Routing optimization based on DRL and Generative Adversarial Networks for SDN environments

Juan Francisco Chafla Altamirano, Mariem Guitouni, Hassan Hassan, Khalil Drira

Routing optimization based on DRL and Generative Adversarial Networks for SDN environments

Juan Chafla Altamirano[1,2], Mariem Guitouni[1,3], Hassan Hassan[1], and Khalil Drira[1]

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

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

***Abstract*—Traditional routing protocols and analytical routing optimization models face limitations in adapting to dynamic and complex environments such as SDN. Deep Reinforcement Learning (DRL) offers promise for addressing these challenges, but its intensive training phase hinders practical implementation. This paper presents a distributed DRL-based routing optimization solution in SDN, enhanced with Generative Adversarial Networks (GAN) to expedite agent training. Our approach, evaluated on a Containernet and OpenAI Gym-based testbed, effectively optimizes network traffic routes for diverse traffic classes, maximizing throughput. Activation of the GAN module significantly reduces training times, enhancing the feasibility of our solution for real-world deployment.**

***Index Terms*—SDN, Routing Optimization, Deep Reinforcement Learning, Generative Adversarial Networks**

I. Introduction

In modern networks, traditional traffic routing optimization methods, reliant on analytical models, face challenges due to the networks' complexity and dynamism [1]. Deep reinforcement learning (DRL), a Machine Learning (ML) approach, has gained traction for effectively addressing these uncertainties without requiring analytical models [2], as it utilizes an exploration-exploitation strategy that can be leveraged for path optimization. The integration of software-defined networking (SDN) further enhances adaptability and control in network management eliminating the need for manual intervention [3]. Our prior research [4] demonstrated the efficacy of merging DRL with SDN, resulting in an autonomously managed network architecture. However, DRL's lengthy training times and the need for ongoing environment interaction pose complex challenges in production networks. To overcome these issues, this paper introduces a novel approach combining distributed DRL agents with a generative adversarial network (GAN) module in an SDN context. This combination leverages GAN's ability to generate synthetic observations, significantly reducing training durations and enhancing efficiency in traffic routing optimization.

The main contributions of this paper are:

1) The proposal of a network self-management architecture based on SDN, DRL, and GAN.
2) The proposal of a one-agent-per-traffic-class model, where each DRL agent associated with a traffic class determines its optimal packet forwarding logic.
3) The proposal of a single action approach for multiple nodes, where a DRL agent can define the optimal next hop of a flow on all network elements (NE) on the route in a single operation.
4) The proposal of using GAN during DRL agents' training to speed up convergence times and not interfere with network operation.
5) The evaluation of our proposal using a testbed deployed with Containernet and OpenAI Gym. We implemented a framework to integrate these two environments.

II. Related Work

In this section, we highlight key works on applying DRL techniques for SDN traffic routing optimization. In [5], a Multi-Plane Routing method with a centralized DDQN agent optimizes end-to-end delay and link utilization by focusing on per-link traffic load observations and finding the optimal routes between all sources and destinations. [6] introduces RL-Routing which requires one agent per switch to predict network behavior and optimize throughput and delay per flow; precomputed routes using shortest path algorithms reduce the action space. [7] proposes a DDPG-based solution that optimizes end-to-end delay and packet loss, and employs an M/M/1/K queue-based network model for offline training of the centralized DRL agent. [8] features a GAN-based transfer RL approach for adaptive routing with no need to retrain the DRL agent; GAN is incorporated into the transfer learning scheme for faster agent convergence. Conversely, our approach presents distributed DRL agents for each traffic class, implementing a single action strategy for multiple nodes. Our goal is throughput maximization and can be tailored to a variety of objectives. In addition, we leverage GAN to improve agent convergence from the beginning of agent training, eliminating the need for costly direct network metric measurements, as opposed to GAN's use of [8] to adapt pre-trained DRL agents.

III. Problem definition

A network, represented by $G(V, E)$, comprises nodes (S) and links (L), forming a connected graph. In this feed-forward network, nodes are not revisited to avoid loops. Each full-duplex link (l) has finite bandwidth (B), i.e., $B = b_1, b_2, \ldots, b_m$. Two neighboring switches (s_i and s_j) share a bidirectional link ($l_{i,j}$) with bandwidth $b(l_{i,j})$. The problem is to find the optimal path (subset of L) to transfer data packets for a given pair of edge switches, s_{in} and s_{out}. The goal is to maximize the throughput (th_i) for data flow between s_{in} and s_{out}, trying to achieve a target bandwidth (bw) by identifying the optimal packet forwarding sequence from $G(V, E)$.

IV. Architecture

The proposed self-management architecture (Figure 1) comprises four key components: Data plane, Control plane (SDN controller), Management plane (DRL agents), and GAN module. Details are as follows:

- Data plane forwards traffic flows, encompassing network elements, links, flows, and end users.
- Control plane, led by the SDN controller (SDN-C), centralizes important functions, collecting network data and commanding the data plane via OpenFlow.
- Management plane generates policy for packet forwarding based on the optimization objective, utilizing distributed DRL agents interacting with SDN-C.
- GAN Module accelerates DRL agents' learning by generating synthetic observations from real ones, reducing training times.

Data and control planes were implemented in Containernet [9], while the management plane used OpenAI Gym [10]. The architecture operates as follows: SDN-C collects network information and conveys observations to the control plane and GAN module. GAN generates synthetic observations for DRL agent training. The management plane organizes real and synthetic observations for DRL agent processing. DRL agents determine network actions based on current observations, conveyed to SDN-C, which translates and implements actions via OpenFlow. Network elements adapt their flow tables accordingly. Subsequent observations and associated rewards refine DRL agent actions against the objective.

A. DRL-based routing module

Our DRL agents employ the Double Deep Q-Learning (DDQN) algorithm, mitigating Q-value overestimation issues found in DQN [11]. The DRL agent's structure contains two neural networks (Main and Target NNs) interacting with environment E at discrete intervals (Δ_t). At each Δ_t, the Main NN selects an action (A_t) using the ϵ-greedy strategy and its Q-values, while the Target NN estimates the Q-value for the next state (S_{t+1}). The loss calculation relies on the difference between these Q-values, followed by parameter updates for the Main NN through backpropagation and stochastic gradient descent (SGD). Actions (A_t) applied to the current state (S_t) in environment E yield rewards

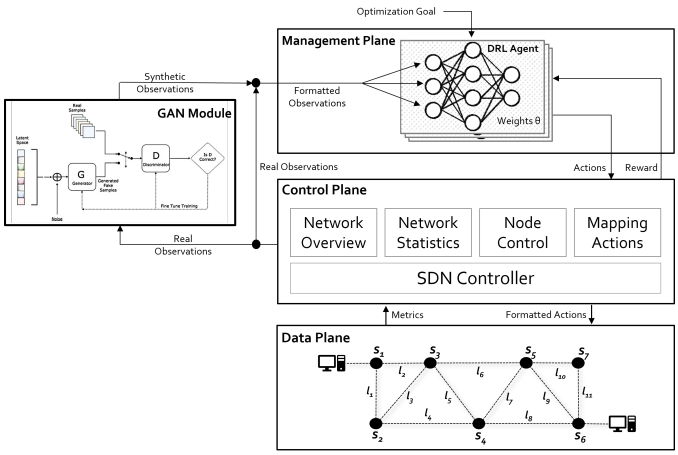


Fig. 1: High-level diagram of our architecture. Adapted from [4]

(R_{t+1}) and the next state (S_{t+1}). These parameters compose an experience tuple S_t, A_t, R_{t+1}, S_{t+1}, stored in Reply Memory. Training involves extracting random mini-batches from Reply Memory, reducing DRL-agent interactions with the environment [12], thereby expediting learning.

1) Observation, Action, and Reward Spaces: Observation, action, and reward spaces for our DRL agents are defined as:

- Observation Space (S_t): Represents the state of environment E as input data to each DRL agent. It is defined as a vector of vectors: $S_t = \{O_1, O_2, \ldots, O_n\}$, where $O_n = \{F_i, s_i, bw_i\}$. Here, F_i is the data flow identifier, s_i is the node identifier, and bw_i is the bandwidth, providing the status of all nodes associated with a specific flow or traffic class.
- Action Space (A_t): Defines the set of actions available to an agent at time Δ_t and state S_t. Our DRL agents' action space is defined as $A_t = \{a_1, a_2, \ldots, a_m\}$, where, each action $a_i = [pth_i | pth_i = \{l_1, l_2, ...l_m\}]$ (for $1 \leq i \leq m$) is a set of links that form a path connecting the ingress node s_{in} and egress node s_{out} for a given flow or traffic class.
- Reward (R): Evaluates the effectiveness of an action a_i in state S_t. It estimates how well pth_i aligns with the goal bw for flow f_i. We define R as:

$$R = \begin{cases} +r_1 & \text{if } th_i \geq bw \\ 0 & \text{if } 0 \leq th_i \leq bw \\ -r_1 & \text{if } th_i = 0 \end{cases} \quad (1)$$

  Here, r_1 represents the reward's numerical value (e.g., 10). When th_i exceeds the defined target bw, the reward is positive (+10). If th_i is less than bw but non-zero, indicating a valid but non-optimal path, the reward is zero. When th_i equals zero, signifying no throughput, the reward is negative (-10).

2) One-agent per traffic class, single-action approach: Our approach distinguishes itself by using a one-agent per traffic

class strategy to determine optimal routes based on throughput targets (i.e., a multi-flow joint optimization scheme). With this method, a single agent simultaneously dictates packet forwarding logic across multiple nodes. This agent selects nodes, determines the optimal next hop, and assigns the associated output port, establishing a forwarding route for a given flow/class in a single action. This approach significantly reduces the observation space, simplifying training and enhancing scalability since a small number of agents can handle all traffic classes and flows.

3) Implementation Framework: To implement our routing optimization solution, we've developed a framework that serves as an abstraction layer bridging the network environment emulator (Containernet) and the DRL agent (OpenAI Gym). This framework comprises three main components:

- GymEnv: This component creates a custom Gym environment by sub-classing the $gym.Env$ class. It is responsible for initializing the Gym environment and defining key aspects such as the observation space, action space, reward, and more. GymEnv acts as the interface between the network environment (Backend) and the DRL agent.
- Agent: Here, DRL agents are initialized, and parameters and hyperparameters for the neural network (NN) are defined. This includes characteristics like the type of NN (e.g., DDQN), the number of layers and neurons, the gamma factor, the Replay Memory size, etc. Importantly, GAN libraries are imported here for generating synthetic observations.
- Backend: This component creates the Containernet network environment and defines essential functions. Notable functions include network topology creation, SDN-C activation, flow activation in Containernet, network measurement collection, and action enforcement.

B. GAN module

An essential feature of our work is the integration of a Generative Adversarial Networks (GAN) module to expedite agent training. During experiments, we observed prolonged training times, primarily attributed to executing actions on network elements (NEs) and measuring throughput. To address this, we incorporated GAN, a deep learning-based generative model comprising two neural networks: the generator and discriminator. These networks engage in adversarial training, competing in a zero-sum game [13]. The generator produces synthetic samples from random noise input, while the discriminator discerns real from synthetic data. Ultimately, GAN learns the genuine data distribution, enabling the generator to produce indistinguishable synthetic data. GAN has demonstrated efficiency in various domains, such as image generation and natural language processing. Tabular GAN (TGAN), a model capable of synthesizing tabular data, was successfully tested in [14]. It follows the adversarial training principle but is specialized in generating structured data with diverse distributions. A pre-trained TGAN model is available on GitHub [15], simplifying its utilization. In our architecture, we deploy the TGAN module between the control and management planes (Figure 1). This module learns to generate synthetic training data in the format $\{S_t, R_{t+1}, S_{t+1}\}$. Consequently, TGAN produces synthetic observations S_t, corresponding rewards R_{t+1}, and subsequent observations S_{t+1}. Through experimentation, we found that the optimal results, in terms of time savings and valid sample generation, occur when we apply the rule that for every 10 real training data points, we generate 10 synthetic ones for DRL agent training.

V. Evaluation

A. Emulation environment

We employed Containernet [9], an open-source docker-based SDN network emulator, to test our solution. Containernet deploys NEs as Open vSwitch (OVS) and independent Docker hosts, customizable for emulating various applications and network services. Other components include Ryu SDN-C [16], OpenFlow 1.3, iPerf [17] for traffic generation and throughput measurement, OpenAI Gym [18] for DRL agent implementation, and TGAN [15] for GAN module.

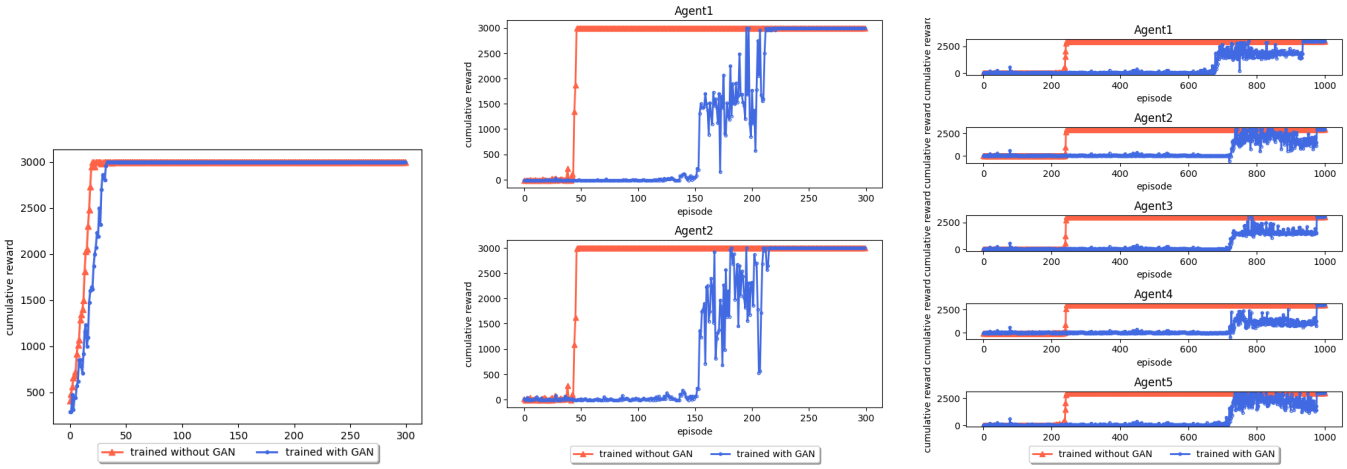
We used a generic network topology (Figure 1) with 7 nodes and 22 links, inspired by our previous work [4]. Each node represents users generating different flows of traffic. Link bandwidths $b(l_i)$ differ but never exceed $5Mbps$ to emulate congestion.

For each traffic class, we deployed a DDQN-based agent, composed of two 3-layer NNs, with 50 connections in the hidden layer. Adam optimizer with epsilon $1e-2$ enhanced training stability. ReLU activation was applied. The $\epsilon-greedy$ strategy decayed from 1.0 to 0.0 over 42.000 steps. Replay Memory had a fixed capacity of 10^9 transitions. We executed multiple episodes during DRL agent training, up to 1000 for various traffic classes, with at least 300 interactions per episode. The GAN module is used to generate 10 synthetic training data for every 10 real ones, reducing direct action and measurement time.

B. Results

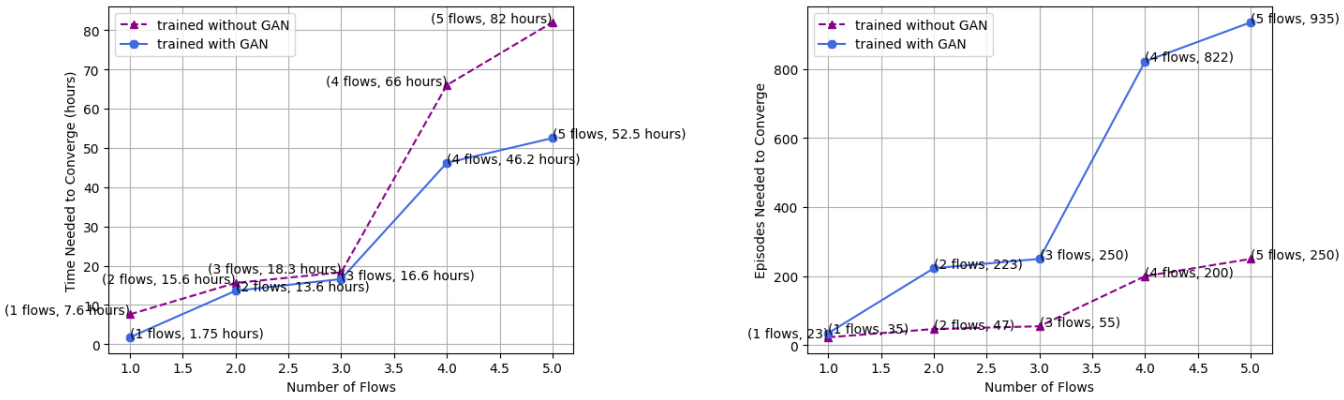
Two sets of experiments were conducted to evaluate our architecture: i) Without the GAN module, and ii) With the GAN module. The aim was to determine how synthetic training data impacts our solution performance, including convergence time and the number of episodes required for convergence. Scalability was also tested by increasing the number of traffic flows in the network.

We considered three key metrics: i) Cumulative reward, ii) Convergence time, and iii) Throughput. Cumulative reward represents the total reward received by an agent in an episode, intending to maximize it. Convergence time is the time taken by the agent to reach an optimal routing policy for a given traffic class. Throughput is the effective data transfer rate experienced by a flow, influencing reward calculation (Equation 1). In initial experiments, using the topology in Figure 1, a single traffic class was generated. Two tests were performed: one without the GAN module (red curve in Figure 2a) and one with the GAN module (blue curve). Both agents reach maximum reward, but the agent without GAN



(a) One class of traffic scenario (b) Two classes of traffic scenario (c) Five classes of traffic scenario

Fig. 2: Cumulative Reward for one, two and five classes of traffic with and without GAN



(a) Time needed for DRL agents to converge (b) Episodes needed for DRL agents to converge

Fig. 3: Time and episodes to converge curves with and without GAN

achieves it from episode 35. Despite initial appearances, GAN reduces convergence time, as seen in Figure 3a. Without GAN, convergence takes 7.6 hours, while with GAN, it only takes 1.75 hours. Subsequently, scalability was tested by introducing two traffic classes. Agents without GAN converge before episode 50 (Figure 2b), while agents with GAN take more episodes to reach significant rewards, stabilizing after episode 200. Despite the longer episode count, agents with GAN still achieve shorter convergence times (Figure 3a), with a difference of two hours. Similar patterns were observed when scaling up to 3, 4, and 5 traffic classes. Agents converged independently for all flows. The number of episodes needed for agents without GAN to converge (Figure 3b) is lower, but convergence time favors GAN. As the number of flows increases, this difference becomes more pronounced, with a nearly 30-hour gap for 5 traffic classes.

VI. Conclusions

We introduced a self-managing network architecture using distributed DRL agents and SDN technology. Each agent independently determines optimal routes for specific traffic classes across multiple network nodes in a single action, ensuring scalability with minimal agent deployment. To enhance training efficiency, we incorporate a GAN module to reduce direct network interactions for metric acquisition during agent training. We evaluated our architecture on a Containernet and OpenAI Gym-based testbed, creating a custom framework for seamless network integration with OpenAI Gym. Results demonstrate the feasibility of our self-management architecture as agents successfully optimize routes for various traffic classes, maximizing throughput. However, we observed extended training times, especially under high-traffic conditions. By activating the GAN module, training times significantly decreased, addressing this challenge as the network's traffic classes scaled up. In the future, we aim to test our framework in complex topologies with increased flows and precise traffic generators, incorporating additional optimization metrics like end-to-end delay and packet loss. Comparison with traditional routing protocols will also be carried out.

REFERENCES

[1] 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. *IEEE Communications Surveys Tutorials*, 21(4):3133–3174, 2019.

[2] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, 2018.

[3] Yichen Qian, Jun Wu, Rui Wang, Fusheng Zhu, and Wei Zhang. Survey on reinforcement learning applications in communication networks. *Journal of Communications and Information Networks*, 4(2):30–39, 2019.

[4] Juan Chafla Altamirano, Mohamd Amine Slimane, Hassan Hassan, and Khalil Drira. Qos-aware network self-management architecture based on drl and sdn for remote areas. In *2022 IEEE 11th IFIP International Conference on Performance Evaluation and Modeling in Wireless and Wired Networks (PEMWN)*, pages 1–6, 2022.

[5] Jiaqi Tang, Andrej Mihailovic, and Hamid Aghvami. Constructing a drl decision making scheme for multi-path routing in all-ip access network. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 3623–3628, 2022.

[6] Yi-Ren Chen, Amir Rezapour, Wen-Guey Tzeng, and Shi-Chun Tsai. Rl-routing: An sdn routing algorithm based on deep reinforcement learning. *IEEE Transactions on Network Science and Engineering*, 7(4):3185–3199, 2020.

[7] Gyungmin Kim, Yohan Kim, and Hyuk Lim. Deep reinforcement learning-based routing on software-defined networks. *IEEE Access*, 10:18121–18133, 2022.

[8] Tianjian Dong, Qi Qi, Jingyu Wang, Alex X. Liu, Haifeng Sun, Zirui Zhuang, and Jianxin Liao. Generative adversarial network-based transfer reinforcement learning for routing with prior knowledge. *IEEE Transactions on Network and Service Management*, 18(2):1673–1689, 2021.

[9] M. Peuster, H. Karl, and S. van Rossem. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 148–153, Nov 2016.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

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

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

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[14] Insaf Ashrapov. Tabular gans for uneven distribution, 2020.

[15] Insaf Ashrapov. Github repo tabular gans for uneven distribution, 2020.

[16] Ryu-SDN Org. Ryu sdn.

[17] Vivien GUEANT. iPerf - The TCP, UDP and SCTP network bandwidth measurement tool, 2013.

[18] OpenAI. Gymnasium documentation.