Automated Saturation Mitigation Controlled by Deep Reinforcement Learning

Elkin Aguas*†, Anthony Lambert*, Grégory Blanc†, Hervé Debar†
*Orange Labs, Châtillon, France
{name.surname}@orange.com
†Télécom SudParis, Institut Polytechnique de Paris, Evry-Courcouronnes, France
{name.surname}@telecom-sudparis.eu

***Abstract*—Recent developments in orchestration and machine learning have made network automation more feasible, allowing the transition from error-prone and time-consuming manual manipulations to fast and refined automated responses in areas such as security and management. This article investigates the capabilities of a deep reinforcement learning agent to learn how to automatically share prefix announcements of an Autonomous System to its neighbors, in order to mitigate undesired network behaviors and therefore increase network resiliency and security. Our work focuses on network saturation, tackling the problem of network responsiveness in today's massive content delivery context. Results not only prove feasibility of such an agent, but also demonstrate its ability to minimize traffic loss as well as the number of actions to be performed by the automation process.**

***Index Terms*—deep reinforcement learning, network, automation, security, management, network resiliency, saturation.**

I. Introduction

Automation and machine learning have already proven they could improve detection and correction of security threats in networks [1]–[3]. Recent developments also suggest they are capable of optimizing security, management and performances of networking infrastructures [4]–[6]. Taking advantage of latest machine learning techniques, would especially allow automation, transitioning from error-prone and time-consuming manual manipulations to faster and more refined automated actions in responses to events. This could, for instance, alleviate the control problem that Carriers and Internet Service Providers (ISPs) face in today's massive content delivery context and which can result in traffic congestion issues. Indeed, Content Delivery Networks (CDNs), are not only responsible for a large part of today's traffic, but they also totally control how content is delivered by their geographically distributed cache servers. On the contrary, Carriers and ISPs are unable to predict where content traffic will enter their Autonomous System (AS). This lack of control increases their vulnerability to traffic saturation over their inter-domain links and can result in serious security issues (inter-domain sessions termination, for instance) that put at risk the good, working state of the network. They can however try to retain some control by sharing the announcements of their prefixes to their neighbors in order to try to influence the entry points of traffic into their AS. But this is a very error-prone and complex task to achieve manually, as it requires to find and maintain the correct sharing of prefixes over time.

978-1-7281-6992-7/20/$31.00 ©2020 IEEE

To this end, this paper evaluates the capability of a deep reinforcement learning (DRL) agent to improve the dynamic prefix load sharing of an AS in order to efficiently mitigate saturation, an issue in today's massive content delivery context. More precisely, we (i) propose a generic network automation architecture that automatically handles events that put at risk its proper operation, (ii) design a DRL agent which controls the choice of actions to be executed in case of saturation, (iii) evaluate its performance under three scenarios and with two different reward functions.

This paper is organized as follows: Section II provides a description of the delivery and saturation issue. Section III introduces our solution. Section IV details how we evaluate its performance and discuss the results from our experiments. Section V positions our work with respect to related works. Section VI concludes the paper.

II. Delivery and Saturation Issue Description

CDNs are built as overlay networks of geographically distributed servers. They implement complex and dynamic delivery strategies to select which server will deliver a given content to a given end-user at a given time. Such strategies range from choosing the closest servers to users, or the ones with the highest capacity, through optimizing the load on the CDN or even the CDN provider transit costs. A direct consequence is Carriers and Internet Service Providers (ISP) turning into "dumb pipes", as they have neither control nor insight on how traffic is delivered to their customers. Therefore, even if content delivery is legitimate traffic and cannot be considered an attack, the massive and unpredictable traffic shifts they trigger can have serious security issues. Indeed, they can lead to inter-domain links saturation, preventing users to access some services. Worse, control traffic can be lost, such as BGP keep-alive messages, resulting in external BGP (eBGP) sessions termination.

Fig. 1 depicts the root cause of such a situation leveraging a basic example. *ISP AS* owns three prefixes: *2.0.0.0/24*, *2.0.1.0/24* and *2.0.2.0/24*. The content is replicated into three possible caches: *ISP Cache* physically placed in the *ISP AS*'s network, but seen as another AS from a connection point of view (state-of-the-art practise); *Direct Cache* located in the

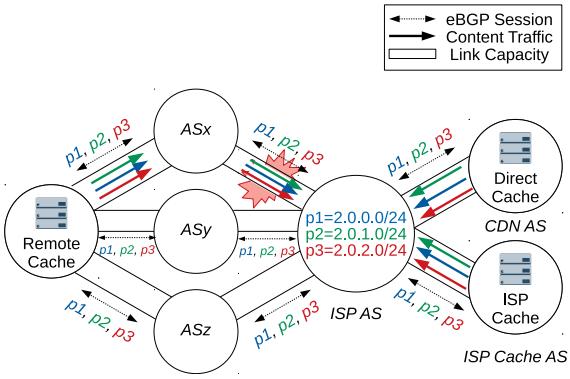

Fig. 1: Delivery and Saturation with Prefix Load Balancing.

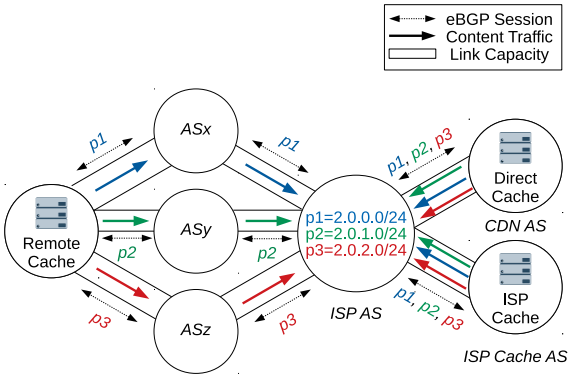

Fig. 2: Saturation avoidance through Prefix Load Sharing.

CDN AS, directly connected to *ISP AS*; and finally, *Remote Cache* located in some AS in the inter-domain[1]. The delivery strategy of the CDN is unknown, it might balance delivery from the three caches equally or prioritize delivery, serving from *ISP Cache* first, then from *Direct Cache* and finally from *Remote Cache* to offload traffic during peak hours, for instance. Depending on its strategy and the cache involved in the delivery, more or less serious saturation is likely to arise:

- In the case of *ISP Cache* (resp. *Direct Cache*), the cache capacity (resp. link capacity between *ISP AS* and *CDN AS*) is meant to fit the volume of traffic to be delivered. Moreover, the ISP and CDN provider are eager to upgrade it in case of saturation and no other traffic is likely to be impacted.
- In the case of *Remote Cache*, CDN traffic will be delivered to *ISP AS* either via *ASx*, *ASy* or *ASz*, depending on the BGP best path selection of the AS where the *Remote Cache* is located. First, none of these links is dedicated to content delivery traffic which will therefore be "in competition" with other traffic for capacity. Saturation, if it occurs, will as a consequence, not only impact content delivery traffic but other traffic too. Second, these links do not necessarily have the same capacity depending on the interconnection agreement settled between *ISP AS* and *ASx* (resp. *ASy*, *ASz*). Therefore, if the traffic is delivered via a link with a low capacity, then saturation is not only more likely to happen but also with more serious consequences. Finally, upgrading link capacity might prove hard to achieve, as agreements are renegotiated few times a year and due to peering ratio in the case of settlement-free peers, for instance[2]. This situation arises on the example of Fig. 1 because the ISP announces all its prefixes to *ASx*, *ASy* and *ASz* (prefix load balancing). The Remote AS has three possible paths to reach *ISP AS* (in Fig. 1, it prefers the path via *ASx* resulting in traffic being sent on that path). On the contrary, if the ISP shares its prefixes, announcing *2.0.0.0/24* (resp. *2.0.1.0/24*, *2.0.2.0/24*) to *ASx* (resp. *ASy*, *ASz*), then content will be delivered as depicted in Fig. 2 preventing saturation[3].

As a conclusion, depending on the CDN delivery strategy and its daily time evolution, saturation can arise especially on peering links. As previously explained, not only this issue is hard to prevent, but also to fix. Prefix load sharing is a possible workaround, but is error-prone and complex to set up, as finding and maintaining the appropriate load sharing is cumbersome. Work presented in [7] has proven automation is an efficient and scalable solution for prefix load sharing to balance traffic upon providers in a multi-homing situation. In this paper, we also apply automation to prefix load sharing but to mitigate saturation and further investigate how reinforcement learning can be used to optimize its actual control.

III. Proposed Solution

We now present our proposal of a generic automation solution that executes actions in a network in order to maintain its optimal level of service. We especially describe the agent that controls the execution of actions, and learns through reinforcement those that improve the achievement of the expected operating condition.

A. Generic Architecture Overview

The generic architecture of our solution is depicted on Fig. 3. A *Monitoring* block periodically performs measurements on the network and stores them in a first-in, first-out *Events* stack. A *Reasoning* block consumes the collected information to create a representation of the state of the network, and subsequently computes the probability of choosing each one of the possible actions of the *Actions* list to optimize the operating state of the network. Based on these probabilities, it selects the action to be carried out by the *Execution* block, via the *Orchestration* block. This architecture can therefore be

[1]For ease of understanding, we only consider one remote cache and three neighbors *ASx*, *ASy* and *ASz*. A more realistic situation would involve many more caches and neighbor ASes.

[2]ASes are generally connected either through a *customer-provider* relationship (the customer pays the provider for incoming and outgoing traffic), or as *settlement-free peers* (the ASes only exchange traffic towards their customers and for free). In the latter situation especially, a peering traffic ratio can be implemented to reach a balanced traffic between peers.

[3]Please note that, to ensure full connectivity, a less specific prefix, covering *2.0.0.0/24*, *2.0.1.0/24* and *2.0.2.0/24*, should be announced to *ASx*, *ASy* and *ASz*. For purposes of simplicity, it was not depicted in the examples.

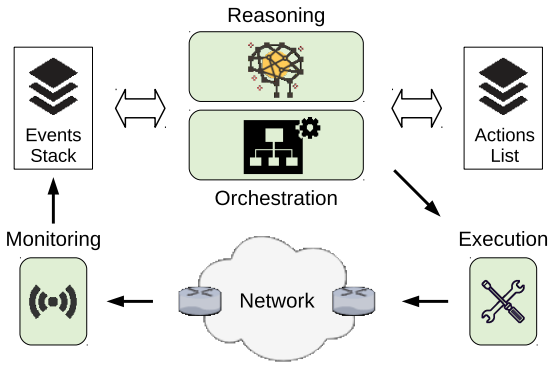

Fig. 3: Proposed Solution Generic Architecture.

seen as an evolution of the one presented in [7]. In particular, it leverages the use of a *Reasoning* block to control the execution of actions. While other works, like the one presented in [7], highlight the feasibility and scalability of automating actions in response to network events, our work focuses on the ability of machine learning to control the automation process.

B. Reasoning Block's Algorithm

The *Reasoning* block's algorithm is based on Q-learning, a model-free reinforcement learning (RL) algorithm, where, for each time step t, an *agent* observes its environment's *state* s_t, selects and executes an *action* a_t, receives a *reward* r_t and transitions to a new environment's state s_{t+1} [8]. Q-learning is goal oriented, and aims to learn a *policy* (which action to take under each state) that will lead the agent to increase its cumulative reward. However, simple Q-learning exhibits poor performance when *state* and *action* spaces are continuous and large. Deep neural networks (DNN) solve this problem by allowing the agent to learn more difficult policies [9]. To cope with complex policies and the constantly changing and sometimes chaotic nature of computer networks, we more precisely choose deep Q-learning (DQL) as the RL method for this work. The model generated by the DQL algorithm will work for the scenario showed in Fig. 1. Changing the scenario will require retraining the model. The target quality, Q, of a state-action combination is iteratively computed using (1),

$$Q_{t+1}(s_t, a_t) = r_t + \gamma \max_a Q_t(s_{t+1}, a) \tag{1}$$

where a belongs to the set of actions, and γ is the discount factor and determines the importance of future rewards. We set γ to 0.99, in order to favor future rewards.

Finally, we initialize the DNN's weights randomly. since we are interested in analyzing the learning performance of the algorithm. Using Fig. 1, we now define the states, actions and rewards of our algorithm.

1) States: At each time step t, the network's state is represented by nine metrics:

- $c_{ASx,t}$: (resp. $c_{ASy,t}$, $c_{ASz,t}$), the maximum capacity of the link between ISP AS and ASx (resp. ASy, ASz);
- $p_{ASx,t}$: (resp. $p_{ASy,t}$, $p_{ASz,t}$), the number of prefixes announced over the link between ISP AS and ASx (resp. ASy, ASz);
- $tr_{ASx,t}$: (resp. $tr_{ASy,t}$, $tr_{ASz,t}$), the volume of traffic through the link between ISP AS and ASx (resp. ASy, ASz).

2) Actions: The agent can perform seven actions:

- $m_{ASx \to ASy}$: (resp. $m_{ASx \to ASz}$), move prefixes from the link between ISP AS and ASx to the link between ISP AS and ASy (resp. ASz);
- $m_{ASy \to ASx}$: (resp. $m_{ASy \to ASz}$), move prefixes from the link between ISP AS and ASy to the link between ISP AS and ASx (resp. ASz);
- $m_{ASz \to ASx}$: (resp. $m_{ASz \to ASy}$), move prefixes from the link between ISP AS and ASz to the link between ISP AS and ASx (resp. ASy);
- *do nothing*.

The number of prefixes to be moved is controlled by a parameter: the *slope* denoted sl. It is computed from successive throughput measures:

$$sl_t = tr_t - tr_{t-1} \tag{2}$$

where tr_t is the total traffic entering the ISP AS at t:

$$tr_t = tr_{ASx,t} + tr_{ASy,t} + tr_{ASz,t} \tag{3}$$

The slope, therefore, represents the trend (increasing, decreasing, steady) of the total traffic entering the ISP AS.

3) Rewards: We opted for using reward functions that depend on network parameters. In this way, the reward is directly dependent on the state of the network, and we circumvent potential biased behaviors of the algorithm, that might arise when using predefined reward values. In this paper, we are interested in evaluating two different reward functions:

- *Balanced* (4): considers every link equally to redistribute prefixes.

$$\begin{aligned} r_t = -|c_{ASz,t} - tr_{ASz,t}| \\ -|c_{ASy,t} - tr_{ASy,t}| - |c_{ASx,t} - tr_{ASx,t}| \end{aligned} \tag{4}$$

- *Priority Aware* (5): prioritizes links over others (*ASz* over *ASy* over *ASx*) in order to enforce some routing policy, for instance, inter domain relationships (customers over peers over providers).

$$\begin{aligned} r_t = -w_0|tr_{ASz,t} - min(tr_t, c_{ASz,t})| \\ -w_1|tr_{ASy,t} - min(|tr_{ASz,t} - tr_t|, c_{ASy,t})| \\ -w_2|tr_{ASx,t} - min(|tr_{ASz,t} + tr_{ASy,t} - tr_t|, c_{ASx,t})| \end{aligned} \tag{5}$$

where w_0, w_1 and w_2 are the weights used to enforce priority.

IV. Performance Evaluation

A. Experiments setup

Our evaluation is based on a series of 1000 simulations of the environment depicted in Fig. 1. This environment is a

connection scenario between a CDN and an ISP. Its simplicity aims to characterize the behavior of the proposed solution. At each simulation we generate one day of traffic, represented by the total traffic in Fig. 6, using the model based on the traffic from a European tier-2 ISP (characterized in [10]). Moreover, we add random traffic variations to the model using a uniform probability distribution function to obtain a certain degree of uncertainty, like in real traffic.

$c_{ASx,t}$ (resp. $c_{ASy,t}$, $c_{ASz,t}$) is set to 20% (resp. 30%, 50%) (20-30-50), of the maximum generated daily traffic volume. We consider that ISP AS owns 100 prefixes and that all prefixes have the same importance in terms of traffic, i.e. each prefix is responsible for $1/100$ of the total traffic delivered. The ideal prefix load sharing to avoid saturation is therefore 20 (resp. 30, 50) prefixes announced over the link between ISP AS and ASx (resp. ASy, ASz). The initial percentage of prefixes per link is set randomly, but making sure saturation on exactly one link arises[4]. Moreover, one prefix is only announced over one link at a time (prefix load sharing).

We measure traffic every 5 minutes, resulting in 288 discrete time steps per day. Considering that making measurements every 30 minutes can highlight daily patterns in traffic [11] and knowing, from performance tests, that our solution can make measurements every second, a 5-minute monitoring frequency is a conservative approach that decreases the blackout period between measurements, compared to other works.

Under the conditions described above, for each simulation we compare the performance of the following algorithms:

- *No-function (NOF)* algorithm: At each time step, it does nothing. It represents today's situation when saturation arises.
- *Naive function (NAIF)* algorithm: At each time step, it evaluates the state of saturation of any two links, and if a link saturates it moves one prefix from that link to a non-saturated one. It is therefore a basic straightforward automation of today's manual prefix load sharing.
- *Balanced DRL (BDRL)* algorithm: At each time step, it aims to maximize the total reward given by (4). BDRL considers every link equally to redistribute prefixes.
- *Priority Aware DRL (PADRL)* algorithm: At each time step, it aims to maximize the total reward given by (5). PADRL prioritizes some links over others.

At each simulation, the performance of the four algorithms are evaluated using the following metrics:

- *Traffic loss*: cumulative loss of traffic due to saturation.
- *Number of actions*: cumulative number of actions taken.

It is in our interest to avoid traffic loss, but also to reduce as much as possible the number of actions. Indeed, using a large number of actions can harm the processing capacity, and therefore, the performance of the orchestration block. Thus, an algorithm that performs less actions will improve the scalability of the solution. It must also be noted that NOF (resp. NAIF) are actually used as baselines to estimate the traffic loss (resp. number of actions to be taken) in the worst case and better understand how our proposed DRL controllers perform as compared to these worst cases.

[4]To ensure comparability between the algorithms, since NAIF can only treat one saturated link at a time.

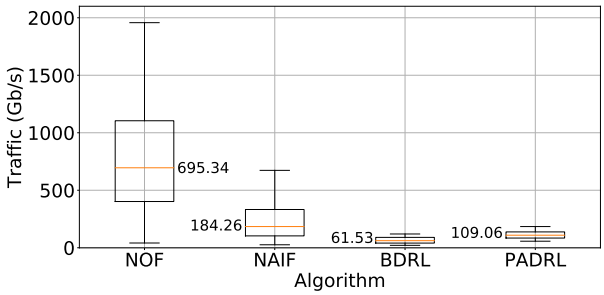

Fig. 4: Traffic loss per algorithm.

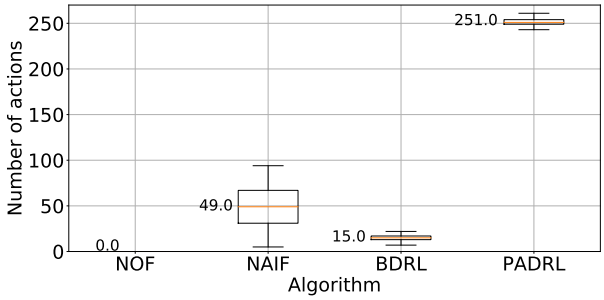

Fig. 5: Number of actions per algorithm.

Before running the simulations, we train the BDRL and PADRL models with 20000 episodes. At the beginning of each episode, the prefix distribution for the links is randomly selected, making sure only one link is saturated. Additionally, a random traffic curve is generated for each episode. Trained models and statistics of their performance are collected and saved every five episodes. When training is finished, models with the highest validation accuracy, lowest validation loss and highest average reward are chosen and used for the simulations. Simulations are implemented in Python, relying on TensorFlow and Keras libraries for DRL algorithms. For traffic related matters, we create a custom package that generates traffic and traffic peaks and computes the slope.

B. Performance Analysis

Boxplots on Fig. 4 (resp. 5) graphically depict distributions of traffic loss (resp. number of actions) over one thousand simulations for the four algorithms evaluated. As expected, in terms of traffic loss, NAIF, PADRL and BDRL outperform NOF which does nothing and is used as a baseline for traffic loss. NAIF (resp. PADRL, BDRL) reduces traffic loss by a factor of 3.7 (resp. 6.4, 11,3) when compared to NOF (using medians values displayed next to each boxplot). This gain however has a cost in terms of number of actions the automation process has to perform. The number of actions shows an increase, going from 0 to 15 (resp. 49, 251) for BDRL (resp. NAIF, PADRL) as compared to NOF. As a consequence, DRL based algorithms prove able to significantly

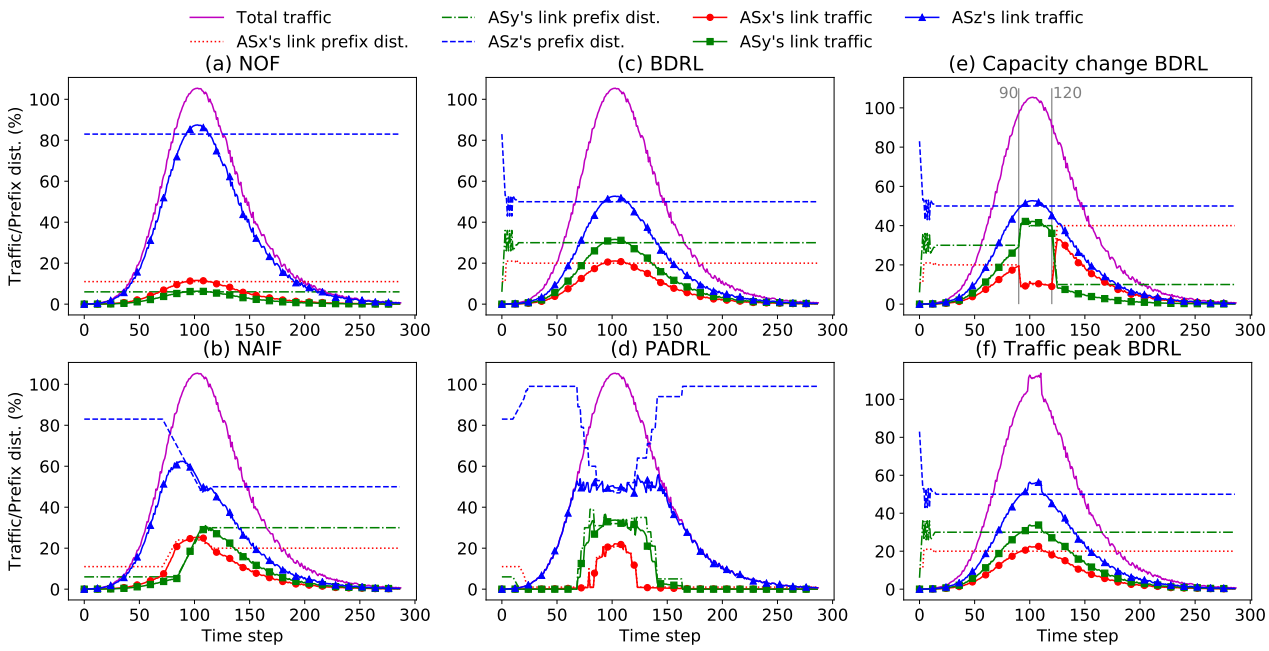

Fig. 6: Prefix and traffic distribution of NOF, NAIF, BDRL, and PADRL, Capacity change and Traffic peak of BDRL.

reduce the loss of traffic due to saturation. BDRL, especially is the best performing algorithm with the lowest traffic loss and number of actions. The latter is indeed reduced by a factor of 3.3 when compared to NAIF, our baseline for the number of actions. PADRL, on the contrary requires a lot more actions to be taken (5.1 times more than NAIF) as a consequence of its more complex reward function.

Fig. 6a (resp. b, c, d) details the prefix and traffic distribution of NOF (resp. NAIF, BDRL, PADRL) over one simulation, providing a deeper look into each algorithm behavior. As expected, NOF, our baseline for traffic loss, does not move any prefix and never reaches the appropriate (20-30-50) prefix distribution for ASx, ASy, and ASz's link resulting in a high traffic loss. NAIF, BDRL and PADRL on the contrary do redistribute prefixes over time and reach the expected distribution. NAIF, our baseline for the number of actions to be taken by an automation process, starts redistributing prefixes once saturation arises and progressively converges until reaching the ideal (20-30-50) distribution. This causes a rather important traffic loss and number of actions. BDRL on the contrary rapidly converges at the beginning of the simulation before any saturation arises (due to its training) which results in a low traffic loss and number of actions taken. The convergence however is not smooth, Fig. 6c exhibiting oscillations. This phenomenon, which we intend to investigate as part of our future work, might be improved by fine tuning the number of prefixes to move using for instance an extra parameter that would complement the use of the slope. Finally, Fig. 6d shows how PADRL successfully maintains a dynamic prefix distribution that both reduces traffic loss while prioritizing links over others. Implementing complex reward functions that mimic real-life cases where ISPs want to enforce priority to certain providers due to service quality or financial reasons therefore seems achievable. However this complexity has a heavy cost in terms of number of actions to be taken and affects the stability of the distribution process. Future work will explore the entanglement between the complexity of the reward function and the stability and scalability of the automated distribution process. Focusing on BDRL as it has exhibited the best behavior in terms of traffic loss reduction and number of actions taken, we finally are interested in evaluating the resiliency of such DRL controller to sudden (i) links' capacities change (Fig. 6e) and (ii) traffic peaks (Fig. 6f). First, BDRL is trained inducing random changes of 10%, 20%, 30%, 40%, and 50% of the maximum traffic capacity of the links. This implies an increase of the exploration space, making necessary a readjustment of the neural network characteristics to avoid potential underfitting. Fig. 6e details BDRL behavior over a simulation with capacity changes occurring at times 90 and 120. It appears BDRL rapidly and smoothly redistributes prefixes when these changes happen. Second, BDRL is trained inducing random traffic peaks in terms of occurrence time and duration, making traffic volume go beyond maximum capacity for all the links. Fig. 6f details BDRL behavior over a simulation. Redistributing prefixes is of no help in such a situation. Interestingly BDRL learns doing nothing and maintains the previously ideal (20-30-50) distribution, instead of trying to move prefixes which would lead to useless actions. Experiments therefore suggest that a DRL based controller is able to handle sudden stress without deteriorating overall performance of the solution. However, a wider range of actions has to be considered to get a deeper insight into DRL based controller ability to face critical and potentially contradictory decisions, such as dropping prefixes

to avoid saturation during peaks or not dropping them to keep service up for users.

V. Related Work

Reinforcement learning is nowadays commonly used for threats detection. Dionisio et al. have for instance recently proposed a processing pipeline that uses deep neural networks to retrieve IT infrastructure security-related information coming from Twitter and raise security alerts [3]. But, over the past few years the ability of reinforcement learning to control or optimize the control of networking infrastructures has also become an active area of research.

Some works have for instance proved the efficiency of DRL to optimize routing in (i) software defined networks [12], [13] in order to reduce delays, (ii) wireless networks [14] to reduce congestion, (iii) data centers [15] to load balance traffic.

Others have been investigating the ability of RL to control and optimize the quality delivered to end-users. Jay et al. have proposed for instance a congestion control protocol that leverages DRL to dynamically modulate traffic sources' data-transmission and efficiently utilize network resources [6]. While Sciancalepore et al. have been using a multi-agent RL to meet the required Quality of Experience (QoE) level of an application by dynamically selecting Classes of Services [16].

Finally, some works have been investigating how to leverage RL to control and optimize security responses to events. For instance, Liu et al. [17] have proposed a DRL based framework to defend against a wide range of Distributed Denial-of-Service (DDoS) flooding attacks by intelligently learning patterns of attack traffic, throttling this traffic and letting pass normal user traffic.

In the same vein of these works but different from them, we investigate the capability of a DRL agent to control the dynamic prefix load sharing of an AS in order to efficiently mitigate saturation.

VI. Conclusion

Automating actions to mitigate critical events such as saturation therefore appears feasible and do benefit from Machine Learning to optimize its control. We have evaluated the capability of a DRL agent to optimize the dynamic prefix load sharing of an AS to efficiently mitigate saturation, an issue in today's massive content delivery context. More precisely, we (i) have proposed a generic network automation architecture that automatically handles events that put at risk its proper operation, (ii) have defined a DRL agent which controls the choice of actions to be executed in case of saturation, (iii) have evaluated its performance under three scenarios and two different reward functions. Results have (i) proven feasibility of such an agent, (ii) demonstrated its ability to minimize traffic loss as well as the number of actions to be performed by the automation process, (iii) pointed out that distributing prefixes to every link equally was the most efficient strategy. Despite promising results, further aspects must however be explored, such as the relation between the complexity of the reward function and the stability of the distribution process, or the ability of our agent to adapt to network real time changes.

References

[1] S. Hao, N. A. Syed, N. Feamster, A. G. Gray, and S. Krasser, "Detecting spammers with snare: Spatio-temporal network-level automatic reputation engine," in *Proceedings of the 18th Conference on USENIX Security Symposium*, ser. SSYM'09. USA: USENIX Association, 2009, p. 101–118.

[2] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for dns," in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security'10. USA: USENIX Association, 2010, p. 18.

[3] N. Dionísio, F. Alves, P. M. Ferreira, and A. Bessani, "Cyberthreat detection from twitter using deep neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, July 2019, pp. 1–8.

[4] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez, "z-torch: An automated nfv orchestration and monitoring solution," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1292–1306, Dec 2018.

[5] T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink, and R. Mathar, "Deep reinforcement learning based resource allocation in low latency edge computing networks," in *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, Aug 2018, pp. 1–5.

[6] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 3050–3059. [Online]. Available: http://proceedings.mlr.press/v97/jay19a.html

[7] E. Aguas, T. Green, and A. Lambert, "Poster abstract: On the feasibility of event-driven network automation scenarios for bgp," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2019, pp. 1031–1032.

[8] M. Sewak, *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*. Springer, 2019, p. 2.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[10] Q. Grandemange, O. Ferveur, M. Gilson, and E. Gnaedinger, "A live network as-level traffic characterization," in *2017 International Conference on Computing, Networking and Communications (ICNC)*, Jan 2017, pp. 901–905.

[11] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix CDN," *CoRR*, vol. abs/1606.05519, 2016. [Online]. Available: http://arxiv.org/abs/1606.05519

[12] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *CoRR*, vol. abs/1709.07080, 2017. [Online]. Available: http://arxiv.org/abs/1709.07080

[13] C. Yu, J. Lan, Z. Guo, and Y. Hu, "Drom: Optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 64 533–64 539, 2018.

[14] R. Ding, Y. Xu, F. Gao, X. Shen, and W. Wu, "Deep reinforcement learning for router selection in network with heavy traffic," *IEEE Access*, vol. 7, pp. 37 109–37 120, 2019.

[15] A. R. Doke and S. K, "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)*, Aug 2018, pp. 1–5.

[16] B. Stefano, D. P. Francesco, G. G. Claudio, M. Salvatore, P. Martina, P. Antonio, R. C. Lorenzo, and S. Vincenzo, "A multi-agent reinforcement learning based approach to quality of experience control in future internet networks," in *2015 34th Chinese Control Conference (CCC)*, July 2015, pp. 6495–6500.

[17] Y. Liu, M. Dong, K. Ota, J. Li, and J. Wu, "Deep reinforcement learning based smart mitigation of ddos flooding in software-defined networks," in *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Sep. 2018, pp. 1–6.