Impact Evaluation of Control Signalling onto Distributed Learning-based Packet Routing

Redha A. Alliche*, Tiago Da Silva Barros*, Ramon Aparicio-Pardo*, Lucile Sassatelli†
* Université Côte d'Azur, CNRS, Inria, I3S, France
† Université Côte d'Azur, CNRS, I3S, Institut Universitaire de France, France
Contact: alliche@i3s.unice.fr

***Abstract*—In recent years, several works have studied Multi-Agent Deep Reinforcement Learning for the Distributed Packet Routing problem, with promising results in various scenarios where network status changes dynamically, is uncertain, or is partially hidden (e.g., wireless ad hoc networks or wired multi-domain networks). Unfortunately, these previous works focus on an ideal scenario where the impact of control signalling is neglected, and network simulation is tailored to simplistic assumptions. In this article, we present the first experimental investigation of control signalling mechanisms for distributed learning-based packet routing. We rely on `PRISMA`, our open-source simulation `ns-3`-based module. We formulate two signalling mechanisms between agents (value sharing and model sharing). We investigate the net gains considering in-band signalling and show that routing policies close to those provided by an oracle with full knowledge of traffic and network topology can be discovered with a control overhead of 150 % with respect to injected data packets, if neighboring agents share their Deep Neural Network models. We discuss the generality of our results to underline the importance of assessing net gains of Multi-Agent Deep Reinforcement Learning (MA-DRL)-based routing.**

***Index Terms*—Multi-Agent Learning, Deep Reinforcement Learning, packet routing, control signalling**

I. Introduction

After the breakthrough results obtained by Deep Reinforcement Learning (DRL) [1] in solving highly complex tasks [2], DRL has been also started to be applied to communication networks problems. One of them is the Distributed Packet Routing (DPR) [3]–[6]. This is a challenging problem where no complete and centralized view of network topology and traffic demands is available (e.g., multi-hop wireless networks [7] or multi-domain optical networks [8]). In the DPR problem, per-packet forwarding decisions are made locally by distributed agents, exploiting local information, such as packet headers and neighboring node states. Nevertheless, these first works [3]–[6] focused on improving the learning performance by modifying the training mechanisms and the model design, neglecting the impact of control signalling on routing performance. Indeed, the *distributed multi-agent* setting of the DPR problem imposes a non-negligible level of communication between the neighboring agents during the training phase, which is meant to run continuously or frequently to adapt the routing policy to any network change (traffic, topology). This information exchange constitutes the control signalling, and it introduces a certain level of *overhead*. Thus, a trade-off appears between this *overhead* and the *quality* of the learned routing: a minimal amount of signalling (*overhead*) is required to make the agents learn a routing policy at a cost of increased bandwidth requirements. On the other hand, an excessive control *overhead* could slow down data packets and impede the learning process.

This paper hence focuses on the impact evaluation of this control signalling. To perform the above-mentioned impact evaluation, a realistic modelling of the communication is necessary, in contrast to the simplified simulations performed with ad hoc network simulators in the previous works. For this reason, we make use of our `PRISMA` tool, detailed in Section III-F. This is a discrete time packet-level simulator designed to apply MA-DRL to the DPR problem. It is based on the `ns-3` [9] network simulator and a multi-threaded implementation of each agent, enabling a more realistic network simulation where the control signalling can be implemented in a reproducible manner. In the MA-DRL framework, neighboring agents collaborate by exchanging information, particularly the estimated packet end-to-end delay. In this work, we consider that this information can be shared as the current value of the estimate (*value sharing*), or as the estimating model itself (*model sharing*).

Our contributions are:

- We present, to the best of our knowledge, the first experimental investigation of control signalling mechanisms for distributed learning-based packet routing. We rely on `PRISMA`, our open simulation `ns-3`-based module enabling testing MA-DRL-based routing in a reproducible and realistic manner [10].
- We formulate two signalling mechanisms for the *DQN routing* [3] framework, where routing nodes periodically communicate copies (*target*) of their Machine Learning (ML) models to their neighbors (named *model sharing*), or only communicate delay estimates (named *value sharing*). The control overhead induced by these exchanges can be modulated by tuning the period between two consecutive exchanges.
- We investigate net gains considering in-band signalling and show that (i) *model sharing* between agents is necessary to find routing policies close to the optimal routing of an Oracle with perfect knowledge of topology and traffic, but at the cost of

The author acknowledges the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-19-CE-25-0001-01 (ARTIC project). This work was performed using HPC resources from GENCI-IDRIS (Grant 2021-AD011012577).

possibly significant control overhead (150 % of extra traffic in the considered network settings with respect to useful traffic); and (ii) *value sharing* incurs in much more reduced overhead (10 %), but barely improves a Shortest Path (SP) routing policy. We discuss the generality of our results to underline the importance of assessing net gains of MA-DRL-based routing.

Related works are presented in Section II, the MA-DRL framework and the control signalling mechanisms are described in Section III. Simulation results are analyzed in Section IV. Section V provides some discussion and Section VI concludes the article.

II. Related works

In recent years, works addressing the DPR problem by making use of DRL have been published in [3]–[6], [11]. The seminal paper [11] was the first to apply a Reinforcement Learning (RL) method, the Q-learning [12], to DPR, giving rise to the *Q-routing* paradigm. In this work, the routing decisions are taken based on the packet destination. The study in [3] applies the Deep Q-Network (DQN) framework to this *Q-routing* by using a Deep Neural Network (DNN) to approximate the Q-value function, yielding its deep learning version: the *DQN routing*.

After *DQN routing*, other proposals of MA-DRL applied to DPR were published in [4]–[6]. These works concentrate on how to improve the quality of the learned routing from a ML perspective (i.e., changing the model, the input features or the RL algorithm), neglecting networking aspects, as the signalling. *DQRC (DQR with Communication)* [4] adds the action history, the future destinations, and the most loaded neighboring node to the packet destination as input. The neural network architecture of *DQRC* also considers a LSTM layer to exploit the new time series features (action history and future destinations). Authors in [5] enlarge the input algorithm with *relational features*, such as the node buffers' occupancy, the distance from the routing device to the packet destination or the packet TTL field. The values of these features can vary for individual packets and devices, but the features' list (and its size) is the same for all the routing nodes, which helps to improve the generalization among the nodes (the same neural network model is used for all the nodes). Finally, the work in [6] adapts the Proximal Policy Optimization (PPO) [13] RL algorithm to the multi-agent scheme of the DPR problem, yielding the Multi-Agent Proximal Policy Optimization (MAPPO) algorithm. In MAPPO, the task to learn is considered as *to find a routing for a given traffic matrix*, using as input the packet destination and the buffers' occupancy. Finally, this work also aims to generalize the routing policies to different traffic matrices (i.e., tasks) by extending MAPPO to the meta-learning framework.

In all the above-mentioned works, the impact of the network control signalling onto the training process is ignored or superficially considered. In the more recent works [5], [6], the training is centralized in a node that gathers all the past experiences (forwarding decisions) of all the forwarded packets at all the routers. The model for each routing node is learned at this central node using this large training set, and is eventually pushed to the routing agents that will forward the packets. The significant overhead associated to these data transfers is not analyzed, questioning the actual gains of the proposed methods. On the contrary, the first works [3], [4] assumed a distributed training operation where training data are exchanged only locally between neighboring agents at a cost of the *non-stationarity* of multi-agent environments [14], which makes harder the agents' policies convergence. To tackle this problem, the authors in [15] propose that the neighboring agents share their policies (models) to stabilize the learning when larger datasets of past experiences are used (i.e., replay memories). However, current DQN routing such as [3] operates by sharing the values of model estimates (*value sharing*), instead of sharing the model at every neighboring node (*model sharing*). Value sharing reduces the signalling overhead, but yields poor results when replay memories are used (known to stabilize the training of DQN [2]). On the contrary, *DQRC* [4] share the neighbors' routing models (*model sharing*), but at each training step, which introduces a significant communication overhead not properly evaluated. In the present work, we share copies of the neighboring models (i.e., policies) combined with replay memories, which improves policy convergence [2]. The overhead entailed by sharing the models between neighbors can be modulated with the sharing update period. In this article, we investigate the performance tradeoff entailed by tuning this update period in MA-DRL.

III. RL for Distributed Packet Routing: Formulation and Signalling

In this section, we first describe the network model and formulate the DPR problem. We then express the considered Multi-Agent Reinforcement Learning (MA-RL) approach problem based on [14]. We detail the required control signalling, proposing two different versions and showing how signalling overhead can be traded with routing performance. We finally introduce our reproducible framework for a realistic evaluation of the impact of signalling in MA-RL.

A. Network Model

Let $\mathcal{G}(\mathcal{N}, \mathcal{E})$ be a directed network graph, where $\mathcal{N}$ is the nodes set and $\mathcal{E}$ is the unidirectional links set. A link $(a, b) \in \mathcal{E}$ initiates at node a, terminates at node b and has a capacity of C traffic units (i.e., $Mbps$ or $packets/sec$). The incoming and outgoing neighbor nodes of the node $n \in \mathcal{N}$ are denoted as $\mathcal{N}_n^{in}$ and $\mathcal{N}_n^{out}$, respectively. Each packet is originated from a source node $s \in \mathcal{N}$ and targeted to a destination node $d \in \mathcal{N}$. The traffic matrix $\boldsymbol{H} = \{h_{sd}, (s, d) \in \mathcal{N} \times \mathcal{N}\}$ counts up the average volumes h_{sd} in traffic units (i.e., $Mbps$ or $packets/sec$) of packet flows between the node pairs $(s, d) \in \mathcal{N} \times \mathcal{N}$. Each node $n \in \mathcal{N}$ is a router equipped with $|\mathcal{N}_n^{out}|$ outgoing network interfaces. Each interface $i \in |\mathcal{N}_n^{out}|$ has its own buffer queue of size B. The queue follows a $FIFO$ (*First-In First-Out*) policy. Hence, if a packet arrives at a full buffer, it is rejected (i.e., *lost*). Otherwise, the packet

Name	Description
$x_{ab}^{sd} \in [0,1]$	Fraction of the packet flow between the node pair (s,d) to be forwarded via the link (a,b).
$y_{sd} \in [0,1]$	Fraction of the packet flow between the node pair (s,d) that is rejected.
$h_{sd} \in \mathbb{R}_{\geq 0}$	Average flow volume between the node pair (s,d) in traffic units (i.e., $Kbps$).
$C \in \mathbb{R}_{\geq 0}$	Link capacity in traffic units (i.e., $Kbps$).
$B \in \mathbb{R}_{\geq 0}$	Buffer size in data units (i.e., $Bytes$).
$M \in \mathbb{R}_{\geq 0}$	Large number to penalize a loss more than a large delay.

TABLE I: Notation

is admitted, eventually getting to the buffer head, where it is forwarded to a next-hop node $n' \in \mathcal{N}_n^{out}$. At each hop, this procedure is repeated until the packet finally reaches its final destination d (or is lost elsewhere en route). Table I gathers the notation.

B. An Oracle Routing Policy

In this subsection, we propose an *oracle* policy to be used as a routing optimality benchmark. We devise a centralized version of the DPR problem, where an *Oracle* observer (different from the routing nodes) has a full knowledge of the network topology and the traffic matrix. This centralized version is formulated as the Minimum Cost Multi-Commodity Flow (MCF) (Min Cost MCF) problem solved by the Linear Programming (LP) model (1). The optimal solution of this model provides a lowest-cost routing policy, that we call *oracle routing* in the reminder of this paper.

$$\min_{\{\boldsymbol{x},\boldsymbol{y}\}} \sum_{\substack{s\in\mathcal{N}\\ d\in\mathcal{N}}} h_{sd} \left(\sum_{(a,b)\in\mathcal{E}} x_{ab}^{sd} + M \cdot y_{sd} \right) \tag{1a}$$

$$\text{s.t.} \sum_{a\in\mathcal{N}_n^{in}} x_{an}^{sd} - \sum_{b\in\mathcal{N}_n^{out}} x_{nb}^{sd} = \begin{cases} -1 + y_{sd}, & \text{if } n = s \\ 1 - y_{sd}, & \text{if } n = d \\ 0, & \text{otherwise} \end{cases} \quad s \in \mathcal{N}, d \in \mathcal{N}, n \in \mathcal{N} \tag{1b}$$

$$\sum_{\substack{s\in\mathcal{N}\\ d\in\mathcal{N}}} h_{sd} \cdot x_{ab}^{sd} \leq C, \qquad (a,b) \in \mathcal{E} \tag{1c}$$

$$x_{ab}^{sd} \in [0,1], \qquad s \in \mathcal{N}, d \in \mathcal{N}, (a,b) \in \mathcal{E} \tag{1d}$$

$$y_{sd} \in [0,1], \qquad s \in \mathcal{N}, d \in \mathcal{N} \tag{1e}$$

The objective function (1a) minimizes a twofold objective: (i) primarily, the total amount of rejected traffic $\sum h_{sd} \cdot y_{sd}$; and, (ii) secondly, the average hop count $\sum h_{sd} \sum x_{ab}^{sd}$. Minimizing (ii) leads to minimize the average end-to-end delay in this model. The flow conservation constraints (1b) ensure that all the traffic admitted at source is routed until destination. The constraints (1c) limit the link capacity. Finally, the constraints (1d) and (1e) define the lower and upper bounds on the variables.

The Oracle policy is not necessarily optimal for the DPR problem, since it was computed to solve a centralized and flow-based version of the routing problem. The LP model (1) works with averaged packets' flows and is unaware of the traffic dynamics at the packets' scale (e.g., a burst of packet arrivals whose data rate exceeds nominal link capacities C). Hence, a per-packet routing algorithm could find policies exploiting outgoing buffers, which are unknown for the LP model (1), able to beat this *Oracle routing* in some circumstances (as we will see in Section IV-C). Anyway, we use it since it constitutes a high quality benchmark hard to attain without a global view of the exact network topology and the precise traffic matrix.

C. The DPR Problem and our DQN Routing Algorithm

The DPR problem consists of deciding, at each node, the output port (next hop) for each upcoming packet. The objective is to find a global routing policy that minimizes the end-to-end delay for all the packets over the network. This problem can be formalized as Multi-Agent Partially Observable Markov Decision Process (POMDP) [14]: a Markov Decision Process where each agent can only locally observe its environment. We use an approach based on *Q-routing* [11], but approximating the *Q-value* as a neural network (as in [3], [4]), and using as input the packet destination and the node buffers' occupancy. We will use this scheme to describe the POMDP and the RL approach used to solve it.

Let $\mathcal{N}$ be the set of routing nodes (*agents*), where each *agent* n has its own local observation space $\mathcal{O}_n$ and its own action space $\mathcal{A}_n$. When a new packet arrives at time t at the node n, the node n selects as action a_n the *next hop node* n' to forward the packet to. This decision is taken depending on the local observation of the router o_n. As a consequence of the decision, the *agent* n receives a reward $r_{n'}$ from the next hop node n': the *next-hop packet delay* (i.e. the packet delay to travel from n to n'). A transition is hence made to a new state. When the packet arrives to a node, this procedure is repeated. The action a_n is selected to minimize an estimate of the expected *end-to-end packet delay* from the node n to its final destination. Under the DRL scheme, this estimate, denoted as $Q_n(o_n, a_n; \theta_n)$ (the *Q-value*), is the output of a DNN with weights θ_n. This DNN is trained to fit the target value Y_n^Q below:

$$Y_n^Q = r_{n'} + \gamma \cdot \tau_{n'} \cdot (1 - f) \tag{2}$$

where f indicates if the next hop is the packet destination, $\gamma \in [0,1]$ is a discount factor, r is the *next-hop packet delay* and $\tau_{n'}$ is *the remaining end-to-end delay* from the next hop n' to the destination computed as follows:

$$\tau_{n'} = \min_{a_{n'} \in \mathcal{A}_{n'}} Q_{n'}(o_{n'}, a_{n'}; \theta_{n'}) \tag{3}$$

where $Q_{n'}(\cdot; \theta_{n'})$ is the *Q-value* estimate of the next hop agent.

In order to approach the estimated $Q_n(\cdot;\theta_n)$ to the optimal Q_n^*, at each agent n, the weights θ_n are updated by stochastic gradient descent when minimizing the next square loss called *Temporal Difference (TD) error*:

$$L_{DQN} = \left(Q_n(o_n, a_n; \theta_n) - Y_n^Q\right)^2 \quad (4)$$

In the next paragraphs, we detail (i) the *node observation* representation, (ii) the *node action*, and (iii) the *reward* definition.

Node observation. The *node observation* $O_n \in \mathcal{O}_n$ is represented as $o_n = (d, \{b_i, i \in 1 \dots |\mathcal{N}_n^{out}|\})$. This is the concatenation of *two* components: (i) *the current packet destination* d and, (ii) *the buffers' occupancy* b_i, in $Bytes$, of each node interface i at the node n.

Node action. The *node action* $a_n \in \mathcal{A}_n$ is the choice of the out-neighbor $n' \in \mathcal{N}_n^{out}$ of the node n (i.e., the outgoing interface to this next hop node n') to forward the packet to the buffer head.

Reward. The *reward* $r_{n'}$ is the *next-hop packet delay*, defined as the time required by the packet to travel from the buffer tail of n up to the buffer tail of the next hop node n'. Then, it is computed as $r_{n'} = l + q$, where: (i) l denotes *the link transmission delay*, the transmission latency in the link connecting n and n'; and, (ii) q refers to *the queuing delay*, the time spent by the packet in the outgoing buffer of node n taking to n'. If, in the next hop node n', the outgoing buffer where the packet should be forwarded is full, the packet is lost, and $r_{n'}$ is considered as infinity, since the packet did not arrive at n'. In practice, we use the *worst-case* end-to-end delay: $\frac{|\mathcal{N}|\times(B+1)}{C}$, that is, the delay of traversing over the $|\mathcal{N}|$ nodes when all the output buffers are full. Thus, the reward can be measured in time units (typically, in $seconds$). This reward definition allows the *end-to-end delay* estimate $Q_n(\cdot;\theta_n)$ to account for both buffer delays and packet losses, but giving more weight to the packet losses to favor its minimization in priority, similarly to the LP model of the *oracle routing* (see Section III-B).

D. Neural Network Architecture

In this subsection, we detail the architecture of the neural network $Q_n(s_n, a_n; \theta_n)$, which is depicted in Figure 1.This architecture is inspired from [4].

The input layer is split into two parts according to the description of the node state s_n in Section III-C: (i) *the packet destination*, as a $|\mathcal{N}|$-element vector in *one-hot encoding* (i.e. the position corresponding to the destination is set to one, the rest to zero); and, (ii) *the buffer occupancy of outgoing buffers of node n*, as an $|\mathcal{N}_n^{out}|$-element vector, whose values are layer normalized.

The first hidden layer is also split into two parts, where each one is a 32-neuron fully connected layer fed by their respective input. Afterward, their outputs are concatenated before feeding two 64-neuron fully connected layers.

Finally, the output layer is fully connected with as many neurons as the node action space (i.e., out-degree $|\mathcal{N}_n^{out}|$ of n). The value of each output neuron is simply the estimate of the *Q-value* $Q_n(s_n, a_n; \theta_n)$. All the activations are ELUs (Exponential Linear Units).

E. MA-DRL Signalling: Value Sharing or Model Sharing

DPR consists of forwarding decisions taken in a distributed manner without cooperation between the routing nodes. However, with MA-DRL, control signalling exchanges are necessary to enable agent training, since agents need to share their past observations (i.e., experiences), network metric estimates and ML models before the training takes place. We consider two agent information sharing techniques: (i) *value sharing* and (ii) *model sharing*. In the ***value sharing*** method, a node n' receiving a packet from n sends back to n a control packet containing the pair $(r_{n'}, \tau_{n'})$. When this packet arrives at n, it is stored with its corresponding observation-action pair (o_n, a_n) in the *replay memory* $\mathcal{M}_n$. We refer to these packets as *replay memory update packets*. We recall that $\tau_{n'}$ is the estimate of *the remaining end-to-end delay* computed by the next hop agent n' at the reception of packet at n' as Equation (3). In the ***model sharing*** technique, the *replay memory update packet* contains the pair $(r_{n'}, o_{n'})$. In this case, the $\tau_{n'}$ estimate is not provided by the neighbor n' but computed at the node n by a copy of the next hop agent DNN $Q_{n'}(\cdot, o_{n'}; \theta_{n'})$. We refer to this copy as *target network*, and we denote it as $\hat{Q}_{n'}(\cdot; \theta_{n'}^-)$ Thus, the model weights $\theta_{n'}$ have to be sent from n' to n periodically to update this *target network*, adding a heavier *control overhead* with respect to the *value sharing* technique. We call the packets carrying the model weights *target update packets*. The time between two consecutive target updates is the *target update period* U, which controls the overhead level of the *model sharing* method. The more often the target networks are updated, the heavier is the control overhead, since control and data packets compete for the same bandwidth, which supposes that both packet types will experience additional queuing delays.

Finally, we conclude this section with Algorithm 1 that depicts the pseudocode of the training routine of our approach. We highlight in red and blue italics the lines corresponding to the *value sharing* and *model sharing* method, respectively.

F. A Realistic Evaluation of Signalling with the PRISMA tool

To study the impact of the control signalling, we need a simulation framework reproducible and realistic enough to capture the tradeoffs between the control overhead and the quality of the learned routing. This is enabled by our new `PRISMA` tool, detailed in [16] and made available to the research community [10].

In the previous literature [3]–[6], the DRL approaches were evaluated on simulation environments (typically implemented in `python`) tailored to the assumptions and simplifications made by each study. Namely, these works do not generally implement the RL agents as separated threads or processes; and, they usually assume a unique buffer queue per node and a fix time slotted operation. As a consequence, only one packet per router is transmitted at each time step, which introduces an artificial synchronization among the state transitions along

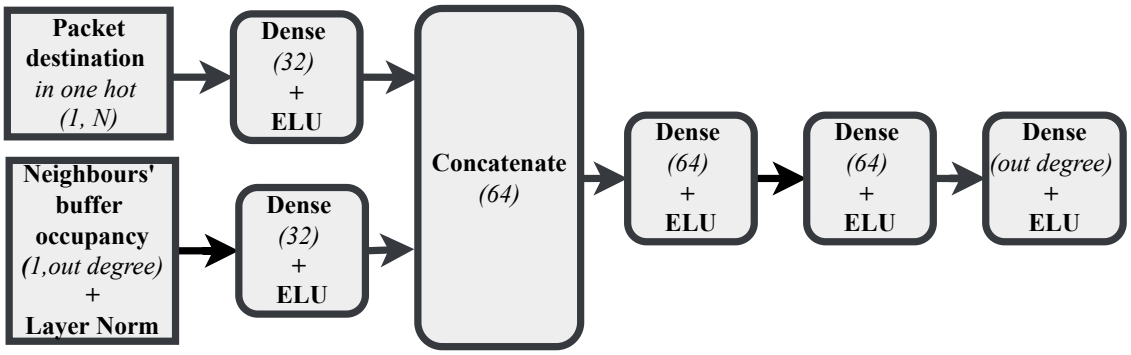

Fig. 1: NN layers architecture

Algorithm 1: DQN Routing training routine at a node n with value sharing (or with model sharing)

Input: Training session duration S; Gradient descent update period T; Target update period U; Weights $\theta_v^{SP}, v \in \{n \cup \mathcal{N}_n^{out}\}$
Output: Final routing model weights θ_n

Initialize experience replay memory $\mathcal{M}_n$;
Initialize $Q_n(\cdot)$ with weights $\theta_n \leftarrow \theta_n^{SP}$;
Initialize targets $\hat{Q}_v(\cdot), v \in \mathcal{N}_n^{out}$ with weights $\theta_v^- \leftarrow \theta_n^{SP}$;
while *current simulation timestamp $t < S$* **do**
  for *each arrival packet* **do**
    Observe the current node state o_n;
    Select action $a_n = n'$, where
    $n' = \begin{cases} \text{a random neig. } v \in \mathcal{N}_n^{out}, \text{with prob. } \epsilon \\ \text{argmax}_{v_n \in \mathcal{N}_n^{out}} Q_n(o_n, v_n), \text{ otherwise} \end{cases}$
    Forward packet p to next hop node n';
    Receive back from n' next hop estimate τ and reward r_n;
    Receive back from n' next hop state $o_{n'}$ and reward r_n;
    Set packet destination flag f;
    Store transition $(o_n, a_n, r_{n'}, \tau_{n'}, f)$ in memory $\mathcal{M}_n$;
    Store transition $(o_n, a_n, r_{n'}, o_{n'}, f)$ in memory $\mathcal{M}_n$;
    if $t \mod T$ **then**
      Sample random batch $\mathcal{B} \overset{i.i.d.}{\sim} \mathcal{M}_n$;
      Set target values Y_n^Q as (2) for $\mathcal{B}$;
      Update weights θ_n by gradient descent on TD error for $\mathcal{B}$;
    if $t \mod U$ **then**
      Get neighgbors' weights: $\theta_v, v \in \mathcal{N}_n^{out}$;
      Update target weights: $\theta_v^- \leftarrow \theta_v, v \in \mathcal{N}_n^{out}$;

the network, constraining the diversity of state transitions that the simulator permits to observe. This has two main consequences: (i) a reduced realism of the simulated communication process; and, (ii) an obstacle to fairly comparing these DRL proposals to state-of-the-art approaches or even against each other. This lack of standardized ML tools in the networking community [17] is becoming a major issue to guarantee reproducibility. As a consequence, in the last years, some tools [17], [18] have been developed to apply RL to networks. These tools allow `python` RL agents to interact with the popular `ns-3` network simulator [9]. Nevertheless, these proposals are not natively compatible with the *collaborative and distributed multi-agent* setting of the DPR problem. To bridge this gap, `PRISMA` has been proposed in the work [16].

`PRISMA` is an open-source RL simulation environment designed specifically for considering the distinctive characteristics of the DPR problem, serving as a playground where the community can easily validate their RL approaches and compare them. It introduces a more realistic modelling of the communication process based on: (i) the `ns-3` [9] network simulator; and, (ii) a multi-threaded implementation for each agent. It makes use of a modular code design, which allows researchers to test their own RL algorithms, without needing to work on the implementation of the environment. PRISMA also provides visualization tools based on `tensorboard` [19] allowing to track training and test phases. All these reasons make `PRISMA` better suited to evaluate the control signalling exchanges, since their evaluation requires: (i) a higher level of realism to correctly consider the extra delays and traffic introduced by control packets; (ii) a standardized tool where different control signalling techniques between agents can be easily implemented and compared.

IV. Simulation Experiments

In this section, we present the experimental approach to answer quantitatively the following research questions:

1) How does the cost and overhead of *value sharing* and *model sharing* strategies evolve when modulating the target update period U?
2) How much overhead is required to make the *model sharing* solution close to the Oracle cost solution?

A. Experiment Settings

Hardware and Software settings. We ran the tests in a Dell Precision 7920 workstation equipped with an Intel Xeon Gold 6230R Dual CPU (26 Cores, 2.1-4.0GHz Turbo, 128 GB RAM) with 2 NVIDIA RTX A5000 GPUs, running Linux Ubuntu 20.04. The DRL agent model is implemented in `TensorFlow` [20] version 2.8 as a described in III-D. The implementation of the MA-DRL method is based on the OpenAI™ `Baselines` library [21]. The `PRISMA` tool [10], [16] is used as RL simulation environment.

Competitors. We compare the routing optimality of the two versions of the *DQN routing* in Algorithm 1 (*value sharing* and *model sharing*) to the *Shortest Path routing* [22], and the *Oracle routing* from LP model (1). The *SP routing* and *Oracle routing* solutions represent performance benchmarks used to

assess the quality of the DRL solution. The *DQN routing* with *value sharing* is, in practice, an extension of the original DQN routing [3], where buffers' occupancy is added to packet destination in the node observation o_n. We cannot compare directly our *DQN routing* to the literature algorithms [4]–[6] since they are based on different assumptions. In *DQRC* [4], there is a unique buffer at each node, which constraints the outgoing node throughput, since, at each time, only one packet can be transferred from the buffer head to one of the forwarding ports. In the works [5], [6], the training is centralized in a node collecting all the past experiences, in contrast to us, where the training is done on the distributed node agents.

Evaluation metrics. To measure the performance of each routing policy, we use the *average cost per packet*, which is calculated as the accumulated reward along the simulation divided by the number of generated packets. We point out that during a network simulation, we can reward each packet arrival using the same reward definition as in Section III-C. Thus, the *average cost per packet* represents an average end-to-end delay per packet, where packet losses are also accounted as *worst-case* delays (see Section III-C).

Topology and traffic. Simulations are performed over the Abilene network ($|\mathcal{N}|$=11) [23]. We fix link propagation delay and link rate C to 1 ms and 500 $Kbps$, respectively.

Four traffic matrices $\boldsymbol{H}$ are generated by sampling each element h_{sd} from a uniform distribution $U(0,1)$. We scale up these matrices by multiplying by a coefficient α. We increase α up to the largest value α_{max} for which an optimal routing with no packet loss can still be found by the LP model (1). The matrix $\alpha_{max} \cdot \boldsymbol{H}$ is associated to a load factor $\rho = 1$. Data packets are generated as `UDP` over `IP` datagrams. Their payload is 512 B long. The `UDP` and `IP` headers are 8 B and 20 B long, respectively. Then, the total packet size is 540 B. Packet traces are produced assuming that packet *inter-arrival time* follows an exponential distribution with mean $540B/h_{sd}$. And, finally, the output buffer size B is fixed to $16,200$ B (i.e., 30 data packets of 540 B long).

Control signalling packets. As explained in Section III-E, DRL agents share information in the form of control signalling packets. For the *replay memory update packets*, we encode each float or integer data type unit in 8 B. Then, a pair $(r_{n'}, \tau_{n'})$ (value sharing) is 16 B long and a pair $(r_{n'}, o_{n'})$ (model sharing) is $16 + 8 \cdot |\mathcal{N}_n^{out}|$ B long. Control packets are also encapsulated into `UDP` over `IP` datagrams. For a *target update*, the size of a DRL agent model (as described in Section III-D) is around 36 KB, which we split in *target update packets* of 512 B long.

Training procedure. The training is performed in two phases: a *supervised pre-training phase* followed by the *main reinforcement learning phase*.

Supervised pre-training is done to improve the convergence of the DQN routing , as authors in [3] demonstrated. They pre-trained the Q-network $(Q_n(\cdot;\theta_n)$ using a dataset of tuples $\{d, L_n^{SP}(d,n')\}$, where d is the packet destination, n' is the next hop node and $L_n^{SP}(d,n')$ is the length of the shortest path between n and d, which contains the next hop node n'. The Q-network is trained till the square loss is minimized. The so-obtained weights are denoted as θ_n^{SP}.

Main reinforcement learning (see Section III-C). The weights θ_n are initialized with θ_n^{SP}. The model is trained using *ADAM* optimizer with a learning rate of 0.001, batch size of 512, and γ of 1. The total duration of the training session S is 1 $minute$ (in `ns-3` simulation time), which is sufficient to cancel the agents' TD error. The gradient descent is launched every $T = 10$ ms (in `ns-3` simulation time). Moreover, we use an ϵ-greedy approach (ϵ decays from 1 to 0.1) to trade between exploration and exploitation. We execute different *training sessions* for each traffic matrix, information sharing technique (*value sharing* and *model sharing*), *replay memory size* and *target update period*. We consider several values for the *replay memory size* (512, 1024, 2500, 5000, 10000, and 15000 experiences), and for the *target update period* U (from $1s$ to $9s$ with $1s$ step). Each session corresponds to a packet trace generated using a traffic matrix scaled to the load factor ρ of 0.4. After each training session, the corresponding model weights are saved.

Testing procedure. For each trained model, corresponding to the tuple {*traffic matrix, sharing technique, replay memory size, target update period*}, a test phase is performed as the following. We ran *nine* simulations for load factors ρ from 0.6 up to 1.4 with 0.1 step. The rationale behind testing for traffic loads higher than 1.0 is to evaluate the model in highly saturated scenarios, where buffer delays become huge. The test packet traces are generated using the same traffic matrix as training, but scaled to the corresponding load factor ρ. In other words, with respect to the training procedure, we test each DNN model with the same traffic distribution among nodes but with higher loads. The duration (in simulation time) of each *testing* simulation was 20 s, which is sufficient to reach a stationary state in the simulation.

B. Tradeoff between Overhead and Optimality

In this subsection, we evaluate the impact of the control signalling on the learning performance by studying the tradeoff between *average cost per packet* and *control overhead ratio*. The *control overhead ratio* is computed during the training, as the ratio between the total byte count of signalling packets and the total byte count of useful data packets. The results presented are computed as an average over the four traffic matrices, and the nine load factors.

Figures 2 and 3 show the *average cost per packet* versus the *replay memory size* and the *target update period* U, respectively. From their observation, first, we see that *DQN routing with value sharing* has a slightly better average cost than SP when choosing the experience replay memory size of 5000 samples. We recall that the original DQN routing algorithm presented in [3] uses value sharing. However, in [3], the routing decisions are taken based only on the packet destination, which yields routing policies close to the SP routing since features depending on traffic dynamics, as the buffers' occupancy, are not provided to the neural network.

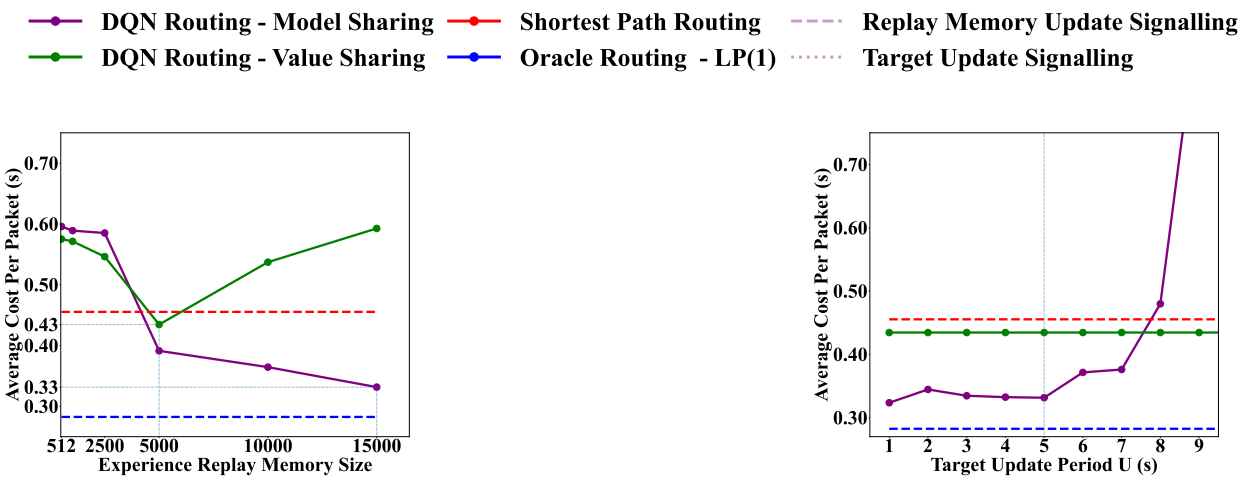

Fig. 2: Avg. Cost Per Packet vs Replay Memory Size

Fig. 3: Avg. Cost Per Packet vs Target Update Period

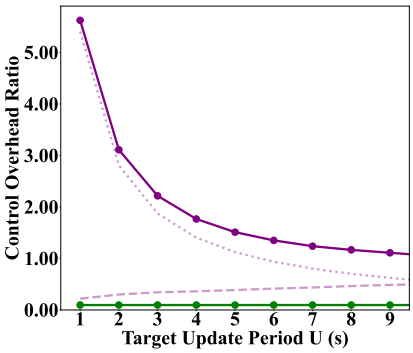

Fig. 4: Control Overhead vs Target Update Period

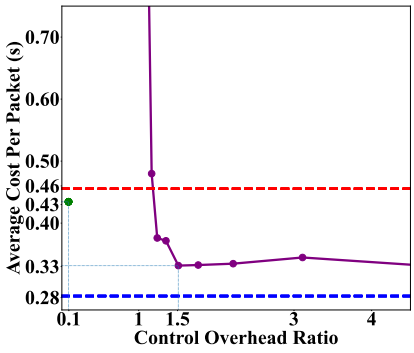

Fig. 5: Avg. Cost Per Packet vs Control Overhead

Interestingly, despite the fact that, in the current paper, we also add the buffers' occupancy to the neural network input, our enriched version of the original DQN routing in [3] does not manage to improve significantly over the SP policy, obtaining even worse results for any replay memory size value except 5000, which seems large enough to store the states' diversity, and small enough to not consider too many outdated experiences. In contrast, *model sharing* outperforms the SP routing and gets close to the Oracle routing cost when the replay memory is large enough (more than $5,000$ experiences) and the target update period U is small enough (less than $8s$). This result suggests that routing agents need to share their DNN models to be able to learn a routing policy close to an Oracle routing. Previous literature [15], [24] has shown the importance of conditioning the learning of each agent on an estimate of the other agents' policies to alleviate the *non-stationarity* effect of MA-DRL. Since other agents' policies are continuously updated over time during the training, their *Q-value* (and, then τ) estimates become obsolete with time. For value sharing, that means that the replay memory can be populated with outdated $\tau_{n'}$ conducting to have different estimates of the *end-to-end delay* from next hop n' to destination for the same state transition $(o_n, a_n, o_{n'})$. Model sharing overcomes this problem by sharing the model $Q_{n'}(\cdot, o_{n'}; \theta_{n'})$. Since $\tau_{n'}$ estimates are computed using Equation (3) at the gradient descent update, two identical transitions $(o_n, a_n, o_{n'})$ stored at the replay memory will be associated to the same estimate $\tau_{n'}$ of the *end-to-end delay* from n'.

Figure 4 depicts the relation between *target update period* U and the *control overhead ratio*. In model sharing strategy, the control overhead strongly decreases with U, since the number of *target update packets* diminishes as well. For value sharing, only the much lighter *replay memory update packets* are sent, giving a much smaller control overhead (0.1). This overhead is shown as constant since it corresponds to only one point (value sharing does not depend on U). Analyzing in detail the overhead for model sharing, we see two different behaviors for the signalling packets. The overhead due to the *target model update packets*, the main overhead source, decreases with the target update period, but the *replay memory update packets* overhead increases linearly. This indicates a growth in packet hops during the training, which is related to a worse training performance for larger U values: the agents do not achieve to find short routing policies (see Figure 3).

Figure 5 depicts the tradeoff between *average cost per packet* and *control overhead ratio* for model and value sharing

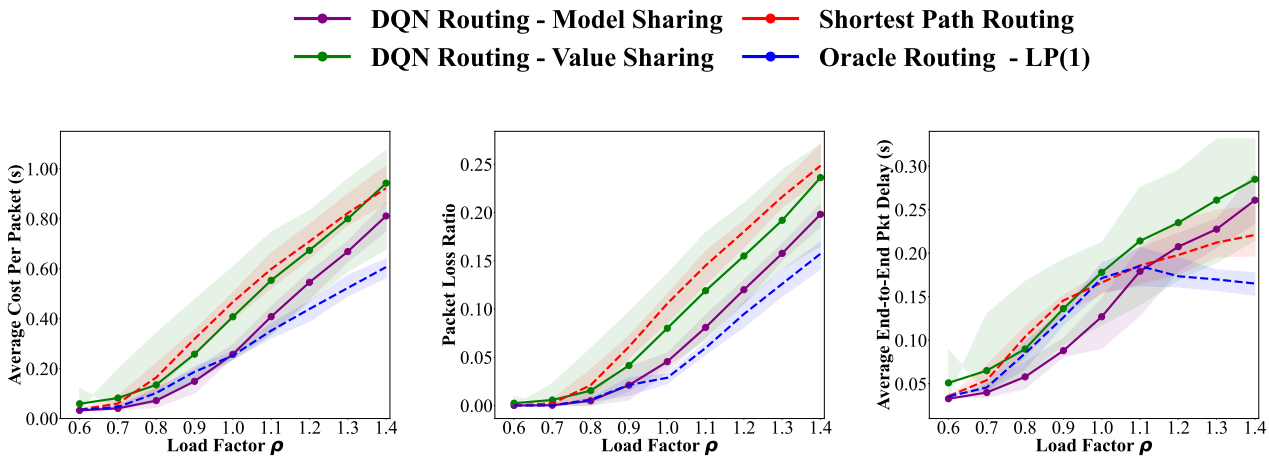

Fig. 6: Average Cost Per Packet.

Fig. 7: Packet Loss Ratio

Fig. 8: Avg. End-to-End Delay.

techniques. Replay memory sizes are set to the best values in Figure 2 for each sharing technique: $5,000$ and $15,000$ for value and model sharing, respectively. Each point in the curve for DQN routing with model sharing corresponds to a given target update period value U, which is associated to a given cost per packet (see Figure 3) and a given overhead (see Figure 4). For value sharing, we have a unique point corresponding to the cost per packet for the best memory size $(5,000)$ and a constant value sharing overhead (0.1). Two observations can be made from this figure: (i) value sharing incurs in a very small overhead (0.1) but cannot improve significantly the SP routing performance, and (ii) model sharing can outperform SP routing and approach the Oracle performance but with much higher overhead (for $U = 5s$, an overhead of 1.5 and a cost 28 % smaller than SP routing and model sharing). These observations provide an important insight: *for the presented settings, the price to pay to become close to the Oracle performance is a control overhead larger than the useful data volume during training*.

C. Packet Cost Performance in Detail

We now analyze the performance of DQN routing for both signalling techniques in terms of: (i) *average cost per packet* (Figure 6), (ii) *packet loss ratio* (Figure 7), and (iii) *average end to end packet delay* (Figure 8). The *x-axis* represents the load factor, ranging from 0.6 up to 1.4 with 0.1 step, and the *y-axis* represents the average, the minimum, and the maximum value of the corresponding performance metric over the four traffic matrices. A solid line is used to depict the average value, whereas the area between the minimum and the maximum value is presented with a shaded color. In these figures, the replay memory sizes are fixed to the best values found in the previous subsection: $5,000$ and $15,000$ for value and model sharing, respectively. Similarly, the target update period for model sharing is set up to the selected value of $5s$.

From the observation of Figure 6, we confirm the remarks stated in the previous subsection: in terms of average cost per packet, the model sharing is always better than value sharing and SP routing, and close to the Oracle routing performance. We recall that average cost per packet is computed as an accumulated reward along a simulation session. Then, it accounts for both buffer delays and packet losses, but giving more importance to the latter to minimize it in the first place (see Section III-C) as the LP model of the Oracle routing also does (see Section III-B). Consequently, Figure 7, that shows the packet loss performance, presents the same trends.

Figure 8 shows the average end-to-end packet delay for the different loads. Interestingly, DQN routing manages to outperform the Oracle routing for medium loads (from 0.7 to 1.1). This can be partly explained by a superior performance in terms of packet loss (if present, the main contributor to the average cost per packet) of the Oracle, namely for the loads 1.0 and 1.1, which leads the Oracle to have a poorer behavior in terms of delay. But, for the loads between 0.7 and 0.9, DQN routing outperforms the Oracle in delay and matches its packet loss, yielding a better average cost per packet in Figure 6. This better behavior of DQN routing can be explained by the flow-based nature of the LP model of the Oracle routing policy. This model has a coarse-grained view of the packets' flows where the notion of packets' buffers is absent: the LP model works with flows defined by an average value in traffic units (i.e., *bps*). On the contrary, the DQN routing has a fine-grained view of the flows, being aware of individual packets. The packets do not arrive at a regular and periodic basis, but randomly following an exponential distribution, giving rise to situations where the instantaneous value of flow in traffic units (i.e., *bps*) exceeds the average flow. In this case, buffers allow absorbing the temporary traffic peaks. Particularly, the packet's buffers can have a bigger impact on the network performance when the average traffic becomes close to the nominal network capacity (medium loads between 0.7 and 0.9), since buffers are mostly empty for lower loads, and saturated for higher loads. Therefore, the DQN routing can benefit from the observation of the buffers' occupancy in these medium loads to overcome the Oracle routing, which is unaware of the buffers.

V. Discussion

The results presented in this study, especially the ratio between the signalling packets and useful data packets, may change if we change the experimental settings, such as using a larger link rate. Doing so will increase the number of data packets per second and hence the number of *replay memory update packets*. The ratio between *replay memory update signalling* and useful data will remain the same, and so the overhead for *target value sharing*. However, when keeping the same *target update period*, the ratio between *target update signalling* and useful data will decrease, and so the gap in overhead between *model sharing* and *value sharing* will narrow, but the gradient step period T may need to be reduced to maintain the number of experiences per gradient step, and hence possibly have a smaller *target update period*. When changing network topology, the overhead, particularly the one due to *target update signalling*, will scale with the number of nodes, and so if we add more nodes in the network, the overhead will increase. The relationship between the overhead, the performances, and the topology settings will be studied in future works. To conclude, when changing the network configurations, DRL based methods needs to be re-evaluated in a simulation environment taking into account the overhead of control packets, in order to have a precise estimation of the net gains of such a MA-DRL routing policy.

VI. Conclusion

In this article, we have presented an experimental framework to investigate, for the first time to the best of our knowledge, the control signalling mechanisms of distributed learning-based packet routing. In this routing paradigm, node agents must forward packets using only local information, solving the so-called Distributed Packet Routing (DPR) problem. To learn their routing policies, they make use of the DQN routing, a Multi-Agent Deep Reinforcement Learning (MA-DRL) algorithm. This framework imposes the neighboring agents to collaborate during the learning by sharing estimates of the end-to-end delays. We have considered two main control exchange strategies, *value sharing* and *model sharing*, which we have evaluated in terms of routing cost combining packet delay and loss rate, each component, considering the impact of the overhead packets due to control. On the Abilene topology for various loads and traffic matrices, we have shown that *model sharing* yields routing policies able to approach the optimal one provided by an oracle within 18%, considering the additional control packets. These represent a control overhead of 150% of extra traffic in the considered network scenarios. Conversely, *value sharing* adds a very small control overhead (approximately 10 %), but it cannot learn a routing policy close to the Oracle one, only managing to slightly improve a Shortest Path (SP) routing. We finally provide a discussion on the impact of our results, detailing how the ratio of overhead would evolve when changing base link rates and topologies. From the present study, we conclude by underlying the importance of assessing rigorously the signalling entailed by multi-agent DRL packet routing to obtain a precise estimation of net gains considering traffic overhead due to signalling. After this experimental analysis, our future works will investigate how MA-DRL performance scales with changes of network parameters, in particular link rates, topology and load.

References

[1] Y. Bengio, *Learning deep architectures for AI*. Now Publishers, 2009.

[2] V. Mnih, K. Kavukcuoglu *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540.

[3] D. Mukhutdinov, A. Filchenkov *et al.*, "Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system," *Future Gen. Computer Systems*, vol. 94, pp. 587–600, 2019.

[4] X. You, X. Li *et al.*, "Toward packet routing with fully distributed multiagent deep reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–14, 2020.

[5] V. Manfredi, A. P. Wolfe *et al.*, "Relational deep reinforcement learning for routing in wireless networks," in *2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2021-06, pp. 159–168.

[6] L. Chen, B. Hu *et al.*, "Multiagent meta-reinforcement learning for adaptive multipath routing optimization," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2021.

[7] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE transactions on automatic control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[8] X. Chen, B. Li *et al.*, "Demonstration of distributed collaborative learning with end-to-end qot estimation in multi-domain elastic optical networks," *Optics express*, vol. 27, no. 24, pp. 35 700–35 709, 2019.

[9] nsnam, "Ns-3 documentation website," 11/05/22. [Online]. Available: https://www.nsnam.org/documentation/

[10] "Prisma tool: An open marl framework for packet routing." 11/05/22. [Online]. Available: https://github.com/rapariciopardo/PRISMA

[11] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Proc. Neural Information Processing Systems (NeurIPS)*, 1993, pp. 671–678.

[12] C. J. C. H. Watkins, "Learning from delayed rewards." Ph.D. dissertation, University of Cambridge, 1989.

[13] J. Schulman, F. Wolski *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[14] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proc. Intl. Conf. on Machine Learning (ICML)*, 1994, pp. 157–163.

[15] J. Foerster, N. Nardelli *et al.*, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. Intl. Conf. on Machine Learning (ICML)*, 2017, pp. 1146–1155.

[16] R. A. Alliche, T. Da Silva Barros *et al.*, "PRISMA: a packet routing simulator for Multi-Agent reinforcement learning," in *2022 IFIP Networking WKSHPS Network Intelligence)*, Catania, Italy, Jun. 2022.

[17] P. Gawłowicz and A. Zubow, "ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research," in *Proc. ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, November 2019.

[18] H. Yin, P. Liu *et al.*, "Ns3-ai: Fostering artificial intelligence algorithms for networking research," in *Proc. ACM 2020 Workshop on Ns-3 (WNS3)*, New York, NY, USA, 2020, p. 57–64.

[19] "Tensorboard: Tensorflow's visualization toolkit," 11/05/22. [Online]. Available: https://www.tensorflow.org/tensorboard

[20] "Tensorflow: An end-to-end open source machine learning platform," 11/05/22. [Online]. Available: https://www.tensorflow.org/

[21] "Openai baselines: high-quality implementations of reinforcement learning algorithms," 11/05/22. [Online]. Available: https://github.com/openai/baselines

[22] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[23] "Sndlib: Library of test instances for survivable fixed telecommunication network design." [Online]. Available: http://sndlib.zib.de

[24] R. Lowe, Y. Wu *et al.*, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Neural Information Processing Systems (NeurIPS)*, 2017.