the performance of Mystique. However, herein we summarize some common settings and remarks valid throughout the performed experiments. For the sake of simplicity, we limited the QoS parameters x_i of user i to the end-to-end throughput, and the mapping function B is an identity function. By doing so, we limit our focus to metrics that can be collected smoothly, and we leave the study of a system based on more indicators (to be summarized for example using a neural network) as future work.

Evaluation Settings: The topology and the switches are emulated over Mininet [34]. Mininet is a network emulator that creates a realistic virtual network, running real kernel, switch and application code, on a single machine. This networking tool makes use of namespaces, a feature of the Linux kernel that partitions kernel resources. In current setup, switches and ports are not powered off, but are only deactivated, since the switches are virtual switches. Nonetheless, in a real deployment, it is possible to exploit existing mechanisms to control the power, such as SNMP, command line interface, or recent mechanisms of power control over OpenFlow.

Traffic Workload: The energy, performance, and robustness of the system heavily depend on the traffic pattern. In the following, we explore how a variety of communication patterns affect system behavior. Specifically, we evaluate a *uniform demand*, where every host sends one flow to another host of the network. We use two more types of traffic patterns to evaluate performance, *moderate increase* and *sharp increase*. During the former, the hosts linearly increase the traffic sent to double it within 20 seconds. In the latter, instead, traffic is doubled in 5 seconds. Finally, due to the lack of public traces specific for this problem, we generate traffic *synthetically*, where each sender-receiver pair runs *TCP iperf3* for 100 seconds, alternating between different rates.

Hyper-Parameter Settings: We developed the agent's neural network with Keras [35], a high-level neural networks API written in Python. The neural network is composed of the input layer with three hidden layers with respectively 256, 128 and 64 neurons. The input layer number of neurons corresponds to the number of links in the managed network, whereas the number of output layer neurons is the amount of links of action set, E^a. Such a neural network runs over Tensorflow [36], an end-to-end open-source platform for machine learning. For hyper-parameter setting, we set the discount factor $\lambda = 0.99$ and the learning rate $\alpha = 0.9$. The batch size is 256 to enable

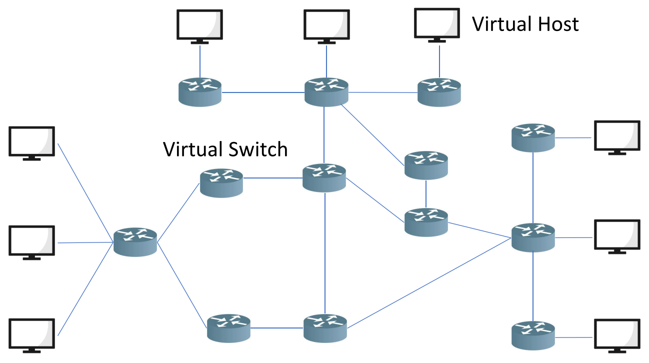


Fig. 3. A sample of emulated network topology used during the experiment campaign. The design reflects the desire of allowing a variety of test conditions and a multitude of available paths.

a high learning rate. If not otherwise specified, cost saving, application performance, and fairness are equally important, i.e., $\mu = \lambda = \omega = 0.3$. We set the default interval for collecting statistics $P = 0.5$ s, and the action decision interval to $I = 1$ s. Nonetheless, in what follows, we also evaluate the consequence of alternative intervals over the system performance.

Network Use Case: The network topology considered throughout experiments should offer sufficient path redundancy, enough hosts to deploy the desired traffic patterns, and be reasonably suitable for an edge network. Taking all these considerations in mind, we converged on a network use case shown in Fig. 3. This topology is the default environment utilized, but also other settings with different link densities are considered. OVS switches are utilized for the measurement of the throughput per each ongoing flow, in turn, mapped to a single QoE value, following considerations of Section VI-B.

Benchmark Algorithms: To validate our solution we compare against three state-of-the-art solutions adequately adapted to our use case: an ML classifier-based method to perform auto-scaling, [6], SRSA [8], and ElasticTree [5]. In [6], an ML-based method converts the auto-scaling decision to an ML classification problem, so that it can learn from the insights and temporal patterns hidden inside measured data from the network. As shown in their study, Random Forest (RF) is the algorithm performing better; thus, we apply this method, and, for simplicity, we refer to this solution as Auto-RF. Conversely, SRSA is a reinforcement learning approach to auto-scaling VMs in a telco cloud platform [8]. Even though our solution shares the RL formalization, substantial differences involve the model, the objective (and reward), the use case, actions associated, and a single agent vs. multi-agent version. Finally, ElasticTree attempts to solve a power optimization problem and compares multiple solutions strategies. A greedy bin-packing heuristic has been advocated as an adequate solution, and for this reason, we employ the version of ElasticTree using such a heuristic. However, it is relevant to note that our difference with the state-of-the art not only resides in a multi-agents configuration, but also in an optimized algorithm which can handle more complexity for more conscious decisions, as seen in Section VII.

VII. Performance Evaluation

In the following, we compare the goodness of our data-based approach with respect to model-based solutions, usually solved via heuristics, and other data-based techniques. After a brief explanation of the considered metrics, we measure the impact of a MARL approach over network performance. Then, we extensively compare Mystique versus related solutions in several network conditions. Lastly, we also run sensitivity experiments by varying some algorithm parameters.

A. Evaluation Metrics

In this section, we make use of metrics and quantity defined in Sections III and IV, such as the QoE fairness and reward function.

Besides them, one of the primary metrics we inspect is the percentage of power savings, computed as:

$$\begin{aligned}\%\ \text{power savings} &= 100 - \%\ \text{original power}\\ &= 100 - \frac{\text{power consumed by solution} \times 100}{\text{power consumed in original network}}, \quad (17)\end{aligned}$$

which gives an accurate idea of the overall power saved by turning off switches and links. The original power, indeed, represents the consumption for the *always-on* baseline.

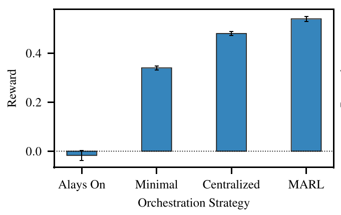
Clearly, the savings depend heavily on the network utilization, u, defined as the average of the load on the link, $L_{u,v}$, over all links weighted by their capacity. In practice, this is the sum of the link flows over the entire network divided by the sum of link capacities, formally:

$$u = \frac{\sum_{(u,v)\in E} \sum_i^k f_i(u,v)}{\sum_{(u,v)\in E} c(u,v)}. \quad (18)$$

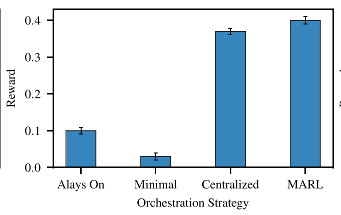
B. Centralized Versus Multi-Agent Reinforcement Learning

Firstly, we compare the performance of our distributed solution based on multiple agents with a centralized version build upon the same model (Section V). For the sake of completeness, to understand the advantages of self-learning capabilities, we also compare them against *always-on* and *minimal orchestration* baseline algorithms. In the former setting, all the switches and links are maintained during the experiment. This policy ensures the best applicative performance but high energy consumption. On the other hand, in a minimal orchestration, no redundancy is exploited, and only the minimal subset to let the network works is kept. This configuration leads to a minimum in management cost, but a degradation in the performance.

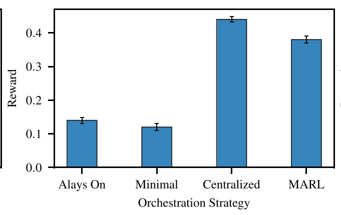
Figure 4 compares the reward function defined in (14), for the four algorithms: always-on and minimal orchestration as baseline algorithms, and the centralized and decentralized version of the RL process, i.e., MARL. In the centralized, a single controller handles the switches of the network, while for the decentralized setting, three controllers are in charge of managing the network and instruct the switches. Evaluating their strengths and weakness implies appraising the behavior for different traffic demands. For this reason, we analyze the reward function across four traffic patterns: uniform, moderate increase, a sharp increase, and synthetic generation for a



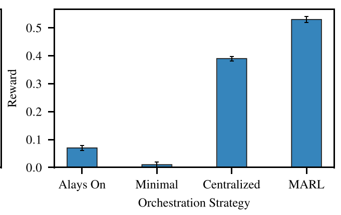
(a) Uniform Traffic Pattern



(b) Moderate Traffic Pattern

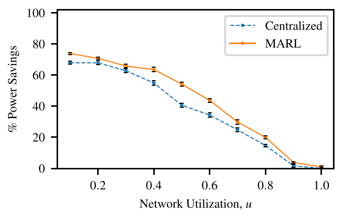


(c) Sharp Traffic Pattern

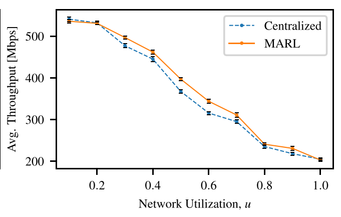


(d) Synthetic Traffic Pattern

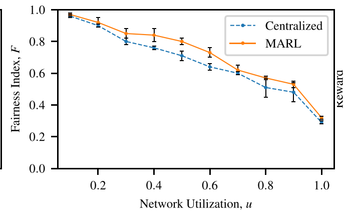
Fig. 4. Reward function for different traffic patterns: (a) uniform, (b) moderate increase, (c) sharp increase, (d) synthetic generation. Four available strategies are compared, highlighting differences between centralized versus decentralized (MARL) model.



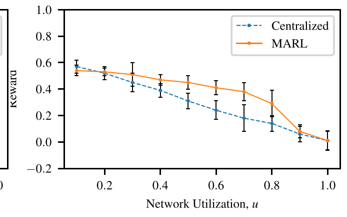
(a)



(b)

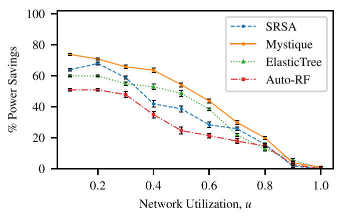


(c)

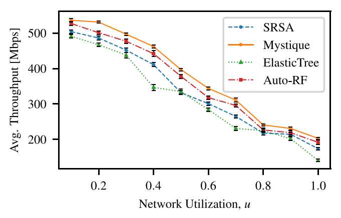


(d)

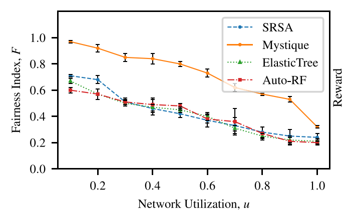
Fig. 5. Centralized versus decentralized (MARL) approach for diverse network utilizations. The comparison entails (a) energy saved with respect to the original setting, (b) mean application throughput achieved, (c) fairness index for the flows, (d) system reward.



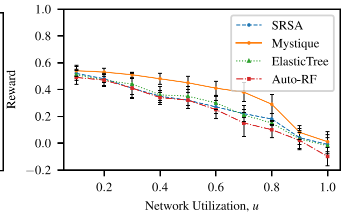
(a)



(b)



(c)



(d)

Fig. 6. Performance comparison with other benchmark algorithms in terms of (a) energy saved with respect to original setting, (b) mean application throughput achieved, (c) fairness index for the flows, (d) system reward.

more realistic use case. We can observe how the distribution of the concerns across multiple agents, as in the decentralized version, produces higher rewards in almost every context. The only exception resides in the sharp increase of the traffic demand, given the limited knowledge of the network status.

Besides this preliminary analysis, for a more extensive comparison, we also consider the differences between centralized and decentralized (MARL) models for increasing network utilization. We report in Fig. 5 the (a) energy efficiency, (b) application throughput, (c) QoE fairness, and (d) reward as computed by the RL agent, for the two alternative approaches. We utilize synthetic traffic generation to mimic a more real workload. When u is close to 100%, then all links and switches must stay active, preventing the savage of power (Fig. 5a). With lower utilization, traffic can be concentrated over a reduced number of links, and the unused ones can be switched off.

The QoE parameter considered is the average end-to-end throughput (the mean between the users). While in the centralized version this value can be easily accessed by the controller, with more agents it is required to exchange topology and application-level information. However, this overhead is minimal in the process. In fact, we can observe in Fig. 5b how the MARL setting regularly enables higher throughput compared to the centralized one. The same result holds when considering the QoE fairness index F in Fig. 5c. Since one of the parameters to optimize is fairness among users, this value is consistently high and drops down only for intensive network utilization. All these considerations are then reflected in the reward behavior shown in Fig. 5d, in which we can recognize that, especially for a medium utilization, the benefits of MARL are significant.

In light of all these results, we can remark that the multi-agents setting does not degrade the performance, quashing the possible decentralization limits. Because of an enhanced procedure in the action-reaction logic, our MARL model, on the contrary, improves performance in more contexts. This further motivates our distributed Mystique solution described in Section V-C.

C. State-of-the-Art Comparison

After a first assessment of Mystique performance, we evaluate our solution against the benchmark algorithms stated above (Section VI-B). In a similar way to the previous evaluation, we report in Fig. 6 the (a) energy efficiency, (b) application throughput, (c) QoE fairness, and (d) reward function, for the considered methods. By considering the plots, we can notice how our system outperforms the related algorithms in

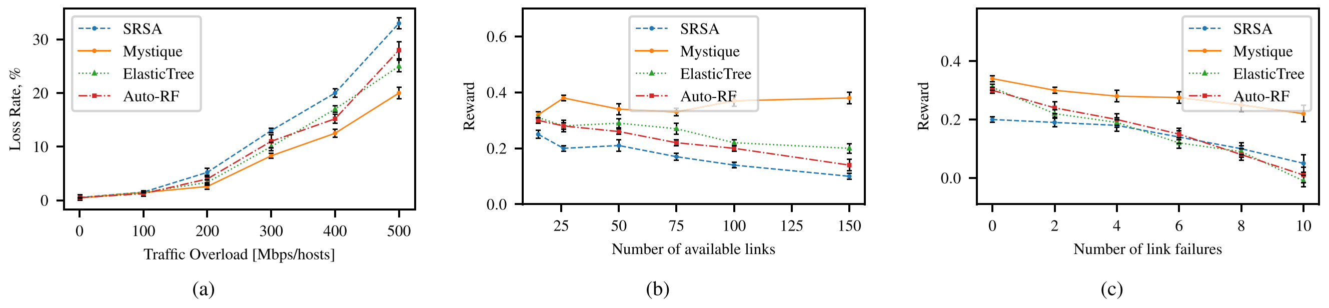


Fig. 7. (a) Drops vs. traffic overload for a variety of methods. (b) Algorithms performance among different topologies with increasing connectivity for benchmark algorithms. (c) Obtained reward for redundant but failing topologies. Our solution outperforms other algorithms, especially for an increased number of link failures.

all the examined metrics. In particular, fairness is distinctly one of the most improved quantities in Mystique, as one of the desiderata. Furthermore, none of the other algorithms can efficiently optimize more metrics simultaneously, but they can successfully improve only some of them. Conversely, Mystique stably outperforms other solutions, demonstrating its ability to optimize management costs and QoS parameters altogether in multiple network scenarios.

Congestion Mitigation: We also compare alternative procedures responding to increasing traffic demand. Fig. 7a shows packet drops as a function of traffic overload, which refers to the amount each host sends and receives in each flow. During the trial, all hosts send 4 flows to 4 randomly chosen hosts, where the bandwidth of the flow is the traffic overload quantity. All tests complete in 60 seconds. Although the number of lost packets is not used as a metric nor in the objective function, we can observe that Mystique can reduce the percentage of losses, due to its utility function, which comprises congestion-related indexes.

Network Diversity Adaptation: In order to generalize our findings, we deployed a more random topology in which links among switches and hosts are randomly generated when the number of nodes increases (reaching the feasible limits on Mininet). Specifically, the number of links between the nodes is a parameter in the generation phase, and it affects the density of the network. The number of hosts matches the number of switches, and each of them generates traffic synthetically. We change the number of available links to evaluate the system performance and how it scales over more dense networks. We train the models over the new networks and assess the reward function changes in these circumstances.

Results are depicted in Fig. 7b and show how our solution is portable over unknown topology compared to other methodologies. As the density of the network increases, the quality of related methods decreases. In contrast, Mystique delivers regular performance even for complex topologies outperforming the other approaches. We can thus conclude that our model can adapt to diverse conditions and reasonably accomplish its mission in a multitude of scenarios.

Failure Reaction: Aside from the auto-scaling mechanisms to address the network congestion, Mystique makes use of self-healing techniques to appropriately react to network failures. In the following, we evaluate the ability of our system to jointly tackle the overwhelming traffic demand and the failure

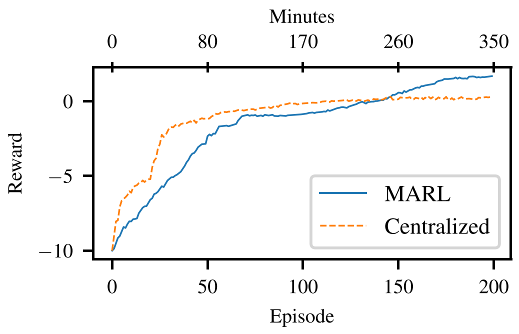


Fig. 8. Training curves for the centralized and decentralized version of the solution.

scenario. Thus, in addition to synthetic traffic generated, we fail links uniformly at random, in numbers also increasing. Fig. 7c reports the average reward obtained for the tested algorithms over a topology based on the scenario in Fig. 3, in which the single points of failure are conveniently redundant. In this new scenario, even if some nodes might be disconnected, other paths to the destination remain. From the graph, we can observe how Mystique notably improves the system robustness and can also handle very frequent failures. These results validate our approach of an autonomous network with both the capability of auto-scaling and fault tolerance.

Convergence Time: Another key aspect to consider is the time required for the RL model to converge to a stable reward. In Fig. 8 we compare the training time for the centralized and MARL settings of the solution. Results are obtained on a host machine running Ubuntu, Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, with NVIDIA GeForce 920MX with CUDA 5.0. We can observe how the centralized algorithm converges more rapidly to a stable reward value. However, this value is lower than the reward obtained via MARL. In fact, in this setting, the model requires a longer time to converge, but it can outperform the competitor. Moreover, the time required to converge is in line and significantly shorter if compared to other DQN models [37], [38].

D. Sensitivity Analysis

Finally, we conduct an analysis of the reward function by varying the parameters of Algorithm 1. Other hyper-parameters of DQN, i.e., batch size and the neural network

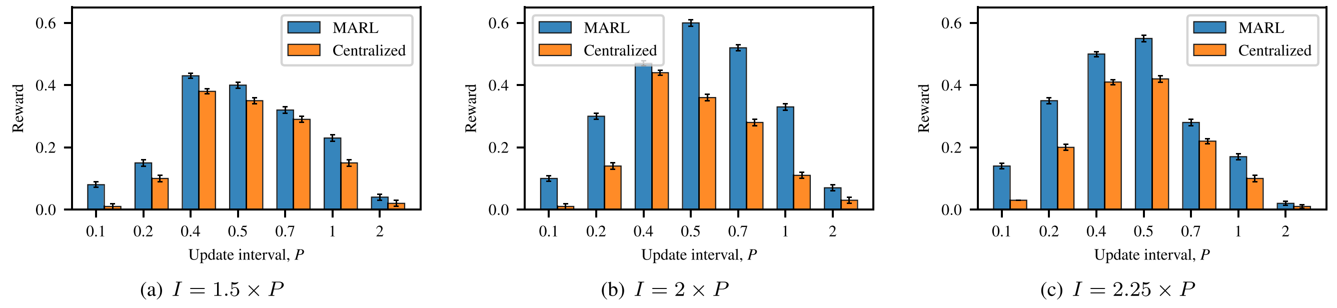


Fig. 9. Reward function for diverse combinations of update interval P and decision interval I. The best trade-off, i.e., maximum reward, arises for MARL in graph (b) with $P = 0.5$s. A similar behavior occurs for the centralized version, with the slightly different of at maximum in graph (b) for $P = 0.4$s.

structure, are selected via grid searching in a standard procedure when dealing with neural networks. Thus, their selection is not detailed in the manuscript. Instead, the target network update interval I is an obvious place where instability could arise and requires more effort to be set, because it is tightly coupled with the frequency in decisions P. For this reason, we report the study of the effects of the frequency in updating the metrics collection for different target net update intervals. Fig. 9 illustrates the reward obtained for different intervals P in three alternative strategies for setting I, respectively when I is (a) 1.5, (b) 2, (c) 2.25 times the value of P, and both centralized and MARL settings are evaluated.

Our aim is to find the best interval that relaxes the overhead in requesting the metrics while being reactive to network changes and operating wise decisions. At first glance, we can observe that an interval P close to 0.4 and 0.5 s is the optimal trade-off in our MARL system, providing the highest reward across the three settings. Likewise for I, delaying the update of the target network may conduct in losing the accuracy. On the contrary, if it is updated too frequently, then the benefit of using the target network (which is to boost the training rate and lower training time) starts reducing, and the training will likely require a more considerable amount of time. The results suggest that when the target network is updated at a half rate of refresh information rate, i.e., $I = 2 \times P$, the reward achieved is at its maximum. Conversely, both higher and lower rates seem to worsen performance. These considerations led us to set the combination $P = 0.5$s and $I = 1$s as the default for the MARL model.

We can also observe differences in the two settings of centralized and decentralized versions. The reward evolution for the centralized approach resembles the MARL, with a maximum reward slightly moved towards the combination $P = 0.4$s - $I = 0.8$s.

VIII. Related Work

Managing network or application resources elasticity implies a first mapping performance requirements to the underlying available resources. Such a process of adapting resources to the on-demand requirements of an application, called scaling, can be very challenging. Resource under-provisioning will inevitably hurt performance and generates QoS violations, while resource over-provisioning can result in idle instances, thereby incurring unnecessary costs. Auto-scaling techniques, i.e., resource allocation strategies that automatically scale resources according to demand, are more than a need and can be differentiated into two classes: *reactive* and *proactive*. While the former class refers to algorithms reacting to system changes, but not anticipating them, the latter stands for strategies that predict and anticipate the future needs and consequently acquire or release resources in advance, to have them ready when they are needed. In the literature, auto-scaling solutions have been extensively discussed from several points of view, especially for cloud computing [39]–[41].

Reactive and Proactive Techniques: Threshold-based policy is a common example belonging to the reactive category, whereas time-series analysis, reinforcement learning, queuing theory, and control theory can be examples of proactive approaches. Queuing theory, given its ability of estimating performance metrics such as the queue length or the average waiting time for requests, has been largely applied to model applications, e.g., general Internet or cloud infrastructure applications [7], [42], [43]. For example, [42] solves an optimization problem by distributing servers among different applications, while maximizing the revenue. The authors characterize the arrival process of requests to an application using a real trace of an e-commerce system, where the arrival process is adequately described by a Poisson process.

Control theory has been applied to automate the management of Web server systems, data centers/server clusters, storage systems, cloud computing platforms, and other systems, showing interesting results across this variety of systems. Many papers have discussed adaptive control techniques, by adjusting the controller tuning parameters online [44]–[47]. For instance, [46] combines two proactive adaptive controllers for scaling down with dynamic gain parameters based on the input workload, and a reactive approach for scaling up.

Time-series are massively used in finance and economic domains to represent the change of a measurement over time. Recently, this technique has also gained attention in engineering and workload or resource usage prediction problems [48]. At the very basic, a time-series is a sequence of data points, e.g., number of requests that reaches an application, measured at successive time instants spaced at uniform time intervals, e.g., one-minute intervals. The time-series analysis is able to find repeating patterns in the input workload and to forecast future values. The auto-regression method has been largely used [49]–[53] and time-series forecasting can be combined with reactive techniques [54]. For example, [52] proposed a hybrid scaling technique that, based on CPU usage, utilizes reactive rules for scaling up and a regression-based approach for scaling down.

Lastly, reinforcement learning (RL) approaches for dynamic resource allocation problems were successfully applied in the literature. RL can well fit auto-scaling problems by online capturing the performance model of a target application and its policy without any *a priori* knowledge. However, these methods have mainly focused on allocating tasks, services, and Virtual Machines (VMs), especially to face the greater or smaller demand, where [7], [55], [56] are examples of a profitable usage of RL. As such, little work has been proposed to address the problem of network resources.

Dynamic Resource Creation of Network Agents: Recent studies have explored scaling softwerized or virtualized network functions in telco and cloud networks. Among them, [6] proposes a proactive ML-based approach to perform auto-scaling of VNFs in response to dynamic traffic changes. The classifier learns from historic auto-scaling decisions and measured network loads, and outputs the number of VNF instances required to serve future traffic without violating Quality of Service (QoS) requirements and deploying unnecessary VNF instances. Reference [5] describes ElasticTree, a network-wide energy optimizer that continuously monitors data center traffic conditions and chooses the set of network elements that must stay active to meet performance and fault tolerance goal. To decide which subset of links and switches to use, a fast heuristic is used. The primary goal of ElasticTree is the savage of energy in data centers containing thousands of nodes. Although we share the general approach with these solutions, we propose a self-learning model that embraces more the QoS aspects.

A reinforcement learning approach is described in [8], where the authors present SRSA, a resource-efficient approach to auto-scale telco-cloud. The decision of allocating or deallocating VMs is performed to guarantee the QoS and to reduce the cloud cost. Our solution is also built upon an RL framework, but differs in the modeling aspects and enables us to scale to more complex networks by learning in a distributed fashion.

IX. Conclusion

This article presents Mystique, a system that allows scaling network resources up and down to track network utilization. The network controller can dynamically activate or deactivate links and nodes in an "as needed" fashion with the aim of minimizing the energy consumption and improving QoE and fairness among users. Furthermore, the system can promptly react to network failures as these happen. In this context, the paper highlights the benefits of splitting the decision logic across multiple controllers, for a distributed and fault tolerant architecture. We show how this approach can improve the management when the quantity of information needed for the model becomes large and can lead to more accurate actions.

References

[1] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," 2017. [Online]. Available: arXiv:1710.11583.

[2] A. Sacco, F. Esposito, and G. Marchetto, "A federated learning approach to routing in challenged sdn-enabled edge networks," in *Proc. 6th IEEE Conf. Netw. Softwarization (NetSoft)*, 2020, pp. 150–154.

[3] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "An architecture for adaptive task planning in support of IoT-based machine learning applications for disaster scenarios," *Comput. Commun.*, vol. 160, pp. 769–778, Jul. 2020.

[4] A. Sacco, F. Esposito, and G. Marchetto, *A Distributed Reinforcement Learning Approach for Energy and Congestion-Aware Edge Networks*. New York, NY, USA: Assoc. Comput. Mach, 2020, pp. 546–547.

[5] B. Heller *et al.*, "ElasticTree: Saving energy in data center networks," in *Proc. NSDI*, vol. 10, 2010, pp. 249–264.

[6] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling VNFs using machine learning to improve QoS and reduce cost," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2018, pp. 1–6.

[7] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.

[8] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient auto-scaling approach in the telco cloud using self-learning algorithm," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2015, pp. 1–6.

[9] E. Ayanoglu, I. Chih-Lin, R. D. Gitlin, and J. E. Mazo, "Diversity coding for transparent self-healing and fault-tolerant communication networks," *IEEE Trans. Commun.*, vol. 41, no. 11, pp. 1677–1686, Nov. 1993.

[10] P. Le Callet *et al.*, "Qualinet white paper on definitions of quality of experience," in *Proc. Eur. Netw. Qual. Exp. Multimedia Syst. Services (COST Action IC 1003)*, vol. 3, 2012.

[11] O. G. Aliu, A. Imran, M. A. Imran, and B. Evans, "A survey of self organisation in future cellular networks," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 336–361, 1st Quart., 2013.

[12] R. W. Thomas, D. H. Friend, L. A. Dasilva, and A. B. Mackenzie, "Cognitive networks: Adaptation and learning to achieve end-to-end performance objectives," *IEEE Commun. Mag.*, vol. 44, no. 12, pp. 51–57, Dec. 2006.

[13] A. Mestres *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 2–10, 2017.

[14] J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "Unleashing the potential of data-driven networking," in *Proc. Int. Conf. Commun. Syst. Netw.*, 2017, pp. 110–126.

[15] P. Kalmbach, J. Zerwas, P. Babarczi, A. Blenk, W. Kellerer, and S. Schmid, "Empowering self-driving networks," in *Proc. Afternoon Workshop Self-Driving Netw.*, 2018, pp. 8–14.

[16] B. Chandrasekaran and T. Benson, "Tolerating SDN application failures with LegoSDN," in *Proc. 13th ACM Workshop Hot Topics Netw. (HotNets)*, 2014, pp. 1–7.

[17] A. Sacco, G. Marchetto, R. Sisto, and F. Valenza, "Work-in-progress: A formal approach to verify fault tolerance in industrial network systems," in *Proc. IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, 2020, pp. 1–4.

[18] *Openflow Switch Specification 1.3.1*. Accessed: Oct. 2, 2020. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf

[19] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck, "Qoe-driven rate adaptation heuristic for fair adaptive video streaming," *ACM Trans. Multimedia Comput. Commun. Appl. (TOMM)*, vol. 12, no. 2, pp. 1–24, 2015.

[20] T. Hoßfeld, L. Skorin-Kapov, P. E. Heegaard, and M. Varela, "Definition of QoE fairness in shared systems," *IEEE Commun. Lett.*, vol. 21, no. 1, pp. 184–187, Jan. 2017.

[21] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination," Eastern Res. Lab., Digit. Equip. Corp., Hudson, MA, USA, Rep. DEC-TR-301, 1984.

[22] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, 2007.

[23] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[24] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[25] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*. Berlin, Germany: Springer, 1990, pp. 227–236.

[26] D. P. Bertsekas and J. N. Tsitsiklis, "Neuro-dynamic programming: An overview," in *Proc. 34th IEEE Conf. Decis. Control*, vol. 1, 1995, pp. 560–564.

[27] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*. Boston, MA, USA: PWS Publ. Co., 1997.

[28] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[29] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.

[30] L.-J. Lin, "Reinforcement learning for robots using neural networks," Sch. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Rep. CMU-CS-93-103, 1993.

[31] *Cisco IOS NetFlow*. Accessed: Oct. 2, 2020. [Online]. Available: http://www.cisco.com/web/go/netflow

[32] *Open vSwitch: An Open Virtual Switch*. Accessed: Oct. 2, 2020. [Online]. Available: http://www.openvswitch.org/

[33] *Ryu Controller*. Accessed: Oct. 2, 2020. [Online]. Available: https://ryu-sdn.org/

[34] 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 *Proc. IEEE Colombian Conf. Commun. Comput. (COLCOM)*, 2014, pp. 1–6.

[35] A. Gulli and S. Pal, *Deep Learning With Keras*. Birmingham, U.K.: Packt Publ. Ltd, 2017.

[36] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[37] K. Yao, L. Zhang, T. Luo, and Y. Wu, "Deep reinforcement learning for extractive document summarization," *Neurocomputing*, vol. 284, pp. 52–62, Apr. 2018.

[38] N. Jiang, Y. Deng, A. Nallanathan, and J. A. Chambers, "Reinforcement learning for real-time optimization in NB-IoT networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1424–1440, Jun. 2019.

[39] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Trans. Services Comput.*, vol. 11, no. 2, pp. 430–447, Mar./Apr. 2018.

[40] A. N. Toosi, J. Son, Q. Chi, and R. Buyya, "ElasticSFC: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds," *J. Syst. Softw.*, vol. 152, pp. 108–119, Jun. 2019.

[41] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *J. Grid Comput.*, vol. 12, no. 4, pp. 559–592, 2014.

[42] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," *ACM Trans. Internet Technol. (TOIT)*, vol. 7, no. 1, p. 7, 2007.

[43] T. Phung-Duc, Y. Ren, J.-C. Chen, and Z.-W. Yu, "Design and analysis of deadline and budget constrained autoscaling (DBCA) algorithm for 5G mobile networks," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, 2016, pp. 94–101.

[44] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A multi-model framework to implement self-managing control systems for QoS management," in *Proc. 6th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst.*, 2011, pp. 218–227.

[45] P. Bodík, R. Griffith, C. A. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in *Proc. HotCloud*, vol. 9, 2009, p. 12.

[46] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Proc. IEEE Netw. Oper. Manag. Symp.*, 2012, pp. 204–212.

[47] P. Padala *et al.*, "Automated control of multiple virtualized resources," in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, 2009, pp. 13–26.

[48] A. Sacco, F. Esposito, and G. Marchetto, "RoPE: An architecture for adaptive data-driven routing prediction at the edge," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 986–999, Jun. 2020.

[49] G. Chen *et al.*, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. NSDI*, vol. 8, 2008, pp. 337–350.

[50] Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for cloud systems," in *Proc. Int. Conf. Netw. Service Manag.*, 2010, pp. 9–16.

[51] A. Sacco, F. Esposito, and G. Marchetto, "Resource inference for task migration in challenged edge networks with RITMO," in *Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet)*, 2020, pp. 1–7.

[52] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Gener. Comput. Syst.*, vol. 27, no. 6, pp. 871–879, 2011.

[53] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in *Proc. Int. Workshop Qual. Service*, 2003, pp. 381–398.

[54] S. Khatua, A. Ghosh, and N. Mukherjee, "Optimizing the utilization of virtual resources in cloud environment," in *Proc. IEEE Int. Conf. Virtual Environ. Human-Comput. Interfaces Meas. Syst.*, 2010, pp. 82–87.

[55] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *Proc. IEEE Int. Conf. Auton. Comput.*, 2006, pp. 65–73.

[56] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow," in *Proc. 7th Int. Conf. Auton. Autonomous Syst.*, 2011, pp. 67–74.

Alessio Sacco (Graduate Student Member, IEEE) received the M.Sc. degree in computer engineering from the Politecnico di Torino, where he is currently pursuing the Ph.D. degree in computer engineering. His research interests include architecture and protocols for network management, implementation and design of cloud computing applications, algorithms and protocols for service-based architecture, such as software defined networks, used in conjunction with machine learning algorithms.

Matteo Flocco received the M.Sc. degree in computer science from Saint Louis University, where he worked as a Research Assistant with the Networking Group with Dr. F. Esposito. He currently works as a Full Stack Developer for Blue Reply, where he develops logistics softwares for large companies. His research interests mainly involve computer networks, with a particular focus on software-defined networking and congestion control algorithms, and machine learning applied to network management.

Flavio Esposito (Member, IEEE) received the M.Sc. degree in telecommunication engineering from the University of Florence, Italy, and the Ph.D. degree in computer science from Boston University in 2013. He is an Assistant Professor with the Department of Computer Science, Saint Louis University (SLU). He also has an affiliation with the Parks College of Engineering, SLU. He worked in the industry for a few years, and his main research interests include network management, network virtualization, and distributed systems. He is a recipient of several awards, including four National Science Foundation Awards and two Best Paper Awards, one at IEEE NetSoft 2017 and one at IEEE NFV-SDN 2019.

Guido Marchetto (Member, IEEE) received the Ph.D. degree in computer engineering from the Politecnico di Torino, in 2008, where he is currently an Associate Professor with the Department of Control and Computer Engineering. His research topics cover distributed systems and formal verification of systems and protocols. His interests also include network protocols and network architectures.