prisma-v2: Extension to Cloud Overlay Networks

Redha A. Alliche, Tiago da Silva Barros, Ramon Aparicio-Pardo, Lucile Sassatelli

prisma-v2: Extension to Cloud Overlay Networks

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 this paper, we present **prisma-v2**, a new release of **prisma**, a Packet Routing Simulator for Multi-Agent Reinforcement Learning. **prisma-v2** brings a new set of features. First, it allows simulating overlay network topologies, by integrating virtual links. Second, this release offers the possibility to simulate control packets, which allows to better evaluate the overhead of the network protocol. Last, we integrate the modules along with the core (`ns-3`) to a docker container, so that it can be run in any machine or platform. **prisma-v2** is, to the best of our knowledge, the first realistic overlay network simulation playground that offers to the community the possibility to test and evaluate new network protocols.

Keywords: ns-3, Multi-Agent, Packet Routing, Reinforcement Learning, Network Simulation, ML tool.

1. INTRODUCTION

Overlay networks are virtual or logical networks built on top of a physical network (called underlay networks). Overlay networks provide flexible and dynamic traffic routing between nodes that are not directly connected by physical links, but rather by virtual or logical links that correspond to paths in the underlying network. Those virtual links can be established using different technologies, like Generic Routing Encapsulation (GRE), Virtual Private Network (VPN) or network virtualization. The underlay topology is managed by a third party, typically one or more network operators. One particular example of overlay networks is Software-Defined Wide Area Network (SD-WAN) [1], which fully utilizes the bandwidth of all available transport networks serving one location, like Multiple Protocol Label Switching (MPLS) fabric, Internet and 5G, considering each one of them as an overlay link. In the context of overlay networks, the problem of routing the traffic between the overlay links, especially in muti-hop scenarios, becomes challenging, since the underlay routing policies are unknown and can involve different protocols, like Open Shortest Path First (OSPF), Border Gateway Protocol (BGP) and others. The absence of information about the underlay network topology and routing policies, yields the existence of Triangle Inequality Violation (TIV) [2]: it is highly possible to find another path relayed by cloud servers which has a much lower delay than following the shortest path in the overlay topology. There are classical routing protocols that can be used to route in overlay, like Cisco's Overlay Management Protocol (OMP), which is a control protocol developed by Cisco and working as BGP. This class of protocols highly depends on pre-defined metrics, and they do not handle multi-hop overlay network. To deal with the above limitations and challenges, a promising approach is using Machine Learning (ML) methods, especially Reinforcement Learning (RL) [3], which provides the ability for an agent (typically a network device) to learn from its environment, and to adapt it policy to meet the dynamically changing demand. In the context of overlay networks, the agent can exploit the information gathered from the network to overcome the lack of knowledge about the underlay network.

The Distributed Packet Routing (DPR) is the problem of finding optimal paths for packets, where each node decides locally which neighbor to forward a packet to. Multi-Agent Reinforcement Learning (MA-RL) is a suitable approach for addressing DPR. Each node can work as an agent that learns a local policy based on its observations and rewards, resulting in scalable and robust solutions. Moreover, MA-RL can leverage the advances of Deep Reinforcement Learning (DRL) techniques, which enables the agents to handle complex and high-dimensional state and action space. Several studies applied MA-RL and DRL to DPR in many situations, ranging from a general multi-hop routing [4], to specific application like mobile ad-hoc networks (MANets) [5]. These works can be extended to the case of overlay networks [6]. However, developing and testing MA-RL solutions for DPR requires a realistic and flexible simulation environment in which the researchers can evaluate the performance, and also the overhead of their solutions.

To support this research field, we made available the first version of **prisma** [7], which is an open-source realistic network simulation environment based on `ns-3` [8] and `openAI-Gym` [9]. This tool allows fast prototyping of Multi-Agent Deep Reinforcement Learning (MA-DRL) solutions, while assuring close to real-world behaviors. **prisma** is, along with other network simulation tools providing data driven based control, not adapted to the overlay scenarios. Given the challenging aspect of this context, we propose in this paper, a new release of **prisma**, namely **prisma-v2**, which offers the possibility to experiment MA-DRL approaches in the context of cloud overlay networks. The new features brought by **prisma-v2** are listed below:

- Overlay topology simulation and control management.
- Ability to add dynamic underlay traffic along with the overlay one.
- High reproducibility of results by supplying containerizing capability using `docker` [10].

- Refactoring the code for better readability.
- Improve `Tensorboard` [11] logging by incorporating both training and testing phases.
- Implement control packets to realistically simulate the communication and evaluate the overhead.

The **prisma-v2** code source is publicly available as **v0.2** release of **prisma** at [12].
The following of the paper will be structured as follows. First, we will talk briefly about the related work in terms of network simulating tools. Second, we will present **prisma-v2** features and code structure. Then, we will present how to install the tool and present some use cases.

2. Related works

One of the most popular network simulating tools is `ns-3` [8], which is a discrete-event simulator implemented in `C++`. It allows researchers to model and test different aspects of network configuration and protocols. However, `ns-3` does not support deep learning algorithms, since most of these algorithms rely on open-source `python` frameworks like `TensorFlow` and `PyTorch`. To enable the interaction between `ns-3` and the latter frameworks, many extensions have been proposed, such as `ns3-gym` [13] and `ns3-ai` [14]. The first one implements a socket communication channel between `ns-3` and `python`, while the second uses a shared memory. Another direction for extending `ns-3` is to focus on specific domains or applications of network systems, such as wireless and IoT networks. These domains pose new challenges and opportunities for network research. To address these challenges, some extensions have been developed such as `GrGym` [15] which is designed for radio communication, `MR-iNet Gym` [16] and `mobile-env` [17] which enable the use of DRL for wireless networks and wireless mobile networks, respectively. In this paper, we present `prisma-v2`, which is the first tool to our knowledge to bring DRL, especially MA-DRL, to cloud overlay networks.

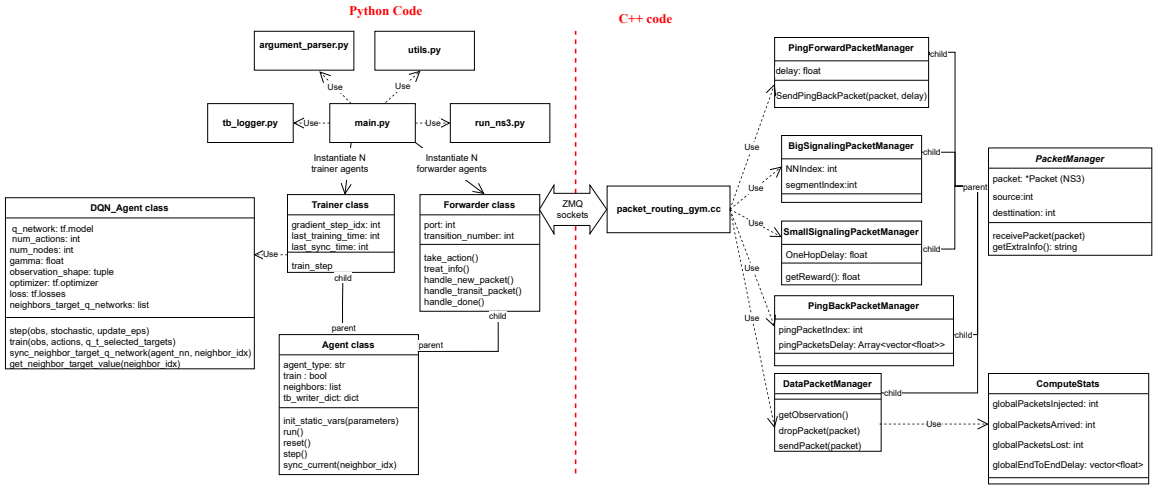

Figure 1: **prisma-v2** code structure

3. prisma-v2 Structure

The main features of this release are illustrated in the Fig. 1. **`ns-3` part**. The `prisma-v2` allows controlling the packet routing policy in overlay networks and, also simulating control signalling packets. We implemented three types of control signalling packets: the *big signalling packets* generated at a fixed time interval, and used to simulate the transfer of neural network weights between neighbors; the *small signalling packets* generated as a response to data packets, in order to transport the new observation and the reward; and the *ping packets* created according to a parameterized number of data packet transmissions and used to collect the overlay link's delay. For managing the data and control signalling packets, we developed the class `PacketManager`, which is inherited by five specific packet manager classes. The `SmallSignallingPacketManager` manages the arriving of small signalling packets. The class extracts the overlay link's delay measured during the sending of the data packet. The `BigSignallingPacketManager` manages the big signalling packet, extracting the information about the neural network weights shared by a node. The `PingForwardPacketManager` receives a ping packet sent, and calculate the overlay link's delay in the overlay scenario. The `PingBackPacketManager` receives the response of the ping packet containing the tunnel delay and stores it. The `DataPacketManager` manages the data packets. It collects the observation information to transmit to the agent. Furthermore, it sends the packet for the next hop based on the agent action. This class communicates with the `ComputeStats` class, which is responsible for the statistics of the network simulation, such as the number of injected packets by the node, the number of lost packets by the node and the end-to-end delay of the arrived packets.

python part. The left side of figure 1 presents the `python` code structure of **prisma-v2**. Like the first version, the program is launched by calling `main.py` with the simulation arguments. Packet Forwarder agent objects are instantiated from the `Forwarder` class stored in `forwarder.py`, while trainer daemon agent objects are instantiated from the `Trainer` class stored in `Trainer.py`. Both classes are children of the `Agent` class stored in `agent.py`. The `Agent` class implements the methods `run, reset, and step`, which will be overridden by the children's classes: For the `Trainer` class, the run method will launch the daemon main loop that checks if it is the time to train, and calls `step` method to run a training step; For the `Forwarder` class, the `run` method will start the agent episode and connects it to the network node in `ns-3`. It calls the `step` method to run a transition (waiting for a packet to arrive to the node, taking an action using the `take-action` method, and handling the information returned as a response to the action). The `reset` method of the `Forwarder` class will reset the agent environment before starting a new episode. Along with the agent's classes, we provide `DQN_Agent` class in `agent.py`, which provides the necessary methods to use an adaptation of Deep Q-Network (DQN) model for the DPR problem like in [4] and [18]. This class relies on the files `replay_buffer.py` and `models.py` for the experience replay buffer and the neural network architecture, respectively.

Tensorboard logger part. In **prisma-v2** release, the network metrics(end-to-end delay, packet loss, control overhead) are computed by the `ns-3` part and transmitted to the `python` for being displayed in the tensorboard logger. Moreover, the tensorboard displays in real time all the metrics to the user during the simulation.

We also improved the logger to automatically store in the same instance the train and the test stats. The logs are saved in the session folder, which is given by the argument `session_name`, respectively.

4. USAGE

In this section, we will go through different parts of the code, and explain the basic use cases that a user may encounter to launch a simulation.

4.1 Installation

After cloning the repository, the user is able to run **prisma-v2** by three different ways:

- Run locally by installing the required packages and dependencies by calling `install.sh`.
- Create a local docker image, by running the docker build command on the root folder. This is possible since we provide a docker definition file. This will copy all the folder in the image and so that user may run **prisma-v2** by calling the docker run command and binding the results' folder to be able to retrieve the results in the host machine.
- Pull a docker image provided in Docker Hub. This image only contains the Linux environment with the requirements installed, so the user need to bind the **prisma-v2** folder to the image.

We have provided an illustrative example in the `readme.md` file, guiding the user from the installation to training a Multi-Agent Deep Q-Network (MA-DQN) model to solve the DPR problem in an overlay network.

4.2 Use cases

1) Modifying or Adding parameters: Like the previous version, the parameters are organized by groups and can be visualized by calling "python3 main.py –help". They are accessible in the file `argument_parser.py`, where the user can add a new parameter or modify an existing one. In the find `run_ns3.py`, we parse the arguments to `ns-3`, so the user can modify this function to pass arguments to the NS-3 part.

2) Changing the DRL algorithm: The state can be modified for in `GetObservation` method of the `DataPacketManager`. The user may modify the DRL model input shape to match the new observation shape. Pre-defined neural networks models are already implemented in `models.py`, and a DRL algorithm is implemented in the `DQN_Agent` class. The user may change the latter class to change the learning algorithm.

3) Sharing the information between the agents: The agent forwarder and trainer objects share information with the main process using the Agent class static variables. In those variables, we can find agents which stores the `DQN_Agent` objects, and `pkt_tracking_dict` to keep track of transiting packets in the network.

4) Changing the topology settings: To change the overlay topology configuration, the user may change some parameters: `physical_adjacency_matrix_path`, `overlay_adjacency_matrix_path` and `map_overlay_path` for the paths to the underlay topology's adjacency matrix file, the overlay topology's adjacency matrix file, and the map between the indices of overlay and underlay nodes file, respectively; The traffic in underlay topology is handled, by default, using the OSPF protocol, which is used for computing the routing tables. Changing the routing policy is possible by the native `ns-3` method `RecomputeRoutingTables`.

5) Customize control packets: For customizing the control packets, the user may create a new class that inherits the class `PacketManager`. For example, The `receivePacket` method recovers the packet information at its arriving and the method `GetInfo` encapsulate the useful collected information in order to send it to the agent. For generating control signalling packets, the user may be inspired by the methods `sendSmallSignalingPacket` and `sendPingForwardPacket`, providing the packet size and the information which the control signalling packet should encapsulate. For the `bigSignallingPacket`, the parameters `syncStep` and `bigSignalingSize` contain the period time for sharing the model weights and weghts total size, respectively. For creating new control signalling mechanisms, the user may be inspired in the current mechanisms developed in **prisma-v2**.

5. CONCLUSION

In this paper, we have presented **prisma-v2**, which is a new release of **prisma**, extending this tool to overlay networks. This version adds a new set of features, like offering the possibility to add control signalling packet and measure the impact of having communication between the agent, and so, evaluate the real cost of implementing MA-DRL solutions for DPR in overlay networks. We hope that this initiative will motivate future research works tackling this challenging problem or adapting the tool to new problems involving decentralized solutions.

ACKNOWLEDGMENTS

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).

REFERENCES

[1] Z. Yang *et al.*, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019.

[2] C. Lumezanu *et al.*, "Triangle inequality and routing policy violations in the internet," in *Passive and Active Network Measurement: 10th International Conference, PAM 2009, Seoul, Korea, April 1-3, 2009. Proceedings 10*. Springer, 2009, pp. 45–54.

[3] R. S. Sutton *et al.*, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] X. You *et al.*, "Toward Packet Routing With Fully Distributed Multiagent Deep Reinforcement Learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 2, pp. 855–868, Feb. 2022, conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems.

[5] S. Kaviani *et al.*, "Robust and scalable routing with multi-agent deep reinforcement learning for manets," *arXiv preprint arXiv:2101.03273*, 2021.

[6] O. Houidi *et al.*, "Amac: Attention-based multi-agent cooperation for smart load balancing," in *2023 IEEE/IFIP Network Operations and Management Symposium (NOMS 2023)*, 2023.

[7] R. A. Alliche *et al.*, "Prisma: a packet routing simulator for multi-agent reinforcement learning," in *2022 IFIP Networking Conference (IFIP Networking)*. IEEE, 2022, pp. 1–6.

[8] nsnam, "Ns-3 documentation website." [Online]. Available: https://www.nsnam.org/documentation/

[9] G. Brockman *et al.*, "OpenAI Gym," *arXiv:1606.01540 [cs]*, Jun. 2016, arXiv: 1606.01540.

[10] "Docker main page website." [Online]. Available: https://www.docker.com/

[11] "Tensorboard visualization toolkit." [Online]. Available: https://www.tensorflow.org/tensorboard

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

[13] P. Gawłowicz *et al.*, "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.

[14] H. Yin *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.

[15] A. Zubow *et al.*, "Grgym: When gnu radio goes to (ai) gym," in *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, 2021, pp. 8–14.

[16] C. Farquhar *et al.*, "Marconi-rosenblatt framework for intelligent networks (mr-inet gym): For rapid design and implementation of distributed multi-agent reinforcement learning solutions for wireless networks," *Computer Networks*, vol. 222, p. 109489, 2023.

[17] S. Schneider *et al.*, "mobile-env: An open platform for reinforcement learning in wireless mobile networks," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–3.

[18] R. A. Alliche *et al.*, "Impact evaluation of control signalling onto distributed learning-based packet routing," in *34th Intl. Teletraffic Congress, ITC 2022*, 2022.