QoS-aware Network Self-management Architecture based on DRL and SDN for remote areas

Juan Francisco Chafla Altamirano, Mohamd Amine Slimane, Hassan Hassan, Khalil Drira

QoS-aware Network Self-management Architecture based on DRL and SDN for remote areas

Juan Chafla Altamirano[1,2], Mohamd Amine Slimane[1,3], Hassan Hassan[1], and Khalil Drira[1]

[1]LAAS-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France
{jfchaflaal, maslimane, hhassan, khalil}@laas.fr

[2]Pontificia Universidad Católica del Ecuador
[3]Ecole Polytechnique de Tunisie

***Abstract*—Connectivity in remote areas remains an unsolved problem, especially in developing countries. An adequate communications infrastructure can provide important services (e.g., telemedicine, virtual education, etc.) to communities of these regions. However, the management of these networks is complex because, on the one hand, they experience highly variable environmental conditions that require the continuous intervention of a human operator and, on the other hand, most of them are deployed in inaccessible areas. Self-management is proposed as an alternative solution to the management problem for these networks, thus reducing human intervention and, conversely, operational costs. This paper shows a network self-management architecture based on the SDN paradigm and Deep Reinforcement Learning algorithms that can learn the network dynamics and make autonomous decisions to optimize the network performance and adapt to the changing conditions of the environment to meet the QoS demands of the different network services. The proposed architecture has been successfully implemented in a simulated environment and was tested using a case study of QoS-aware routing optimization in a rural scenario.**

***Index Terms*—SDN, QoS, Routing, Deep Reinforcement Learning, self-management**

I. Introduction

Providing connectivity and internet access to people living in remote/rural areas is a major issue that needs to be overcome globally because communication networks can offer them the opportunity to develop and improve their quality of life through services such as virtual education, telemedicine, e-commerce, etc. Connectivity projects, in these constrained scenarios, face several challenges including: i) Low population density and low purchasing power do not justify the business case, ii) High investment in infrastructure (e.g. towers and base stations) and alternative energy sources (e.g., generators, solar panels, etc.) increases capital expenditures (CAPEX), and iii) Complex network management increases operating expenses OPEX [18], since rural areas are often inaccessible and require specialized personnel. These factors reduce the return on investment (ROI) making the construction of rural networks even more complicated [16].

Despite these challenges, several efforts can be found in the literature that have focused on providing network architectures for rural areas in a cost-effective manner such as the ones presented in [18]; where it is possible to notice that several of these solutions are based on current wireless technologies such as 5G/4G, unmanned aerial vehicles (UAVs), stratospheric balloons, or the latest advances in satellite communications, and also on more traditional approaches such as WiMAX or WiLD [10] that can further reduce costs by using unlicensed frequencies and low-cost equipment.

There is not a single solution that fits all rural scenarios; on the contrary, it is necessary to study which technology or combination of technologies can solve the connectivity problem for each case, considering that cost reduction is one of the most important aspects.

An important part of the operational costs is related to the need for specialized human resources for network management. Network management includes tasks such as continuous review of network Key Performance Indicators (KPIs), network configuration and optimization to ensure that Quality of Service (QoS) levels meet the demand of several network services (i.e., telemedicine, virtual education, etc.). Our work aims to contribute to this matter by presenting and evaluating a network management architecture that uses state-of-the-art mechanisms to add self-management functionalities to the network to reduce human intervention and, therefore, reduce OPEX while guaranteeing the desired QoS levels. Self-management is a desired feature in modern networks, and both the scientific community and industry have developed novel paradigms to achieve it. One of them is the software-defined networking (SDN) paradigm which aims to decouple the data and the control plane of network devices. This allows the creation of a programmable network by instrumenting the control plane in the SDN controller (SDN-C) [14]. In addition, the SDN-C can collect centralized information to be used by optimization programs running over it through the northbound interfaces [3].

Although SDN can automate many of the repetitive management tasks that have traditionally been carried out by human operators, using programs, these do not adapt to changing network conditions (e.g., sudden changes in traffic loads, unexpected link failures, changes in the environment, etc.) typical of remote zones. Therefore, we consider including Artificial Intelligence (AI), specifically Deep Reinforcement Learning (DRL), since these are

techniques that learn optimal behaviors from their continuous interaction with the environment [1], adapting to unforeseen operational conditions, and handling the decision-making process autonomously.

Figure 1 shows the high-level diagram of our self-management network architecture. It can be seen that the SDN-C collects the metrics and the current state of the entire network (e.g., the state of interfaces, flow tables, delay, etc.), and delivers this information to the DRL agents distributed in the backhaul Network Elements (NEs). The DRL-agents analyze the collected data and plan the actions to be taken to meet high-level management objectives. For testing purposes, we have adapted our architecture to a case study of QoS-aware routing optimization in a rural scenario, where the DRL-agents find the optimal route for transmitting local flows through the backhaul network as is explained in *Our architecture* section.

The remainder of this article is organized as follows. First,

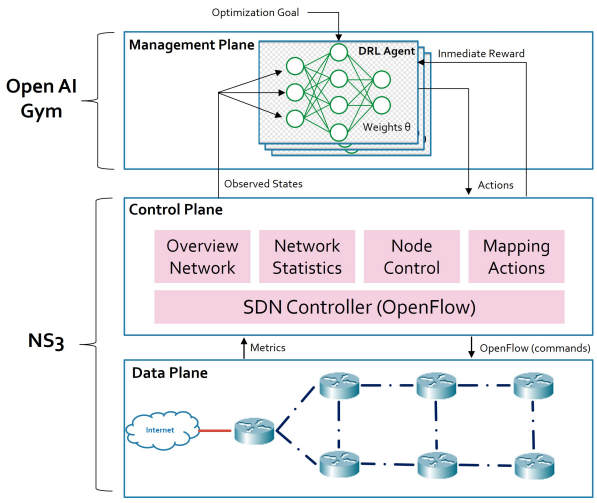


Fig. 1: High-level architecture of proposed traffic routing optimization solution

a review of the related articles is presented in Section II. Then, in Section III we present the proposed architecture in detail showing aspects such as the design of the DRL agents, the QoS-aware Routing optimization solution for the case study, among others. Section IV shows the evaluation of our architecture, here we describe the case study scenario, the simulation environment setup, and the main results. Finally, the conclusions and future work are presented in Section V.

II. Related Work

This section presents the most relevant proposals that integrate DRL and SDN algorithms for routing optimization like our case study. A summary of the main features of these papers in terms of the DRL algorithms used, the type of agent deployment (i.e., centralized or distributed), the observation space, the action space, and the metrics with which they calculate the reward of their neural networks can be seen in Table I.

In [13] the authors present a DRL-based routing optimization solution on top of the SDN controller (SDN-C). The DRL agent uses actor-critical, Deterministic Policy Gradient (DPG) algorithm to provide the SDN-C with the ability to adapt to changes in the environment by calculating proper routes that minimize the delay for all source-destination pairs. The observation space of the DRL agent is represented by a Traffic Matrix (TM) which is composed of the bandwidth requests between each source-destination pair, the actions are based on changing link weights and the average network delay defines the reward. OMNeT++ discrete event simulator was used to test the routing solution and showed that, after the training process, the agent can define near-optimal routing paths in a single step, making it more suitable for real networks compared to traditional approaches, such as OSFP, which needs several steps to converge. In [17] the authors presented a DRL-based control framework, called DRL-TE, to solve the TE (Traffic Engineering) problem (i.e., given several flows find a routing solution that maximizes the utility function). The authors propose two new techniques to optimize DRL for TE: i) TE-aware exploration which leverages a good TE solution as the baseline during exploration, and ii) Actor critic-based prioritized experience replay. The observation space of the DRL agent is composed of the throughput and delay of each communication session, the actions are based on traffic load split technique (i.e., specifies the amount of traffic load going through each of the paths) and the reward uses a α-fairness utility function which goal is to maximize the total utility (i.e., the throughput and delay) of all the communication sessions. They implemented DRL-TE in the NS-3 simulator and conducted a comprehensive simulation study to evaluate its performance in three network topologies: NSFNET, ARPANET, and a random topology, proving that DRL-TE improves the performance of certain traditional methods such as Shortest Path (SP), Load Balance (LB), etc. In [19] the authors state that routing optimization should be considered as a continuous actions problem, so they propose DROM; a centralized DRL agent for SDN that combines the DQN method with DPG (i.e., DDPG) in an actor-critical framework. The observation space of the DRL agent consists of a TM containing information about the current network load, the actions are based on the calculation of the link weights, which causes the SDN-C to generate and configure new routes for all considered flows. The reward is based on how well the new routes maximize throughput and reduce delay throughout the network. DROM was tested using OMNeT++, and the results were compared to traditional approaches such as OSPF, showing that it provides better routing configurations in terms of delay reduction and performance improvement. In [8] the authors propose a DRL routing optimization solution similar to that of [19], i.e., a centralized DRL agent for SDN that is based on neural networks of actors-critics and DDPG algorithm. The observation space is an aggregated traffic volume matrix (ATVM) which shows the traffic volume of each switch, and the actions are based on link weights calculation. The goal

TABLE I: Related Articles on DRL and SDN for Routing Optimization Use Cases

Ref.	DRL Algorithms	Agent Deployment	Observation	Actions	Reward Metrics
[13]	DQN/actor-critic/DPG	Centralized	Traffic Matrix	Link-weights	Mean network delay
[17]	DQN/actor-critic	Centralized	Throughput and delay	Traffic load split	Throughput and delay
[19]	DQN/actor-critic/DDPG	Centralized	Traffic Matrix	Link-weights	Throughput and delay
[8]	DQN/actor-critic	Centralized	Traffic Matrix	Link-weights	E2E delay and packet loss
[12]	DQN/actor-critic/CNN/DDPG	Centralized	Traffic Matrix	Link-weights	Mean latency, packet loss
[2]	DDQN	Centralized	Network topology, statistics	E2E path	Path cost
[5]	DQN/Multi-Agent	Distributed	Buffer size of neighbor routers	Next hop	Congestion, number of hops

of the reward is to optimize end-to-end delay and network packet loss. The novelty of this solution is the construction of a model based on the M/M/1/K queue for offline training of the DRL agent to avoid the long learning process of DRL in case of topology change, thus preventing degradation of the actual network performance. In [12] the authors propose a centralized QoS-aware routing decisions method based on actor-critic and DDPG framework for SDN networks. The DRL agent uses a TM as an observation space which provides the traffic demand of the flows in the network. Using this TM, the DRL agent calculates the actions and arranges them into a vector of link weights which defines the path that optimizes the latency and packet loss for each flow. The main contribution of this work is that it considers the influence that a flow has on others; this is possible thanks to a multi-layer convolutional module that learns the inter-flow impact when making routing decisions on network elements (e.g., switches, routers) shared by multiple flows. In [2], the authors propose a routing algorithm based on DRL and SDN called DRSIR. The centralized DRL uses Online and Target Neural Networks (NNs) for reducing estimation errors, and Experience Replay Memory to increase the pace of learning. The DQN algorithm has an observation space that is based on path-state metrics (i.e., path bandwidth, path delay, and path packet loss ratio). The action space is used to create a routing plan, i.e., a specific end-to-end path that connects the source and destination nodes and that is chosen from a list of all possible paths between that specific origin and destination. The DRSIR reward is calculated based on how well the agent avoids packet loss and delay by prioritizing routes that have more bandwidth available to avoid congestion. Compared to other solutions (e.g., RL-based routing solutions, Dijkstra's algorithm, etc.), the routes calculated by DRSIR are on average shorter and less congested, so the mean delay and losses are also lower. In [5] authors present a Deep Multi-Agent Reinforcement Learning (MARL) packet routing solution that can be deployed on different networks. They propose a distributed approach in which each network element has a DRL agent to perform routing decisions. The observation space is local to each router/agent and consists of the size of the packet and its destination, and the buffer capacity of the neighboring routers. The action space allows each agent to choose the most proper next-hop for the transmission of the packet. The packets are then transferred to the chosen routers and the agent receives the reward (i.e., how well that decision reduces the probability of congestion and the number of hops to the destination) and observation of the next hop routers. Tests showed that this distributed DRL agent model reduces the computational complexity compared to solutions using centralized agents, and that the agents can improve network performance and the probability of congestion under heavy traffic conditions.

In summary, most of the proposals (i.e., [13], [17], [19], [8], [12] and [2]) use a centralized architecture for intelligent routing management. In these proposals, a central DRL agent is used, together with the SDN-C, to compute the routing policy for the entire network. Obviously, this approach is not scalable because as the network grows, so does the complexity of the DRL agent, increasing the convergence time and the computational resources needed. In contrast, our proposal follows the distributed approach shown in [5], i.e., it is based on multiple DRL agents distributed in the network elements, each one making independent local decisions, which keeps their level of complexity low, making it suitable for large-scale networks. On the other hand, there is a tendency to use DRL techniques based on actor-critics and DDPG algorithms (i.e., [13], [17], [19], [8] and [12]) that use continuous action spaces which are difficult to implement in contrast to discrete action spaces. In our case, we use Deep Double Q-Network algorithm with discrete action spaces that eases the transformation of these actions into network configurations. Furthermore, it is worth noting that most of the proposals base their actions on calculating link weights (i.e., [13], [19], [8], and [12]) which makes them dependent on an underlying traditional routing protocol that takes those weights and reconfigures the routing tables (e.g., using OSPF). This approach does not have the best performance in terms of convergence time, since in addition to the link-weights computation time, the convergence time of the underlying protocol must be added. To overcome this problem, actions based on next-hop calculation [5] or on the choice of the optimal path from the group of all possible paths [2] have been proposed. Our approach does not rely on underlying routing protocols, or on generating a list of all possible paths with anticipation; rather, decisions are based on choosing the outgoing interface for packets on each network element; a simple but effective approach. Finally, it is common to see that DRL agents are provided with complex observation spaces in the form of a Traffic Matrix having specific information about the entire network environment [i.e., [13], [19], [8] and [12]] which in large-scale networks is difficult to handle; instead, our proposal uses a simple observation space that still allow

agents to find optimal routes and keep them computationally light (see Section *Design of the DRL agent*).

III. Our Architecture

The proposed architecture is shown in Fig. 1. The control plane contains the network elements that perform the basic function of forwarding packets based on instructions from the upper layers. The control plane, represented by the SDN Controller, handles the logic by which the data plane behaves. The SDN-C uses the OpenFlow protocol to send commands to the network elements and install in them the necessary flow tables to establish the appropriate policies that optimize network operation. Likewise, the SDN-C oversees the collection of network monitoring metrics necessary to evaluate the effectiveness of the optimization policies as well as the general state of the network. It should be noted that both the data and control plane have been implemented in the NS-3 environment. Finally, at the top layer, we have the Management Plane, where DRL agents work in a distributed fashion analyzing network metrics, and autonomously making decisions on network reconfiguration actions needed to meet high-level optimization objectives (e.g., guaranteeing QoS levels). In particular, for our case study, each distributed agent (i.e., one DRL agent for each community in Fig. 2 and for each class of traffic) receives the network monitoring metrics, in the form of observations, and through its artificial neural network structure, calculates the best routing policy for each traffic class, in the form of discrete actions that are transmitted to the SDN-C, which maps them to specific commands that configure/reconfigure the flow tables of the corresponding network elements. Once these actions are applied, the environment is checked again to evaluate the effect of the actions on the target metrics and provide feedback to each DRL agent in the form of a reward. If the optimization goals are not met, each agent adjusts its policy looking to increase rewards in the long term.

A. Design of the DRL agent

The proposed DRL Agents are based on the Double DQN (DDQN) algorithm, which uses two Neural Networks (NN) for simultaneous calculation and evaluation of the values of the actions (i.e., Q-values or $Q(s_t,\ a_t)$) through the loss function (i.e., the difference between the predicted and the actual value) [9]. Van Hasselt et al [15] proposed DDQN to solve the overestimation of action values and low performance typical of traditional Deep Q-Learning algorithms. Fig. 3 shows that the two NNs (i.e., the Main and Target NN) have the same structure and are used to improve the learning process and guarantee the stability of the algorithm (i.e., to avoid oscillations and divergences of the policy) [11]. The Main NN selects the a_t actions based on the ϵ-greedy strategy and its current parameters (i.e., the weights θ_1 of the NN), and the Target NN (with weights θ_2) calculate its own Q-values. The difference between the Q-values predicted by the Main and the Target NN is calculated by the Loss function; then backpropagation and stochastic gradient descent (SGD) algorithms are used to optimize this function and adjust the parameters of the Main NN. The Target NN parameters are copied from the Main NN and periodically updated every certain number of steps.

Each action a_t calculated for the current state s_t is applied to the environment and in return the next observation/state s_{t+1} and the immediate reward r_t are received. These four parameters form a tuple $\{s_t, a_t, r_t, s_{t+1}\}$ which is stored in the Reply Memory as an experience. When there is an adequate number of experiences, random mini-batches are taken from the Reply Memory to train the NN, thus minimizing the interactions of the DRL-agent with the environment [6] and accelerating its learning process.

B. DRL agent for Routing Optimization

During the training process, the DRL agents learn through direct interaction with the network environment by calculating actions and executing them according to the ϵ-greedy with linear decay policy mentioned before, which allows exploring the environment with random actions (i.e., an action a_t is randomly selected with probability ϵ) at the beginning and then exploiting the learning gained and supplying actions based on experience. The observation space (i.e., the current observable state s_t of each DRL agent) represents the state of the communication path set up for a traffic class/flow. This state is defined by a vector of the form S_t=$\{Fi,\ Rx,\ NE_{j=1},\ Pout_{j=1},\ \ldots,\ NE_{j=n},\ Pout_{j=n}\}$, where Fi is the flow identifier, Rx is a binary number whose value of one shows that packets are reaching their destination on the current path, NE_j is the identifier of the jth network element ($1 \leq j \leq$m; m is the total number of NEs), and $Pout_j$ is the port identifier of the jth NE port through which the packets of Fi are being forwarded. The action space (i.e., a_t) represents the set of possible actions that the agent can take given the state s_t, in our case each agent chooses the output port $P_k \in \{0,\ \ldots,\ k\}$, where k is the number of interfaces of each NE, through which the packets of the flow Fi should be sent. This action is computed for each NE thus establishing a path between source and destination. The reward r_t that each agent receives immediately after applying the actions is calculated based on the target QoS metric, in our case based on the average end-to-end delay for the flow Fi and the number of packets received (i.e., an indirect measure of packet loss). The monitoring module of our solution continuously measures the number of packets received at the destination for each Fi and the delay, and when a new path is defined by a DRL agent, the average E2E delay (D_t) is calculated with the equation: D_t=(D_A-D_L)/(R_A-R_L), where D_A is the cumulative delay of flow Fi packets, D_L is the last delay measured before changing the path, R_A is the number of cumulative packets received, and R_L is the number of packets received before changing the path. If D_t meets the target QoS levels (e.g., the delay for flow Fi is less than 10ms) the reward will be positive otherwise negative.

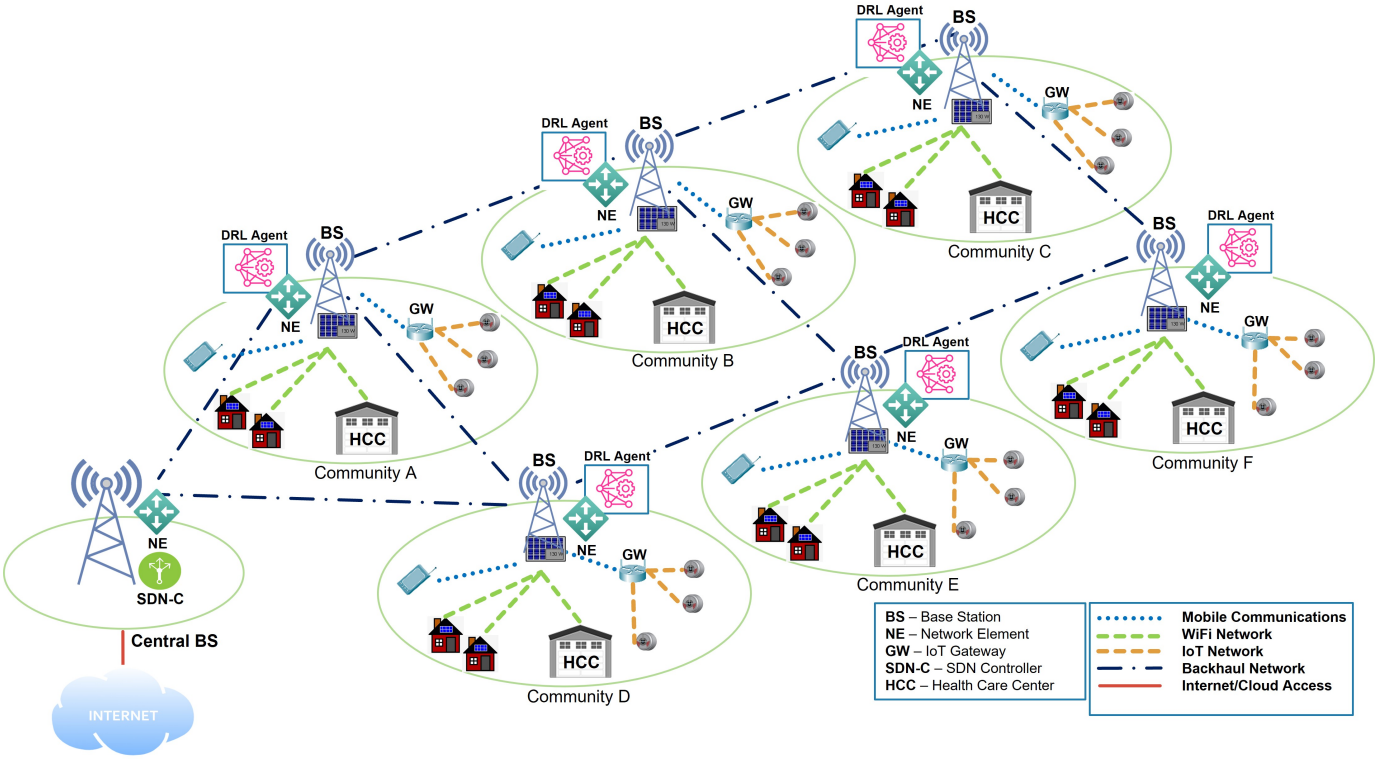


Fig. 2: Case Study Scenario: Multi-hop Topology Network for Rural Connectivity

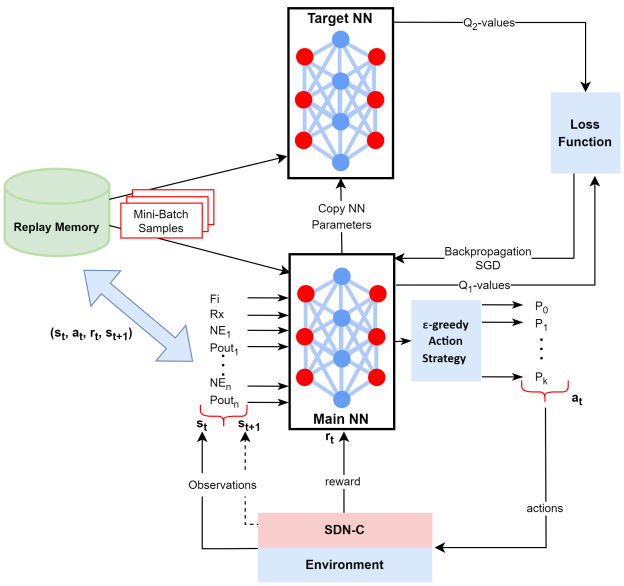


Fig. 3: Structure of the neural network used in the DDQN agent

C. Implementation framework

The framework used to implement our solution is ns3-gym [7], a middle-ware between NS3 (Network Simulator 3), a discrete network simulation tool, and OpenAI Gym, a Reinforcement Learning framework. In order to simulate SDN, we opted for a community module OpenFlow1.3 [4] that offers up to date implementation of the OpenFlow standard. OpenAI Gym aims to implement DRL agents to interact with independent environments (in our case NS3). To develop the agent, we used an additional PyTorch-based library called PFRL (version 0.3.0) that implements Deep Reinforcement Learning algorithms.

IV. Architecture Evaluation

This section shows the results of the evaluation of our QoS-aware Network Self-management Architecture adapted to our case study for routing optimization, in the scenario of the network for remote areas showed in Fig. 2. First, the case study scenario is described, and then the simulation environment setup. Finally, we show the performance of our architecture and evaluate its effectiveness not only in calculating the optimal routes for different classes of traffic considering their QoS objectives, but its ability to self-optimize the network configuration without human intervention.

A. Case Study Scenario

To test our routing optimization solution, we have considered the scenario shown in Fig. 2. The topology is inspired by earlier work on connectivity in rural areas of the Amazon region, such as the one presented in [10] that was implemented in Peru, which is described as a multi-hop topology for the backhaul network and has proven to be effective in reaching

very distant communities with minimal infrastructure investment. The backhaul network can be built over WiFi for Long Distance (WiLD), WiMAX or any other wireless technology that use a free license spectrum for costs savings. As can be seen, each community has a base station (BS) that provides local connectivity and at the same time serves as gateways for communication to the central BS that has an Internet connection. Each community is expected to have different network services (aka traffic classes), among the most important services we consider: telemedicine, remote monitoring of vital signs (i.e., eHealth), virtual education, among others. Each of these services demands different requirements from the network in terms of delay, bandwidth, packet loss, etc., and shows distinct levels of criticality as shown in Table II. The multi-hop topology brings congestion problems due to the limitation of resources (e.g., bandwidth, links, etc.), the confluence and aggregation of traffic especially in the links to the central BS, making it difficult to guarantee the QoS levels for all traffic and requiring permanent adjustments to the network every time there is congestion or environmental changes typical of the Amazon climate. Therefore, our proposal is to deploy DRL agents in a distributed manner in the network (i.e., one agent per community for each traffic class), which can calculate the best route for each class of traffic based on its criticality and needed QoS level. The SDN-C, which is in the Central BS, handles the implementation of the flow tables in each network element based on the route calculated by the agents locally.

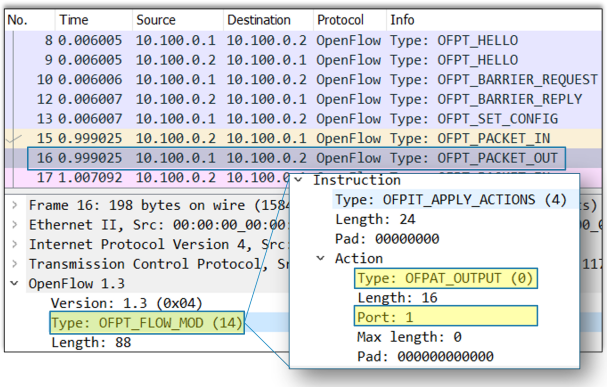


Fig. 4: OpenFlow message exchange in the Data Plane

B. Simulation environment Setup

The scenario presented earlier was implemented in the NS3 simulator with different traffic classes in each community (see Table II). The DDQN agent was implemented in the NS3-Gym framework and deployed in each community independently for handling each traffic class. The simulated network has seven network elements/nodes and nine links creating a partial-mesh topology. The bandwidth of each link is fixed on 50 Mbps, and the delay of each link varies between 0.5 ms and 10 ms, therefore, there are routes that show a high delay and others a lower one depending on the path defined and the confluence of more than one flow in the same link. A traditional FIFO (First In First Out) queuing technique is implemented in each NE interface. It should be noted that all NEs attempt to reach the Internet through the central BS. The SDN-C functionality and OpenFlow protocol have been integrated into the network environment to add the programmability feature of the NEs. The DPCTL (Data Parallel Control) commands contains the instructions to configure the flow tables in each NE and are generated by the SDN-C. The DPCTL commands are transported by OpenFlow to the NEs where they are finally converted into configurations. This command exchange can be seen in Fig 4 which shows a capture of the OpenFlow messages that the SDN-C sends to the NEs to reconfigure them after the DRL agent made the calculation of the best route. We also implemented a traffic management module to avoid loops, incoming packet management module, among others, as part of the SDN-C functionality. For traffic generation (i.e., to simulate different traffic classes) we used the NS-3 class "OnOffApplications" which emulates a traffic source of any type (e.g., TCP, UDP, etc.) and allows defining the bit rate, packet size, among other options. In our case we use several traffic sources with different rate settings depending on the type of traffic (e.g., 100Kbps for Best Effort, 512Kbps for Real-Time, etc.). Each DRL-agent manages two 3-layer neural networks (i.e., the Main and Target NN). The hidden layer has 50 connections to the first and last layer. The size of the first layer is flexible and depends on the size of the observation space, likewise the last layer depends on the action space. ReLU was used as an activation function. To reduce the error rates during the training of the agents, the Adam optimizer PyTorch was used with a value of $eps = 1e-2$ which provided better stability of the algorithm. The gamma discount function was set to 0.9 to give value to future rewards. The ϵ-greedy function was set to 1.0 (i.e., 100% chance of choosing a random action at the beginning), a final value of 0.0 with a decay of 0.99. Finally, the size of the Replay Memory was fixed with a capacity of 10E6 transitions. For the training of the DRL-agents, several episodes were run (e.g., up to 100 episodes for the first experiment, 20 episodes for the second one, etc.) and each episode with at least 300 interactions with the simulation environment.

C. Results

The first simulation results prove that the proposed architecture design, as well as the DRL agents converge to optimal routes as can be seen in Fig. 5. To test this, a first experiment was performed by generating one traffic flow from each NE of each Community in Fig. 2. In this case, the distributed DRL agents must find the best route for each flow trying to meet the QoS condition, e.g., that the delay experienced by the highest priority flow is no more than 10 ms. Fig. 5, shows the reward received by an agent vs the number of episodes. It can be seen that in the first episodes the agent does not converge to an optimal route, therefore the Agent's reward remains low, but there is also an exponential growth of the curve (i.e., the DRL-agent is learning quickly) until episode

TABLE II: Traffic Classes and QoS Demands

Class	Priority	Packet Loss Tolerance	Delay Tolerance	Bandwidth demands	Jitter Tolerance
eHealth	Very High	Low	Low	Low	Low
Telemedicine	Very High	Low	Low	High	Low
Virtual education	High	Low	Low	High	Low
Best Effort	Low	High	High	High	High

10 when it obtains the maximum reward proving that it has found the optimal route. Our architecture is not limited to the computation of the optimal route, but to the implementation of it in the network elements. In Fig. 4, it can be seen how the SDN-C, once it receives the actions calculated by the DRL-agent, converts them into DPCTL commands and reconfigures the network through OpenFlow setting the output port on each NE for the flow F_i.

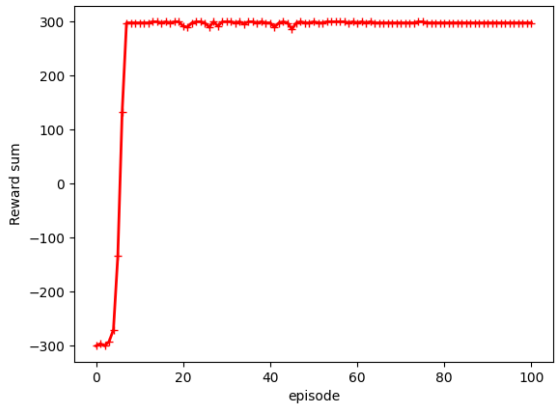


Fig. 5: Agent Reward Sum in the One Flow per Community Experiment

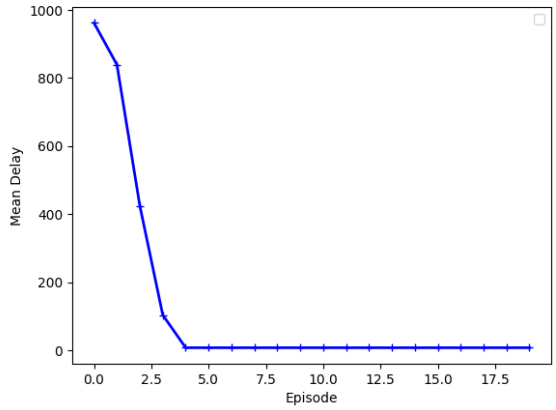
In a second experiment, we increase the complexity of the agent's task by implementing different traffic classes (i.e., real-time (RT), best effort (BE)) in two different communities. Each class contains two flows with the Internet as a destination (i.e., passing through the central BS). The BE flows belong to the lower priority class, so no QoS level is guaranteed for them and any route to reach the destination from any community is valid, but they can affect the higher priority classes. As for the RT class (e.g., telemedicine), the two flows generated, at 512kbps each, from Community B to central BS require the delay to be less than 10 ms. Two DRL agents are created; one to handle the RT class and the other for the BE. The first one must calculate a path where the sum of the delays of each link does not exceed the 10 ms delay limit, which is not a trivial decision. Fig. 6a and Fig. 6b show the behavior of the mean delay of the RT and BE classes, respectively, as the number of episodes increases. It can be seen, in the case of the agent for class RT (Fig. 6a), how the delay goes from very high values (i.e. above 100ms) to values that meet the QoS demands of the RT class (i.e. delay lower than 10 ms) from the fourth episode and remains stable until the end of the simulation. On the other hand, for the BE class the agent is able to find an optimal path that does not affect the RT class and achieves an acceptable delay for best effort flows above the 10ms threshold. In Fig. 6c and Fig. 6d, we can see the reward received by the RT and BE class agent respectively; it is possible to notice that for the latter the convergence is much faster starting with a value of 50 compared to -300 for the RT agent. This is because the BE class has no QoS constraints so it is simpler to find a route to transport these flows. In contrast, the RT class has a significant delay constraint (i.e., a maximum delay of 10ms), which forces the agent to try various combinations of routes until it finds the optimal one. In conclusion, although RT traffic had to be routed together with BE flows, both agents were able to converge to optimal routes demonstrating that our architecture is able to satisfy the QoS demands of priority flows while avoiding the adverse effect of less important traffic.

Finally, a third scenario was implemented to test the effect of having two agents working on two classes in the same NE (i.e., in the same Community). In this scenario, we added one more BE class in the first Community with two flows compared to the previous scenario. Again, our architecture proved to be effective, as the agents converged for all flows of all the classes, finding optimal routes even in the presence of several BE flows.

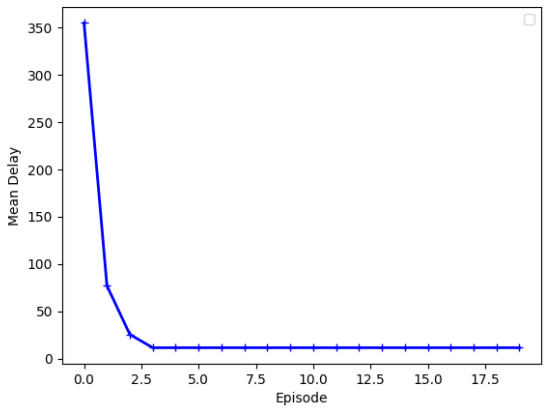
D. Discussion

The DRL agent model proposed in this paper is the best design we obtained, but it is not the only design we tested in our simulations. The first model we tried took as an input the "Flow ID", the "Network Element ID", the "Input Port" of the NE, the "Output Port" of the NE and the measured "Flow Delay" as a QoS parameter. It is notable to state that the configuration of the simulation was also different as the model was receiving the different possible combinations of the configuration of the NEs. This approach, although it yielded positive results, was discarded because it required the generation of all possible configuration combinations. Furthermore, this approach is not scalable because adding another community (i.e., another NE in the backhaul) to our network requires manual adjustments in the generation of the above mentioned combinations.

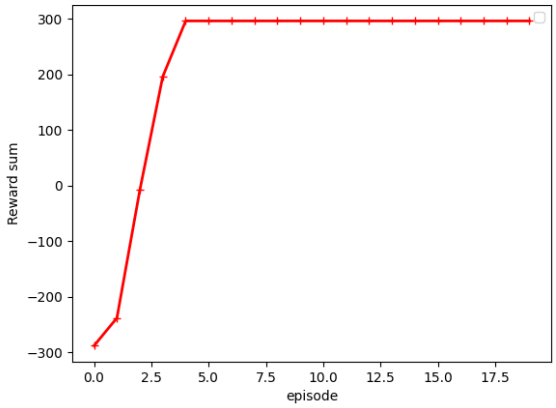
The second model we implemented followed a traditional scheme where the agent receives only the observation of the environment and takes actions accordingly. Even though this approach gave good results, it took a lot of time for the model to converge. By removing the "Input Port" as an input and



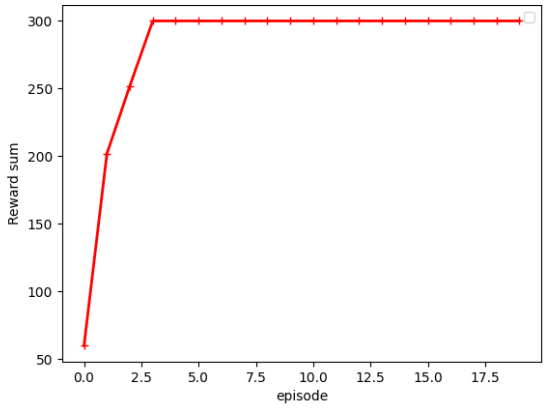
(a) Mean Delay for the Real-time Class



(b) Mean Delay for the Best Effort Class



(c) Reward Sum for the Real-Time Class Agent



(d) Reward Sum for the Best Effort Class Agent

Fig. 6: Results for the Second Experiment: Real-Time and Best-Effort classes

adding the received packet information instead, the speed of convergence of the model was improved. However, the agent showed signs of anomalies as the "Flow Delay" input swayed the model away from choosing optimal routes.
Essentially, in the first steps of the simulation, when the agent found valid routes with non optimal delays, it focused on those configurations rather than finding optimal routes. By removing the "Flow Delay" from the input, and using it only to attribute rewards to the agent after applying actions, we finally produced a viable agent that could find optimal routes that respect QoS levels. This model could be qualified as scalable as it can be used in any kind of topology regardless of the number of NEs and the QoS criteria of every class of traffic.

V. Conclusions

In this paper we presented a QoS-aware Network Self-management Architecture based on DRL agent and SDN with a special focus on remote areas. The architecture was tested in a routing optimization case study, where the DRL-agents act in a distributed manner without any prior knowledge of possible routes in the environment. Based on QoS demands, our solution proposes optimal routes for the flows of each traffic class. The agents adapt easily to the change of QoS constraints and flows number. In future work we would like to test the scalability of our architecture and extend the functionalities of DRL-agents to implement more advanced traffic engineering mechanisms such as Call Admission Control (CAC), queuing techniques, traffic shaping, etc., that allow not only optimization of traffic routes but also granular control of network resource consumption (i.e., bandwidth, queue space, etc.). Finally, the proposed architecture can be considered as a framework not only for solving the routing optimization problem, but also for implementing other AI algorithms that leverage the self-management approach and autonomously solve complex network problems, such as computational offloading, security, resource scheduling, etc.

Acknowledgment

The authors thank the FSPI - Doctoral Schools Project of the French Embassy in Ecuador, financed by the Ministry of Europe and Foreign Affairs.

REFERENCES

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Processing Magazine*, 34(6):26–38, aug 2017.

[2] Daniela M. Casas-Velasco, Oscar Mauricio Caicedo Rendon, and Nelson L.S. da Fonseca. DRSIR: A Deep Reinforcement Learning Approach for Routing in Software-Defined Networking. *IEEE Transactions on Network and Service Management*, 2021.

[3] Ranganai Chaparadza, Tayeb Ben Meriem, Benoit Radier, Szymon Szott, Michal Wodczak, Arun Prakash, Jianguo Ding, Said Soulhi, and Andrej Mihailovic. SDN enablers in the ETSI AFI GANA Reference Model for Autonomic Management & Control (emerging standard), and Virtualization impact. *2013 IEEE Globecom Workshops, GC Wkshps 2013*, pages 818–823, 2013.

[4] Luciano Jerez Chaves, Islene Calciolari Garcia, and Edmundo Roberto Mauro Madeira. Ofswitch13: Enhancing ns-3 with openflow 1.3 support. In *Proceedings of the Workshop on Ns-3*, WNS3 '16, page 33–40, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Ruijin Ding, Yuwen Yang, Jun Liu, Hongyan Li, and Feifei Gao. Packet Routing Against Network Congestion: A Deep Multi-agent Reinforcement Learning Approach. *2020 International Conference on Computing, Networking and Communications, ICNC 2020*, pages 932–937, feb 2020.

[6] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A Theoretical Analysis of Deep Q-Learning. jan 2019.

[7] Piotr Gawłowicz and Anatolij Zubow. Ns-3 meets openai gym: The playground for machine learning in networking research. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '19, page 113–120, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Gyungmin Kim, Yohan Kim, and Hyuk Lim. Deep Reinforcement Learning-Based Routing on Software-Defined Networks. *IEEE Access*, 10:18121–18133, 2022.

[9] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying Chang Liang, and Dong In Kim. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey, oct 2019.

[10] Andrés Martínez-fernández, Josep Vidal, Javier Simó-reigadas, Ignacio Prieto-egido, Adrián Agustín, Juan Paco, and Álvaro Rendón. The TU-CAN3G Project: Wireless Technologies in Developing Countries Based on 3G Small Cell Deployments. *IEEE Communications Magazine*, (July):36–43, 2016.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature 2015 518:7540*, 518(7540):529–533, feb 2015.

[12] Tran Anh Quang Pham, Yassine Hadjadj-Aoul, and Abdelkader Outtagarts. Deep Reinforcement Learning Based QoS-Aware Routing in Knowledge-Defined Networking. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 272:14–26, 2019.

[13] Giorgio Stampa, Marta Arias, David Sanchez-Charles, Victor Muntes-Mulero, and Albert Cabellos. A Deep-Reinforcement Learning Approach for Software-Defined Networking Routing Optimization. sep 2017.

[14] Arun Prakash Tayeb Ben Meriem, Ranganai Chaparadza, Benoît Radier, Said Soulhi, José-Antonio Lozano- López. *ETSI White Paper No. 16 GANA - Generic Autonomic Networking Architecture Reference Model for Autonomic Networking, Cognitive Networking and Self-Management of Networks and Services*. Number 16. 2016.

[15] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[16] World Telecom Labs. World Telecom Labs Launches Annual Survey About Rural Connectivity in Africa - WTL, 2018.

[17] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven Networking: A Deep Reinforcement Learning based Approach. *Proceedings - IEEE INFOCOM*, 2018-April:1871–1879, jan 2018.

[18] Elias Yaacoub and Mohamed Slim Alouini. A Key 6G Challenge and Opportunity - Connecting the Remaining 4 Billions: A Survey on Rural Connectivity. *arXiv*, pages 0–129, 2019.

[19] Changhe Yu, Julong Lan, Zehua Guo, and Yuxiang Hu. DROM: Optimizing the Routing in Software-Defined Networks with Deep Reinforcement Learning. *IEEE Access*, 6:64533–64539, 2018.