On Deep Reinforcement Learning for Traffic Engineering in SD-WAN

Sebastian Troia, Federico Sapienza, Leonardo Varé and Guido Maier

***Abstract*—The demand for reliable and efficient Wide Area Networks (WANs) from business customers is continuously increasing. Companies and enterprises use WANs to exchange critical data between headquarters, far-off business branches and cloud data centers. Many WANs solutions have been proposed over the years, such as: leased lines, Frame Relay, Multi-Protocol Label Switching (MPLS), Virtual Private Networks (VPN). Each solution positions differently in the trade-off between reliability, Quality of Service (QoS) and cost. Today, the emerging technology for WAN is Software-Defined Wide Area Networking (SD-WAN) that introduces the Software-Defined Networking (SDN) paradigm into the enterprise-network market. SD-WAN can support differentiated services over public WAN by dynamically reconfiguring in real-time network devices at the edge of the network according to network measurements and service requirements. On the one hand, SD-WAN reduces the high costs of guaranteed QoS WAN solutions (as MPLS), without giving away reliability in practical scenarios. On the other, it brings numerous technical challenges, such as the implementation of Traffic Engineering (TE) methods. TE is critically important for enterprises not only to efficiently orchestrate network traffic among the edge devices, but also to keep their services always available. In this work, we develop different kind of TE algorithms with the aim of improving the performance of an SD-WAN based network in terms of service availability. We first evaluate the performance of baseline TE algorithms. Then, we implement different deep Reinforcement Learning (deep-RL) algorithms to overcome the limitations of the baseline approaches. Specifically, we implement three kinds of deep-RL algorithms, which are: policy gradient, TD-λ and deep Q-learning. Results show that a deep-RL algorithm with a well-designed reward function is capable of increasing the overall network availability and guaranteeing network protection and restoration in SD-WAN.**

***Index Terms*—Software Defined Networking (SDN), Software-Defined Wide Area Network (SD-WAN), Deep Reinforcement Learning, Enterprise Networking.**

I. Introduction

A Wide Area Network (WAN), is a telecommunication network interconnecting multiple Local Area Networks (LANs), that are distributed over different geographic areas. Enterprises use WANs to connect their different branches to the headquarter and to reach Cloud services that are provisioned from a Cloud computing provider. The rapid evolution of Enterprise Networking (EN) and Information Technologies (IT) increases consistently the demand of higher capacity and higher quality WANs [1].

Sebastian Troia and Guido Maier are with the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano and SWAN networks, Milan, Italy. At the time this work was developed, Leonardo Var and Federico Sapienza were with the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Milan, Italy. E-mail: sebastian.troia@polimi.it

The first public WANs were deployed in the early 1980s and underwent continuous development. Initially, wide area connections were designed to connect two remote sites by using leased lines that had high costs and limited speeds [2]. Consequently, later many different technologies have been proposed with the purpose of letting enterprises to lease capacity from the operators of the public network infrastructure for building their inter-site connections. The enterprise WAN became an overlay network with similar performance as leased lines, but without the need of actually deploying a company-owned physical infrastructure. The main network technologies adopted have been: Asynchronous Transfer Mode (ATM), Frame Relay (FR), Multi-Protocol Label Switching (MPLS) and Virtual Private Networks (VPN). Among these technologies, MPLS is the one currently widely deployed that can provide guaranteed Quality of Service (QoS) with high efficiency. However, the cost of MPLS for enterprise users is quite high: we can say MPLS is at one extreme of the trade-off between cost and QoS. On the other extreme, we can place the VPN technology. A VPN overlay WAN can be created on virtually any kind of underlay network, and in particular also connecting the enterprise sites with simple and inexpensive Internet connections. The drawback is that there is no possibility of providing QoS. The high cost of MPLS is pushing enterprises towards the new technology named Software-Defined Wide Area Networking (SD-WAN). SD-WAN promises to be able to achieve QoS with low cost by taking advantage of broadband Internet [3].

SD-WAN has various advantages with respect to MPLS, such as reduced costs, simplified WAN configuration, easy access to cloud services and efficient WAN utilization. The basis of SD-WAN relies on Software-Defined Networking (SDN), which separates the physical forwarding elements, called data plane, from the network's control logic, called control plane, which is now implemented in a logically centralized controller; this simplifies network control, management, and enables innovation through network programmability. The control logic is what differentiate SD-WAN from the VPN solution, allowing to provide a good degree of QoS and reliability at the same access cost of VPN. That is achieved by the centralized controller, which constantly monitors the network conditions and consequently adapts the choice of connections. It's very important to highlight that SD-WAN is implemented only at the edges of the overlay network, i.e. inside the Customer Premises Edge (CPE) devices which are located in the enterprise's branches, headquarters and main offices. Therefore, the SD-WAN solution is totally independent of the service provider network. On the contrary, MPLS requires on a large extend the support of the network equipment located in the nodes of the carrier operator.

Nowadays, SD-WAN is gaining momentum as connectivity solutions across the enterprise landscape. Currently, there are around 30 SD-WAN vendors in the market with varying level of product maturity. Notably, those with mature product offerings include VeloCloud (owned by VMware), Viptela (owned by Cisco), Nuage (owned by Nokia), and Silver Peak. SD-WAN market is projected to surpass USD 17 billion by 2025[1]. For instance, VeloCloud offers an SD-WAN cloud-based solution delivering a broad set of integrated capabilities as a virtual network function (as opposed to the traditional hardware-centric model), which includes network overlay control, dynamic path selection, application performance monitoring, and other services.

If on the one hand SD-WAN introduces many advantages, on the other it brings numerous technical challenges, such as: 1) Placement of the software controller in cloud or at premises; 2) Reliability of the controller in case of failure; 3) Scalability of the network architecture in case of growing number of CPEs, hosts, overlay networks; 4) Traffic engineering methods to efficiently orchestrate network traffic among the CPEs; 5) Monitoring of WAN performance; 6) Security against common network attacks (as DDoS); and many others.

In this work, we target the Traffic Engineering (TE) features of an SD-WAN solution that uses broadband internet to connect its edge devices (CPEs). TE is crucial for network availability and reliability. Enterprises can orchestrate their traffic in consideration of the monitoring measurements of WAN performance, such as packet delay, loss, jitter, and service requirements. The main goal of this work is to study how to improve TE's algorithms in the SD-WAN context to achieve similar performance provided by the current MPLS. In this regard, we aim at improving the service availability by exploring the adoption of state-of-the-art and Machine-Learning (ML) algorithms in the SD-WAN controller to assess their advantages and limitations.

Specifically, we first evaluate the performance of baseline TE algorithms. Afterwards, we implement different deep Reinforcement Learning (deep-RL) algorithms to overcome the limitations of the baseline approaches. We implement three kinds of deep-RL algorithms, that are: Policy gradient, TD-λ and deep Q-learning. Deep-RL is a sub-field of ML that aims at finding the optimal actions in a given environment in order to reach specific targets. In the context of SD-WAN, the network data plane represents the environment, while the actions and the reward functions are specific software modules running on top of the SD-WAN controller. The main reason that led us to implement deep-RL algorithms is that they are capable of predicting the network performance degradation and acting on the environment by changing pro-actively the routing information. Thanks to this capability, we are able to build intelligent TE's algorithms capable of achieving similar performance with respect to guaranteed QoS WAN solutions. This work focuses on a basic SD-WAN network in which an enterprise needs to connect two remote offices through two different networks. The goal is to improve service availability by dynamically "switching" the traffic flows between the two networks. The switching occurs based on the status of the networks, such as the packet delay, loss, jitter and available bandwidth.

[1]Software-Defined Wide Area Network (SD-WAN) Market 2019 In-Depth Analysis of Industry Share, Size, Growth Outlook up to 2025. Market Study Report LLC, USA, 2020.

The paper is organized as follows: section II shows the related works. Section III introduces the technical challenges when deploying an SD-WAN solution and the paper contribution. Section IV introduces the SD-WAN architecture being considered in this work and the problem statement. Then, in section V we will show the proposed implementation by introducing the environment, reward functions and the agents. Section VI is devoted to the simulations and results. Section VII present a discussion on open issues and future work. Finally, Section VIII draws the conclusions of this work.

II. Related work

The academic research on SD-WAN seems to be at a primordial state: at the best of our knowledge, there are very few research works addressing traffic engineering in SD-WAN, and in particular using ML-based algorithms. Consequently, besides considering works strictly dedicated to SD-WAN solutions, we have included in our state-of-the-art survey those papers that propose TE (or decision-making) algorithms in SDN, where there is complete control over the network devices and not only at the edges.

Authors in [4] propose a cloud network architecture in which three data-centers of a Cloud Service Provider (CSP) are connected through different public Internet Service Providers (ISPs). An overlay network is created which nodes, corresponding to data-centers, can connect to the Internet through two different networks. Moreover, the three nodes are managed by a centralized SDN controller that sets forwarding rules over the overlay network topology. The authors focus on the design of the network architecture and on the resilience. The latter is ensured by the fact that the datacenters are multihomed, and that multiple distinct paths can be established between datacenters within the overlay. The controller is able to detect network failures, and react dynamically by diverting network flows towards alternative routes. Phemius et al. [5] propose a 1:1 protection scheme in an SD-WAN architecture composed by two CPEs and two WANs. Traffic flows are divided into critical and non-critical. Whenever a failure occurs on a path, non-critical traffic is stopped, while critical flows are directed to the working path. Their architecture is composed of different software modules working above the Floodlight Controller [6], a Java-based SDN Controller. As in [4], the authors focus on network resiliency by introducing traffic re-routing according to link failures, flows' priorities and QoS requirements. Authors in [7] propose a Dynamic Traffic Management (DTM) as a generic concept to tackle the problem of minimizing traffic transit expenses. It refers to different monetary cost of inter-domain traffic, such as the one related to the network energy consumption, and other kind of costs related to the volume of traffic. The latter may reflect network conditions, utilization of resources, distance of content location, congestion, etc. The authors focus on the optimization of monetary costs related to traffic transfer via WANs. The ability of SD-WAN to switch traffic flows from one link to another efficiently minimizes transit expenses. Authors in [8] propose an SD-WAN testbed made by two datacenters and two interconnecting WANs. The goal of their work is to verify that an adequate level of QoS can be guaranteed and to provide traffic priority in an SD-WAN network. In the deployed scenarios, the controller can efficiently manage 300 VoIP calls, using a maximum of 16% CPU load at the edge servers. Authors in [9] perform a techno-economic analysis from implementing SD-WAN with 4G/LTE for the Automated Teller Machine (ATM) networks. Most ATMs use only the Very Small Aperture Terminal (VSAT) access to connect their WAN via satellite. Having only VSAT access on most ATMs can be risky, especially if the satellite connection goes down. With SD-WAN, ATM will have at least two WAN connections to its network, as a result, if one of the connections is down, network traffic will not be interrupted. Based on the techno-economic analysis provided by the authors, the implementation of SD-WAN with 4G/LTE for the ATM network is feasible and profitable.

The studies presented so far implement basic SD-WAN network architectures based on two or three CPEs and two WAN interconnections. Furthermore, the TE mechanisms implemented on the aforementioned networks come into play only when there is a failure. In this work we develop intelligent TE algorithms capable, not only of intervening during network failures, but also before they occur, significantly increasing the overall system performance with respect to the baseline algorithms. In addition, we show a performance analysis in terms of service availability that is currently lacking in research works on SD-WAN. In the next paragraphs, we review state-of-the-art TE algorithms in SDN.

In Ref. [10], the Google's network infrastructure team shows how SDN can be exploited to optimize their Google's internal WAN. This WAN is fully controlled by an SDN controller and connects a dozen of data-centers across the planet. It has some unique characteristics: massive bandwidth requirements, elastic traffic demand and full control over the edge servers and data-center networks. SDN allows advanced centralized TE policies that allocate bandwidth among competing services based on applications priorities. In particular, they build a TE application with the aim of running their WAN links at near 100% of utilization, corresponding to 2-3x efficiency improvements relative to standard practice. Their solution enables to deploy cost-effective WAN bandwidth maximizing the network utilization. Another relevant paper has been published by Microsoft [11]. They present SWAN, a system that improves the utilization of inter-datacenters networks by centrally controlling the traffic and frequently reconfiguring the data plane, to match the current network needs. Their solution enables QoS by differentiating three priority classes (interactive, elastic and background), in which bandwidth and paths are allocated according to traffic priorities while maintaining fairness. Since uncoordinated data plane reconfigurations result in severe congestion and heavy packet loss, they demonstrate that leaving 10% free capacity on each link allows for a congestion-free reconfiguration plan in just a few steps. The adoption of SDN brings new chances to exploit Artificial Intelligence (AI) and Machine Learning (ML) to optimize the network usage. In particular, ML algorithms are applied to different network problems, such as: traffic classification, routing optimization, QoS prediction, resource management, and security [12]. As stated in [13], the idea of applying ML techniques to networks' optimization problems has first appeared in [14]. The authors proposed an intelligent Knowledge Plane (KP), able to perform network's optimizations in situations that are too complicated to be addressed by humans. In the last few years, ML algorithms have been proposed to tackle different traffic engineering problems, e.g. traffic prediction and routing decision processes. According to the specific problem to be solved, both supervised learning and reinforcement learning approaches have been investigated. Ref [15] shows the application of an RL-based approach to optimize routing. In particular, considering the traffic matrix as network state and the choice of a source-destination path as action, the objective of the RL is to select the optimal routing paths, with the final goal of minimizing the network delay. Their experimental results show an improvement compared to a deterministic routing scheme. Ref [16] proposes an RL-based agent to optimize routing strategy without human intervention. To exploit the periodical nature of the traffic [17], they make use of Recurrent Neural Networks (RNN) to build the proposed RL-agent with the aim of optimizing the network usage. In Ref [18], the authors make use of deep-RL for automatic routing in optical transport networks. The role of the deep-RL agent is to optimally route new traffic demands through specific end-to-end paths. The agent has to choose among the candidate paths that connect the source and the destination node of each traffic request by maximizing the network usage. Authors of [19] proposed an RL algorithm to perform efficient and privacy-preserving embedding of virtual graphs over multi-operator telecom infrastructures. Doke et al. [20] exploit deep-RL to design a time and cost-effective network load balancer to optimize inter-data center traffic. With the goal of avoiding hand-crafted load balancing policies, they exploit deep-RL to continuously update the implemented policies in presence of a dynamic environment. They compared the performance of their deep-RL agent, based on deep Q-learning, with other deterministic load balancing policies, such as Round Robin. As a result, they show that the deep reinforcement agents achieve similar performance with other existing load balancing policies highlighting the fact that the policy is self-learning, so it will adapt to dynamic environments. Many research works presented in this section assume the complete control over the devices in the network, thanks to which different kind of ML algorithms can be trained with heterogeneous and historical network data.

III. Motivations and paper contribution

The SD-WAN control plane has a global, centralized view of the network and thus also can access network statistics and properties through monitoring algorithms. With centralized TE solutions, this data can be leveraged to find globally optimal path assignments. Traditional TE mechanisms such as

RSVP-TE or LDP for MPLS rely on the local, limited view of the network from the ingress router. Programmability in SD-WAN offers possibilities to implement custom, fast, and efficient adaptive routing schemes that optimize for low end-to-end latency, as well as failure recovery mechanisms that can be achieved with low communication overhead and little controller interaction.

This work focuses on the development of TE algorithms to understand to what extend SD-WAN is able to actually provide service availability by exploiting broadband Internet. Specifically, we address the following technical issues related to TE that have not yet been investigated within the SD-WAN context, that are: the maximization of the service availability and the mitigation of the continuous switching of traffic flows among the overlay networks due to the high variation of traffic characteristics (e.g. packet delay). The latter will be considered hereafter as: channel flipping problem.

To do so, we first evaluate the performance of baseline TE algorithms. Afterwards, we implement different deep-RL algorithms to overcome the limitations of the baseline approaches. In other words, we study how historical network information acquired by the SD-WAN controller, by means of a monitoring system, can be exploited to improve the overall SD-WAN performance. In this regard, we implement three basic model-free deep-RL algorithms, that are: Policy gradient [21], TD-λ and deep Q-learning [22]. The implementation of more advanced deep-RL algorithms are left for the future work. As we mentioned early, in SD-WAN the control is limited to network devices (CPEs) placed at the network edges, without having any direct knowledge of the state of the network elements (nodes and links) between the edges. This condition also constrains the amount and type of data that we can use to train the ML algorithms, compared to the situation when we have control also over the inner portion of the network. That makes network optimization by ML more challenging than e.g. in MPLS. This work aims at showing that, despite such limitations, implementing deep-RL algorithms by making the best use of network data available at the edges of the network can effectively improve performance in terms of service availability. In this work, we refer to the basic SD-WAN architecture presented in [5], and focus on the implementation of ad-hoc reward functions.

The main contributions of this paper can be listed in the following points:

1) Implementation of deep-RL based TE algorithms in the SD-WAN context. With this contribution we prove the effectiveness of deep-RL in assessing the technical issues regarding the TE in terms of the maximization of the service availability and mitigation of the channel flipping problem.
2) Design of ad-hoc reward functions based on static and dynamic features. In the context of deep-RL, a well-designed reward function is very important since it determines the overall learning procedure. In this work we compare deep-RL algorithms performance based on different reward functions.

This investigation is timely, giving the enormous commercial success of SD-WAN technology, and at the same time is novel, since very few research scientific works have been published so far about this approach.

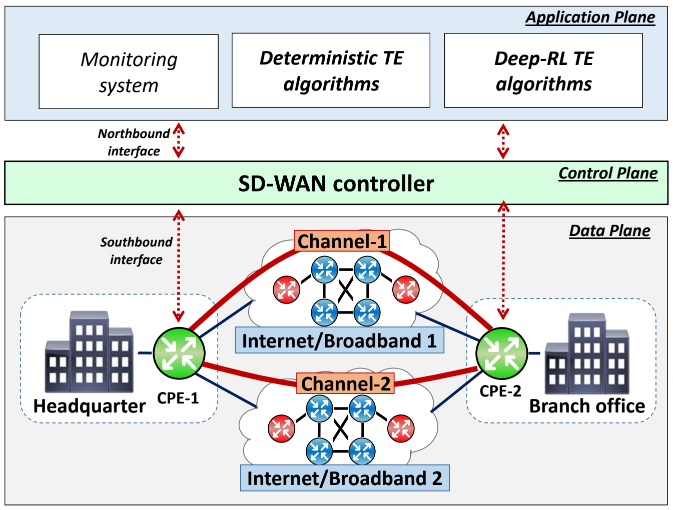

Fig. 1: SD-WAN network architecture. The CPE devices, which are located in the enterprise branches, are in charge of routing the traffic data into different WAN's.

IV. SD-WAN Architecture and Problem Statement

In our previous work [23], we have proposed an early implementation of SD-WAN based on open source components, such as OpenDaylight [24] as SDN controller, OpenvSwitch (OvS)[25] as switching element in the CPEs and a set of services for network monitoring and policy-based path selection. We presented a proof-of-concept through a simple emulated but realistic network environment composed by one headquarter and one branch office connected by two alternative networks (or channels). Data exchanged between headquarter and branch office flow through the most appropriate of the two channels, selected by the SD-WAN controller according to the status. The emulator was used to show new features and advantages for the enterprise in terms of resource optimization.

In this work, we exploit the same emulated SD-WAN scenario to focus on the deep-RL TE algorithms. SD-WAN architecture comprises three planes: data, control and application, see Fig. 1.

The data plane includes the network elements that carry user traffic, namely two CPEs and two WAN connections. CPE-1 and CPE-2 are network devices located at the customer locations and physically connected to the border routers of the service providers. The two WAN connections, Channel-1 and Channel-2, exploits two overlay tunnels configured over (two possibly different) underlay Internet/broadband networks. In our testbed we consider the overlay tunnels are obtained by the Generic Routing Encapsulation (GRE) tunneling protocol [26]. We simulate the two Internet/broadband connections so that they are characterized by different network performance in terms of packet delay. The control plane is embodied by the SD-WAN controller, that receives instructions from the application plane and relays them to the CPEs in the data plane. The controller is aware of the functional models of the CPEs and is therefore able to convert the high-level TE decisions received by the applications into appropriate instructions for the CPEs. In our case, these instructions are commands that make each CPE to switch on the appropriate channel when exchanging data with the remote CPE. Communications between CPEs and controller occurs across the South-Bound Interface (SBI) using an SBI protocol (in our case, OpenFlow). The application plane runs on top of the control plane and can be referred as the intelligence of the network. Applications are software programs that communicate policies and resource requests with the controller via Northbound Interfaces (NBIs). In our implementation, there is a single application performing TE and monitoring. More specifically, the application is running the algorithms that decides at each instant which one of Channel-1 and Channel-2 is the WAN channel the CPEs should use to communicate.

The problem statement of TE in SD-WAN is the following: *Given N the number of CPEs and C the number of channels interconnecting the CPEs, the goal of the TE algorithm is to pro-actively orchestrate the traffic among the CPEs to maximize the end-to-end service availability.*

In other words, the goal of this work is to develop a TE algorithm that minimizes service downtime by pro-actively modifying the traffic routes among CPEs. The pro-activity consists of predicting network issues and acting on traffic routing accordingly. State-of-the-art deterministic TE algorithms are not suitable for solving this type of problems as they do not perform any kind of prediction and take always the same decisions according to deterministic rules. For this reason, in this work we implement RL-based algorithms, which unlike deterministic ones, are able to learn how to orchestrate traffic over time to increase service availability.

In order to precisely define our TE problem, we assume the enterprise user has specified the QoS requirements of the applications running in the headquarter and in the branch office in the form of a Service Level Agreement (SLA). The SLA defines a set of thresholds to some performance parameters of the WAN connection between headquarter and branch office (e.g. packet loss, delay, jitter, minimum bandwidth, etc.). If the values of all the parameters measured over the connection are within the acceptable range (e.g. below the threshold), than the service is in up conditions, otherwise it is in outage. Since we want to maximize the service availability, the TE application will have to efficiently orchestrate the routing of the traffic flows between the two channels[2], so that the channel that is performing badly (i.e. has performance parameters outside the acceptable range) is possibly not chosen. In order to simplify our study, we have decided to consider just a single performance parameter, that is packet delay. The choice is motivated by that packet delay is simpler to control in our emulated testbed, as it will become apparent later on. As such, we define QoS thresholds based on packet delay. For instance, if the packet delay of channel-1, measured by the monitoring application, exceeds a certain threshold, then, the path-switching application instructs the controller to move the traffic flows from channel-1 to channel-2. This mechanism will eventually prevent the service from being down. We point out that we assume a monitoring application able to report the status of the two channels to the application plane where the TE applications run. In the context of a TE application, we can design several policies which define the expected behavior of the SD-WAN solution. In this work we consider four different TE policies:

[2]From now on we will use *channel* and *network* interchangeably.

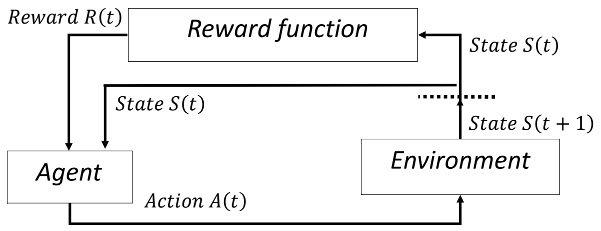

Fig. 2: Basic reinforcement learning system with an agent, a reward function and the environment.

1) Policy A: service flows can be steered into the two channels without any preference as long as the QoS thresholds are met.
2) Policy B: service flows are preferably routed into channel-1, that is the working channel. If the packet delay on channel 1 exceeds the threshold, then the service flows are moved into channel-2. They will come back to the first channel as soon as it becomes available (i.e. packet delay value under the threshold).
3) Policy A-1 and Policy B-1: they are an extension of policy A and policy B respectively; after a channel switch, no more changes are allowed for a fixed period of time. These policies eventually mitigate the flipping between the two channels when the packet delay variations are too high.

V. Deep Reinforcement Learning for SD-WAN

Reinforcement learning, in the context of artificial intelligence, is an approach to machine-learning that trains algorithms using a system of positive and negative rewards. Contrary to what happens with the supervised and unsupervised machine-learning algorithms, in RL we do not have training, validation and test set of data. A reinforcement learning algorithm, or *agent*, learns how to perform a task by interacting with its *environment*. The interaction is defined in terms of specific *actions*, *observations* and *rewards*. As we can see from Fig.2, an agent performs an action at time t based on the reward and state (or observations) obtained by the environment. The action performed by the agent will produce another state of the environment and a reward at a time $t+1$ and so on.

In this work, we developed a TE application based on RL algorithms. Therefore, we need to define several components such as: the environment (including the set of observations), the reward functions and the agents. In the context of an RL-based system, the environment is the SD-WAN data plane. As stated in section IV, we consider an environment with one headquarter and one branch office connected by two networks that can transport data. The agent comprises TE application implemented over the SD-WAN controller and the controller itself, as explained earlier. The reward function is computed by the same software module implementing the TE application in the application plane.

A. Environment

At each time step t, i.e. every second, the environment exposes its state $s(t)$ composed by 5 *observations*:

1) $O_1(t)$: used channel at time t.
2) $O_2(t)$: packet delay of channel 1 at time t.
3) $O_3(t)$: packet delay of channel 2 at time t.
4) $O_4(t)$: packet delay threshold of channel 1.
5) $O_5(t)$: packet delay threshold of channel 2.

The state is forwarded to the agent by means of the monitoring system. As shown in Fig. 2, the agent takes as inputs the state of the environment $s(t)$ and the result of the reward function $r(t)$. The TE application of the agent decides whether to trigger a switching of the channel in which the enterprise packets are flowing or not. This decision is forwarded to the controller, which in turn performs the switching action $a(t)$ over the environment by issuing commands to the CPEs thorugh the SBI. In our case, since we are considering CPEs equipped by OpenFlow switches (OVSs), an action will ultimately trigger an update in the forwarding tables of the CPEs at the customer locations.

Given the different combination of observations, we identify 10 environment states of interest used to develop the reward functions.

- $s_1 : O_2(t) \leq O_4(t) \wedge O_3(t) \leq O_5(t) \wedge O_1(t) = 1$. This state corresponds to the case in which the service flows are steered into channel 1 and both channels are in a good shape; this means that their actual packet delay value are under the QoS thresholds defined for each channel.
- $s_2 : O_2(t) \leq O_4(t) \wedge O_3(t) \leq O_5(t) \wedge O_1(t) = 2$. This state is the same of s_1 with the only difference that the service flows are steered into channel-2
- $s_3 : O_2(t) \leq O_4(t) \wedge O_3(t) > O_5(t) \wedge O_1(t) = 1$. This state corresponds to the case in which the service flows are steered into channel-1 and packet delay of channel-2 exceeds the threshold.
- $s_4 : O_2(t) \leq O_4(t) \wedge O_3(t) > O_5(t) \wedge O_1(t) = 2$. This state is the same of s_3 with the only difference that the service flows are steered into channel-2.
- $s_5 : O_2(t) > O_4(t) \wedge O_3(t) \leq O_5(t) \wedge O_1(t) = 1$. This state corresponds to the case in which the service flows are steered into channel-1 and packet delay of channel-1 exceeds the threshold.
- $s_6 : O_2(t) > O_4(t) \wedge O_3(t) \leq O_5(t) \wedge O_1(t) = 2$. This state is the same of s_5 with the only difference that the service flows are steered into channel-2.
- $s_7 : O_2(t) >> O_4(t) \wedge O_3(t) > O_5(t) \wedge O_1(t) = 1$. This state corresponds to the case in which the service flows are steered into channel-1 and packet delay of channel-1 exceeds the threshold much more than in channel-2.
- $s_8 : O_2(t) >> O_4(t) \wedge O_3(t) > O_5(t) \wedge O_1(t) = 2$. This state is the same of s_7 with the only difference that the service flows are steered into channel-2.
- $s_9 : O_2(t) > O_4(t) \wedge O_3(t) >> O_5(t) \wedge O_1(t) = 1$. This state corresponds to the case in which the service flows are steered into channel-1 and packet delay of channel-2 exceeds the threshold much more than in channel-1.
- $s_{10} : O_2(t) > O_4(t) \wedge O_3(t) >> O_5(t) \wedge O_1(t) = 2$. This state is the same of s_9 with the only difference that the service flows are steered into channel-2.

The reward function represents the feedback of the action performed by the agent. It depends on how many times the system is in the wrong channel, that is when the delay on the used channel overrun the threshold.

B. Reward functions

The goal of an RL-agent is the maximization of the cumulative sum of rewards. At each step, the action leads to a reward: the better the action, the better the reward. The design of efficient reward functions is a challenging problem [27][28] since it determines the overall learning procedure.

The output value of our reward function at a time t depends on the effect produced by the action performed by the agent on the environment. We define the reward $r(t)$ as follows:

$$r(t) = \sum_{n=0}^{N} \left(\frac{1}{2}\right)^n f(t-n) \tag{1}$$

where f represents the actual reward function and N is the number of previous environment states being considered to weight the final reward. For instance, if we consider $N = 0$, the final reward is: $r(t) = f(t)$; while if $N = 2$ the final reward will be a weighted sum of 3 reward values assigned in the previous time steps: $r(t) = f(t) + 0.5 \cdot f(t-1) + 0.25 \cdot f(t-2)$.

Based on the TE policies and the considered environment states introduced in section IV and V-A respectively, we design two kind of reward functions based on static $f_s(t)$ and dynamic $f_d(t)$ features, see table I.

1) Static reward functions: A static reward function is characterized by a fixed reward assigned according to the environment states. Specifically, based on the implemented policy, we assign positive, negative and neutral reward values (i.e. near zero or zero). Table I shows the reward assignment based on the strategy (static or dynamic) and the policy (A or B).

a) (Static) Policy A: We assign positive rewards when the action of the agent allows the achievement of the states: s_1, s_2, s_3, s_6. These states occur when the packet delay value of the channel, in which the service flow is steered, is under the QoS threshold. Rewards are negative when the agent reaches the states: s_4, s_5, s_7, s_{10}. These states occur when the QoS threshold is overrun in channel-1 or channel-2 respectively. Then, we assign neutral rewards, i.e. zeros, when states are considered neither good nor bad, such as: s_8, s_9. These states occur when both channels overrun the QoS thresholds, hence we assign zero if the used channel corresponds to the one with less packet delay value. In this work, we have adopted a greedy method to choose the value of α, i.e. we tried different values until we reached the one, $\alpha = 1$, that shows the best results in terms of actions performed by the agent.

TABLE I: Reward table

Environment state	Static reward $f_s(t)$		Dynamic reward $f_d(t)$	
	Policy A ($\alpha = 1$)	Policy B ($\beta = 3$)	Policy A	Policy B
s_1	α	β	$x(1,t)^{w_1}$	$x(1,t)^{w_1}$
s_2	α	$-\beta$	$x(2,t)^{w_1}$	$-x(1,t)^{w_2}$
s_3	α	β	$x(1,t)^{w_1}$	$x(1,t)^{w_1}$
s_4	$-\alpha$	-3β	$-x(2,t)^{w_2}$	$-x(1,t)^{w_2}$
s_5	$-\alpha$	-2β	$-x(1,t)^{w_2}$	$-[x(1,t)+x(2,t)]^{w_3}$
s_6	α	0.3β	$x(2,t)^{w_2}$	$[x(1,t)+x(2,t)]^{w_3}$
s_7	-2α	0	$-min[x(1,t),x(2,t)]^{w_3}$	$-min[x(1,t),x(2,t)]^{w_4}$
s_8	0	0.5β	$min[x(1,t),x(2,t)]^{w_3}$	$min[x(1,t),x(2,t)]^{w_4}$
s_9	0	0.5β	$min[x(1,t),x(2,t)]^{w_3}$	$min[x(1,t),x(2,t)]^{w_4}$
s_{10}	-2α	0	$-min[x(1,t),x(2,t)]^{w_3}$	$-min[x(1,t),x(2,t)]^{w_4}$

b) (Static) Policy B: As introduced in section IV, service flows are preferably routed into channel-1, that is the working channel, hence the reward function must be designed accordingly. Positive rewards are assigned when the action of the agent allows the achievement of the states: s_1 and s_3. These states occur when the packet delay value of channel-1, in which the service flows are steered, is under the QoS threshold. Rewards are negative when the agent reaches the states: s_2, s_4 and s_5. These states occur when either the QoS threshold is overrun or when the service flows are steered into channel-2 even if channel-1 is available (i.e. packet delay under the QoS threshold). Then, neutral rewards are assigned for the other states. In particular, we assign 0.3β when state s_6 occurs, i.e. when service flows are steered into channel-2 while channel-1 is unavailable. This kind of reward assignment is done to be consistent with the fact that the agent should prefer channel-1 to channel-2. The previous considerations apply to states s_7, s_8, s_9 and s_{10} as well. In this case, we choose $\beta = 3$ with the same greedy method being considered for policy A.

2) Dynamic reward functions: A dynamic reward function is characterized by one or more variables that change according to the conditions of the environment. Let us define the distance $m(c,t)$ between the actual packet delay and the QoS threshold for each channel c as follows:

$$m(c,t) = \frac{|d(c,t) - d_{TH}(c,t)|}{d_{MAX}(c,t)} \quad \forall c \in [1,2] \tag{2}$$

where $d(c,t)$ is the actual packet delay value, $d_{TH}(c,t)$ represents the QoS threshold value and $d_{MAX}(c,t)$ the maximum delay experienced by channel c obtained by historical observations of the channel.

Let us now define $x(c,t)$ as follows:

$$x(c,t) = \begin{cases} 10 \cdot m(c,t) & m(c,t) \leq 0.05 \\ 0.5 & m(c,t) > 0.05 \end{cases} \tag{3}$$

$x(c,t)$ represents the actual reward value; then, it can be positive or negative and weighted by exponential factors w_z (see Table I) based on the strategy and on the environment state. $x(c,t)$ depends on the difference between the packet delay and the QoS threshold. When the difference is high ($x(c,t) = 0.5$), the action choice is simple, since the packet delay is either way far under the threshold or over the threshold. Instead, we give more importance to the cases in which, being $m(c,t)$ small, the action choice is not straightforward ($x(c,t) = 10 \cdot m(c,t)$).

a) (Dynamic) Policy A: The considerations made for the static reward (Policy A) are the same with respect to the positivity or negativity of the reward value, therefore we consider $x(c,t)$ positive for those states in which the threshold is not exceeded, negative otherwise (see Table I).

In this case, the reward value is dynamic and depends on the actual distance between the packet delay of the channel and the QoS threshold, i.e. $m(c,t)$. Based on equation 3, we assign a higher value when $m(c,t)$ is less than 0.05, that is when the decision on which channel to choose is not straightforward.

Then, we raise the reward values to the power of w_z, which are weights used to distinguish the states of the environment. In this work, we assign the weights as follows: $w_1 = 0.25$, $w_2 = 0.5$, $w_3 = 0.25$, such that $\sum_{z=1}^{3} w_z = 1$, in order to give different level of importance to the states. By assigning the weight w_2 to the states s_4, s_5 and s_6, we are giving more importance to those events in which the threshold is overrun; while, for state $s_7 - s_{10}$ we choose the channel with minimum value of $x(c,t)$ in order to route the traffic into the best channel.

b) (Dynamic) Policy B: The considerations made for the static reward (Policy B) are the same with respect to the positivity or negativity of the reward value, therefore we consider $x(c,t)$ positive for those states in which the threshold is not exceeded and when service flows are steered into channel-1, negative otherwise (see Table I).

In this case, we assign the weights as follows: $w_1 = 0.1$, $w_2 = 0.4$, $w_3 = 0.4$ and $w_4 = 0.1$ such that $\sum_{z=1}^{4} w_z = 1$. Weights w_2 and w_3 give more importance to those states in which the threshold is overrun and the wrong channel is used. Moreover, we penalize/boost state s_5 and s_6 in which the working channel is unavailable.

3) Reward function with change penalty: Whenever the agent makes a channel switching, there is a short period of time in which the service flows are disrupted. This is mainly due to the fact that the agent's action is not instantaneous and consequently there is a small time frame in which service is down. Moreover, if the variation of the packet delay value is too high to fluctuate several times around the QoS threshold, it will generate the flipping problem, i.e. a continuous channel switching. The reward functions introduced in the previous sections do not avoid these issues. As consequence, we introduce a small variation to the equation 1 based on the number of channel switching performed in the previous time steps. In particular, we add a penalty value whenever there is a channel switching. This value changes based on the policy.

Considering the static policy A and B, the reward value decreases by adding a negative value equal to -0.5, see equation 4.

$$r(t) = \begin{cases} \sum_{n=0}^{N} \left(\frac{1}{2}\right)^n f_s(t-n) & O_1(t) = O_1(t-1) \\ \sum_{n=0}^{N} \left(\frac{1}{2}\right)^n f_s(t-n) - 0.5 & O_1(t) \neq O_1(t-1) \end{cases} \tag{4}$$

where O_1 is the used channel at a time t.

While, considering the dynamic policy A and B, the reward value decreases by adding a negative value equal to $-0.3C_N$, where C_N is the number of changes performed in the previous N time steps, see equation 5.

$$r(t) = \sum_{n=0}^{N} \left(\frac{1}{2}\right)^n f_d(t-n) - 0.3C_N \tag{5}$$

C. Agents

The agent, or network agent, takes as input the observations obtained from the environment and returns the decision whether or not a service should be steered into another channel. Modern RL algorithms can be divided into two families: model-free and model-based RL. By a model, we mean a function which emulates the environment being able to predict state transitions and rewards.

The main advantage of a model-based algorithm is that it allows the agent to plan ahead its actions by trying a range of possible strategies and explicitly deciding between its options. A famous example of this approach is AlphaZero [29]. When this works, it can result in a substantial improvement over methods that do not have a model. The main drawback is that a ground-truth model of the environment is usually not available to the agent. If an agent wants to use a model in this case, it has to learn the model purely from experience, which creates several challenges. On the other hand, model-free methods give up the potential gains due to the presence of an environment model and tend to be easier to implement and optimize. In this work, we focus on model-free RL algorithms which are more popular and have been more extensively developed and tested than model-based methods. These methods differ from each other from what they need to learn in order to accomplish the initial goal, for instance: policies, action-value functions (Q-functions), value functions, and/or environment models. In this work we implement three approaches to representing and training agents with model-free RL:

1) Policy Optimization: These methods learn a policy in order to accomplish the initial goal. A policy is a rule used by an agent to decide what actions to take. It can be deterministic or stochastic, in which case it is usually denoted by π. Policy optimization methods represent a policy explicitly as $\pi_\theta(a|s)$, where a is the action; s is the state and θ the set of parameters. In deep-RL, we deal with parametrized policies whose outputs depend on a set of parameters θ (such as the weights and biases of a neural network) which we can adjust (or learn) to change the behavior of the agent.

In this work we have implemented a policy optimization method called policy gradient [21]. In particular, the parameters θ are represented by the weights and bias of a neural network with one hidden layer composed by 40 nodes; while, the learning procedure is driven by the gradient descent algorithm [30].

2) Value functions: Differently from policy optimization algorithms, value functions tend to find a numerical representation of a state. By value, we mean the expected reward E for a state s under a policy π. Intuitively, it measures the total rewards achievable from a state following a specific policy. The value function is denoted by $V^\pi(s)$ and it is expressed as follows: $V^\pi(s) = E_\pi[r(t)\,|s]$.

Algorithms based on value functions replace all state space with values. Then, the agent chooses the action that moves to the higher value. The V value can be derived by different methods, such as Monte Carlo (MC) and Temporal-Difference (TD) [22]. Agents based on MC method assume that the interaction with the environment is divided into finite episodes (i.e. a collection of actions, states and rewards), therefore value estimates and policies are updated only at the end of each episode. MC estimates the value function $V^\pi(s)$ for a given policy π using the empirical mean of the rewards; in other words, the value function $V^\pi(s)$ is estimated by averaging the rewards obtained after the visit to the state s.

Agents based on TD method, differently from MC, exploits incomplete episodes. The $V^\pi(s)$ is updated before the end of the episode; as consequence, the value estimation of one state is built upon the estimation of upcoming states. This technique of using estimations of other states is called bootstrapping. The main advantage of this technique is that the learning procedure can be done online after every step (without waiting for the end of the episode), allowing continuous learning. However, the TD-λ method does not work well for problems with infinite number of states, so we need an approximation function which, as we will see in the next paragraphs, uses neural networks.

We implement the TD-λ [22] algorithm which is a constrained version of the MC method. The parameter λ ($0 \leq \lambda \leq 1$) measures how many states are used for the value estimation, starting from TD-0 that uses the estimate of the next-state. In this work, we set $\lambda = 0.5$.

3) Action-Value functions: Methods in this family learn a value for the action instead of a state. We aim at finding an optimal policy $\pi^* : S \rightarrow A$ for the agent to maximize the expected long-term reward function for the system. Accordingly, we first define value function $V^\pi : S \rightarrow \mathbb{R}$ that represents the expected value obtained by following policy π for each state $s \in S$. The value function V for policy π quantifies the goodness of the policy through an infinite horizon that can be expressed as follows:

$$V^\pi(S) = E_\pi\left[r_t(s_t, a_t) + \gamma V^\pi(s_{t+1}) \mid s_0 = s\right] \tag{6}$$

Since we aim to find the optimal policy π^*, an optimal action at each state can be found through the optimal value function expressed by $V^*(s) = max_{a_t}\{E_\pi\left[r_t(s_t, a_t) + \gamma V^\pi(s_{t+1}) \mid s_0 = s\right]\}$. If we denote

$Q(s,a) \triangleq r_t(s_t,a_t) + \gamma E_\pi [V^\pi(s_{t+1})]$ as the optimal Q-function for all state-action pairs, then the optimal value function can be written by $V^*(s) = max_a\{Q^*(s,a)\}$. Now, the problem is reduced to find optimal values of Q-function, i.e., $Q^*(s,a)$, for all state-action pairs, and this can be done through iterative processes. In particular, the Q-function is updated according to the following rule:

$$\begin{aligned} Q_{t+1}(s,a) &= Q_t(s,a) \\ &+ \alpha_t \left[r_t(s,a) + \gamma max_{a'} Q_t(s,a') - Q_t(s,a)\right] \end{aligned} \tag{7}$$

In eq. 7, the learning rate α_t is used to determine the impact of new information to the existing Q-value. The algorithm then yields the optimal policy indicating an action to be taken at each state such that $Q^*(s,a)$ is maximized for all states in the state space, i.e., $\pi^*(s) = \arg\max_\alpha Q^*(s,a)$.

Q-learning is based on the concept that the agent knows what will be the expected reward of each action at every step. It will perform the sequence of actions that will eventually generate the maximum total reward. Imaging an environment with 10000 states and 1000 actions per state, the Q-value methods are not able to infer new states from already explored ones. This shows two problems: 1) First, the amount of memory required to save and update all the Q-values would increase as the number of states increases; 2) Second, the amount of time required to explore each state would be unrealistic. In this work we implement the deep Q-learning [31] algorithm which use a neural network to approximate the Q-value function $Q^*(s,a)$. The state is given as the input; the hidden layer is made by 120 nodes; and the Q-value of all possible actions is generated as the output. Specifically, section V-D shows the implemented deep Q-learning algorithm with Experience Replay (ER). The ER is a feature needed to make the algorithm more stable since a nonlinear function approximator is used. The reader can refer to Ref. [31] and [32] for the mathematical background of this method.

4) Note on the hyper-parameters: Hyper-parameters are defined as parameters that are not learned by the agent but set before the start of learning. In order to find the numerous hyper-parameters, besides the standard trial and error procedure that is prone to errors, we have used the Hyperopt [33] library. Hyperopt uses sequential model-based optimization which is a Bayesian optimization technique that uses information from past trials to decide the next set of hyper-parameters to explore in the parameter space. As such, the hyper-parameters of the three agents have been evaluated by implementing this method.

D. Pseudo-code

Algorithm 1 shows the pseudo-code of the proposed deep Q-learning with ER based TE algorithm. First of all we initialize the parameters regarding the network and the algorithm. After that, the monitoring system is started, which has the task of collecting the network statistics in real-time (i.e. packet delay values of WANs). Then the loop begins, that is the agent reads the status from the environment, then, it checks if the policy is met (for example if the delay on channel 1 is below the threshold). If the policy is not met (i.e. it is FALSE), then the agent performs the action on the environment. After that, the status and the reward are updated, while the neural network weights are optimized by using the stochastic gradient descent [30]. This process is performed in a loop. In order to also take into consideration the cases in which the deep Q-learning algorithm does not give good results (i.e. service uptime under a threshold for long time), we have included a backup algorithm, that is one of the baseline algorithms in section VI-A3, that comes into play only when the global variable $auth$ is set to $DENY$, which means either there is a change in some settings of the TE algorithm (such as the policy), or the agent is not performing as expected.

Algorithm 1 Deep Q-learning with ER based TE algorithm

```
1:  Network initialization: policy (A or B), number of WANs (C), number of CPEs (N);
2:  Algorithm initialization: replay memory D, Q-network with random weights $\Theta$;
3:  Run monitoring system;
4:  $auth = ALLOW$;
5:  loop:
6:  Get state: $s_t = [O_1(t), O_2(t), O_3(t), O_4(t), O_5(t)]$;
7:  if $policy = FALSE \quad and \quad auth = ALLOW$ then
8:      Compute action: $a_t = \arg\max_\alpha Q^*(s_t, a_t, \Theta)$;
9:      Observe reward $r_t$ and state $s_{t+1}$
10:     Perform the action: $a$ on $n \in (1, N)$;
11:     Store $(s_t, a_t, r_t, s_{t+1})$ in D;
12:     Set $s_{t+1} = s_t$
13:     Sample random minibatch of transitions $(s_t, a_t, r_t, s_{t+1})$ from D;
14:     Set $y_j = r_j + \gamma \arg\max_\alpha Q^*(s_t, a_t, \Theta)$
15:     Optimize neural network weights by using stochastic gradient descent [30] on $(y_j - Q(s_t, a_j, \Theta))^2$;
16: end
17: if $auth = DENY$ then
18:     Stop the agent;
19:     Run backup algorithm;
20: end
21: goto loop;
```

VI. Simulations and results

In this section we focus on the results obtained by applying the algorithms shown in section V-C. However, before starting our analysis, we discuss the technical setup of the simulations, including: traffic data generation, reward-testing algorithm, baseline algorithms used for comparisons.

A. Simulations setup

1) Data generation: In this work we assume that the monitoring system is able to report real-time network traffic statistics to the software-defined controller, as in [34][35][36]. As previously mentioned, we consider the packet delay as metric to characterize the condition of the channels. As a result, we generate a dataset V containing packet delay values at each second of two days. In the dataset V, where $V \in \mathbb{R}^{[N \times T]}$, each row is in the form $v_i = [v_{i,1}, v_{i,2}, v_{i,3}, ..., v_{i,T}]$ where $N = 2$ represents the number of channels, $T = 172800(seconds)$ stands for the time interval and $v_{i,j}$ means the packet delay value of channel i_{th} during the time interval j_{th}.

To generate our packet delay dataset, we start to generate network traffic samples for both channels according to the evaluations made by the authors in [17]. Specifically, they assess that the transport network traffic is composed by a linear combination of 3 different components denoted as *eigenflows*. It is defined as a time-series that captures the network traffic variability between two nodes in a telecommunication network. Thus, the eigenflows can be divided into 3 classes: 1) Deterministic eigenflow: which captures the periodic trends between the couple of nodes; 2) Spike eigenflow: which captures the short-lived bursts between couple of nodes; 3) Noise eigenflow: which generates a random noise resulting from the stochastic nature of the traffic.

Therefore, as in [17], we generated our traffic dataset for the two channels as a linear combination of the 3 eigenfows (50% deterministic, 25% spike and 25% noise), and then we derived the delay according to the linear correlation between the two [37]. We assumed a minimum delay value equal to zero and a maximum value equal to 2 ms, and set the following delay thresholds: $O_4(t) = 1.1ms$ (channel-1) and $O_5(t) = 1.9ms$ (channel-2).

To test the performance of the proposed algorithms, we use the dataset to run our environment. Recalling the fact that the environment simulates the SD-WAN dataplane, we use the generated data to reproduce the daily life of an SD-WAN based network by simulating the traffic exchanged between the two CPEs. Hence, the proposed agents can act on the environment in order to learn how to maximize the service uptime through trials and errors.

2) Reward-testing algorithms: Beside the reward functions introduced in section V-B, we use the Inverse Reinforcement Learning (IRL) algorithm [38] to test the performance of our reward functions. IRL extracts a reward function "a posteriori" from the optimal behaviour of the agent. The optimal behaviour can, therefore, be used as expert actions and taken as the input to the IRL algorithm in order to compute the best reward function. We use this method as ground-truth to score our reward functions.

3) Baseline algorithms: In this work, we implement baseline TE algorithms as mean of comparison with the proposed deep-RL ones. We investigated the use of threshold methods proposed in literature for real-time protection and restoration schemes in networks, i.e. naive threshold-based methods [39][36]. By threshold-based we mean those methods that make use of QoS thresholds to solve connectivity issues. For instance, if the end-to-end packet delay over a network path exceeds a defined threshold, then the deterministic algorithm will deterministically react by always implementing an action, such as changing the routing towards a path with a packet delay value under the threshold.

We implement two different kind of deterministic algorithms:

- Naive: If the network traffic is flowing into channel-1 and the delay *threshold* is exceeded, then the Naive algorithm will steer the traffic into the second channel.
- Threshold with Weighted Moving Average and weighted moving Variance as margin (WMA-V): given the WMA denoted by $\hat{\mu}(x)$:

$$\hat{\mu}(x) = \frac{\sum_{i=1}^{i=N} w_i x_i}{\sum_{i=1}^{i=N} w_i} \tag{8}$$

and the weighted variance denoted by $\hat{\sigma}^2(x)$ [40]:

$$\hat{\sigma}^2(x) = \frac{\sum_{i=1}^{i=N} w_i \left(x_i - \hat{\mu}(x)\right)^2}{\sum w_i - \frac{\sum_{i=1}^{i=N} w_i^2}{\sum_{i=1}^{i=N} w_i}} \tag{9}$$

where $N = 8$ represents the number of time steps considered for the computation of $\hat{\mu}(x)$ and $\hat{\sigma}^2(x)$; $w_i = \frac{i}{N}$ are the weights, being $i = N$ the most recent packet delay sample and $i = 1$ the last one. Finally, the WMA_V is equal to:

$$WMA_V(x) = \hat{\mu}(x) + \hat{\sigma}^2(x) \tag{10}$$

The channel switching is triggered when the delay is greater or equal to $threshold - WMA_V(x)$.

B. Simulations results

In this section we show the results of the proposed algorithms. The goal is to evaluate the impact of different reward functions by considering 3 popular deep-RL algorithms. We run the simulations in an Ubuntu 18.04 x64 machine with 64GB of RAM, Intel(R) Core(TM) i7 - 7700 CPU @3.6GHz using openAI Gym Python library[3]. Moreover, we repeated the simulations 10 times as best practice [41][42].

We perform the following analysis:

1) Algorithm analysis: we evaluate how many samples are needed for different deep-RL algorithms to converge to a stable solution in terms of Service Uptime (SU). The last is defined as number of time steps in which the service is flowing into the channel with a packet delay value under the threshold. The main outcome of this analysis is to show which is the best-suited algorithm for TE in SD-WAN.
2) Reward function benchmark: we evaluate the performances of deep-RL algorithms according to the reward functions introduced in section V-B. Moreover, we implement the IRL to compare our solution with the optimal one.
3) Prediction capability: we show the prediction rate of the proposed algorithms. By prediction we mean the ability of the agents to switch the channel before the threshold overrun. With this analysis we aim at evaluating the number of times in which the agent triggers the channel switch on time, before the threshold gets actually overrun.
4) Channel flipping: we show two different approaches to solve the issue of channel flipping due to the high variation of packet delay. First, we introduce policy A-1

[3]Website: https://gym.openai.com/ (last access 15/07/2020)

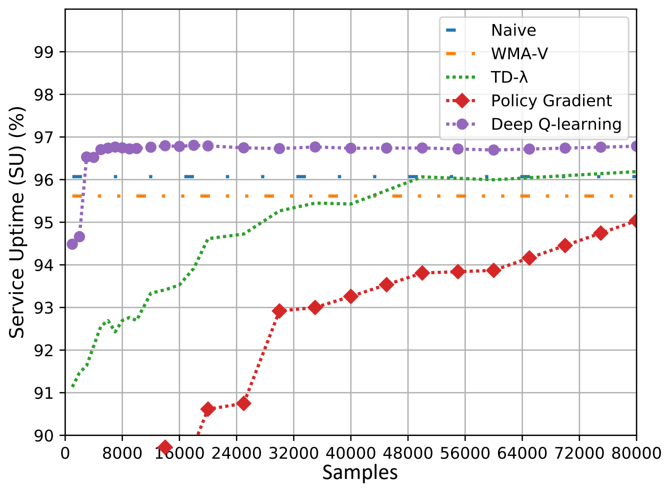

Fig. 3: Service uptime considering policy A and the proposed TE algorithms.

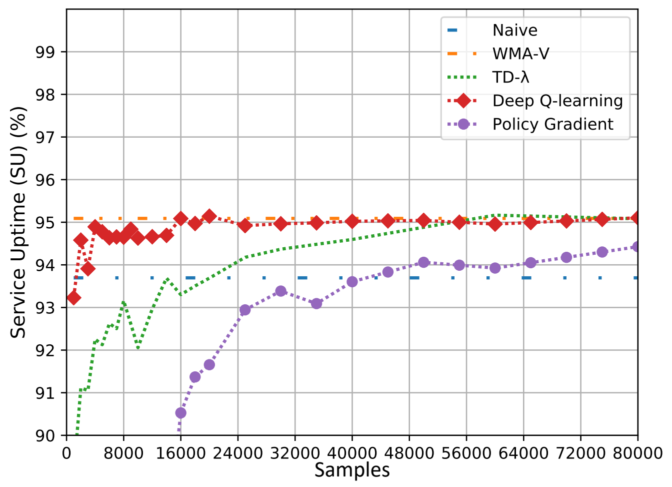

Fig. 4: Service uptime considering policy B and the proposed TE algorithms.

and B-1 in order to avoid consecutive channel changes, therefore we act directly on the agent's action; second, we modify the reward function in order to add a penalty in case the agent makes too many consecutive changes. This analysis aims at evaluating the most effective approach.

5) Scalability: we evaluate the scalability issues of this work by looking at the convergence time of the proposed algorithms as the number of channels increases.

1) Algorithms analysis: Figure 3 shows the behaviour of the proposed algorithms following policy A. We evaluate the percentage of time in which the service is up, as a function of packet delay samples received by the environment. We can clearly see that the deep Q-learning algorithm achieves better performances with respect to the other TE algorithms, such as policy gradient and TD-λ. We can also observe that deep Q-learning performs slightly better than naive threshold, (the best suited deterministic algorithm for this policy), by achieving almost 97% of SU.

Figure 4 shows the behaviour of the proposed algorithms following policy B, i.e. the agent prefers to steer the traffic into channel-1 rather than channel-2. In this case, the best deterministic TE algorithm is always the Naive, but the best deep-RL algorithms are deep-Q learning and TD-λ. The SU is a bit lower with respect to policy A, 95% for deep-Q learning and TD-λ. The convergence time, i.e. the time needed to reach the best results in terms of SU, of the deep Q-learning algorithm is much smaller w.r.t all the other deep-RL algorithms for both policies.

Furthermore, we evaluate the number of times in which policy B has been satisfied; in other words, the number of time steps in which the agent steered the service into channel-1. The agent was compliant with the policy during 92% of the time.

Figures 3 and 4 are obtained considering the best reward function that is dynamic (N=0) with change penalty and dynamic (N=7) for policy A and B respectively.

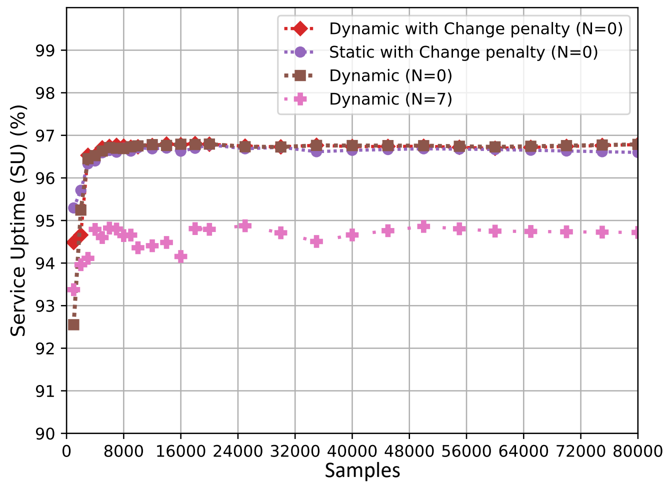

Fig. 5: Service uptime considering policy A and the proposed reward functions.

2) Reward function benchmark: In the previous subsection we have shown that, among the implemented deep-RL algorithms, deep Q-learning is the most promising one. Here we focus on analyzing the performances of this algorithm with the most suited reward functions. In this evaluation, we found that some of the proposed reward functions for policies A and B do not converge towards an acceptable solution, but remain highly variable. This means that they are not suitable for our purpose, therefore we have compared those reward functions that have proved to be suitable for our use case.

Figure 5 shows the performance of deep Q-learning with the following reward functions: dynamic with change penalty (N=1), static with change penalty (N=0), dynamic (N=0) and dynamic (N=7). As we can see, there is no advantage in increasing N, on the contrary, the reward functions with N = 0 shows a more efficient learning by the agent, with a gap of about 2% more in terms of SU. We can state that policy A does not need so much historical information to run efficiently.

Figure 6 shows the opposite situation w.r.t. the fig. 5.

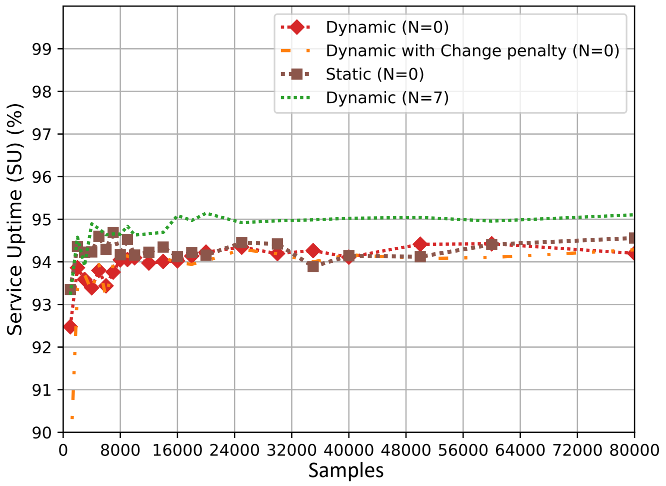

Fig. 6: Service uptime considering policy B and the proposed reward functions.

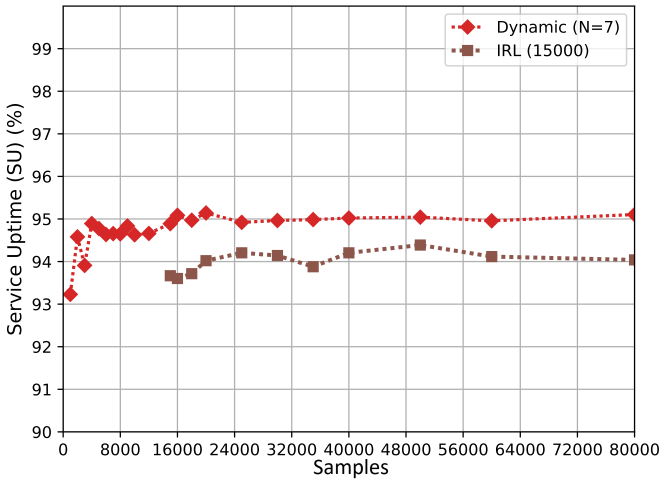

Fig. 7: Service uptime considering policy B, the dynamic reward functions with $N = 7$ and the IRL.

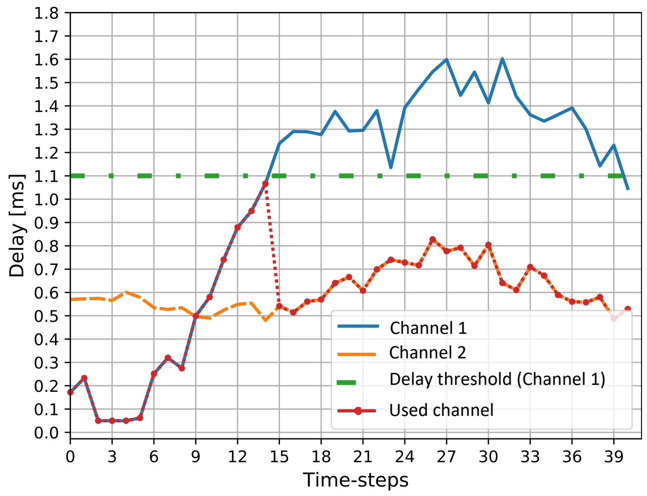

Fig. 8: Effect of deep Q-learning in predicting the channel switching in time.

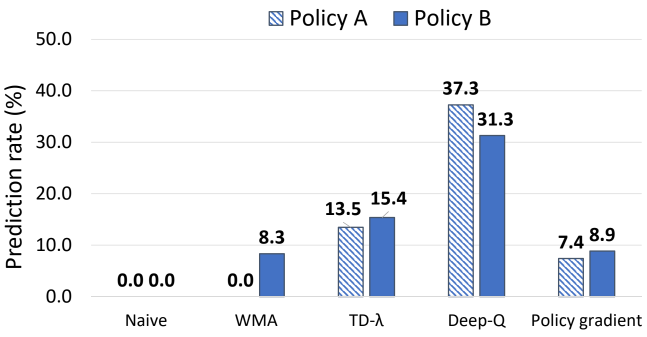

Fig. 9: Prediction rate of the proposed algorithms following both policies.

Considering policy B, the best reward function is the dynamic (N=7). After a first period of learning, the SU is stable at 95%, which means policy B benefits from a history of reward data since the agent must promptly redirect traffic into the first channel.

After testing the proposed reward functions, we compare the best one with the one approximated by the IRL. It allows to evaluate an optimal approximation of the reward function if we know in advance the future packet delay values of both channels and the best actions to keep the SU as high as possible. The IRL algorithm used the first 15000 samples to compute the reward function. Figure 7 shows that our proposed dynamic (N=7) reward function outperforms the best approximation made ”a posteriori” with IRL.

3) Prediction capability: The prediction rate of the algorithms is defined as the number of times in which the agent triggers the channel switch in time. This situation is depicted in Fig. 8, where we can see the effect of deep Q-learning on channel switching. In particular, we notice that the controller commands a transition from channel-1 to channel-2 at time-step 14, before the delay of channel-1 actually gets over the threshold (at time-step 14). Deep Q-learning reaches the best results in prediction by managing to switch the channel in time about 37.3% of times, as shown in Fig. 9. This result clearly outperforms all the other deterministic and deep-RL based approaches.

4) Channel flipping: In this paragraph, we evaluate the flipping problem. Policies A-1 and B-1 have been added with the aim of mitigating the effect of flipping when the packet delay of both channels is too high. Despite this, we have modified the reward functions by adding a penalty with the aim of constraining the agent’s actions to face this issue. Figures 10 and 11 show that it is wiser to modify the reward function rather than the policy to mitigate the channel flipping. As such, we obtain better results if we penalize channel changes when the variance of packet delay value is too high.

5) Scalability: Finally, we evaluate the scalability issues of this work. Figure 12 shows the convergence time of the proposed algorithms as the number of channels in the network increases. The convergence time is defined as the time it takes to reach the highest value of SU. As expected, the algorithm based on deep-Q learning increases the convergence time very slowly as the number of channels increases, from less than 1

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2020.3041385, IEEE Journal on Selected Areas in Communications

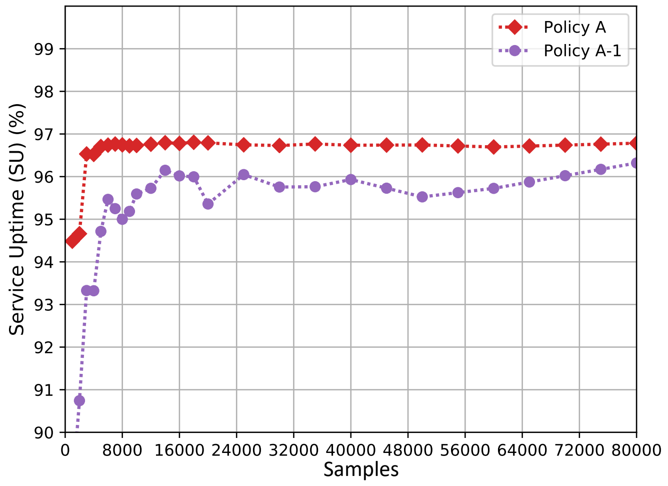

Fig. 10: Service uptime considering policy A and A-1 with a dynamic reward functions ($N = 0$) with change penalty.

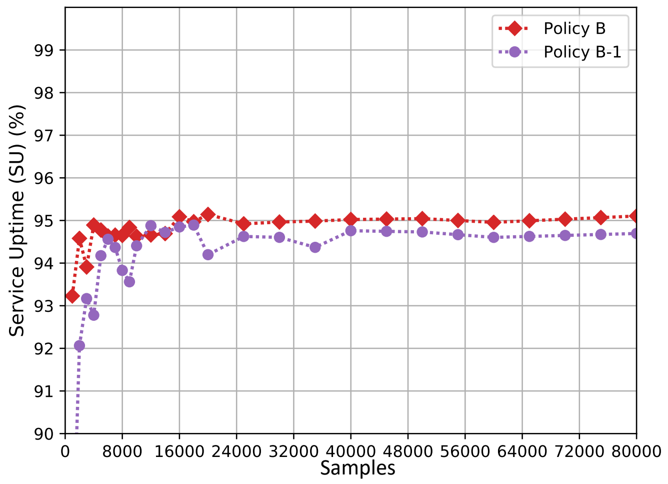

Fig. 11: Service uptime considering policy B and B-1 with a dynamic reward functions ($N = 0$) with change penalty.

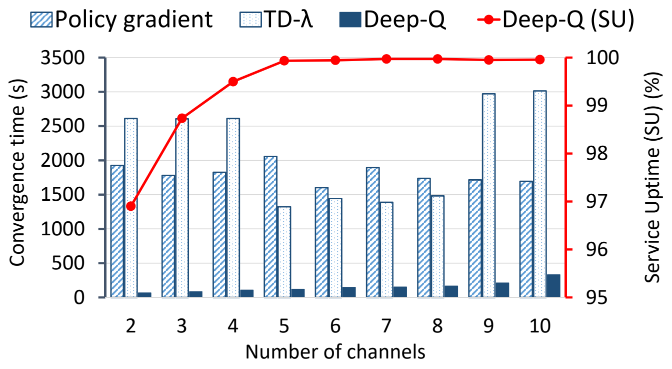

Fig. 12: Convergence time and service uptime for different number of channels considering policy A.

minute with 2 channels to about 5 minutes with 10; moreover, the SU increases up to 99.97% with 10 channels (SU=99.95% with 5 channels). This result suggests that the algorithm is able to maximize the SU with only 5 channels available. The algorithms based on policy gradient and TD-λ fail to converge in a reasonable time, keeping the SU always less than 96%.

VII. Open Issues and Challenges

In this section, we provide a discussion on the open issues that can be considered to improve our methodology and the future work. While algorithm 1 computes the set of actions aim at improving the service availability, it does not specify how to perform those actions to enforce a quick reconfiguration of the CPEs. To do so, it is necessary to make a testbed with a dataplane made by CPEs and WANs, a control plane made by a software controller and an orchestration plane where TE algorithms can run in form of software modules. The testbed is feasible if the algorithms are fast enough to make the reconfiguration of the CPEs useful. This work shows that deep-Q learning is efficient in the SD-WAN context (see figure 3) and also fast-running (see figure 12). Furthermore, we show a basic network architecture (2 CPEs and 2 WANs) in line with the state of the art research works. However, we have to consider more complex networks to be able to generalize the proposed TE algorithms. Specifically, we need to deal with different kind of topologies (e.g. mesh) and study how to efficiently orchestrate the services. In this case, more advanced deep-RL algorithms [43] for TE can be investigated, such as: double deep Q-learning, deep Q-learning with prioritized experience replay, distributional deep-Q learning, etc.

As mentioned in the Introduction section, there are several technical research challenges that, in our view, need to be addressed before deploying an SD-WAN solution as they heavily affect the overall performance. They are the following:

- The SD-WAN controller can be placed in the cloud or at the customer premises. Its placement can result in delays for the control plane operations, such as asynchronous switch of the traffic flows into the available channels given by the different delays between the controller and the CPEs. We would like to evaluate the effect of this positioning on the performance of the services flowing into the overlay networks.
- In an SD-WAN-based network, the failure of the central controller may collapse the overall network. In contrast, the use of multiple controllers in a physically distributed (but logically centralized) controller architecture alleviates the issue of a single point of failure but makes the control plane management more complicated.
- Scalability in terms of number of CPEs represents an important aspect to be considered when deploying an SD-WAN solution. In fact, as the network grows in size (e.g., CPEs, hosts, etc.), the centralized SD-WAN controller becomes highly solicited (in terms of events/requests) and thus overloaded (in terms of bandwidth, processing power and memory).
- An efficient network monitoring is required for the development of control and management applications in SD-WAN-based networks. However, collecting the appropriate data and statistics without affecting the network performance is a challenging task. In fact, the continuous monitoring of network data and statistics may generate

excessive overheads and thus affect the network performance whereas the lack of monitoring may cause an incorrect behavior of management applications.

- Network security is another crucial challenge for SD-WAN. The use of VPN tunnels among the CPEs and the controllers can reduce the risk associated to the most common network attacks such as DDoS. However, if a malicious CPE plugs into the network, this can put all security at risk. Consequently, it is of fundamental importance to study secure authentication mechanisms and management of overlay tunnels in order to avoid that malicious people put network security at risk.

VIII. Conclusions

An enterprise WAN is a corporate network that connects geographically spread sites that could be anywhere in the world. MPLS has been so far the main WAN optimization technology for EN, allowing to support various services such as Layer 3 MPLS-VPN over thousands of sites. Although MPLS has many advantages, SD-WAN is a new and growing paradigm that could achieve similar performance of MPLS more cost-effectively. In this work we aim at developing intelligent TE algorithms for SD-WAN to deliver service flows with certain QoS over broadband Internet by using software-defined capabilities. Due to the predictive nature of ML algorithms, we have demonstrated that TE algorithms based on RL outperforms all the other deterministic ones in terms of service uptime. Furthermore, we have demonstrated the importance of the reward function and how it can be designed to improve the general performance of an RL algorithm.

In an increasingly Cloud-centric world, SD-WAN represents a game-changer in driving improved return on network investments for EN services. It is universally acclaimed as a new and unprecedented way to easily implement policies across large WANs at a fraction of the cost of traditional solutions. This work represents one of the first scientific papers that address the TE features of an SD-WAN based network by exploiting deep-RL.

References

[1] O. group. (2019) Five ways sd-wan is transforming cloud connectivity. [Online]. Available: https://www.oracle.com/a/ocom/docs/five-ways-sd-wan-is-transforming-cloud-connectivity-wp.pdf

[2] R. Graziani and B. Vachon, "Quality of service and multi-protocol label switching," *White Paper*, 2014.

[3] S. Uppal, S. Woo, and D. Pitt, "Software defined wan for dummies," 2015.

[4] A. Fressancourt and M. Gagnaire, "A sdn-based network architecture for cloud resiliency," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, 2015, pp. 479–484.

[5] K. Phemius and M. Bouet, "Implementing openflow-based resilient network services," in *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*. IEEE, 2012, pp. 212–214.

[6] (2013) Floodlight. [Online]. Available: http://www.projectfloodlight.org/floodlight

[7] Z. Duliski, R. Stankiewicz, G. Rzym, and P. Wydrych, "Dynamic traffic management for sd-wan inter-cloud communication," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1335–1351, 2020.

[8] R. E. Mora-Huiracocha, P. L. Gallegos-Segovia, P. E. Vintimilla-Tapia, J. F. Bravo-Torres, E. J. Cedillo-Elias, and V. M. Larios-Rosillo, "Implementation of a sd-wan for the interconnection of two software defined data centers," in *2019 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2019, pp. 1–6.

[9] S. Andromeda and D. Gunawan, "Techno-economic analysis from implementing sd-wan with 4g/lte, a case study in xyz company," in *2020 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, 2020, pp. 345–351.

[10] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[11] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, 2013, pp. 15–26.

[12] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2018.

[13] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *Ieee Network*, vol. 32, no. 2, pp. 92–99, 2017.

[14] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 3–10.

[15] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[16] P. Sun, J. Li, J. Lan, Y. Hu, and X. Lu, "Rnn deep reinforcement learning for routing optimization," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*. IEEE, 2018, pp. 285–289.

[17] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," in *ACM SIGMETRICS Performance evaluation review*, vol. 32, no. 1. ACM, 2004, pp. 61–72.

[18] J. Suárez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routing based on deep reinforcement learning in optical transport networks," in *Optical Fiber Communication Conference*. Optical Society of America, 2019, pp. M2A–6.

[19] D. Andreoletti, T. Velichkova, G. Verticale, M. Tornatore, and S. Giordano, "A privacy-preserving reinforcement learning algorithm for multi-domain virtual network embedding," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020.

[20] A. R. Doke and K. Sangeeta, "Deep reinforcement learning based load balancing policy for balancing network traffic in datacenter environment," in *2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2018, pp. 1–5.

[21] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[22] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[23] S. Troia, L. Zorello, A. Maralit, and G. Maier, "Sd-wan: an open-source implementation for enterprise networking services," in *2020 22th International Conference on Transparent Optical Networks (ICTON)*, 2020, pp. 1–4.

[24] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 2014, pp. 1–6.

[25] Open vswitch. [Online]. Available: http://www.openvswitch.org/

[26] Rfc 2784 - generic routing encapsulation (gre) - ietf tools. [Online]. Available: https://tools.ietf.org/html/rfc2784

[27] M. Grzes, "Reward shaping in episodic reinforcement learning," 2017.

[28] A. D. Laud, "Theory and application of reward shaping in reinforcement learning," Tech. Rep., 2004.

[29] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017.

[30] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[33] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, “Hyperopt: a python library for model selection and hyperparameter optimization,” *Computational Science & Discovery*, vol. 8, no. 1, p. 014008, 2015.

[34] K. Phemius and M. Bouet, “Monitoring latency with openflow,” in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 122–125.

[35] D. Sinha, K. Haribabu, and S. Balasubramaniam, “Real-time monitoring of network latency in software defined networks,” in *Advanced Networks and Telecommuncations Systems (ANTS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1–3.

[36] J. M. Llopis, J. Pieczerak, and T. Janaszka, “Minimizing latency of critical traffic through sdn,” in *Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[37] M. Moudi and M. Othman, “On the relation between network throughput and delay curves,” *Automatika*, vol. 61, no. 3, pp. 415–424, 2020. [Online]. Available: https://doi.org/10.1080/00051144.2020.1774731

[38] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[39] T. Chin, M. Rahouti, and K. Xiong, “End-to-end delay minimization approaches using software-defined networking,” in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. ACM, 2017, pp. 184–189.

[40] GNU, “Gnu scientific library.” [Online]. Available: https://www.gnu.org/software/gsl/doc/html/statistics.html, note =

[41] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control,” in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[42] S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau, “Benchmarking batch deep reinforcement learning algorithms,” *arXiv preprint arXiv:1910.01708*, 2019.

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

Sebastian Troia received the Bachelor, Master and Ph.D. degree in Telecommunication Engineering from Politecnico di Milano, (Italy), in 2013, 2016 and 2020 respectively. He is currently a postdoctoral researcher with the Department of Electronics, Information and Bioengineering (DEIB) at the same university. From 2018 he joined to SWAN networks, a spin-off of Politecnico di Milano developing network orchestration and advanced algorithms for SDN and SD-WAN based networks. His current research interests are in the field of SD-WAN orchestration and control-plane architectures; Machine-Learning for communication networks; SDN and NFV.

Federico Sapienza received Master Degree in Telecommunication Engineering on December 2019, with final grade 110/110 cum laude. His main interests are Data networking and Computer Science with a particular focus on Software-Defined Networking (SDN) and Network Function Virtualization (NFV).

Leonardo Varé received the Master degree in Telecommunications from the Politecnico di Milano, Milan, Italy, in 2019. He is currently a Technical IP Engineer at Huawei, Milan. His academical interests are in the field of Machine Learning and software development with a main focus on Software-Defined Networking (SDN) and Network Function Virtualization (NFV).

Guido Maier received his Laurea degree in Electronic Engineering at Politecnico di Milano (Italy) in 1995 and his Ph.D. degree in Telecommunication Engineering at the same university in 2000. Until February 2006 he has been researcher at CoreCom (research consortium supported by Pirelli in Milan, Italy), where he achieved the position of Head of the Optical Networking Laboratory. On March 2006 he joined the Politecnico di Milano as Assistant Professor. In 2015 he became Associate Professor. His main areas of interest are: optical network modeling, design and optimization; SDN orchestration and control-plane architectures; SD-WAN and NFV. He is author of more than 150 papers in the area of Networking published in international journals and conference proceedings (h-index 23) and 6 patents. He is currently involved in industrial and European research projects. In 2016 he co-founded the start-up SWAN networks, spin-off of Politecnico di Milano. He is editor of the journal Optical Switching and Routing, General Chair of DRCN 2020 and DRCN 2021, guest editor of a special issue of the IEEE Open Journal of the Communications Society and TPC member in many international conferences. He is a Senior Member of the IEEE Communications Society.