Constrained Deep Reinforcement Learning for Smart Load Balancing

Omar Houidi*, Djamal Zeghlache*, Victor Perrier†‡, Pham Tran Anh Quang†,
Nicolas Huin†, Jérémie Leguay†, Paolo Medagliani†
*Telecom SudParis, Samovar-Lab, Institut Mines-Telecom, Institut Polytechnique de Paris, France
†Huawei Technologies Ltd., Paris Research Center, France
‡Telecommunication for Space and Aeronautics (TESA) Laboratory, Toulouse, France
Email: {omar.houidi, djamal.zeghlache}@telecom-sudparis.eu, victor.perrier@tesa.prd.fr,
{phamt.quang, nicolas.huin, jeremie.leguay, paolo.medagliani}@huawei.com

***Abstract*—In this paper, we explore the use of an actor-critic architecture for Deep Reinforcement Learning (DRL) to improve load balancing beyond traditional algorithms. Some centralized Reinforcement Learning (RL) algorithms have targeted in the reward function expression the Quality of Experience (QoE) for video flows, but this requires access to clients, or the Maximum Link Utilization (MLU) for other types of flows. In our approach, we tune the actor-critic algorithm to only leverage on QoS parameters in order to load balance traffic in the network and maximize the QoE experienced by the users. This avoids having to collect observations and performance measurements from client applications, as it only focuses on network metrics that can be easily measured. We explore both centralized and distributed solutions to assess the feasibility of the proposed smart load balancing solutions. We compare them to ECMP, QoE-based reward methods, and RILNET that uses an underlying DDPG optimization approach. The proposed algorithms are shown to outperform previous approaches.**

***Index Terms*—Deep Reinforcement Learning, Smart Load Balancing, Intelligent Routing, QoE Optimization.**

I. Introduction

Traffic engineering plays a key role in load balancing to enhance Quality of Service (QoS) by using more efficiently network resources [1]. The most popular load balancing mechanism, i.e., Equal-Cost Multipath routing (ECMP) [2], splits flows evenly over candidate paths between origin (source) and destination pairs, but performs poorly when confronted with bursty traffic or in the presence of elephant flows. Uneven flow splitting across multiple paths, i.e., Unequal Cost Multipath routing (UCMP), slightly improves the performance but does not resolve much longer-term optimization for load balancing when traffic patterns are unknown and hard to forecast or predict. In UCMP, a weight is associated with each next-hop (i.e., path), and traffic is distributed across the next hops in proportion to their weight. An alternative to these traditional approaches consists of tackling the load balancing problem through a model-free approach that can learn and reinforce its decisions over time to provide stable and improved performance, no matter the applications flow and traffic patterns emerging in the networks.

Model-free approaches are also motivated by the difficulty of tuning load balancing weights to optimize Quality of Experience (QoE) as well as predicting well future conditions. Even if network calculus or queuing models can estimate latency, packet loss, and jitter, the associated models remain quite complex when deriving end-to-end performance. The behavior of the applications and transport-layer mechanisms are also difficult to capture. All these reasons lead us to network metrics that are correlated and inherently reflect QoE.

To this end, we propose a set of Deep Reinforcement Learning (DRL) algorithms that optimize QoS factors that are correlated with QoE. We propose to constrain our reward with service level agreements in terms of flow delivery delay and packet dropping for video traffic. The idea is to learn the policies and adjust them to maximize throughput and minimize flow delivery delays and packet dropping. Our goal is to design and evaluate centralized, semi-distributed, and distributed algorithms based on actor-critic and multi-agent DRL architectures.

Our QoS-aware load balancing approach uses a DRL algorithm with a constrained policy optimization that learns the reward parameters to meet QoS requirements with the main objective of controlling load balancing weights over a set of pre-computed paths for the multiple access devices. We resort to an actor-critic approach that uses centralized learning and distributed execution and that relies on a Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [3] philosophy to realize robust and improved load balancing performance. More specifically, we expand actor-critic schemes to address this multi-objective optimization problem to meet quality targets and avoid SLA violations. We actually explore several possible centralized, semi-distributed, and fully distributed execution schemes for smart load balancing.

In the literature, most centralized methods control load balancing weights to minimize a routing cost or the Maximum Link Utilization (MLU) (e.g., RILNET [4]). These methods do not include SLA requirements nor constraints in the policies optimization nor any addition of several QoS factors. Besides adding these factors and constraints, we also explore and show the relevance of adopting distributed approaches. In order to take these SLA obligations, we embed in the reward function penalties when traffic flow delivery delays and packet dropping targets are violated or are not met to provide performance guarantees. We devise several reward functions and assess

their performance by comparing them to the popular and practical ECMP algorithm and a fully centralized actor-critic approach acting as a benchmark. We show that introducing these constraints and adopting semi and fully distributed approaches using only network-collected QoS metrics improves QoE compared to ECMP and achieves very close performance to centralized QoE-based reward methods.

For the simulation environment, NS-3 [5] simulates the network and the routing operations, while the DASH library [6] generates realistic streaming video traffic. Simulation results show that we outperform ECMP and RILNET solutions and match the performance of QoE-based reward approaches by using readily available network collected QoS metrics.

II. Related Work

We can separate the related work into three categories: (1) multi-agent RL algorithms, (2) legacy solutions for load balancing, and (3) RL algorithms application to load balancing.

A. Multi-Agent Reinforcement Learning

With the introduction of deep learning, there was a resurgence of Reinforcement Learning (RL) algorithms based on deep neural networks used as function estimators [7]. Deep Q-Network (DQN) has been improved with a lot of different techniques such as Deep Deterministic Policy Gradient (DDPG) [8], Twin-Delayed Deep Deterministic (TD3) [9] and Soft Actor-Critic (SAC) [10]. The DDPG algorithm allows continuous policy management and introduces the use of experience replay: past experiences are stored into a buffer in order to train again later on the same sample. If making an environment calls for too much computer power, or if no simulator is available to harvest samples in real-time, techniques based on buffers and offline learning become very efficient. TD3 uses multiple critic networks to minimize the estimation error and use delayed (target) networks (that are just the running exponential average of the network) to stabilize training.

MARL (Multi-Agent Reinforcement Learning) framework has been improved by introducing DIAL [11], a technique to pass messages through pre-established communication channels. In this way, local agents can acquire useful information from their neighbors, solving the partial observability problem. The authors also notice that passing the gradient of the messages, in addition to the message itself, improves the training speed. Both COMA [12] and MADDPG [3] use one or multiple central critics and local agents in order to solve the instability caused by the fact that MARL is not necessarily a Markovian problem anymore. It also allows centralized training and decentralized execution, corresponding to our setting.

B. Legacy solution for Load Balancing

ECMP (Equal-Cost Multi-Path) [2], an extension to OSPF, is the most commonly used technique to load-balance traffic. It decouples the problem of routing from the one of load-balancing: the routing protocol first provides a set of equal-cost paths and, then, ECMP equally splits flow aggregates leveraging on the output of a hash function executed over packet-header fields. Albeit simple to implement, it assumes regular topologies with a high number of equal-cost paths to a given destination and an equal amount of available capacity over them. In more general settings, uneven flow splitting may improve network utilization and the use of unequal cost multi-paths techniques can further increase performance. However, these two approaches solve a local problem and do not attempt to optimize QoE / QoS. Another limit for this algorithm arises if the packet length distribution is uneven: the algorithm tries to send the same number of flows on each path, but does take into account the volume of the flow at decision time.

C. Using RL for Load Balancing

Authors of [13] use centralized deep reinforcement learning to estimate new incoming traffic flows and accordingly adjust weights to choose the most appropriate path. In [14], they adopt centralized learning and distributed execution. They propose a multi-agent actor-critic reinforcement learning algorithm to improve load balancing with the aim of minimizing the maximum link utilization ratio, while we investigate other rewards and metrics. The work in [15] proposes an intelligent load balancing architecture that includes link load balancing, server load balancing, and traffic classification. Only the traffic classification is realized using machine learning, the selection of paths is heuristic based on avoiding most critical nodes for elephant flows in the selected paths.

A relevant and closer approach to our algorithms is RILNET (ReInforcement Learning NETworking) [4], a centralized solution for reinforcement-learning-based load balancing using DDPG [8] to minimize the MLU. RILNET adopts RL to control a network on based the learned experience. RILNET operates on aggregations of flows referred to as ”flowlets”, i.e., a flow set that includes all flows going from the same source edge switch to the same destination edge switch. It results from the paper that RILNET can balance traffic load much more effectively than ECMP and DRILL [16]. In this paper, we improve RILNET by enabling the actors (agents in the switches) to cooperate by exchanging information via the DIAL cooperation framework and a multi-agent-based DDPG (MADDPG) approach. RILNET combined with DIAL is used for performance assessment.

D. Our Contributions

We use the MARL framework for multiple reasons: first to explore the distribution of intelligence for smart load balancing using semi-distributed and distributed cooperation of multiple agents, and second to meet contracted SLAs in terms of requested quality metrics. Adopting centralized training and then using agents in a decentralized setting is appealing in our case. MARL can also solve issues linked to partially observable states and stabilize the training. Our environment is not a Markovian process, as each agent is seen as part of the environment from the point of view of other agents. The non-stationarity comes from the fact that if it changes its policy, the behavior of the environment will change as well. Besides, it

is not even trivially a Markov process because load balancing operations influence future states as long as flows still exist.

III. Problem Statement

Let's first introduce the problem we will solve in details:

- We consider a graph with N nodes and E edges.
- Dynamic scenario: we do not know in advance the incoming flows.
- A controller is supervising the network and is in charge of computation tasks (path-finding, training, collecting experience replay samples, updating agents' policy, etc.).
- For each pair of nodes (n_i, n_j), K distinct routes are already pre-computed by the controller.
- When a new flow enters the network, an agent at the entry node, associated with the flow, decides which of the K paths has to be taken.
- An agent located at a node has access to both local information and messages received from the neighbors to make a better decision (in semi and fully distributed scenarios).

The information available for training are mostly network-level KPIs (Key Performance Indicators) such as flow delivery delay, packet dropping, achieved throughput, link utilization, jitter, and video streaming metrics such as stalling and the normalized video bit rate whenever applicable. They are observed, collected, and processed for optimization purposes and for updating the objective functions, the losses, and the reward. The controller can also access ground truth QoE to assess the effectiveness of the proposed algorithms. With this setting, we then have a total of $N(N-1)$ agents, distributed among N nodes.

The main objective is to find the load balancing weights that determine the path (among candidate paths) for every incoming video flow. In this specific work, the ultimate goal is to optimize QoE by leveraging only on network-level KPIs and deriving the probabilities for path selection. For this reason, we identify the combination of network metrics necessary to design an efficient reward function that maximizes experienced QoE. We show that by selecting flow delivery delay and throughput measurements, we can improve the experienced QoE against ECMP and RILNET or against methods using purely QoE metrics in the reward function.

The computation of target split ratios for every source and destination pair is performed at each time new information is received. It takes as input the set of candidate paths and updated traffic information on these paths (i.e., throughput, number of active flows). Once new target split ratios are decided, they can be used to make routing decisions every time a new micro-flow arrives. The decision is made to move actual split ratios towards the targeted ones (outcome of the policies). In some cases, if advanced data plane mechanisms such as FlowLets [17] are used, micro-flows can also be re-routed during their lifetime to accelerate the convergence of split ratios towards their target.

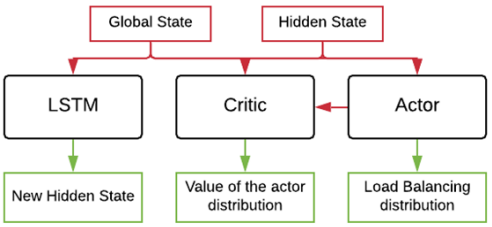


Fig. 1: Centralized Architecture.

IV. Reinforcement Learning Architectures

This section presents the actor-critic architecture used for the centralized, semi-distributed and distributed cases.

A. Centralized and Distributed Actor-critic Approaches

While we considered a centralized training, the architecture can be centralized (Fig. 1), semi-distributed (Fig. 2) or fully distributed (Fig. 3). If a centralized approach is adopted, both the actor and critic are global and are trained to derive policies for all origin nodes for all OD (origin-destination) pairs, namely a probability to select a candidate path per origin demand. In all cases, the execution of the policies is local to each origin node. In the semi-distributed case, the critic is centralized and actors are local and located at the origin node. In the fully distributed approach, there is one critic and one actor in each origin node. Cooperation is assumed in the distributed solutions through message exchanges among the actors (using DIAL as described later in the section).

1) Centralized model: 1 global critic, 1 global actor: The centralized architecture is not related to the MARL framework and only uses classic reinforcement learning algorithms. Compared to the original RILNET architecture, we have added a communicator based on an LSTM [18] neural network sending a local message for the next time iteration. This message passing is useful to address the non-Markovianity of the environment, because the state is artificially expanded to the previous states. RILNET, whose architecture is equivalent to the one presented in Fig. 1, is centralized as well, but it targets the MLU in the reward function.

2) Semi-distributed model with local reward: N^2 local agents, N^2 global critics: As shown in Fig. 2, this setting combines global critics $Qg_{i,j}$ with a set of agents $A_{i,j}$ and communicators $M_{i,j}$ distributed inside the N nodes.

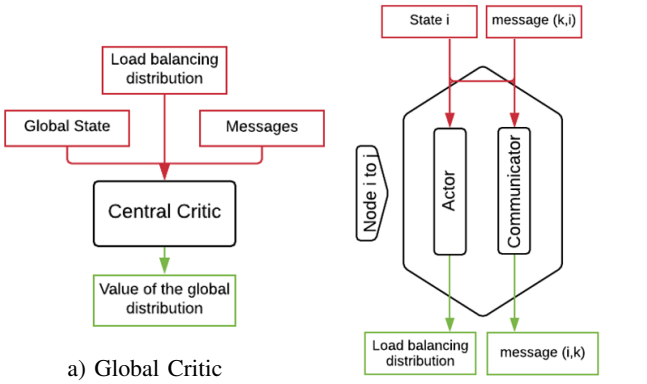
As the input of the critic is the global distribution of actions, the entire state of the network, and all the messages from communicators, it should be able to model with precision the future behavior (and so the future reward) of the network.

If the global reward r can be expressed as the average of local rewards $r_{i,j}$ ($r = \frac{1}{N^2}\sum_{i,j} r_{i,j}$), then we can express the discounted reward R as a combination of local rewards:

$$R = \mathbb{E}\left[\sum_{\mathcal{A}\sim(\pi)} \gamma^t r(t)\right] = \mathbb{E}\left[\sum_{\mathcal{A}\sim(\pi)} \frac{1}{N^2}\sum_{i,j}\gamma^t r_{i,j}(t)\right]$$
$$= \frac{1}{N^2}\sum_{i,j} r_{i,j}$$

As we optimize $A_{i,j}$ and $M_{i,j}$ based on the global critics $Qg_{i,j}$, we follow the same gradient descent as if we have just one global critic Qg. However, this method has different performance:

- Each agent will see its local contribution more easily because the reward is supposed to be local and more correlated to the agent's actions than the global reward. This way, it improves the convergence by a factor N as the critic does not need to differentiate the local reward from the random noise coming from the other N^2 rewards.
- The global reward must be a *fixed* linear combination of local rewards. The local rewards must be weighed in terms of the global traffic through the network, as the critic still needs to learn the behavior of the whole network.



a) Global Critic

b) Local Actor & Communicator

Fig. 2: Semi-Distributed Architecture.

Moreover, we do not really enter a competitive setting: as each agent is associated with a *global* critic, it is aware of the influence of the global situation on the local reward. This means that it will prevent an agent from changing its policy too much if that decreases a local reward located elsewhere in the network (because it will mechanically decrease the global reward). However, we can not achieve this behavior if we do not use global critics. Using local critics leads to a competitive setting, because each agent will selfishly improve their own discounted rewards without seeing the impact on global performance. The difference within the training loop is just the dimension of R and the output of the critics, whose shape will be (N, N).

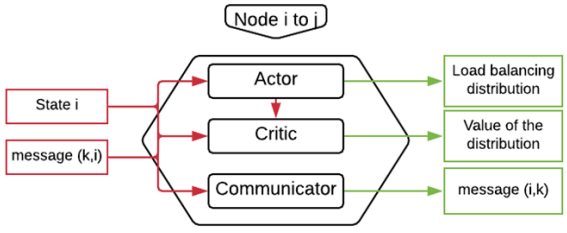


Fig. 3: Distributed Architecture.

3) Fully Distributed model: N^2 local agents, N^2 local critics: The distributed architecture is presented in Fig. 3. In this setting, for each directed pair of nodes (n_i, n_j) exchanging traffic there is a local actor $A_{i,j}$, a set of communicators $(M_{i,k})_{(i,k)\in E}$, each one sending messages to each neighbor node, and a local critic $(Ql_{i,j})$ that aims to estimate J, the global discounted reward.

The training loop is similar, but with N^2 local critics, each taking a different part of the state/action/messages as its inputs. The idea behind this architecture is to make the model more scalable. Indeed, the input of global critic is proportional to N, so it becomes harder to optimize it. With that architecture, it may also be possible to add node(s) to the network without losing all the training made on the critic(s).

However, the drawback of the distributed approach is that we have to make sure that the depth of communications and cooperation between immediate and distant neighbors is optimized and signaling overhead controlled and minimized. Further, if messages are only transmitted to neighboring agents, the messages can be diluted for far apart agents, despite they can have a significant effect on each other even when far away. This optimization is left for future work at this stage.

B. DIAL for Cooperation Between Agents

To implement the communication between agents (i.e., Communicators), we used DIAL an evolution compared with a previous method called RIAL. Reinforced Inter-Agent Learning (RIAL) is based on deep Q-learning with a recurrent network to handle partial observability. Differentiable Inter-Agent Learning (DIAL) relies on the insight that centralized learning affords more opportunities to improve learning than just parameter sharing. As DIAL passes gradients from agent to agent, it is an inherently deep learning approach. DIAL uses the insight that the combination of centralized learning and Q-networks makes it possible, not only to share parameters, but also to push gradients from one agent to another through the communication channel. DIAL works as follows: during centralized learning, communication actions are replaced with direct connections between the output of one agent's network and the input of another. While communication is restricted to discrete messages, during learning the agents are free to send real-value messages to each other. Since these messages function as any other network activation, gradients can be passed back along the channel, allowing end-to-end back-propagation across agents.

C. MARL notations

We consider a multi-agent framework with:

- Cooperative capabilities: the goal of each agent is to maximize the same expected global discounted reward.
- Partially observable state: each agent will only be aware of local information.
- Communication capabilities: we allow the agents to communicate via messages (if they are neighbors in the network graph in the semi and fully distributed cases).

Table I summarizes the notations used in the training description and the modeling framework.

V. Training algorithms

In this paper, all the proposed architectures use a similar training loop. The actor-critic architecture and models that we adopt are mainly inspired by Soft Actor Critic (SAC) algorithm [10], Twin Delayed Deep Deterministic (TD3) [9], DIAL [11], and MADDPG [3]:

1) We use two training critic networks Q_1 and Q_2, and use the value of the critics to compute the loss, in order to keep the $\min(Q_1, Q_2)$ to stabilize the training by reducing the risk of overestimating the value function.
2) We use our own value function and not the one of SAC, since we are addressing a non-stationary and dynamic environment.
3) We rely on a MADDPG architecture (with multiple critics) to train local agents.
4) Delayed networks $\bar{Q}_1$ and $\bar{Q}_2$ are used as sliding targets for the critic and $\bar{M}$ is used for the delayed communicator acting as target for the cooperation.
5) In this setting, as $M_{i,j}$ helps both the agent and the critic, it will be optimized with both losses. We can see the message $m_{i,j}$ as a function that should transmit as much information as possible to the local agents and critics.
6) Stochastic agents are used just as in SAC, the same loss is used. The output of the agent (i, j) is a tuple (μ, σ) with softmax$((\mu_k)_{k<K})$ as the average load balancing distribution for this agent. However, the random action is chosen by sampling $K-1$ random variable from $\mathcal{N}(0, \sigma)$: $(q_1, q_2, ..., q_{K-1})$, and setting $q_0 = 0$. The random distribution is then softmax$((\mu) + (q))$. q_0 is set to 0 to have a bijection between $(q_1, q_2, ..., q_{K-1})$ and the resulting load balancing distribution: softmax$((p)+(q))$. This enables computing the probability density for the SAC algorithm.

In Algorithm 1, the training process works with time episodes because our models involve messages that will influence future decisions. In order to simulate that in the training of our neural network, we then need to select $(\mathcal{S}(t), \mathcal{A}(t), R(t), \mathcal{S}(t+1))$ samples that were generated one after the other. We take batches of multiple time series to have uncorrelated samples. The critics are trained in order to predict the behavior and the future rewards of the network on average. The loss depends on the delayed network to stabilize the training, and to avoid that one bad sample makes the whole training fail. The communicators are trained in the same step. Then the actor training is realized just like in the SAC algorithm that trades off exploration (with the entropy $\mathcal{H}$) and maximizing the future rewards (depending on the critic).

VI. States, Actions and Rewards

A. States

The states are the amount of traffic (average throughput) and the number of active flows on each outgoing path (route). We use 1 agent for each OD flow. For the policy decisions, each agent is positioned at the source node and decides about load balancing weights for the K paths towards the destination.

TABLE I: Notations

$S_i(t)$	local state of the node i at time t
$\mathcal{S}(t)$	global state at t
$r_{(i,j)}(t)$	local reward at t of the agent focusing on routes from node i to j
$r(t)$	global reward at the time step t
γ	discount factor of the RL framework
τ	rate at which the delayed networks are updated
$m_{i,j}(t)$	message passed from n_i to n_j at t
$M_{i,j}$	communicator such that $m_{i,j} = M_{i,j}(S_i, (m_{i,k})_{(i,k)\in E})$
ϕ	network parameters of M
$A_{i,j}(t)$	load balancing distribution between n_i and n_j at t
$\pi_{i,j}$	actor such that $A_{i,j} = \pi_{i,j}(S_i, (m_{i,k})_{(i,k)\in E})$ (the policy)
θ	network parameters of π
$Ql_{i,j}$	local critic $Ql_{i,j}(S_i, A_{i,j}, (m_{i,k})_{(i,k)\in E})$ trained with $r_{(i,j)}$
Qg	global critic $Qg(\mathcal{S}, \mathcal{A}, m)$ trained with r
$Qg_{i,j}$	global critic $Qg_{i,j}(\mathcal{S}, \mathcal{A}, m)$ trained with $r_{(i,j)}$
$\bar{Q}l_{i,j}$	delayed local critic
$\bar{Q}g$	delayed global critic
$\bar{\omega}$	network parameters of the delayed critic

B. Actions

The actions in our case are the load balancing weights for each OD flow. Due to the discrete nature of RL algorithms and message transmission, we choose to discretize the decision-making process for the agents: at each time step/period P a new policy will be selected by the actor and will be followed during the entire time step P: each new flow arriving between the current time t and $t+P$ will be submitted to the same decision process (i.e., will follow the same policy).

We explore as mentioned the use of QoS metrics collected in the network as an alternative to QoE-based DRL methods. The latter assume that access to clients and applications on the end points or terminals is readily available and can be collected to monitor the evolution of the reward at each iteration. In order to verify, confirm and demonstrate that it is feasible to rely exclusively on network QoS metrics, the best evaluation and test is to conduct the study for video flows for which QoE is a relevant performance metric.

C. Rewards

Table II lists the reward expressions that have been considered in the evaluation and used for comparisons in the performance evaluation. We used several algorithms as benchmark. First, we use a centralized actor-critic reinforcement learning approach that uses the QoE as the reward. This algorithm is referred to as Cen-QoE, i.e., QoE-based centralized algorithm, where both the actor and critic are central. The second is RILNET, since it provides an alternative to the QoE-centric DRL approach by using the MLU in the reward and aims

Algorithm 1: Generic training loop used for all models

Draw from the experience replay buffer M batches of time series of lengths $L+1$:
$((\mathcal{S}(t))_{t_k \leq t \leq t_k+L})_{k<M}$;
$((\mathcal{A}(t))_{t_k \leq t \leq t_k+L})_{k<M}$;
$((R_{i,j}(t))_{t_k \leq t \leq t_k+L})_{k<M}$;

Initialize $m_{i,j}(t_k) = 0 \ \forall k < M$;
// Training the critic networks:
for $l \in [0, L]$ **do**
  Compute $(\mathcal{A}(t_k + l))_{k<M}$;
  Compute $(m_{i,j}(t_k + l + 1))_{k<M}$;
end

$QX_1 = \bar{Q}_1(\mathcal{S}(t+1), \mathcal{A}(t+1), m(t+1))$;
$QX_2 = \bar{Q}_2(\mathcal{S}(t+1)), \mathcal{A}(t+1), m(t+1))$;
update $= R + \gamma \min(QX_1, QX_2)$;
$QX_{1,old} = Q_1(\mathcal{S}(t), \mathcal{A}(t), m(t+1))$;
$QX_{2,old} = Q_2(\mathcal{S}(t), \mathcal{A}(t), m(t+1))$;
$L_1 = \mathbb{E}[(update - QX_{1,old})^2]$;
$L_2 = \mathbb{E}[(update - QX_{2,old})^2]$;

optimize ω_1, ω_2 following $\nabla_{\omega_1} L_1$ and $\nabla_{\omega_2} L_2$;
optimize ϕ following $\nabla_\phi L_1 + \nabla_\phi L_2$;
// Update target critics:
$\bar{\omega}_1 = (1-\tau)\bar{\omega}_1 + \tau\omega_1$;
$\bar{\omega}_2 = (1-\tau)\bar{\omega}_2 + \tau\omega_2$;
$\bar{\phi} = (1-\tau)\bar{\phi} + \tau\phi$;

// Update target actors:
for $l \in [0, L]$ **do**
  $\mu, \sigma = \pi(\mathcal{S}(t), m(t))$;
  sample q from $\mathcal{N}(0,1) * \sigma$;
  $A = softmax(\mu + q)$;
  Compute $(m_{i,j}(t_k + l + 1))_{k<M}$;
end
$QX_1 = \bar{Q}_1(\mathcal{S}(t), \mathcal{A}(t))$;
$QX_2 = \bar{Q}_2(\mathcal{S}(t), \mathcal{A}(t))$;
$L_1 = -\mathbb{E}[\min(QX_1, QX_2)] + \alpha\mathbb{E}[\log \text{ prob}(A\|\mu)]$;
$L_2 = \mathbb{E}[\mu\nabla_\mu L_1] + \mathbb{E}[\sigma\nabla_\sigma L_1] + \alpha\mathbb{E}[\log \text{ prob}(A\|\mu)]$;
optimize θ following $\nabla_\theta L_2$;

TABLE II: Reward expressions

Unconstrained rewards	
Algorithm	**Reward expression**
Cen-QoE	$\mathcal{R} = QoE$, ≡ Normalized average video bitrate
RILNET	$\mathcal{R} = -MLU$; minimize MLU
ECMP	$p_1 = p_2 = ... = p_k$; equal weight candidate paths
Cen-Th	$\mathcal{R} = Throughput$
Proposed constrained rewards	
Algorithm	**Reward expression**
Cen/Semi/Dis-QoE+D	$\mathcal{R} = QoE - \lambda e^{(-\lvert Reference_{Delay} - Current_{Delay} \rvert)}$
Cen/Semi/Dis-Th+D	$\mathcal{R} = Throughput - \lambda e^{(-\lvert Reference_{Delay} - Current_{Delay} \rvert)}$

Benchmark rewards Proposed new rewards

at minimizing link utilization as a kernel metric. The third is ECMP that splits flows equally across candidate paths and has been shown to perform fairly well if not faced with very bursty traffic and elephant flows. The last one is Cen-Th, corresponding to Central-based algorithm using the Throughput as the exclusive term in the reward expression.

As shown in Table II, we introduce constrained reward expressions where we penalize decisions if the experienced flow delivery delay violates a tolerable delay bound, called the *reference delay*. Recall that our goal is to resort exclusively on network KPIs to design smart load balancing algorithms that can intelligently distribute traffic to optimize QoE. The constrained rewards use a parameter λ to penalize the algorithms if flow delivery delays and packet dropping violate target SLAs. Ideally, Lagrangian optimization should be used and integrated in the model itself to ensure optimization of policies at every step in the learning and testing phases. While algorithms like RCPO [19] can be implemented to find optimal values of λ, in our case to ease the implementation, we performed an iterative search. The constrained based rewards are referred to as Cen/Semi/Dis-QoE+D and Cen/Semi/Dis-Th+D for the "QoE and Delay penalty" and "the throughput and Delay penalty" based approaches respectively (where "Cen" stands for centralized, "Semi" for semi-distributed, "Dis" for distributed, "Th" for throughput, and "D" for delay).

VII. Performance Evaluation

This section presents the evaluation methodology and the simulation results.

A. Simulation Environment, Training, and Testing

The topology used for the performance evaluation is the Abilene [20] topology (see Fig. 4) and for this set of results the clients generate video flows, generated using the LibDASH library [6] in the NS-3 simulator [5] coupled with a PyTorch environment for the machine learning framework where an MADDPG and RL actor-critic based approaches have been implemented and used to conduct the evaluation.

Fig. 4: Abilene Topology.

TABLE III: Definitions and metrics

Metrics and performance indicators		
Metric	**Definition**	**Comments**
QoE	A normalized index of the average video bitrate of DASH representations	Depends on screen resolution adaptation by DASH
Stalling	Average duration of stalls in the image collected from DASH and client	To minimize
Flow delivery delay	Average flow delay to destination	To minimize
Packet dropping	Average proportion of flow packets dropped	To minimize
Throughput	Proportion of flows that arrive at destination with SLAs met	To maximize
MLU	Maximum Link Utilization amount of consumed bandwidth	To minimize
Jitter	Phase drift in flow frames reception at destination	Should be kept as close to zero as possible

Our simulation platform is based on the Adaptive Multimedia Streaming Simulator Framework (AMust) [21] in NS-3, which implements an HTTP client and server for LibDASH, one of the reference software of ISO/IEC MPEG-DASH standard. As streaming content, we have chosen a representative open movie commonly used for testing video codecs and streaming protocols: Big Buck Bunny (BBB). More specifically, the following simulation scenario and machine learning settings have been used for the reported results:

- **Topology**: Abilene (see Fig. 4).
- **Traffic type**: video streams use DASH [6] that adjusts video encoding according to perceived bearer conditions sensed at the application level in the clients.
- **The selected neural network configuration**: it consists of 2 hidden layers with 64 neurons, a batch size for experience replay sampling set to 32 (L in Algorithm 1) for both distributed and centralized models, even if the centralized model could benefit from more powerful resources, i.e., large network size.
- **Test duration**: 300 time units (seconds).
- **Used training duration**: spanning from 30 000 time units to 300 000 time units.
- **Low vs. High load scenarios**: for the low load scenario, all link capacities are randomly generated between 5 Mbps and 20 Mbps. For the high load scenario, bandwidth values are within 1 Mbps and 4 Mbps.

We generated 3 paths for each source/destination pair, using a k-maximally disjoint shortest path algorithm. The traffic is randomly generated at the beginning of the experiment following these characteristics:

- The duration of flows is 40s and 10 Origin-Destination (OD) pairs are active in each simulation.
- The inter-arrival time of flows follows a Poisson distribution with heterogeneous parameters for each origin-destination pair. The global average arrival rate is 1.1 new streams per second.

The metrics assessing performance are listed in Table III.

1) QoE metrics: We measure multiple QoE factors for video streaming that can accurately highlight the variation in the QoE of videos. In particular, we measure the following QoE metrics:

- **Average video bitrate** of the downloaded chunks.
- **Average video quality**: average on all downloaded chunks of a normalized quality index indicating to which representation they belong. It evolves between 0 and 1 for r^{min} (min video bitrate of DASH representations) and r^{max} (max representations bitrate), respectively (see [22] for more details).
- **Re-buffering ratio**: fraction of the freezing (or stalling) time over the duration of the video session.

2) Collection of statistics: We extensively measure the following metrics for the performance evaluation, and in order to avoid any misunderstanding, these performance metrics are not all used in the reward expressions (the used criteria and reward expressions are specified for each algorithm):

TABLE IV: Algorithm parameters

Actor learning rate	10^{-3}
Critic learning rate	10^{-2}
Batch size	32
τ delayed network update rate	0.1
γ discount factor	0.9
Number of hidden layers	2
Dimension of hidden layers	64
Temperature parameter used for entropy based exploration	10^{-3}
Size of the vector containing messages between neighboring agents	20
Policy refreshing period (time period)	$10s$

- For each route: average delay, average jitter, loss ratio, average throughput, and the number of flows.
- For each flow: QoE statistics (within DASH) such as stalling time of the video and experienced jitter for each segment (10s). Vice versa, the average video bitrate experienced in the network is used in the reward expression for the QoE-based algorithms, since this metric can be directly linked to the experienced QoE.
- For the whole network: the average delay, the average jitter, the loss ratio, and the average throughput.

3) Simulation conditions and settings: The simulation conditions and hyper-parameters used for the performance evaluation of all the algorithms and all network load scenarios are specified in Table IV. The simulation scenario served for the comparison with the central algorithms Cen-QoE, RILNET, and ECMP to benchmark the state of the art solutions.

B. Evaluation Results

We present results of centralized, semi-distributed, and fully distributed algorithms in Table V and VI depicting the medians of average delay, throughput, QoE, video stalling, MLU, and jitter over a test episode of 300 time units (seconds), with a policy refreshing period of 10 seconds (i.e., update frequency of load balancing weights).

1) Actor/Critic loss: Fig. 5 depicts the loss functions for the DDPG critic and actor of the algorithms. It indicates that algorithms converge as losses are stable. Rewards, not shown here for lack of space, follow similar patterns. Optimization and stabilization are realized using Stochastic Gradient Descent (SGD optimizer) instead of the usual Adam optimizer.

2) Performance results: In the rest of the section, we present the performance evaluation of the semi-distributed and fully distributed algorithms using either global critics and multiple local actors (one per OD flow at the origin node) or using multiple local critics and actors (one local critic actor pair per origin node) to realize smart load balancing in networks using reinforcement learning. Results are reported for low and high load conditions.

a) Low load condition scenario:

The results, as depicted in Table V, show that the semi-distributed algorithms achieve the best overall trade-offs across

TABLE V: Low load scenario.

Metric \ Algorithm	ECMP [2]	RILNET [4]	Cen-QoE	Cen-QoE+D	Cen-Th	Cen-Th+D	Semi-QoE	Semi-QoE+D	Semi-Th	Semi-Th+D	Dis-QoE	Dis-QoE+D	Dis-Th	Dis-Th+D
Delay (s) Target delay $\leq$ 0.5s	0.352	0.344	0.315	0.285	0.298	0.291	0.287	0.274	0.286	0.277	0.301	0.282	0.291	0.284
Throughput (bps)	5 032 320	5 731 790	6 348 990	6 742 770	6 821 850	6 970 090	7 240 850	7 432 250	7 159 160	7 069 930	7 094 240	6 424 820	6 510 830	6 421 260
QoE	0.4021	0.4428	0.4564	0.4662	0.4617	0.4644	0.4834	0.4822	0.4542	0.4667	0.4559	0.4633	0.4705	0.4753
MLU (%)	13.08	11.72	13.59	13.51	13.91	12.14	14.03	13.81	14.01	13.57	13.88	12.21	13.39	12.13
Video Stalling (ms)	701.59	578.07	567.05	561.96	581.52	515.81	513.85	525.44	564.51	521.94	559.75	562.18	535.21	570.42
Average Jitter (s)	0.0678	0.0503	0.0132	0.0032	0.0296	0.0195	0.0038	0.0017	0.0067	0.0077	0.0107	0.0221	0.0051	0.0089

TABLE VI: High load scenario.

Metric \ Algorithm	ECMP [2]	RILNET [4]	Cen-QoE+D	Cen-Th+D	Semi-QoE+D	Semi-Th+D	Dis-QoE+D	Dis-Th+D
Delay (s) with Target delay $\leq$ 0.5s	0.5847	0.5613	0.5161	0.5073	0.4977	0.5041	0.5052	0.4939
Throughput (bps)	1 462 885	1 510 425	1 705 395	1 718 895	1 697 205	1 705 875	1 670 470	1 674 010
QoE	0.2775	0.2996	0.3333	0.3277	0.3231	0.3229	0.3128	0.3102
MLU (%)	15.28	14.91	14.89	15.61	15.17	14.09	15.11	14.42
Video Stalling (ms)	1568.22	1304.54	952.61	996.6	987.99	966.36	984.39	1002.35
Average Jitter (s)	0.1501	0.1464	0.0849	0.0966	0.1005	0.1065	0.1145	0.1084

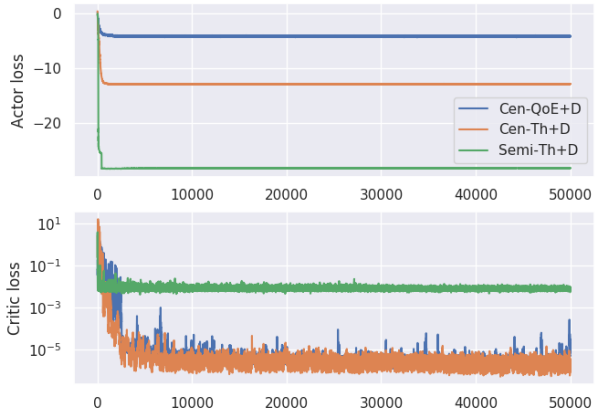


Fig. 5: Critic and Actor loss as a function of training steps.

all performance metrics when jointly analyzed. ECMP and RILNET are outperformed by all the approaches implementing reward constrained policy optimization, as they penalize the rewards as long as target delays are not met. Further, the reported results show that the use of a joint throughput-delay performance criterion and metric allows achieving similar and occasionally better performance across all evaluated metrics. Namely, in end-to-end packet delivery delay, realized throughput, achieved QoE, experienced jitter, and stalling on video flows. As far as MLU optimization performance is concerned, RILNET, that focuses on MLU minimization, outperforms typically all other algorithms on this metric but at the expense of poor performance on all other key performance indicators. Further, the throughput-delay based reward expressions can make better use of the links, they also minimize the MLU but operate at higher link utilization and still meet target delays and improve QoE. They realize better MLU, QoE trade-offs.

In summary, the algorithms, that rely on the tuple (throughput, delay) network metrics, are sufficient to realize the desired performance improvements and show that we do not need to resort to application-layer metrics that measure or assess the QoE (or a related metric such as average bitrate for video flows) to meet SLA and performance requirements. The goals can be met by relying exclusively on the "throughput and the delivery delay" network metrics measured at the end points. This will considerably simplify the learning and exploitation outcomes of reinforcement learning-based intelligence.

b) High load condition scenario:
Under high load, where links are typically congested to some extent, the semi-distributed algorithms continue to perform reasonably well, providing good results for all the metrics. The fully distributed algorithms are more challenged by these high load conditions, as expected. This comparable performance of all algorithms at high load is expected and known, and presents nothing unusual in terms of results. More importantly, the semi-distributed algorithms, using the tuple (delivery delay, throughput) in the reward constrained policy optimization expression, confirm their relevance and adequacy as the best trade-off across all metrics. The fully distributed case performance is not surprising either, since the communication and cooperation across agents are basic in DIAL.

VIII. Conclusions and Perspectives

We have shown in this paper that centralized and multi-agent actor-critic based reinforcement learning algorithms for QoS-aware load balancing are capable of improving performance compared to more traditional load balancing techniques and previous machine learning based methods. The use of semi-distributed multi-agent approaches, with a central critic and local actors in the source nodes of end-to-end connections, is recommended. The use of fully distributed approaches, where a critic and an actor are used at each source node, shall not be dismissed at all. Indeed, the performance of the fully distributed solution is conclusive even with a rather basic cooperation and communications across actors or agents. The use of more efficient cooperation systems, such as ATOC [23] or QMIX [24], may further improve performance, reaching that of the centralized and semi-distributed solutions.

Note also that a lesson, learned from these investigations, is to rely on network metrics to express rewards and objective functions. Further, the use of simply delay and throughput metrics allows meeting the required QoS performance and equals that of algorithms using QoE metrics collected at clients and end devices, which is quite often practically unfeasible. As long as both delay and throughput are used and actions are penalized for excessive delivery delay or policy optimization is constrained by key quality of service metrics such as delay and packet dropping, the results will be quite close and competitive to any other solution relying on less accessible metrics.

REFERENCES

[1] Ning Wang, Kin Hon Ho, George Pavlou, and Michael Howarth. An overview of routing optimization for internet traffic engineering. *IEEE Communications Surveys & Tutorials*, 10(1):36–56, 2008.

[2] Christian Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. Technical report, RFC 2992, November, 2000.

[3] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems, 4-9 December 2017, Long Beach, CA, USA*, pages 6379–6390, 2017.

[4] Qinliang Lin, Zhibo Gong, Qiaoling Wang, and Jinlong Li. RILNET: A Reinforcement Learning Based Load Balancing Approach for Datacenter Networks. In *Machine Learning for Networking*, 2018.

[5] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[6] Christian Kreuzberger, Daniel Posch, and Hermann Hellwagner. A scalable video coding dataset and toolchain for dynamic adaptive streaming over http. In *Proceedings of the 6th ACM Multimedia Systems Conference*, page 213–218, 2015.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[8] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[9] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. ICML*, 2018.

[11] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems, December 5-10, 2016, Barcelona, Spain*, pages 2137–2145, 2016.

[12] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual Multi-Agent Policy Gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2974–2982, 2018.

[13] Abhishek Jha, Kislay Kunal Singh, K. Vimala Devi, and V. Manjula. Reinforcement learning based weighted multipath routing for datacenter networks. *Materials Today: Proceedings*, 2021.

[14] Tianle Mai, Haipeng Yao, Zehui Xiong, Song Guo, and Dusit Tao Niyato. Multi-agent actor-critic reinforcement learning based in-network load balance. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6, 2020.

[15] C Fancy and M Pushpalatha. Proactive load balancing strategy towards intelligence-enabled software-defined network. *Arabian Journal for Science and Engineering*, pages 1–8, 2021.

[16] Soudeh Ghorbani, Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Micro Load Balancing in Data Centers with DRILL. In *Proc. ACM HotNets*, 2015.

[17] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *Proc. USENIX NSDI*, 2017.

[18] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural computation*, 12(10):2451–2471, 2000.

[19] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward Constrained Policy Optimization. *CoRR*, abs/1805.11074, 2018.

[20] Anukool Lakhina, Konstantina Papagiannaki, Mark Crovella, Christophe Diot, Eric D Kolaczyk, and Nina Taft. Structural analysis of network traffic flows. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 61–72, 2004.

[21] C Kreuzberger, D Posch, and H Hellwagner. Amust framework-adaptive multimedia streaming simulation framework for ns-3 and ndnsim, 2016.

[22] Giacomo Calvigioni, Ramon Aparicio-Pardo, Lucile Sassatelli, Jeremie Leguay, Paolo Medagliani, and Stefano Paris. Quality of experience-based routing of video traffic for overlay and ISP networks. In *Proc. of IEEE INFOCOM*, 2018.

[23] Jiechuan Jiang and Zongqing Lu. Learning Attentional Communication for Multi-Agent Cooperation. In *Advances in Neural Information Processing Systems, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 7265–7275, 2018.

[24] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *Proc. of ICML*, 2018.