IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 11, 2022, accepted May 30, 2022, date of publication June 10, 2022, date of current version June 17, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3182009

Deep Reinforcement Learning Based Routing in IP Media Broadcast Networks: Feasibility and Performance

PEDRO AMARAL[1,2] **(Member, IEEE), AND DIOGO SIMÕES**[1,3]

[1]Departamento de Engenharia Electrotécnica e de Computadores, Faculdade de Ciências e Tecnologia (FCT), Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
[2]Instituto de Telecomunicações, 1049-001 Lisboa, Portugal
[3]Skyline Communications, 8870 Izegem, Belgium

Corresponding author: Pedro Amaral (pfa@fct.unl.pt)

This work was supported by Fundação para a Ciência e Tecnologia/Ministério da Ciência Tecnologia e Ensino Superior (FCT/MCTES) through National Funds and When Applicable Co-Funded EU Funds under Project UIDB/50008/2020.

ABSTRACT The media broadcast industry has evolved from Serial Digital Interface (SDI) based infrastructures to IP networks. While IP based video broadcast is well established in the data plane, the use of IP networks to transport media flows still poses challenges in terms of resource management and orchestration. Software Defined Networking (SDN) based orchestration architectures have emerged in the industry that use SDN to route the media flows of a broadcast service across the provider IP network. Several approaches to multimedia flow routing in IP based SDN networks have been proposed in the context of streaming applications over the Internet. These range from model based linear optimization solutions that have high complexity to simple shortest path based routing with either Static Link Costs (SLC) or Dynamic Link Costs (DLC). More recently model-free optimization methods such as Deep Reinforcement Learning (DRL) have been proposed for routing and Traffic Engineering (TE) of multimedia flows in SDN networks. The media broadcast scenario however has specific requirements, with services like Master Control Room (MCR) operation and live broadcasting of events, and it has been rarely addressed in the literature. In this work we propose a DRL based routing method for this scenario and compare it to SLC and DLC algorithms based on Dijkstra shortest paths. This is, to our knowledge, the first work to follow this approach in the context of media broadcast services in IP infrastructures. The algorithm is designed considering the specifications and capabilities of one of the leading SDN orchestrators in the market and considers the more common Service Level Agreement (SLA) requirements in the industry. Three different DRL algorithms are implemented and compared and we evaluate them using a real service provider network topology. The results indicate that DRL based routing is applicable in real production scenarios and that it achieves considerable performance gains when compared to the SLC and DLC shortest path algorithms commonly used today.

INDEX TERMS Media broadcast networks, artificial intelligence, deep reinforcement learning, network orchestration, routing, software defined networks.

I. INTRODUCTION

The media broadcast industry has evolved in recent years, from SDI [1] based infrastructures to all IP broadcast networks with aspects like the transport, synchronization, and description of separated video, audio and ancillary data streams over the IP data plane being well defined in the SMPTE 2110 set of standards [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Jian Song.

A. MOTIVATION

When it comes to the orchestration and management of the underlying infrastructure there are still several challenges to overcome. These come mainly from the use of a non-linear IP network to transport linear media flows that need to be switched quickly and frame-accurately. In some scenarios, like MCR operation, feeds can be booked ahead of time and neither are time critical nor have to be frame accurate, leaving enough time for an orchestrator to make resource management decisions in advance to guarantee capacity and

availability. However, when it comes to live broadcasting, typical vendor-specific broadcast controllers do not handle capacity and underlying network topology constraints, leaving no option but to resort to over provisioning solutions that lead to network under-utilization. This leads to situations were either there is a high cost due to over provisioning or the transport networks might introduce frame drops, jitter and latency due to network congestion caused by bottlenecks. An orchestration layer is needed that is able to manage media broadcast networks considering the underlying infrastructure capabilities, capacities and availability. To achieve this, network capacity must be managed in a centralized way, managing resources that can include legacy hardware, software or even virtual appliances.

Some efforts have been made to propose orchestration architectures for all IP broadcast networks like the works described in [3] and [4], that define use cases and service definitions as well as general architectural functionality. SDN based solutions to provide this orchestration layer have emerged in the market (e.g. the offer by Skyline Communications [5]), in these solutions media flows need to be routed across a media broadcast network to one or multiple destinations. In the SDI world, this was done by setting a cross-point in the SDI router [1], in the all IP scenario an SDN controller routes the media flow that originates from a source to a destination. The problem is how to perform this routing in a near-optimal way.

B. AVAILABLE APPROACHES

Several approaches have been proposed over the years for SDN based multimedia flow routing in the context of streaming applications over the Internet [6]. These can be broadly divided into categories according to the followed approach. A first category are solutions based in linear optimization formulations that are usually NP-hard [7]. This type of solutions usually establish a static model of the network based in a complex analysis of its traffic, which many times leads to an intractable problem that needs to be approximated to be solvable using heuristic approximations. A second type of approach, that is still the most broadly used in real deployments due to its robustness and low complexity, is shortest path routing based on link cost. This approach can be further divided into SLC and DLC routing algorithms with the latter being more difficult to implement due to the need to obtain accurate network state information [8]. A recently explored alternative is the use of model free optimization methods such as DRL to provide routing algorithms to solve TE problems in SDNs. Several works have studied the use of DRL for routing and TE in the context of SDN with promising results. However, there are still challenges: state space can explode with network size [9]; there is still a gap in performance when compared with optimization based heuristics [10]; it can be difficult to generalize training to unseen scenarios [11].

C. CONTRIBUTIONS

Despite this body of research available for the context of Internet streaming applications, the applicability to the all-IP broadcast media network scenario, has not, to our knowledge, been addressed in the literature. Also, most of the existing proposals are evaluated either using numerical or event-driven simulations and either use simple publicly available IP backbone network scenarios or synthetic topologies that do not accurately represent all-IP Broadcast Networks.

In this work we study the use of DLC shortest path routing and DRL based routing for media flow routing in a real production media broadcast network scenario assuming the orchestration capabilities offered by a real commercial orchestrator [5]. We show that DRL based media flow routing is applicable in a real production all-IP media broadcast network scenario and compare its efficiency with both SLC and DLC shortest path routing algorithms.

II. RELATED WORK

Approaches for multimedia flow routing in SDN networks have been proposed mainly in the context of streaming applications over the Internet [6]. The routing of media flows in the context of all-IP broadcast networks is a TE problem were the main objective is to avoid Quality of Service (QoS) and Quality of Experience (QoE) degradation due to frame drops, jitter and high latency caused by link congestion. This can be achieved through routing optimization in order to avoid routing traffic through congested links. Routing optimization has been traditionally approached in terms of a shortest (or k shortest) path algorithm, were link weights can be static (SLC algorithms) or dynamic (DLC algorithms), the routing optimization is then done by optimizing the link costs in order to optimize a metric or a set of metrics.

A. OPTIMIZATION BASED SOLUTIONS

Optimizing the link costs is usually an NP-complete problem [12]. In [13], a survey of QoS routing algorithms in SDN networks is presented with several examples of global offline QoS routing algorithms that involve complex optimization problems, such as integer or mixed integer linear programs [14], [15]. These are complex and hard to implement due to the difficulty in modeling modern networks and their difficulty in dealing with network dynamism. In this work we did not consider this approach due to the dynamism of the scenario, especially in live broadcasting, and also because contrary to the scenarios in [14], [15], multimedia transmission in broadcast media networks is more bandwidth hungry than delay sensitive, and avoiding link congestion is more important than assuring strict delay bounds.

B. DLC APPROACHES

SDN can allow the use of network state information to enhance traditional shortest path routing protocols (based in some variation of the Dijkstra's algorithm), allowing the use of dynamic routing costs according to current network

state [16]. In [8], a comparison of SLC and DLC heuristic routing algorithms in the context of SDN is presented. It is shown that, as expected, DLC algorithms have a better performance than SLC algorithms in terms of accepted flows (i.e. flows that are placed in the network) and total traffic throughput. However, the cost of obtaining accurate network state information in SDN environments is reflected in terms of control overhead. DLC performance is also adversely impacted by network state information inaccuracies [8], resulting in a trade-off between the amount of times the network is probed and the overall efficiency of the protocol.

C. DRL BASED APPROACHES

In recent years there has been an interest in taking advantage of the programmable nature and the ability to gather network information of SDN to use model-free Machine Learning (ML) methods to optimize routing [17]. From these approaches DRL [18] based routing is particularly interesting due to its independence of existing labeled data sets, that are very hard to obtain for routing, and its adequacy for dynamic optimization problems.

In [17], a list of DRL based routing approaches using SDN is reviewed for the general problem of routing optimization. Approaches vary in the used DRL algorithm from value function based methods such as Deep Q-Networks [19], to combined action and policy value algorithms, known as actor-critic methods, such as Deep Deterministic Policy Gradient (DDPG) [20]. From this body of work, we emphasize a set of approaches that tackle a similar problem to ours, although not for the broadcast network scenario [21]–[25].

In [21], a DDPG algorithm is used to optimize the cumulative QoE while routing multimedia flows across an SDN network. The reward is evaluated by a Mean Opinion Score (MOS) of the clients reported QoE. This is difficult to obtain in real time, and the authors use a Deep Neural Network (DNN) trained to map flow statistics to reported QoE using past experiences to solve the problem. The state reflects the bandwidth, delay, jitter and packet loss of the currently transported flows and the actions consist in choosing a path and the respective bandwidth allocation for a flow. Simulations are performed using an event-driven simulator to mimic an SDN network and use topologies from a public available repository whose realism is hard to verify. A DDPG based agent with a very similar model to the one in [21] is proposed in [23], but in this case the problem is formulated as a weight optimization problem for shortest path calculation. The DRL agent provides an action that consists in the set of link weights attributed to the links of the network and the paths are then calculated using this set of link weights. The reward is calculated according to the chosen optimization metric that can be delay or throughput and the traffic matrix is the input state for the agent. The proposal is evaluated via simulation in an event-driven simulator using an old topology (2011) from the IP backbone of an American Internet provider. Xu *et al.* [22] proposed an extension to DDPG, replacing the exploration based in random noise by an exploration based in a baseline method (e.g. shortest path routing) and using a prioritized experience replay instead of the simple uniform sampling of DDPG. The state input for the DRL algorithm is formed by a set of throughput and delay tuples from each communication session, the work is evaluated via simulation using two commonly used topologies, the National Science Foundation Network (NSFNET), the Advanced Research Projects Agency Network (ARPANET), as well as a randomly generated topology.

An issue with DRL approaches is the state and action space sizes that exponentially grow with the size of the network topology. In [9], [25], the authors follow a link pinning strategy to select only a set of critical links to be controlled in order to reduce the state and action space sizes. The link selection algorithm is performed offline and the DRL agent adjusts the link's weights online, thus optimizing the paths that are calculated. The used DRL algorithm is an actor-critic method where the state is formed by the critical links traffic distribution and the action space is the set of link weights for posterior path calculation. Once again, this work is evaluated through an event-driven simulator and uses generated topologies.

These approaches all have in common the formulation of the problem in a continuous control structure, since they either use link weight arrays as their action [9], [23], [25], a path bandwidth continuous value [21], or an array of path choice probabilities [22]. On the other hand, the authors in [24] propose a solution, called RL-Routing, which uses a discrete action space and targets throughput maximization and communication delay minimization. RL-Routing uses a Duelling Double Deep Q-Network (Duelling DDQN) DRL algorithm with an agent for each individual router. The state space is an array of link and router statistics, the action space is the set of available paths (pre-calculated k-shortest paths) and the reward is obtained by measuring the delay and throughput rate. In this case, the work was evaluated via network emulation using a virtual software network built in Mininet [26] and the NSFNET and ARPANET topologies were used for the evaluation scenarios.

III. ROUTING VIDEO STREAMS ACROSS A PRODUCTION ALL-IP BROADCAST NETWORK

To our knowledge, at the time of writing, there are no DRL based proposals in the literature specifically addressing the routing of video streams in the broadcasting scenario. The problem is similar to the routing and TE problems discussed in the related work section, but there are requirements and limitations that need to be considered when dealing with a real production broadcast network scenario.

In this work we consider a scenario as close as possible to a real production deployment. We assume the use of an orchestrator with the characteristics of the Network Management System (NMS) described in [5] and we evaluate the proposed solution in a topology that is a real all-IP broadcast service provider infrastructure. The considered NMS is one of the leading commercial offers in the market for SDN based orchestration of services in Broadcast networks.

It can act as an SDN controller and is capable of using several southbound protocols in networks with different levels of control-plane/data-plane separation. We also use a real service provider IP Broadcast network topology to test the developed algorithms.

In many production broadcast networks in use today a *control plane broker* SDN flavour is used, this means that there is still control plane operation on the hardware devices and the SDN controller acts like a mediator between applications and the network devices. The controller can retrieve information from the devices and manipulate their control planes (e.g. installing routes in the Forwarding Information Base) using protocols like SNMP, NETCONF, RESTCONF or dedicated APIs. In this scenario, depending on the used protocol, obtaining flow statistics can be a costly operation in terms of time overhead and the available information also varies. For this reason we only consider simple network metrics like total bandwidth capacity and current utilization of interfaces (and not individual flows) that can be obtained by even the simpler protocols such as SNMP. This invalidates any approach that needs individual flow statistics in their DRL model like the ones in [21], [22].

Approaches that use link weight arrays as their action [9], [23], [25] are also not suitable for our real deployment scenario, The reason is that the only possible implementation implies using one of the supported southbound APIs to set the new link weights and Dijkstra to calculate the new set of paths. This is computationally heavy when implemented in the considered NMS system [5], so we opted for solutions where the action space is a set of candidate paths available to forward the media flows [24]. The same reasoning applies to DLC shortest path algorithms that have similar implementation implications in terms of setting the new link weights.

Another issue to consider is the nature of the SLAs in the broadcast industry. In most cases, like MCR and live broadcast scenarios, the practice is to stipulate a determined and fixed flow bandwidth that clients expect to be available at all times regardless of the actual utilization. This means that the bandwidth allocation is user defined and cannot be part of the action of the agent, so approaches like the one described in [21] can not be applied.

IV. SYSTEM DESIGN

We consider the scenario of multimedia broadcast services transported over an IP infrastructure network. Additionally, it is assumed that the services are orchestrated by a controller and that this controller has the capabilities and limitations of the controller offered by Skyline Communications [5], so that the implementation is realistic for real production environment use.

Each service is served by a scheduled resource reservation that comprises three distinct time periods: *pre-roll*, *active* and *post-roll*. During the *pre-roll*, resources, such as SDI encoders, SDI decoders, switches, routers, or antennas, are configured for the specific type of requested service. The service requirements are defined in a service definition (see fig. 1) that includes at least one SDI encoder, where the multimedia source is originated and translated into IP, a transport network, a nested service definition that defines how the routing of video stream data is made, and one SDI decoder connected to the exit edge node of the transport network.

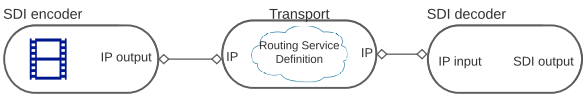


FIGURE 1. Transport service definition.

The service reaches its end when the active time has elapsed and enters the *post-roll* phase where the reserved resources are released.

The media stream routing algorithm runs on top of the transport block of Figure 1. It retrieves the interface metrics of the topology (total bandwidth capacity and current utilization) and uses that information to compute paths that are then installed in the transport devices. Finally, the algorithm decides, for each media stream, which of the available paths are used.

One SLC and two DLC routing algorithms were implemented to serve as baseline methods. These are then compared with the DRL based routing solution that is implemented with three different algorithms.

V. BASELINE ALGORITHMS

The first baseline protocol is a simple SLC solution that uses Dijkstra with fixed link costs that we call Minimum Hop Algorithm (MHA). The second baseline protocol is a DLC algorithm that uses updated information of the interfaces state to dynamically, and accordingly, update the costs of the links. This information is retrieved in 5 second intervals and new paths are calculated according to the current costs. Two different DLC models were implemented: dynamic shortest path (DSP) [8], which considers a link cost $C_{(u,v)}$ that is inverse to the currently available bandwidth $RBW_{(u,v)}$

$$DSP : C_{(u,v)} = \frac{1}{RBW_{(u,v)}}, \tag{1}$$

and Least Interference Optimization Algorithm (LIOA) [8] that adds a metric related to the quantity of flows carried by network links by multiplying the link cost equation of DSP by the number of flows being transported $I_{u,v}$ and raising it to a constant α

$$LIOA : C_{(u,v)} = \left(\frac{I_{(u,v)}}{RBW_{(u,v)}}\right)^{\alpha}. \tag{2}$$

The total bandwidth capacity and the current utilization are metrics that can be obtained directly from the physical interface by all of the southbound protocols supported by the considered controller [5] in our design. The number of transported flows must be calculated in the controller and is updated every time a flow is placed in a path, with the number of flows of an interface being incremented for all interfaces in the chosen path. When a flow stops, this value is decremented.

VI. DRL BASED ALGORITHMS

In a DRL model an *agent* interacts with an *environment*, that generates relevant data for the model's objective. The current state of the environment $S_t \in S$ reflects the environment in a given time t while S is the state space. The *agent* chooses an action $a_t \in A(S_t)$, where $A(S_t)$ is the action space in the current state S_t. The performed action impacts the environment that enters a new state S_{t+1} and produces a reward r_{t+1}. Such reward indicates how the action taken in a given time step and environment state contributes to the agent's objective.

DRL can be modeled has a Markov Decision Process (MDP) since it has no memory requirement, with every state having enough information on its own for the agent to choose the best action. The goal of an MDP is to find the optimal policy that leads to the maximization of the reward function. The policy function π is responsible for mapping states to either a deterministic action or a probability distribution over actions that returns the highest reward. An alternative way to choose the best actions is to calculate the value of a state. In this case, the value function $V_\pi(S_t)$ maps a state to its expected reward that is defined by the long-term reward average from being in a state or taking an action in that state, provided that a certain policy is consistently followed.

A. DEEP Q-NETWORKS (DQNS)

The *Q-function* is an *action-value* function used in a class of DRL algorithms called Deep Q-Network models (DQNs) [27]. This function is defined as $Q(s, a)$, and maps a state-action pair input to the value of action a at state s. The function is updated during training according to the following equation:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R(S_t, A_t) + \gamma maxQ(S_{t+1}, a') - Q(S_t, A_t)]. \quad (3)$$

The $\alpha[R(S_t, A_t) + \gamma maxQ(S_{t+1}, a') - Q(S_t, A_t)]$ term is the Temporal Difference (TD), where $R(S_t, A_t) + \gamma maxQ(S_{t+1}, a')$ is the predicted Q-value and $Q(S_t, A_t)$ the current Q-value. In this equation, γ is the discount factor and α is the learning rate. In other words, the updated Q-value is equal to the current predicted Q-value plus the amount of value expected in the future [27].

Q-values greatly depend on the observed rewards, this can lead to instability in Q-values since rewards are commonly unstable, divergent (i.e same state-action pairs resulting in different rewards) or sparse, depending on the application.

Experience replay is one of the techniques used to try and solve this problem. It uses a replay buffer to continuously store experiences containing the state, the action taken, the reward and the resulting state (s_t, a_t, r_t, s_{t+1}). A few of these experiences are then sampled from the buffer as input to the Q-learning algorithm. Then, actions are chosen according to a renewed policy generating new experiences, which are added to the buffer which starts to contain new and old experiences that can be sampled together stabilizing the algorithm. The sampling of experiences from the replay buffer can be prioritized according to low error values in past transitions or other criteria that makes them more valuable to the training process.

Another technique for training stabilization is the use of a target network, which is a copy of the main DQN that learns the Q-function. This copy $\hat{Q}$ has its parameters updated with a certain lag, which helps stabilizing the back-propagation training process. Additionally, by using this network to calculate the target Q-value to train the main DQN, the effect of recent updates is decreased.

B. DOUBLE DEEP Q-NETWORKS (DDQNS)

This concept is at the origin of the DDQN algorithm that tries to overcome the tendency of DQNs to over-estimate the value of some actions. This over-estimation neglects the possibility of other actions being more suitable in certain conditions [28]. DDQNs use two value functions, which are initialized as a copy of each other, to separately select and evaluate action values. One of these functions will select an action based on the updated weights (i.e. *argmax*) and the other will estimate the value of the current Q-function using lagged weights. The weights of the second Q-network are updated according to the weights stored in the target network $\hat{Q}$, while the target network's weights θ' are updated periodically according to the online weights θ as is done in traditional DQNs. The θ weights are optimized to minimize the loss given by equation 4.

$$[r_j + \gamma \hat{Q}(s_{t+1}, argmax_{a'} Q(s_{t+1}, a'; \theta); \theta') - Q(s_t, a_t; \theta)]^2. \quad (4)$$

C. DUELLING DEEP Q-NETWORKS (DUELLING DQNS)

Duelling DQNs decompose the state action value function $Q(s, a)$ in two separate functions. A state-value function $V(s)$, that gives the value of being in a given state s, and an action-value function $A(a)$ that represents the value of an action compared to its alternatives. These functions correspond to different layers in the DQN that generate a combined Q-value given by:

$$Q(s, a; \alpha, \beta) = V(s; \beta) + \left(A(s, a; \alpha) - \frac{\sum_{a'} A(s, a'; \alpha)}{|A|}\right), \quad (5)$$

where α is a parameter of the advantage function, β of the value function, and $|A|$ is the number of actions in the action space. This division of the Q-value in two functions has the goal of increasing training stability and accelerating convergence.

D. DRL MODEL

The DRL problem is modeled by $M = \langle S, A, R \rangle$, with S being the space state, A the action space and R the reward function. The network is modeled by a graph $G(V, E)$, where V is the set of vertices and E the set of edges. The set of host nodes (i.e. connection points to the SDI encoders/decoders) is denoted by H.

A state $S_t \in S$ is sampled at time step t and is represented by a 3-dimensional tensor with dimensions $[N, N, 1]$, where N is the number of hosts $h \in H$ in the network (i.e. $N = |H|$), and the rightmost index of the tensor dimensions corresponds to a network representation metric. At the start of the algorithm training, k paths $p_k(n_s, n_d)$ are computed, using the Dijkstra algorithm, between each source/destination pair. The network representation metric is equivalent to the available bandwidth in the bottleneck link of one of the pre-computed paths. This value is calculated by determining the available bandwidth in the bottleneck link of each of the k paths, selecting the path p_l with the lowest value and using that bandwidth value, depicted as $L_{bw}(p_l(n_s, n_d))$, to be part of the state. An alternative that was considered and tested was to use each of the bottleneck links for the k paths in the state definition. This changes the state dimensions to $[N, N, k, 1]$, decreasing the scalability of the state space, without visible gains in the results. Using only the smallest value among the k paths still forces the algorithm to pursue path diversification since the algorithm will try to maintain an equilibrium between each path's utilization to optimize (i.e. maximize) the state representation metric value, achieving similar results with a much smaller state space.

The action space A is defined as:

$$A = \{a_1, a_2, \ldots, a_k\},$$

where k is the number of pre-computed paths and

$$a_i = p_{a_i}(n_s, n_d), i \in 1, 2, \ldots, k.$$

$R(s, a)$ is the reward function that translates an action a_i in state S_t to a reward value that is calculated according to the available bandwidth of the selected path's bottleneck link in the resulting state. It is defined as:

$$R(s,a) = \begin{cases} +50, & \text{if } L_{bw}(p_l(n_s, n_d)) \geq 75 \\ +30, & \text{if } L_{bw}(p_l(n_s, n_d)) \geq 50 \\ 0, & \text{if } L_{bw}(p_l(n_s, n_d)) \geq 25 \\ -10, & \text{if } L_{bw}(p_l(n_s, n_d))) \geq 0 \\ -100, & \text{if } L_{bw}(p_l(n_s, n_d)) < 0 \end{cases}$$
$$\forall (n_s, n_d) \in N \text{ and } p_l n_s, n_d \in k. \quad (6)$$

For each combination (n_s, n_d) and $p_l(n_s, n_d)$, the reward function will increase or decrease the total reward according to the respective $L_{bw}(p_l(n_s, n_d))$. This function is designed to maximize available bandwidth while severely penalizing requests that are assigned to paths with links that will become congested.

E. TRAINING LOOP

Training the three Deep Q-Learning algorithms requires multiple environment episodes. Each of these episodes consists of a set of video stream requests, from a service booking, that have to be placed in the environment. The placement of these streams results in new states and rewards that feed the Q-networks learning process. The inputs of the training loop algorithm are the number of video streams requests, the communicating pairs of nodes and the reserved bandwidth for each stream of an episode.

Algorithm 1 Training Loop Algorithm

```
Initialize replay buffer R_b
Q ← Q_Network(θ)
Q̂ ← Q_Network(θ')
for i episodes do
    requests ← 0
    done ← false
    max_requests ← N
    while not done do
        a_t = (1 − ε) × argmax(s_t, a_t, θ)
        s_{t+1} ← take_action(a_t)
        r_t ← evaluate(s_{t+1})
        requests ← requests + 1
        store (s_t, a_t, r_t, s_{t+1}) in R_b
        θ ← update_model(sample(R_b))
        if i == update frequency then
            Q̂ ← Q
        end if
        if requests == max_requests then
            done ← True
        end if
    end while
end for
```

Algorithm 1 provides an overlook of the training process. It starts with the initialization of the replay buffer R_b, the main Q-network Q and the target Q-network $\hat{Q}$, and then runs during the defined i number of episodes. For each episode, the current number of active requests for video stream communication (*requests*), the control variable for the loop (*done*) and the maximum number of requests for the episode (*max_requests*) are initialized.

In each iteration of the episode's training loop, the action a_t that returns the highest value considering the current state s_t and the current model weights θ is selected with probability $(1 - \epsilon)$ (exploration is performed by selecting a random a_t with ϵ probability). That action is then performed in the environment by placing the media stream of the request in the path corresponding to action a_t. This results in an updated state s_{t+1}, that already reflects the placement of this request in the network. This state is then evaluated according to equation 6, resulting in a reward r_t. The current transition (s_t, a_t, r_t, s_{t+1}) is then added to the replay buffer R_b and the model is updated using a sample of transitions from the replay buffer and the update rule of the specific Deep Q-Learning method (this is the only step that differs between the three different DQN methods that were implemented). The target model weights θ' are then equaled to the main model weights θ, with a given update frequency. Finally, the number of requests is incremented and if it has reached the total number of requests of the episode, the loop stops and a new episode starts.

TABLE 1. DQN and DDQN network's layers.

Type	Input Size	Output Size	Activation Function
Linear	STATE_SIZE	l1_out	ReLU
Linear	l2_in	l2_out	ReLU
Linear	l3_in	l3_out	ReLU
Linear	l4_in	N_ACTIONS	-

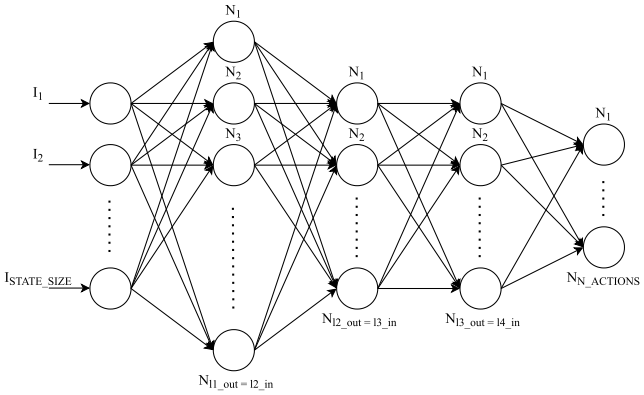


FIGURE 2. DQN and DDQN architecture.

TABLE 2. Duelling DQN network layers.

Type	Input Size	Output Size	Act. Function
Linear	STATE_SIZE	l1_out	ReLU
Linear	l2_in	l2_out	ReLU
Linear (Value)	l3a_in	l3a_out	ReLU
Linear (Value)	l4a_in	1	-
Linear (Advantage)	l3b_in	l3b_out	ReLU
Linear (Advantage)	l4b_in	N_ACTIONS	-

F. Q-NETWORKS ARCHITECTURES

We implemented three different Deep Q-Network algorithms: DQN, DDQN and Duelling DQN. For the DQN and DDQN algorithms, the network layers of the Q-networks both follow the structure summarized in Table 1 and illustrated in Figure 2.

Given its architecture, which separates the value and advantage functions, the layers in the Q-Network for the Duelling DQN algorithm are organized differently, as Table 2 and Figure 3 illustrate. In Figure 3, the top blocks at the final layers correspond to the value segment of the Duelling DQN and the lower blocks to the advantage. After calculating the state value and the advantage of each action, the results are aggregated and the Q-value for each state-action pair is generated. These Q-values are calculated as the sum of the value function and the respective advantage subtracted by the average advantage of all actions.

In both cases the input layer has an input size corresponding to the state size of the model, which corresponds to the size of the flattened tensor of $[N, N, 1]$ dimensions described in section VI-D. The output layer has the size of the number of possible actions, in our case the number of paths between the communicating hosts. The sizes of the inner layers are dependent on the topology the algorithm is applied to. If the network size increases (i.e. additional edges and possible paths between them), the state flattened tensor size increases

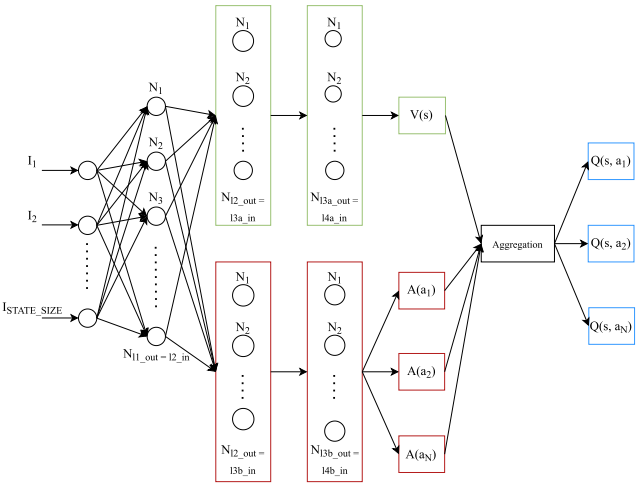


FIGURE 3. Dueling DQN architecture."N" stands for "N_ACTIONS".

TABLE 3. Training loop hyper-parameters.

Parameter	Description
α	Learning rate
ϵ_i	Initial exploration rate
ϵ_f	Final exploration rate
γ	Future reward discount rate
$\|R_b\|$	Number of stored observations
Batch size	Number of selected observations from R_b
Update steps	Periodicity of target network update
Epochs	Number of episodes

meaning that more nodes will be used in each layer. The chosen activation function for every layer in the models is the Rectified Linear Unit function given by $f(x) = max(0, x)$.

The hyper-parameters for the training loop are listed in Table 3.

Finally the loss function used for the update of the Q-networks weights θ is the Mean Squared Error function (MSE):

$$MSE = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y} - y)^2, \tag{7}$$

And the error is minimized using an adaptive stochastic gradient descent algorithm called Adam [29].

VII. RESULTS AND DISCUSSION

We evaluated the proposed DRL solution using the three different deep Q-network algorithms and compared the results with the SLC and DLC base line methods.

A. DEVELOPMENT AND SIMULATION ENVIRONMENT

The system was designed for use in real production scenarios as described in section IV. For that purpose, we selected one of the leading existing SDN orchestrators in the market [5] as the implementation target and designed the system assuming its capabilities and limitations. We also asked providers for information on the Service Level Agreements that are commonly used in the broadcast industry and obtained a real production network topology used to transport broadcast services.

However, it is not feasible to train a DRL based solution in a real production network, so a simulation environment was implemented that mimics those real life conditions as closely as possible.

This environment was built using Mininet to set up a network containing virtual hosts, switches and links, on top of which the different tested approaches will operate. The orchestrator is simulated using the Ryu OpenFlow controller to install forwarding rules into the switches according to the results of the algorithms.

The experiments were performed using two topologies, depicted in Fig. 4: the ARPANET network, and the real media broadcast network topology, depicted as Network RW. Network RW is a real topology from a service provider that uses the SDN orchestrator described in [5] to manage the broadcast services in their infrastructure. Given the respective topologies dimensions, we defined the k number of available paths between end-points as $k = 5$ for ARPANET, and $k = 10$ for Network RW. It is also worth mentioning that all ARPANET links have a 100 Mb capacity, while Network RW uses a set of link capacities that range between 100 and 1000 Mb.

B. EVALUATION METRICS

The evaluation of developed media stream routing algorithms is based on three metrics:

1) *Flow average bitrate*.
2) *Flow average round-trip-time (RTT)*.
3) *Number of uncongested flows*. The number of services that meet the requested requirements. This value is calculated by counting the number of flows with an average bitrate equal to the requested bandwidth in the SLA.

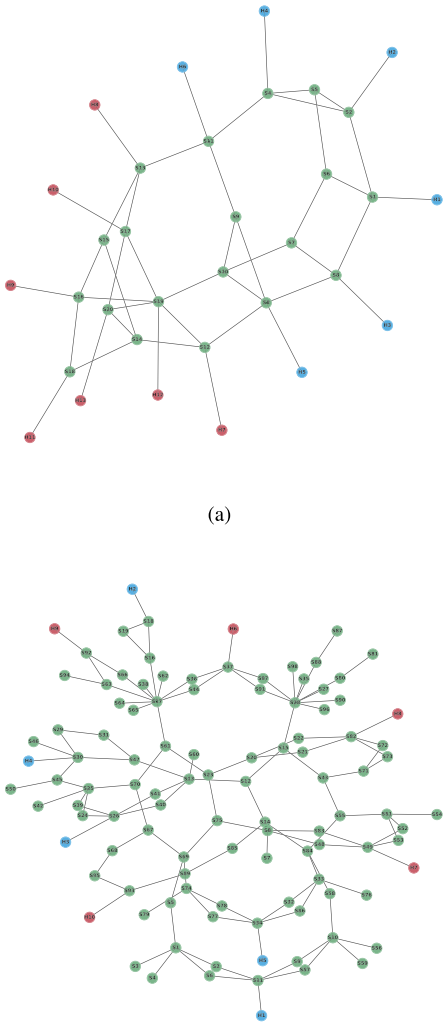
These metrics have a strong influence on each other. When the network becomes congested, the bitrate of a flow and the average RTT decrease. Moreover, that same bitrate drop makes the number of uncongested flows also decrease.

C. TRAINING SETTINGS

Three different DQN-based agents were trained (DQN, Double DQN and Duelling DQN), both for ARPANET and Network RW.

Also, four distinct DRL environment setups were designed, picturing different broadcast service profiles.

Setup 1 uses fixed settings in each training episode, this recreates a scenario where the network use is constant. For this setup each episode will have 32 concurrent stream requests all requiring a 15 Mb Bandwidth for the ARPANET topology and 24 requests of 20 Mb for Network RW. The sources and destinations of the requests are selected by computing the worst-case traffic scenario based on link centrality, resulting in the selection of the host pairs that, when communicating, will make the network suffer from congestion faster. These sources and destinations remain fixed during the entire training process.



(a)

(b)

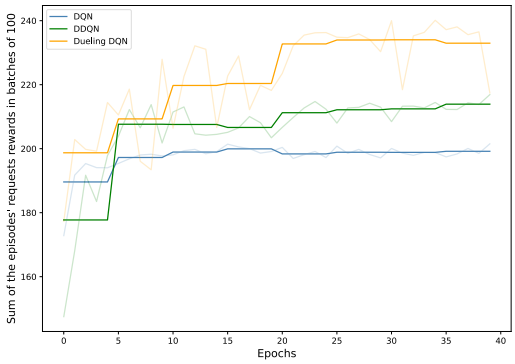
FIGURE 4. Simulation network topologies (source hosts are represented in blue, destination hosts in red and switches in green): (a) ARPANET with 20 switches, 13 hosts and 45 links; (b) Network RW with 10 hosts, 98 switches and 131 links.

Setup 2 was designed to simulate irregular network uses. This is achieved through the randomization of the number of requests per episode between 1 and the number of requests used in setup 1.

Setup 3 simulates a tailored network use where some variation may still occur. Thus, it uses a smoother change of the number of requests per episode that is achieved by sampling values from a normal distribution centered around an average of 24 concurrent requests. Each request, can have a bandwidth requirement of 5, 10, 15 or 18 Mbits/s.

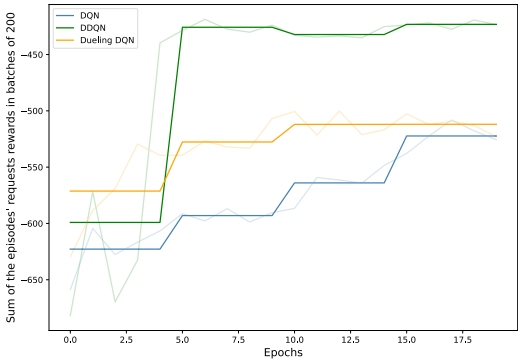
Lastly, setup 4 is the only one that does not use the worst-case traffic scenario for the selection of the source and destination of the requests. Instead, these are sampled from two lists of possibilities. The remaining settings from setup 3 are not altered.

For the Network RW topology only setup 1 was used, since it is the closest to what is defined by the SLAs in real-world



(a)

FIGURE 5. Evolution of the reward during the agents' learning processes in Network RW using setup 1.



(a)

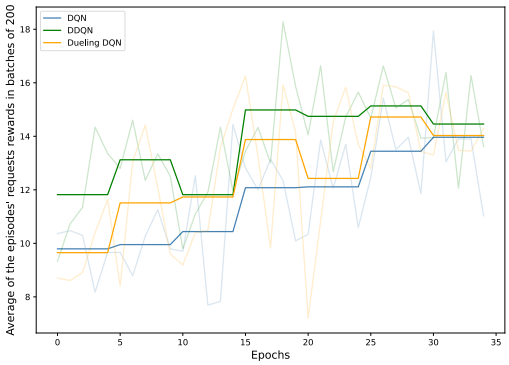
FIGURE 6. Evolution of the reward values during the agents' learning processes in ARPANET. Setup 1.

situations (i.e. fixed bandwidth must be available throughout all of the media stream duration and the number of streams in a service is known). Figure 5 shows the evolution of the reward for the Network RW topology using setup 1.

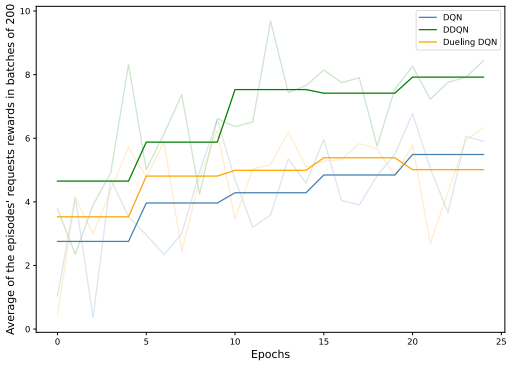
For the ARPANET topology, the agents were trained for all setups. Figure 6 shows the reward evolution for setup 1 and Figure 7 for setups 2, 3 and 4.

In order to smooth the reward plots, averages of batches of epochs were used. Thus, the real horizontal axis value of each plot is found by multiplying it by the size of the batch, indicated in the vertical axis. Also, in figure 7, the average reward per episode is represented instead of the total episode reward due to the reward value oscillation introduced by using a variable number of requests per episode.

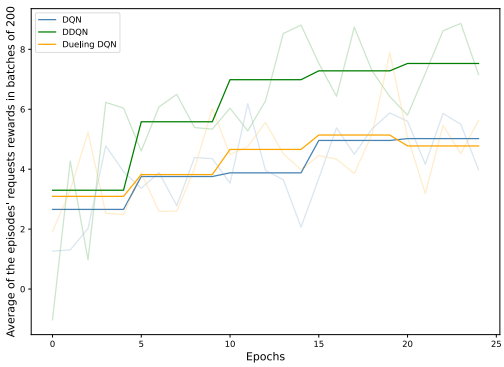
For setup 1, 4000 epochs were used. For setups 2, 3 and 4, the different agents were trained with a higher number of epochs (up to a maximum of 7000 epochs) due to the use of randomization. The training time of each agent varies according to the used Deep Q-Network algorithm variant (DDQN is the slowest and DQN the fastest) and the number of epochs. In the conducted experiments, the training processes took between 3 and 5 hours for the ARPANET topology and between 6 and 8 hours for Network RW. The training was performed in a single machine with an Intel Core i5 processor and 8Gb of RAM, without using a GPU.



(a)



(b)



(c)

FIGURE 7. Evolution of the reward values during the agents' learning processes in ARPANET. (a) Setup 2; (b) Setup 3; (c) Setup 4.

D. TRAINING RESULTS

Figures 6 shows the results for setup 1 in the ARPANET topology. The reward increased for all of the implemented agents that begin the training with similar reward values (the variation is mostly due to the random initial θ weights of the Q-networks). DDQN stabilizes its reward value at around -425, a significant improvement over DQN and Duelling DQN, both stabilizing at around -525. This stabilization occurs faster for DDQN than Duelling DQN while DQN is the slowest to converge. The results observed are negative due

to the chosen reward function and its application to a worst-case scenario, where even if the best paths for each request are chosen, some links will always be overly booked given that multiple requests are concurrently being transported.

For the remaining setups, shown in figure 7, the DDQN agent continues to converge faster and to higher reward values than the DQN and Duelling DQN agents. The reward values are much more unstable in setups 2, 3 and 4. This is due to the variable number of requests these setups use in each episode, resulting in fluctuations in the average reward of the episodes. In episodes with a small number of requests, poor action selection will have a smaller impact on the network state due to the reduced number of requests and can therefore still result in good reward values. On the other hand, in episodes with a high number of requests, poor actions accumulate over the requests and have a higher impact in the network state resulting in lower reward values. For this reason, even though the plots still show an increasing reward value as the agents learn, there is a higher variance in the results.

In setup 1, the request rewards in the beginning of an episode are positive and start decreasing as more flows are allocated due to the increase in congestion. Since our reward function is not linear and severely penalizes congestion, the requests allocated last result in exponentially lower rewards, resulting in a negative sum of rewards per episode. In setups 2 to 4, the average of the episodes requests rewards are positive. This occurs because the number of requests per episode is lower, leading to lower congestion and, consequently, higher rewards.

The performance differences verified between agents can be explained by their distinct behaviours. The superior performance of the DDQN agent can be related to its use of a double estimator for Q-values, which prevents some actions from overshooting and getting constantly selected. This allows for a better exploration of the network with the agent being able to test different actions in situations where the other algorithms do not. The Duelling DQN algorithm only had a marginal increase of performance compared to the DQN algorithm. In Duelling DQN, the state and action values are decoupled, which, in some problems, facilitates the learning of action values, thus improving performance. However, in this topology, the action and the state values are too correlated for this approach to be beneficial.

Fig. 5 introduces the training process of the three agents in Network RW for setup 1. In this case, the Duelling DQN algorithm has the best performance of the three algorithms with a similar convergence time than DDQN, but a much higher reward value. This can be explained by the topology itself that allows the Duelling DQN to benefit from the state-action values separation. DDQN is still marginally superior to DQN in terms of reward, while DQN as a much slower convergence time.

The rewards obtained in the Network RW topology are higher than in the ARPANET topology. This indicates that the agents were able to assign paths that are better at avoiding congestion in this topology. This will be further analysed, but it is easy to see that Network RW has a topology that is more suitable for load balancing than the ARPANET topology. Additionally, the different environment settings, such as the number of requests per episode and their bandwidth requirements, might also positively affect the agents performance.

TABLE 4. Comparison between dynamic link cost algorithms (LIOA and DSP) and Static link cost Dijkstra (MHA).

Metric	MHA	DSP	LIOA
Bitrate (Mbits/s)	9.99	10.50	11.04
RTT (s)	0.331	0.349	0.314
Uncongested flows	6	6	7

E. PERFORMANCE COMPARISON

After the training phase of the DRL based algorithms, three comparative performance analysis were performed:

1) Comparison between the baseline algorithms: MHA; DLC and LIOA.
2) Performance evaluation of the three agents trained in ARPANET and the four setups presented, measured against MHA.
3) Performance evaluation of the three DRL algorithms in Network RW compared with MHA.

1) COMPARISON BETWEEN BASELINE ALGORITHMS

The following results were obtained using 32 stream requests that last 180-seconds and have a 15 Mbits/s bitrate requirement.

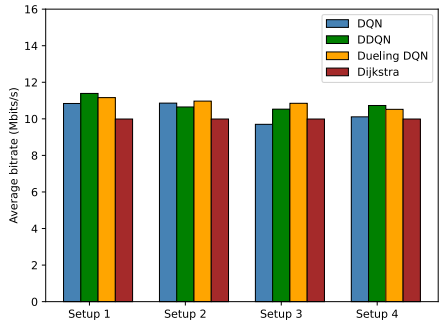
The results in Table 4 show that the two DLC solutions can slightly improve the performance of the MHA approach. This improvement is explained by how DSP and LIOA can better avoid the concentration of all requests in the same popular links by changing their weights. This results in higher bitrate levels because there is less congestion. However, this decrease in congestion is marginal, since they do not show significant improvement on the number of uncongested flows. In terms of RTT, the results are identical among the three algorithms. Finally, a DLC solution is operationally difficult to implement in the realistic scenario that we considered, since obtaining the network state can be a time intensive operation.

2) DRL VERSUS SLC IN THE ARPANET TOPOLOGY

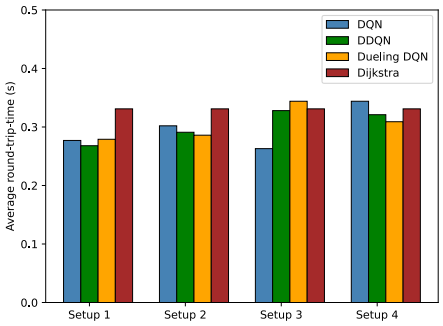
The results in Fig. 8 allow us to compare the performance of the trained DRL agents to the baseline static link weight Dijkstra in the ARPANET topology, for each of the four training setups.

After an analysis of the obtained results, we can see that DDQN is the best performing agent across all setups, which is consistent with its higher reward values in the training phase.

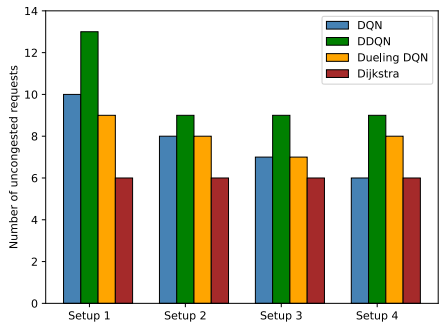
Regarding the average bitrate, it is clear that in setups 1 and 2, all the DRL agents outperform Dijsktra. Additionally, between them, there is not much difference. In setups 3 and 4, however, DQN starts to fall back from the remaining agents' performance and to match Dijkstra based MHA.



(a)



(b)



(c)

FIGURE 8. Performance of the different developed agents in the four setups and Dijkstra in ARPANET, according to the three evaluation metrics. (a) Average bitrate; (b) Average round-trip-time; (c) Number of uncongested flow requests.

In terms of average RTT, the trend continues. In setups 1 and 2, the DRL agents are much better than Dijkstra, but tend to deteriorate in setups 3 and 4.

The last evaluation metric is the number of uncongested flow requests. The optimization of this value is the most important goal of video stream routing algorithm due to the nature of the SLA that are used in real life scenarios. DRL approaches show a strong increase in performance over Dijkstra in this metric, specifically in the DDQN agent in setup 1. The DQN and Duelling DQN agents are also capable of outperforming the baseline routing mechanism in almost every scenario, expect for the DQN agent in setup 4.

TABLE 5. Performance of the DRL agents compared to SLC Dijkstra (MHA).

Metric	Bitrate (Mbits/s)	RTT (s)	Uncongested flows
Dijkstra	14.84	0.412	2
DQN	18.88	0.192	12
DDQN	19.12	0.187	15
Dueling DQN	19.20	0.166	16

A common trend for all metrics and agents is the performance degradation in setups 3 and 4, especially in the RTT values. This can be explained by the introduction of bitrate variation in the training episodes. Since bandwidth is the indicator of the network state used in the DRL model, changes to its value impact the performance of the agents by making it more difficult for them to understand the underlying network. Setup 1 is also clearly the most favorable environment for the algorithms, mainly because of its smoother learning process. Setup 2, despite using a random number of requests per episode, still manages to achieve better results that the last two, which supports the idea of the bitrate requirement per request having a stronger influence on the agents performance than the number of video streams to transmit.

Based on these observations, we can conclude that the DRL approach to routing is especially efficient when the communication sessions occur between the same endpoints and maintain their requirements (i.e. bitrate) throughout a service, something that matches the most common scenario in video broadcast services.

3) DRL VERSUS SLC IN THE NETWORK RW TOPOLOGY

In Table 5, the performance of the DRL agents trained using setup 1 in the Network RW topology is presented and compared to the SLC using Dijkstra. In the evaluation, 25 video stream requests were placed between the source hosts and every destination host in the network, with a bitrate requirement of 20 Mbits/s.

It is clear that all three DRL agents outperform MHA. For all agents, the average bitrate achieved for the 25 stream requests reaches levels close to their bitrate requirement. Additionally, 16 out of the 25 achieved the requested 20 Mbits/s bitrate during the entire stream transmission. The RTT also decreases up to 60% compared to MHA. This is possible because the agents learn to place flows in less congested paths, balancing the load evenly across the network. These results indicate that the use of DRL algorithms is applicable to real production all IP broadcast networks, since not only the network capacity is maximized (i.e. more uncongested video streams), but each stream bitrate and RTT are optimized versus traditional link cost based approaches.

The results achieved by the DRL approach in the Network RW topology also reveal a much more pronounced improvement than in the ARPANET topology. Network RW, is a production network topology of an operator, with more path diversity than in the ARPANET topology, which seems to favour the agents' ability to learn how to make more

efficient routing decisions with more choices of actions to explore.

VIII. CONCLUSION

This work demonstrates the use of model-free DRL based algorithms for routing video streams in IP networks. Our DRL model shows that under the requirements and characteristics of real production environments, we have treatable state and action spaces and that the agents converge and learn to increase the reward. The results show that the routing performance significantly increases with the use of DRL versus the standard methods that are currently used in production networks. They also show that the enhancements introduced by the Duelling DQN algorithm increase the performance of the DRL agent in this scenario. Finally, this work differs from existing DRL based routing proposals by tackling a previously untreated scenario and using conditions very close to real production scenarios, serving as a motivating example for the use of this type of approach in the industry.

REFERENCES

[1] J. Hudson and N. Seth-Smith, "3G: The evolution of the serial digital interface (SDI)," *SMPTE Motion Imag. J.*, vol. 115, nos. 11–12, pp. 472–481, Nov. 2006.
[2] *Smpte Overview Document—Professional Media Over Managed Ip Networks Roadmap for the 2110 Document Suite*, document OV 2110-0:2018, 2019, pp. 1–4.
[3] R. Cabrera, J. Montalban, P. Angueira, Y. Wu, L. Zhang, W. Li, S.-I. Park, S. Kwon, and N. Hur, "ATSC 3.0 broadcast core network for next-generation media delivery," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Aug. 2021, pp. 1–7.
[4] J. Montalban, R. Cabrera, E. Iradier, P. Angueira, Y. Wu, L. Zhang, W. Li, and Z. Hong, "Broadcast core-network: Converging broadcasting with the connected world," *IEEE Trans. Broadcast.*, vol. 67, no. 3, pp. 558–569, Sep. 2021.
[5] *Network Management System Designed for Monitoring, Maintaining and Optimizing your Operation*, Skyline Commun., Izegem, Belgium, 2021.
[6] M. Sanaei and S. Mostafavi, "Multimedia delivery techniques over software-defined networks: A survey," in *Proc. 5th Int. Conf. Web Res. (ICWR)*, Apr. 2019, pp. 105–110.
[7] G. Trimponias, Y. Xiao, X. Wu, H. Xu, and Y. Geng, "Node-constrained traffic engineering: Theory and applications," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1344–1358, Aug. 2019.
[8] E. Akin and T. Korkmaz, "Comparison of routing algorithms with static and dynamic link cost in software defined networking (SDN)," *IEEE Access*, vol. 7, pp. 148629–148644, 2019.
[9] P. Sun, Z. Guo, J. Lan, J. Li, Y. Hu, and T. Baker, "ScaleDRL: A scalable deep reinforcement learning approach for traffic engineering in SDN with pinning control," *Comput. Netw.*, vol. 190, May 2021, Art. no. 107891.
[10] M. Kim *et al.*, "Learning collaborative policies to solve NP-hard routing problems," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 1–13.
[11] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in SDN," in *Proc. ACM Symp. SDN Res.*, Apr. 2019, pp. 140–151.
[12] G. R. Waissi, *Network Flows: Theory, Algorithms, and Applications*. London, U.K.: Pearson, 1994.
[13] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 388–415, 1st Quart., 2018.
[14] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time quality of service with software defined networking," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 70–76.
[15] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications," in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2016, pp. 193–202.
[16] S. Tomovic, N. Lekic, I. Radusinovic, and G. Gardasevic, "A new approach to dynamic routing in SDN networks," in *Proc. 18th Medit. Electr. Conf. (MELECON)*, Apr. 2016, pp. 1–6.
[17] R. Amin, E. Rojas, A. Aqdus, S. Ramzan, D. Casillas-Perez, and J. M. Arco, "A survey on machine learning techniques for routing optimization in SDN," *IEEE Access*, vol. 9, pp. 104582–104611, 2021.
[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
[20] J. N. Witanto and H. Lim, "Software-defined networking application with deep deterministic policy gradient," in *Proc. 11th Int. Conf. Comput. Modeling Simulation (ICCMS)*, 2019, pp. 176–179.
[21] X. Huang, T. Yuan, G. Qiao, and Y. Ren, "Deep reinforcement learning for multimedia traffic control in software defined networking," *IEEE Netw.*, vol. 32, no. 6, pp. 35–41, Nov./Dec. 2018.
[22] 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 *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1871–1879.
[23] 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. 64533–64539, 2018.
[24] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "RL-routing: An SDN routing algorithm based on deep reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 3185–3199, Oct. 2020.
[25] P. Sun, J. Lan, J. Li, J. Zhang, Y. Hu, and Z. Guo, "A scalable deep reinforcement learning approach for traffic engineering based on link control," *IEEE Commun. Lett.*, vol. 25, no. 1, pp. 171–175, Jan. 2021.
[26] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. Rodrigues Prete, "Using mininet for emulation and prototyping software-defined networks," in *Proc. IEEE Colombian Conf. Commun. Comput. (COLCOM)*, Jun. 2014, pp. 1–6.
[27] A. Zai and B. Brown, *Deep Reinforcement Learning in Action*. Shelter Island, NY, USA: Manning Publications, 2020.
[28] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.
[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

PEDRO AMARAL (Member, IEEE) received the M.Sc. degree in computer engineering and the Ph.D. degree in electric and computer engineering from the Universidade Nova de Lisboa, in 2006 and 2013, respectively. He is currently an Assistant Professor with the Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, and a Researcher with the Instituto de Telecomunicações, Lisboa. His current research interests include model free resource optimization algorithms in networks, smart-grid security, and very low delay networks.

DIOGO SIMÕES received the M.Sc. degree in electric and computer engineering, in 2021. He has been an Engineer at Skyline Communications, since 2022, working in the development of network management and orchestration platforms. His current research interest includes AI powered network management systems.