Constrained Policy Optimization for Load Balancing

Ahmed Yassine Kamri, Pham Tran Anh Quang, Nicolas Huin, Jérémie Leguay
Huawei Technologies Ltd., Paris Research Center, France.

***Abstract*—To improve the bandwidth utilization in IP networks, a centralized controller splits flow aggregates over multiple paths and decides load balancing weights. Ideally, load balancing policies should anticipate the impact of their decisions on the Quality of Service (QoS). However, the embedding of accurate performance models into load balancing optimization algorithms is a challenge. In this context, we propose a Deep Reinforcement Learning (DRL) based solution that is able to learn the relationship between traffic and QoS, while providing safety to maximize throughput and avoid violating link capacity constraints. Our safe solution for QoS-aware load balancing integrates DRL algorithms with the Reward Constrained Policy Optimization algorithm. In a scenario where link delays follow the M/M/1 queuing model, we demonstrate, using a non-linear integer program, that our solution can reach a close to optimal end-to-end delay. We also show that our solution automatically learns reward parameters to meet capacity constraints.**

I. Introduction

Traffic engineering (TE) plays a crucial role in load balancing network traffic over time as it helps optimize the use of network bandwidth [1]. Indeed, Quality of Service (QoS) in terms of end-to-end delay and packet loss can be enhanced when network resources are efficiently utilized. The most popular load balancing mechanism, equal-cost multi-path routing (ECMP) [2], uniformly divides traffic across multiple paths between the origin and the destination. Uneven flow splitting mechanisms [3] can even further improve load balancing by using weights to control the amount of traffic sent over each path. However, none of the current approaches consider QoS measurements to adjust load balancing policies.

The emergence of Software-Defined Networking (SDN) [4] paradigm unveiled new capabilities for the global optimization of load balancing. In SDN, a centralized controller is responsible for computing routing and load balancing decisions. Moreover, it is able to monitor the traffic and the availability of resources. Therefore, it is capable of optimizing bandwidth utilization over time. In the literature, centralized methods such as Niagara [5] or IRSR [6] have been proposed to control load balancing weights so as to minimize a linear routing cost or the Maximum Link Utilization (MLU). However, these proposals are not explicitly trying to improve QoS metrics such as the end-to-end latency.

Accurate analytical models for end-to-end QoS performance metrics can be difficult to derive and integrate into routing optimization problems, as they are often intractable. Simple models have been embedded into routing algorithms, but in practice, they cannot properly estimate the latency. For instance, Ben-Ameur and Ouorou [7] considered the Kleinrock function [8] and gave a convex relaxation to compute a lower bound of the fractional routing problem. In a previous work [9], we integrated an M/M/1 model into a single path routing algorithm. The difficulty to embed accurate analytical models makes QoS routing an interesting opportunity for model-free solutions.

Reinforcement Learning (RL) is a sub-field of machine learning well suited for this kind of problems. Instead of having a predefined model, the reinforcement learning agent interacts with the environment and evaluates the consequences of its actions thanks to a reward function; it learns the characteristics of the environment by trial and error. Deep Reinforcement Learning (DRL) [10] combines deep learning and reinforcement learning principles (e.g., parametrize policies with neural networks). It has been applied to solve routing under the umbrella of *experience-driven networking* [10].

Applying Reinforcement Learning (RL) is generally difficult in practice. At each iteration, the agent performs an action based on the current state. The action produces a reward and the goal is to maximize the accumulated reward. Hence, the reward signal implicitly defines the behavior of the agent; misspecifications in the reward can lead to unwanted behaviors. A load balancing policy minimizing the average latency may violate link capacity constraints. Source-nodes may trigger, by overloading some paths, transport layer adaptions that reduces the overall throughput to obtain network states with lower latency. Therefore, we need to craft an appropriated reward signal that trades off latency against accepted traffic.

To deal with such behaviours, the LearnQueue reward [11] has been introduced to minimize the end-to-end delay while penalizing traffic rejections. However, it requires to weight properly the two objectives in the reward function. And tuning reward parameters can be tedious in practice as they heavily depend on the environment. To deal with these limitations and avoid tuning parameters by hand, we propose to use the Reward Constrained Policy Optimization (RCPO) algorithm [12], a novel multi-timescale approach for constrained policy optimization. RCPO guides the policy towards constraint satisfaction using an alternative penalty signal in the reward. The penalty is scaled using a Lagrangian parameter learnt during the training: constraints satisfaction becomes automatic.

Our paper makes the following contributions. We first describe the system architecture and the environment we developed based on the Python toolkit Gym [13]. We present our algorithmic solutions based on DDPG [14] as a base for the actor-critic model and on the RCPO algorithm to automatically enforce link capacity constraints while minimizing the average end-to-end delay. Finally, we compare our solution to LearnQueue and a non-linear integer model on an SD-WAN (Software-Defined Wide Area Network) use case and show

that a close to optimal delay can be achieved.

II. Related Work

Machine Learning (ML)-based techniques have been adopted for numerous networking applications [15]. The most remarkable advantages of ML-based techniques is their capability in coping with complex problems that are difficult to model with conventional approaches. To control a network, one can use supervised or unsupervised learning to predict the network status and give appropriate control decisions based on the predictions. Another way is to use RL to make decisions based on observable network states. In the context of traffic engineering, the latter approach has demonstrated that it outperforms other ML techniques [16].

There have been already a number of attempts to use RL algorithms to solve routing problems. Boyan and Moore [17] were one of the firsts and used tabular reinforcement learning to learn the best routing policy, by maintaining a table of all the current estimations of each node delay. However, the algorithm must be run online to be adaptive because it only stores the policy that is used while it is running, and do not have a function that maps the state of the network to the policy. Lin et al. [18] have also tried to solve the routing problem from a centralized point of view (SDN) with reward engineered from the QoS. The decisions here are taken each time a flow enters the network. Finally, Stampa et al. [19] have focused on the same problem considering the delay as the reward, but taking hop-by-hop routing decisions. Contrary to these works, we consider the load balancing of flow aggregates by a network controller over a set of pre-computed paths.

Only RILNET [20] deals with the load balancing problem using RL. It uses the Maximum Link Utilization (MLU) as the reward and an actor-critic model. However, having a lower MLU may lead to a better QoS, but the algorithm does not explicitly aims at optimizing the QoS.

III. System Architecture

In this section, we introduce the load balancing system architecture we considered and the environment we set up.

As illustrated in Fig. 1, the centralized network controller is equipped with a load balancing agent that periodically updates the load balancing policy to minimize the end-to-end latency for all tunnels. Each source device manages a set of *OD flows* for which they are the origin, also called *tunnels* in the rest of the paper. Tunnels can be split over multiple paths using target split ratios (or load-balancing weights) that are continuously updated by the controller. The set of candidate paths used by a source for a given tunnel can be provided by a local or a centralized path computation module. It is meant to be rather stable over time. The end-to-end delay on each path can be measured at each source with in-band telemetry for instance.

A. Gym Environment

We created a Gym environment [13] to train and test our proposed RL solution. In this environment, a network is represented by a directed graph where each edge e has

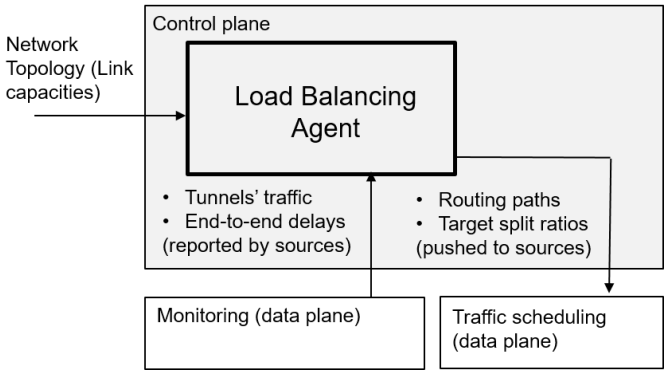


Fig. 1: Load balancing architecture.

a capacity c_e and an instantaneous load l_e. A tunnel k is defined by its source (s_k) and destination (t_k). The set of available paths for the tunnel is $\mathbb{P}_k$. The traffic demand of tunnel k at time t is denoted D_k^t and it can be split over all available paths $\mathbb{P}_k$. A routing function is implemented to enforce the load balancing decisions given by the DRL agent and to compute the resulting load and delay on each link. When links get overloaded, we adopt a max-min fairness rate allocation [21] to mimic TCP's behavior and adjusts the rate of each tunnel to link capacity constraints with a standard water-filling algorithm. Consequently, a portion of the traffic will be rejected and the amount of admitted traffic for tunnel k is denoted $\widehat{D}_k^t$, i.e., $\widehat{D}_k^t \leq D_k^t$.

To measure the amount of dropped traffic at time t, we define the traffic *enqueueing rate* as follow $\texttt{er}^t = \sum_k \widehat{D}_k^t / \sum_k D_k^t$. Also, based on the link load returned by the routing function, we compute d_e, the delay of edge e, using a simple M/M/1 queuing model as follows: $d_e = d_e^{prop} + \frac{1}{c_e - l_e}$ with d_e^{prop} the propagation delay that does not depend on the load. Thus the latency on a path p for a tunnel k is defined by $d_p^k = \sum_{e \in p} d_e$ and we consider that the delay d_k experienced by a tunnel corresponds to the maximum delay over all its paths, i.e., $d_k = \max_{p \in \mathbb{P}_k} d_p^k$.

B. MDP Formulation and DRL Setting

We can formulate the load balancing problem as a Markov Decision Process (MDP) defined by the tuple $(S, A, R, P, \mu, \gamma)$. Where S is the set of states, A the available actions, $R : S \times A \times S \to \mathbb{R}$ is the reward function, $P : S \times A \times S \to [0, 1]$ is the transition matrix and $P(s_0|s, a)$ is the probability of transitioning from state s to s_0 assuming action a was taken, $\mu : S \to [0, 1]$ is the initial state distribution and $\gamma \in [0, 1)$ is the discount factor for future rewards. A policy $\pi : S \to \Delta A$ is a probability distribution over actions and $\pi(a|s)$ denotes the probability of taking action a at state s. For each state s, the value of following policy π is indicated by: $V_R^\pi(s) = \mathbb{E}^\pi[\sum_t \gamma^t R(s_t, a_t)|s_0 = s]$. An important property of the value function is that it solves the recursive Bellman equation: $V_R^\pi(s) = \mathbb{E}^\pi[R(s, a) + \gamma V_R^\pi(s_0)|s]$.

To find the optimal policy, we consider a DRL agent that interacts with the environment by monitoring the observable

states s_t to determine actions. The observation space comprises the traffic demands D_k^t of tunnels, i.e., the state size is the number of tunnels. The action space a_t is determined by the split ratios of all tunnels, thus its size is $\sum_k (|\mathbb{P}_k| - 1)$. At each time step, the DRL agent computes the split ratios, and the routing function enforces them in the environment.

Our goal here is to minimize the sum of delays for all tunnels, i.e., $\sum d_k$. The challenge is to find a safe policy that avoids dropping traffic to minimize the delay. We discuss this issue in the next section and we propose a solution.

IV. Constrained Policy Optimization

To solve the load balancing problem with safety, we propose a solution based on the RCPO algorithm.

A. Motivations

To avoid that the policy rejects all traffic to minimize the delay and to enforce link capacity constraints, a solution called LearnQueue [11] has been proposed in the context of wireless networks. The key idea is to define a reward function that penalizes the actions resulting in large delays without causing a significant increase in the dropping rate. The reward function contains two main components: r_d, related to delay; and r_{enq}, related to packet drops. The total reward is defined as the weighted sum of the two terms as follows:

$$r_d = -\delta \times \sum d_k^t \tag{1}$$

$$r_{enq} = (1 - \delta) \times \mathtt{er}^t \tag{2}$$

Weights $\delta \in [0, 1]$ are used to combine the sum of delays and the enqueuing rate (i.e., the portion of accepted traffic). As one can see, the main issue with LearnQueue is that δ must be specified and the optimal value is not trivial to find. The RCPO algorithm presented next solves this issue.

B. RCPO

The Reward Constrained Policy Optimization (RCPO) [12] algorithm converts a Constrained Markov Decision Process to an equivalent unconstrained problem by adopting Lagrangian relaxation; the reward embeds the constraints, and the DRL agent learns how to obtain the optimal solution while alleviating constraint violations. It is proven that RCPO converges, under mild assumptions, to a constraint satisfying solution [12].

To express the constraints in RCPO, the authors introduced a penalty signal $c(s_t, a_t)$, a constraint $C(s_t) = F(c(s_t, a_t), ..., c(s_N, a_N))$ and a threshold $\alpha \in [0, 1]$. The constraint may be for instance a discounted sum like the reward or an average sum. Here we try to maximize the discounted reward with respect to the constraint $J_C^\pi \leq \alpha$ where $J_C^\pi = \mathbb{E}_{s_0 \sim \mu}^\pi [C(s)]$. The value of the discounted guiding penalty is $V_C^\pi(s) = \mathbb{E}_{s_0 \sim \mu}^\pi [\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)]$. The penalized reward function is then defined as $r'(\lambda, s_t, a_t) = r(s_t, a_t) - \lambda c(s_t, a_t)$ and the value of the new discounted penalized reward is equal to $V'^\pi(\lambda, s_t) = V_R^\pi(s_t) - \lambda V_C^\pi(s_t)$. The critic network can learn this penalized value by using temporal difference-learning. The RCPO is a three-timescale process, in which the actor and critic are updated following the penalized value defined above, and λ is updated using the constraints.

Algorithm 1: RCPO [12]

Input: penalty $c(.)$, constraint $C(.)$, threshold α, learning rates $\eta_1(k) < \eta_2(k) < \eta_3(k)$

Initialize actor parameters $\theta = \theta_0$, critic parameters $v = v_0$, Lagrange multipliers and $\lambda = 0$

for *k=0,1,...* **do**

  Initialize state $s_0 \sim \mu$

  for *t=0,1,...,T-1* **do**

    Sample action $a_t \sim \pi$, observe next state s_{t+1}, reward r_t, and penalties c_t

    $\hat{R}_t = r_t - \lambda_k c_t + \gamma \hat{V}(\lambda, s_t; v_k)$

    Critic update:

    $v_{k+1} \leftarrow v_k - \eta_3(k) \left[\frac{\delta \left(\hat{R}_t - \hat{V}(\lambda, s_t; v_k) \right)^2}{\delta v_k} \right]$

    Actor update:

    $\theta_{k+1} \leftarrow \Gamma_\theta \left[\theta_k + \eta_2(k) \nabla_\theta \hat{V}(\lambda, s) \right]$

  Lagrange multiplier update:

  $\lambda_{k+1} \leftarrow \Gamma_\lambda [\lambda_k + \eta_1(k) (J_C^{\pi_\theta} - \alpha)]$

return policy parameters θ

For our load balancing problem, we set the penalty $c(s, a)$ to be the sum of the differences between the loads and the capacities of the links of the network : $c(s_t, a_t) = \sum_e l_e - c_e$, where l_e is computed based on the traffic demands (s_t) and the split ratios (a_t). In our scenario, the constraint $C(s_t)$ is the average sum of the penalties $c(s, a)$. Since the loads have to be smaller than the capacity, we also have $\alpha = 0$.

C. Integration into DRL Algorithms

Both LearnQueue and RCPO can be combined with any DRL algorithm such as , PPO [22], DDPG [14]. DDPG has shown good performance in benchmarks [23] as well as in solving networking problems [24]. Consequently, we adopt DDPG as a based DRL algorithm to solve the load balancing problem.

V. Performance Evaluation

In this section, we compare the performance of our load balancing solution with two benchmarks, i.e., the LearnQueue reward and a non-linear model.

A. Benchmarks

We first compare our solution to the LearnQueue reward with different values of δ. We shall see that RCPO can automatically tune the reward to enforce capacity constraints and ensure traffic acceptance.

Additionally, we also compare against a model-based approach. Thanks to the use of the simple and popular non-linear M/M/1 queuing model to estimate the delay on each link in our simplified Gym [13] environment, we can directly use a closed-form expression of the delay to optimize load

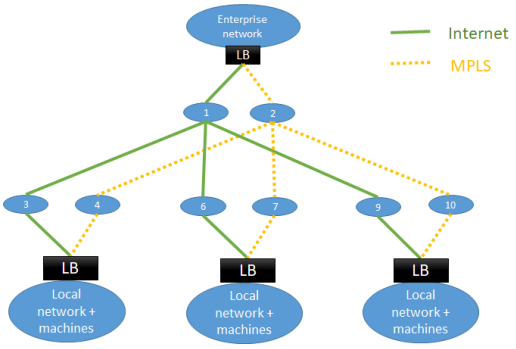


Fig. 2: SD-WAN scenario with an headquarters site and three remote sites connected through broadband Internet and MPLS. Load Balancing (LB) agents are deployed on access routers with two ports (Internet, MPLS).

balancing. Indeed, we can formulate a Non-Linear Integer Problem (NLP) and use it as a theoretical benchmark. To do so, we study the problem minimizing the delay of routed flows.

We can formulate the problem as follows. Let y_{pk} be a boolean variable indicating that the flow k has any amount of bandwidth routed through path p and let variable x_{pk} represents that amount. The delay of link e is given by variable z_e and d_k corresponds to the maximum delay experienced by flow k.

The objective function is :

$$\min \sum_{k \in K} d_k$$

We need to link y and x variables using the following constraints:

$$D_k y_{pk} \geq x_{pk}, \qquad \forall k \in K, p \in \mathbb{P}_k,$$

$$D_k x_{pk} \geq y_{pk}, \qquad \forall k \in K, p \in \mathbb{P}_k$$

Then, to evaluate the delay of a link, we use the M/M/1 queing model formula that gives us the following constraints:

$$z_e \geq \frac{1}{c_e - \sum_{k \in K} \sum_{p \in \mathbb{P}_k : e \in p} x_{pk}}, \qquad \forall e \in E$$

that can be rewritten as

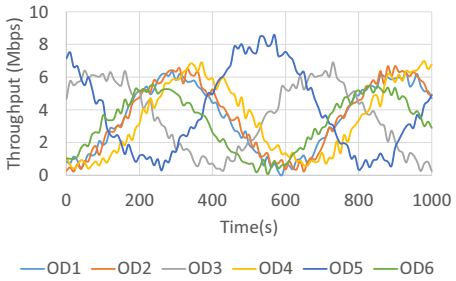
$$z_e \left(c_e - \sum_{k \in K} \sum_{p \in \mathbb{P}_k : e \in p} x_{pk} \right) \geq 1, \qquad \forall e \in E.$$

Capacity constraints are given by

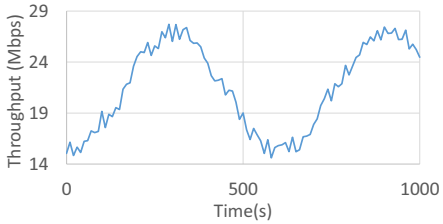
$$\sum_{k \in K} \sum_{p \in \mathbb{P}_k : e \in p} x_{pk} \leq c_e, \qquad \forall e \in E$$

Finally, the end-to-end delay constraints are given by

$$\sum_{e \in p} z_e y_{pk} \leq d_k, \qquad \forall k \in K, p \in \mathbb{P}_k$$



(a) Traffic inside tunnels.



(b) Total traffic.

Fig. 3: Traffic scenario in the simulation.

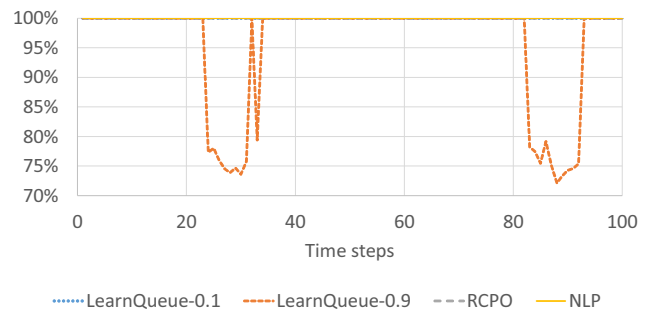
B. SD-WAN Use Case

Fig. 2 illustrates a typical SD-WAN network with one headquarters site and three branch sites that are multi-homed with Multi-Protocol Label Switching (MPLS) and broadband Internet connectivity. Traffic can go between the headquarters and remote sites, or between sites themselves. Origin-Destination (OD) tunnels carry traffic aggregates for the different types of application classes (e.g., real-time critical, elastic critical, elastic non-critical). For the sake of simplicity, we consider tunnels with the same priority.

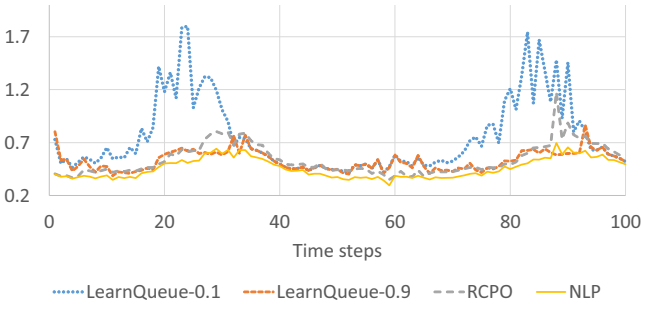
In the simulation, we created six tunnels where three are from HQ to sites, and three are from sites to HQ. The aggregate traffic pattern of each tunnel is diurnal. In Fig. 3, we show a traffic sample of 1000 seconds with the traffic of each tunnel and the total traffic in the network. The traffic of each tunnel can be split over two paths based on Internet and MPLS connectivity. The bandwidth of Internet and MPLS paths are 15 Mb/s and 6 Mb/s, respectively, to keep simulation time reasonable.

C. Hyper-parameters

A bad choice for the neural network architecture can impede the learning procedure in DDPG. For instance, the number of neurons per layer have to be scaled to the dimension of the action and observation spaces [25]. If the number of neurons is too low, it may not be sufficient to detect complicated patterns. Contrarily, overfitting can occur if the number of neurons is too high. Furthermore, the discount factor controls how much importance we give to the future rewards in comparison to the present reward and affects the convergence of the algorithms. Finally, the action noise is also an important parameter to trade-off between exploitation and exploration.



(a) Acceptance rate.



(b) Average delay (seconds).

(c) Maximum Link Utilization.

Fig. 4: Test results over the SD-WAN scenario.

For LearnQueue, we selected the following parameters after testing various configurations. For the architecture of the actor and critic neural networks, we choose two layers of 128 neurons, each with a $relu$ activation function. We set the discount factor γ to 0.7 and we train the model over 1 million time steps. To show two extreme versions, we present results for δ equals 0.1 and 0.9, to respectively focus either on throughput maximization or delay minimization. For the RCPO solution, we also considered actor-critic neural networks with two layers of 128 neurons, each with a $relu$ activation function. We set the initial value of the Lagrangian parameter λ to 0.8 and the learning rate for the Lagrangian parameter to 0.01.

For both algorithms, we used a noise to explore the action space during the training. It follows an Ornstein-Uhlenbeck process, with a mean of zero and a standard deviation of 0.5.

D. Numerical Results

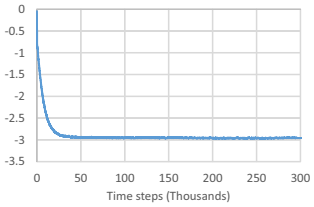
We now present results on the SD-WAN scenario. Fig. 4 shows the acceptance rate, the average tunnel delay and the MLU over a test episode of 100 traffic matrices, each one averaging traffic over 10 seconds. We can first observe that traffic is lost when $\delta = 0.9$ in Learnqueue. Indeed, this parameter scales between latency minimization and enqueueing rate maximization, and in this case, the policy focuses on delay minimization. On the contrary, for $\delta = 0.1$, no traffic is lost. This is also the case for RCPO, which automatically learns about capacity constraints while minimizing the delay. RCPO plots a lower delay than LearnQueue with $\delta = 0.1$, leading to a smaller average delay by 25%. The ideal benchmark solution from the NLP model gives the optimal delay with 100% traffic acceptance. As we can see, RCPO finds a delay not too far from this optimal value. Fig.1 also shows the evolution of the MLU over time steps. We can see that LearnQueue with $\delta = 0.1$ reaches twice high MLU, making the spikes of delay because of the M/M/1 model. Interestingly, we can observe that the MLU is not correlated with the average delay. The NLP model for instance generates a high MLU but yields to the smallest delay. This highlights the need to explicitly consider the delay in the load balancing optimization, instead of minimizing the MLU.

We now present the losses of the DDPG actor and critic networks during the training. We only show the actor and critic losses from beginning to 300 thousands time step because all algorithms converges after 300 thousands time steps. As shown by Fig. 5 for LearnQueue, the losses reveal instabilities during the training procedure, especially for $\delta = 0.9$, which can highlight the difficulty to find a satisfactory policy. However, the algorithms did converge and the best policy is saved. Fig. 6 presents the actor and critic losses for RCPO. RCPO performs better than LearnQueue since the algorithm learns the Lagrangian parameter and simplifies the tuning procedure. However, the choice of the learning rate and the initial value of the Lagrangian parameter remains important.

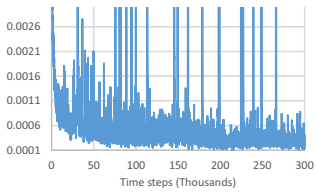
VI. Conclusion

We have presented a deep reinforcement learning solution for load balancing to optimize the end-to-end average latency of a set of tunnels. To provide safety and avoid throughput degradation, we have enforced capacity constraints in the policy optimization using the RCPO algorithm. We have demonstrated that a close to optimal delay can be achieved while automatically learning reward parameters to meet capacity constraints. This approach outperforms the LearnQueue reward, which is difficult to parametrize.

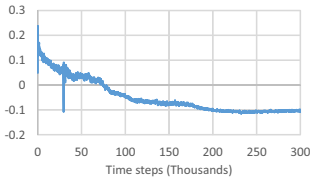
Other approaches can be considered for safe reinforcement learning. For instance, in future works, we may combine control barrier functions and DRL algorithms as proposed by Cheng et al. [26]. We may also test the algorithm with more complicated objectives related to QoE in a real actionable network environment.



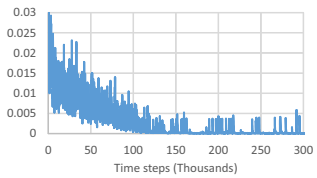
(a) Actor, $\delta = 0.1$



(b) Critic, $\delta = 0.1$

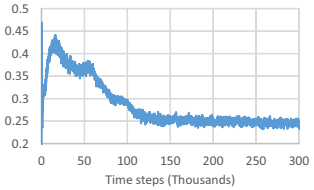


(c) Actor, $\delta = 0.9$

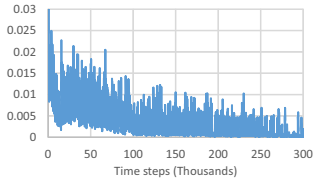


(d) Critic, $\delta = 0.9$

Fig. 5: Losses for LearnQueue on the SD-WAN scenario.



(a) Actor loss



(b) Critic loss

Fig. 6: Losses for RCPO on the SD-WAN scenario.

References

[1] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Com. Surveys Tutorials*, vol. 10, 2008.

[2] D. Thaler and C. Hopps, "RFC 2991: Multipath issues in unicast and multicast next-hop selection," 2000.

[3] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers," in *Proc. ACM EuroSys*, 2014.

[4] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[5] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *Proc. ACM CoNEXT*.

[6] P. Medagliani, J. Leguay, M. Abdullah, M. Leconte, and S. Paris, "Global optimization for hash-based splitting," in *Proc. IEEE GLOBECOM*, 2016.

[7] W. Ben-Ameur and A. Ouorou, "Mathematical models of the delay constrained routing problem," *Algorithmic Operations Research*, vol. 1, no. 2, 2006.

[8] L. Kleinrock, *Communication nets: Stochastic message flow and delay*. Courier Corporation, 2007.

[9] N. Huin, J. Leguay, and S. Martin, "Génération de colonnes pour le problème de routage à délai variable," in *ALGOTEL*, 2020.

[10] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM*, 2018.

[11] N. Bouacida and B. Shihada, "Practical and dynamic buffer sizing using learnqueue," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1885–1897, 2018.

[12] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *CoRR*, vol. abs/1805.11074, 2018.

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *CoRR*, vol. abs/1606.01540, 2016.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control wuth deep reinforcement learning," *ICLR*, 2016.

[15] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, vol. 32, 2018.

[16] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *ACM HotNets*, 2017.

[17] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in neural information processing systems*, 1995, pp. 369–376.

[18] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *IEEE SCC*, 2016.

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

[20] Q. Lin, Z. Gong, Q. Wang, and J. Li, "RILNET: A Reinforcement Learning Based Load Balancing Approach for Datacenter Networks," in *ML for Networking*, 2019.

[21] D. Bertsekas, "Gallager., r. 1992. data networks."

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[23] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," *CoRR*, vol. abs/1709.06560, 2017.

[24] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A Deep Reinforcement Learning Approach for VNF Forwarding Graph Embedding," *IEEE TNSM*, vol. 16, no. 4, pp. 1318–1331, 2019.

[25] K. G. Sheela and S. N. Deepa, "Review on methods to fix number of hidden neurons in neural networks," *Mathematical Problems in Engineering*, vol. 2013, 2013.

[26] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proc. AAAI*, vol. 33, 2019, pp. 3387–3395.