P4-based In-Network RL inference for Efficient Flow-Level Bandwidth Allocation

Arslan Qadeer, (*Member, IEEE*)
aqadeer000@citymail.cuny.edu
Department of Electrical Engineering, The City College of New York of CUNY
New York, NY, USA

Myung J. Lee, (*Life Member, IEEE*)
mlee@ccny.cuny.edu
Department of Electrical Engineering, The City College of New York of CUNY
New York, NY, USA

Daiki Nobayashi
nova@ecs.kyutech.ac.jp
Department of Electrical and Electronics Engineering, Kyushu Institute of Technology, Japan

GLOBECOM 2023 - 2023 IEEE Global Communications Conference | 979-8-3503-1090-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/GLOBECOM54140.2023.10437488

***Abstract*—Software-defined networking (SDN) has notably improved networks by providing Machine Learning-Powered programming capabilities at the control plane (CP), making it easier to dynamically manage the network resources according to varying traffic conditions. However, the geographically remote location of the CP from the data plane (DP) leads to significant round-trip delays in the order of milliseconds, which can adversely impact the performance of delay-sensitive and real-time traffic. To address this issue, this paper proposes a novel in-network reinforcement learning (RL) inference framework that extends programming capability from the CP to the DP for fine-grained control of network resources to meet the Quality of Service (QoS) demands of real-time applications. The in-network RL inference is achieved by adopting a match-action table mapping strategy in the DP and validating it through programming protocol-independent packet processors (P4). A P4 meter extern is utilized to allocate bandwidth to individual traffic flows based on their QoS requirements. Our proposed strategy achieves in-network RL inference at the line rate with negligible processing overhead while reducing packet loss rate and jitter by up to 92% and 57%, respectively, compared to the CP-based approach. Additionally, we evaluate the performance of our proposed bandwidth allocation framework using a state-of-the-art deep-deterministic policy gradient (DDPG)-based RL agent with a heuristic priority experience replay (hPER) technique. Our proposed DDPG agent achieves a faster convergence rate, higher reward, superior training stability, and up to 56% reduction in operational cost compared to two alternative agents.**

***Index Terms*—Software Defined Networking, Data plane programming, In-network RL inference, DDPG, Bandwidth Allocation, Network Resource Management, P4, B5G/6G**

I. Introduction

The advent of next-generation applications such as augmented reality (AR), virtual reality (VR), and extended reality (XR) has paved the way for novel use cases including immersive and semantic communication, remote surgeries, and tactile internet, among others [1]. The successful deployment of these emerging applications heavily relies on intelligent and dynamic network infrastructure capable of supporting various functions with diverse performance requirements. In particular, healthcare applications demand ultra-reliable low-latency communication (URLLC) capabilities for risk-free operations [2]. In this paper, we investigate the design and implementation of a network architecture that can meet the stringent performance demands of emerging AR, VR, and XR applications while providing flexibility, scalability, and reliability.

According to the 6G wireless network architecture, the Radio Access Network (RAN) is connected to the edge-cloud (EC), public data network (PDN), and other centralized control-plane (CP) services with the help of a wired network (the core network) and the user-plane function (UPF), also referred to as the data-plane [3]. To achieve the full potential of URLLC services, it is imperative to implement efficient resource management practices to support an unprecedented number of connected devices and applications in 6G networks. Resource management should be applied not only to the wireless resources but also to the wired devices in the core network, specifically to data-plane (DP) switches, to ensure seamless and uninterrupted communication. As such, the network resource management problem can be addressed by dynamically allocating bandwidth to the traffic flows. To be specific, resources are allocated at the flow-level to address the bandwidth allocation problem. Adopting a flow-based approach to resource allocation enables fine-grained control of the traffic coming from various applications, resulting in enhanced network performance.

Protocol-independent switching architecture (PISA) has emerged as a promising architecture for introducing data plane (DP) programmability in network devices [4]. By leveraging programming protocol-independent packet processors (P4), which is an open-source domain-specific language, PISA enables efficient implementation of advanced network functions in the DP. With the advent of 6G networks, the trend towards achieving greater reusability and flexibility through programmability is anticipated to gain even more momentum [5]. However, the implementation of ML-powered programmability in the DP is hindered by some challenges due to the unavailability of necessary operations in the switch ASIC, such as floating point computations and loops, which are needed for the ML-inference [6], [7]. These challenges underscore the need for novel approaches that can address the limitations of existing hardware platforms and enable efficient implementation of ML in the DP.

Many existing works have proposed in-network ML inference to address conventional routing, anomaly detection, and DDoS attack detection problems using already developed supervised ML models that rely on labeled training data [7]–[13]. However, the future 6G networks will be dynamic, and static training data is not suitable specifically for non-stationary and large-scale environments. Reinforcement learning (RL) is a promising approach for solving complex control problems in dynamic environments where no training data is

available. A recent study proposed an RL-based technique for quickly learning the requirements of individual traffic flows and procuring resources dynamically in a programmable network [14]. However, the training and inference of the RL agent are currently carried out in a centralized CP, which takes round-trip times in the order of milliseconds from the DP to the CP. A novel approach is needed that can implement RL-inference in the DP to ensure the efficient allocation of resources at the line rate to meet the QoS demands and real-time constraints of next-generation mobile and commercial applications in 6G networks.

Various techniques, including binary neural networks (BNNs), modified hardware designs, and match-action table (MAT) mapping-based methods [7], [8], [11], [12] can be utilized to implement in-network RL inference. BNN-based implementation requires at least 12 MATs to represent one layer in the DP [12], which can grow exponentially to keep the accuracy level intact. On the other hand, modified hardware design-based methods can be validated only using FPGAs, which is a complex and costly approach and requires redesigning the existing hardware. However, MAT mapping is a well-suited technique for P4, given its ability to be incorporated into existing hardware. The MAT mapping can support a variety of different forwarding behaviors, which can be rapidly modified by the CP to adapt to non-stationary environments. Additionally, the forwarding process in the PISA architecture enables the sharing of common instructions on MATs, leading to more efficient memory utilization [4] in the DP.

In this study, we aim to undertake the development of an in-network resource allocation framework by utilizing machine learning (ML). Our emphasis is on the execution of reinforcement-learning (RL) inference in the data plane (DP) for efficient flow-level bandwidth allocation at the line rate, ultimately achieving the ultra-reliable and low-latency communication (URLLC) goal in the 6G networks. A deep-deterministic policy gradient (DDPG) based RL agent provides superior state representation for continuous control problems [15], which is better suited to large-scale and dynamic 6G environments. We leverage our state-of-the-art DDPG agent [16] to facilitate a quicker convergence to optimal policies and promote robustness to environmental changes. Our proposed framework successfully implements in-network RL inference with negligible processing time overhead while reducing packet loss rate, jitter, and operational costs for bandwidth allocation. To the best of our knowledge, this is the first work to propose in-network RL inference for flow-level bandwidth allocation using P4.

The main contribution of our work is summarized as follows:

- We present a simple traffic and bandwidth allocation model that takes into consideration the QoS demands of the individual flows. Resource allocation based on individual flows allows for fine-grained control of the traffic from any application and improves overall network performance.
- We formulate the bandwidth allocation problem into a DDPG-based actor-critic framework. The reward maximization objective for resource allocation considers minimizing the bandwidth allocation cost.

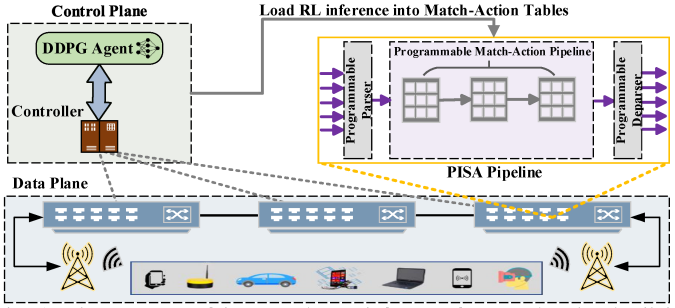

Fig. 1: The structure of in-network RL inference framework. The detail is described in Section III.

- We present a unique procedure to implement RL inference in the DP. The unique state-action design of our bandwidth allocation system is analogous to the match-action table mapping in the PISA pipeline.
- The effectiveness of our proposed method is evaluated in terms of per-packet processing time (negligible overhead), packet loss rate, and jitter. Additionally, our proposed method outperforms two other DDPG-based agents in convergence rate, reward, training stability, and operational cost for bandwidth allocation.

The remainder of this paper is organized as follows: The traffic model and bandwidth allocation model in the DP are described in Section II. Section III describes our proposed framework to achieve in-network RL inference. Section IV presents the experiment scenarios and performance evaluation details, followed by the conclusion and future directions in Section V.

II. System Model

A. Traffic Model

In conventional networking architectures, bandwidth is typically allocated at the link level, which can result in suboptimal use of resources. However, with the flow-level bandwidth allocation in SDN, the CP can assign bandwidth based on the requirements of individual flows, leading to enhanced network performance and reduced congestion. In addition, flow-level bandwidth allocation facilitates better control over network traffic by allowing an SDN controller to dynamically prioritize specific types of traffic, such as voice or video, over others. This ensures that time-sensitive applications receive adequate bandwidth to function effectively while best-effort traffic is assigned a lower priority [17]. The SDN controller accomplishes this by installing forwarding rules and differentiating traffic based on the TCP/UDP port numbers, with a weight ω associated with each flow indicating the quality of service requirement of the traffic belonging to that flow.

B. Bandwidth Model

A generalized model for bandwidth allocation has been developed with the goal of making it applicable to any SDN-based system [14]. To this end, we leverage our previously proposed model as a foundation for the current framework. As shown in Fig. 1, a 5G/6G wireless network comprises N number of core network nodes (i.e., programmable network switches) $\{1, 2, 3, ..., N\}$. Each node n contains P number of ports $\{1, 2, 3, ..., P\}$, and carries F number of flows

$\{1, 2, 3, ..., F\}$ per port p. Based on the importance of the traffic, a network administrator can assign each flow f with a weight parameter ω, which stipulates the QoS preference as described in Section II-A. The total bandwidth or capacity of a switch port p is represented by B_p (e.g., $100Gbps$).

We employ meter extern in P4 to allocate bandwidth to individual flows. The meter is configured with a CIR (committed information rate) and a PIR (peak information rate), along with a burst size in the form of discrete units. Based on the configuration and traffic rate, the meter yields either 0 (GREEN), 1 (YELLOW), or 2 (RED) as an output. P4 targets (e.g., behavioral model V2 (BMv2)) generally use these two-rate, three-color meters, which are associated with the flows to control the packet or data rate, thus called $RateLimiter$ [18]. We set the lower and upper limits; for instance, a minimum of 1 unit (i.e., $1Mbps$) and a maximum of 10 units are allowed for bandwidth allocation. Flows that fall within the same category, such as AR, VR, XR, etc., can be grouped and associated with a single meter. The utility of the switch is evaluated on a per-port basis. At time t, the total usage of the port capacity is the sum of the occupied resources by all the flows and is given as:

$$U_p(t) = \frac{\sum_{f=1}^{F} b_f^p}{B_p}, \tag{1}$$

where b_f^p represents the number of bandwidth units that are allocated to the flow f on port p.

III. In-Network RL inference Framework

In this section, we present a framework to address the RL inference and bandwidth allocation problem in the data plane (DP). As depicted in Fig. 1, the SDN controller is connected to the DP and serves as an intermediary between the DDPG agent and the DP. The controller has three responsibilities: 1) Stores the domain-specific information in the replay buffer of the agent, which is extracted by the DP from real traffic; 2) Updates the CIR and PIR of meters according to the bandwidth allocation policy learned by the DDPG agent; and 3) Performs the RL inference mapping into the forwarding rules and installs them in the MATs. By utilizing this control loop, the DDPG agent interacts with the network environment to learn an optimal bandwidth allocation policy. The following subsections provide further details on the proposed DDPG agent, the state-action space, and the reward model. Subsequently, we explain the RL inference in the DP using a MAT mapping strategy and outline the high-level workflow of our framework.

A. DDPG Agent

Traditional DDPG-based frameworks use fully connected networks (FCNs) that have large trainable weights and capture only global discriminative features [19], which is not suitable for traffic flows with varying QoS requirements that have complex temporal variations. To this extent, we proposed a state-of-the-art model called the temporal feature learning attentional network (TFLAN) for superior state representation and better function approximation of resource allocation systems [16]. Our TFLAN-based DDPG agent consists of three parts: 1) A Conv1D residual block structure to learn the correlations among local features of each input state; 2) A GRU layer to learn temporal dependencies; and 3) An attention mechanism to capture meaningful information at certain moments. Additionally, we proposed a heuristic-based priority experience replay (hPER) to reduce variance and break undesired temporal correlations in the training samples. The hPER stores all experiences ((s_t, a_t, r_t, s_{t+1})) and ranks them using an efficient heuristic function. This approach guides the agent towards useful regions, improving the convergence rate and stability during training.

1) State Space: The state of the system in the DP is defined at the flow-level, which is the observation of the weight, currently assigned date rate, and demand of the flow. The state of each distinct flow at time t is given as: $s_t = \{\omega_f(t), R_f(t), D_f(t)\}$, where ω_f denotes the weight of the flow f, which may or may not change over time. The currently assigned data rate and the demand of the flow are denoted by R_f and D_f, respectively. Initially, the system allocates a default data rate to all the flows according to their weights, which can be modified later according to the demand of the flows. The demand of a flow f at time t is calculated as the rate during the last τ till the current time t and can be determined as:

$$D_f(t) = \frac{Rx_f(t) - Rx_f(t-\tau)}{t-\tau} \tag{2}$$

where $Rx_f(t)$ is the total number of received bytes from a flow f at time t. Eq. (2) gives us an average demand rate of the flow f during an interval $(t-\tau)$. In practice, it is fairly simple to measure the demand rate in the DP using P4. Upon the arrival of a packet, the number of bytes of the packet (excluding header size) is incremented in the relevant counter. The stateful information of counters and timestamps is maintained using the registers.

In order to avoid unnecessary RL inferences in the DP (to minimize overhead), we apply pruning based on the difference between the current rate and demand, which is given as: $\delta_f(t) = |D_f(t) - R_f(t)|$. If $\delta_f(t)$ becomes greater than a set threshold (i.e., θ_f) of the flow, only then RL inference is executed in the DP to reallocate bandwidth at the line rate (more information in Section III-C).

2) Action Space: Based on the above observation sequences, the proposed DDPG agent learns an optimal bandwidth allocation policy. In each state s_t, the system decides how much bandwidth has to be allocated to minimize the system cost while meeting the QoS demands. The action space in the DP, $a_t = \{b_f\}, (f \in [1, F])$, has a parameter b, which represents a new allocation (b units of bandwidth (Section II-B)) at time t.

3) Reward Model: The objective of the DDPG agent is to reduce the total cost of the system by taking a series of actions in all states. To learn an optimal resource allocation policy, the agent executes the action a_t in state s_t, advances the system into state s_{t+1}, and obtains a reward r_t from the environment, which is the inverse of the cost incurred by the action a_t. In our proposed model, the reward is calculated as the inverse of the sum of the cost and penalty of resource procurement in the system.

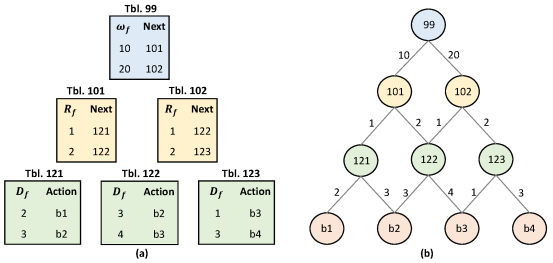

Fig. 2: A high-level structure of in-network RL inference using match-action tables.

$$cost_f = b_f \times C, \tag{3}$$

where C is the system cost per unit of bandwidth, which is adjustable based on the environment. The penalty for a flow refers to the additional cost caused by inaccurate resource allocation and is defined as:

$$penalty_f = \omega_f \times \frac{D_f}{b_f}, \tag{4}$$

The penalty equation helps the DDPG agent learn a precise resource allocation policy by minimizing the penalty ratio. Therefore, the overall reward is given as follows:

$$r_t(s_t, a_t) = \frac{1}{cost_f(s_t, a_t) + penalty_f(s_t, a_t)}. \tag{5}$$

We compute rewards for each flow separately. This method is beneficial because QoS demands may differ across flows, and computing rewards for each flow can aid the agent in deriving an even better resource allocation policy.

The system aims to maximize the long-term reward by minimizing the system cost when allocating bandwidth, while taking into account varying QoS requirements and constrained resources. This optimization problem can be expressed as:

$$maximize \sum_{t=1}^{T} r_t(s_t, a_t) \tag{6}$$

subject to:

$$U_p^n(t) \leq 1, \forall t \in T, \forall p \in P, \forall n \in N \tag{7}$$

$$1 \leq b_f \leq b_{f_{Max}}, \forall f \in F \tag{8}$$

$$\delta_f < \theta_f, \forall f \in F, \tag{9}$$

where constraint (7) limits the ports usage on all the switches to not exceed the available capacity (B_p) at any given time. The constraint (8) guarantees that the flow f is allocated with bandwidth units between 1 and the maximum permissible units. The constraint (9) serves three purposes: Firstly, it ensures that the difference between the demand rate and the currently assigned rate remains below the threshold θ_f, thereby meeting the QoS requirement. Secondly, it helps reclaim the resources in situations where the demand is less than the currently assigned rate. Lastly, it helps minimize per-packet processing time in the DP by avoiding unnecessary inferences.

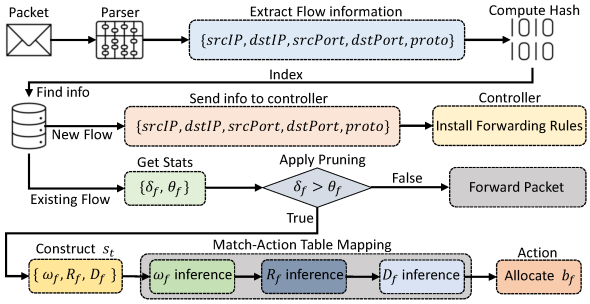

Fig. 3: PISA pipeline workflow for in-network RL inference.

B. RL inference using Match-action tables

The programmable switch ASICs have certain limitations with respect to their operational capabilities. Specifically, the switch ASICs are unable to perform floating point computations and loops, which makes it challenging to implement inference procedures for ML models in the DP [6], [7]. We address this problem by designing an RL model in the DP whose states and actions are tailored to fit well within the PISA pipeline. The state s_t consists of multiple parameters (i.e., ω_f, R_f, D_f), which in turn produces an action a_t. As an example in fig. 2(a), the program applies the first table in the pipeline to match the weight of the flow (ω_f). This is followed by matching the appropriate rate table (R_f), followed by the final demand table (D_f) to output a suitable meter (bandwidth). As illustrated in fig. 2(b), the process of RL inference is analogous to a graph structure, which can be mapped to match-action tables (MATs). Essentially, the MAT mapping strategy facilitates RL inference in the programmable network devices, obviating the need for computationally intensive calculations. Note that the rules in the MATs are installed by the controller according to the bandwidth allocation policy, which is learned by the DDPG agent. As mentioned in Section II-B, multiple related flows can be grouped together and assigned the same weight based on their importance. This helps save memory and table entries, thus enhancing efficiency.

C. Workflow

As depicted in fig. 3, we illustrate the overall workflow of our proposed framework with the help of the PISA pipeline. Upon the arrival of a packet, the programmable parser extracts the flow information. A hash is calculated on the 5 tuple information to get an index value for the registers, where flow information is saved. In the event of a new flow, the packet is cloned and sent to the controller to install a forwarding rule and allocate a bandwidth unit. Conversely, in the case of an existing flow, the demand of flow is measured using the methodology elaborated in Section III-A1. After applying pruning, if δ_f surpasses the predefined threshold (θ_f), the state is formulated (i.e., $\{\omega_f, R_f, D_f\}$). The framework then executes RL inference using the match-action pipeline, as discussed in Section III-B, to reallocate the bandwidth at the line rate.

IV. Experimental Results

In this section, we evaluate the performance of our proposed in-network RL inference framework. We develop the bandwidth allocation model (Section (II-B)) and proposed TFLAN-based DDPG agent (Section (III-A)) in Python 3.8.10. The

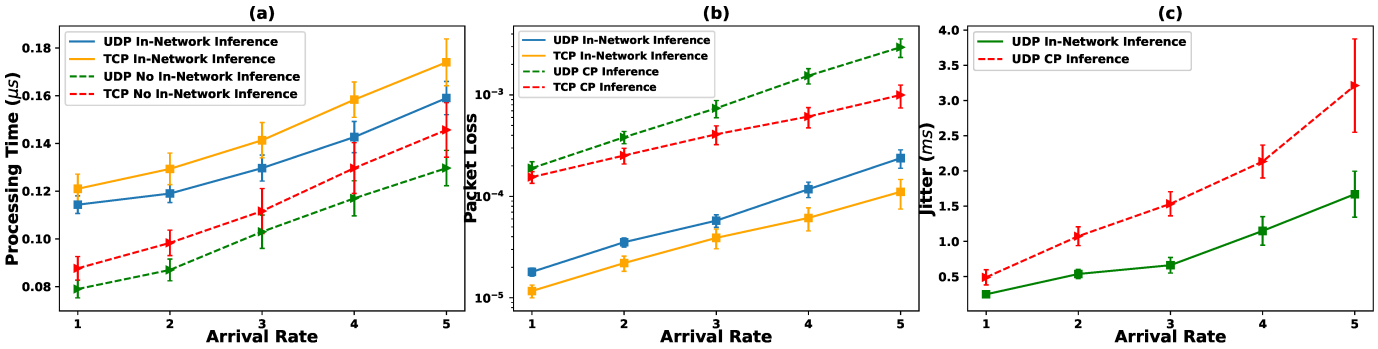

Fig. 4: (a) Per-packet processing time in microseconds with and without in-network RL inference, (b) Packet Loss rate comparison between in-network and CP-based RL inference (Y-axis is in log scale), and (c) Jitter comparison for UDP traffic in milliseconds.

data plane (DP) component of RL inference consists of about 500 lines of $P4_{16}$ code, which runs in the behavioral model V2 (BMv2) simple switch [20], and tested using Mininet [21]. Note that our proposed implementation of RL inference required 3X fewer lines of code compared to the pForest implementation [7]. An SDN controller is also developed in Python, which controls the RL inference via forwarding rules and acts as a mediator between the DDPG agent and the DP with the help of the P4 Runtime API. All the experiments are conducted on a virtual machine (VM) with 12 2.9 GHz CPU cores, 90 GB of memory, and the Ubuntu 20.04 OS. Note that the DDPG agent, controller, and P4 target (BMv2) run in the same VM.

A. *Effectiveness*

We analyze the overhead and the effectiveness yielded due to the proposed in-network RL inference framework. We created a triangle topology using Mininet which emulates the BMv2 switches to run the P4 code. Each switch in the triangle topology is connected to a host, which is used to generate TCP and UDP traffic with the iperf tool. We measure the per-packet processing time in the PISA pipeline with and without RL inference. As depicted in fig. 4(a), the per-packet processing time increases with the traffic arrival rate. When RL inference is applied in the DP, a maximum of $\sim$100 nanoseconds (ns) of overhead is observed in the case of TCP traffic (with maximum arrival rate), compared to the case when no in-network RL inference is applied (UDP with minimum arrival rate). This overhead is negligible compared to the round-trip time from the DP to the CP, which is in the order of milliseconds and can further increase if the CP is located in a distant location.

Fig. 4(b) depicts a comparison of the packet loss rates between our proposed in-network RL inference and CP-based RL inference. In the conventional approach [14], an inference request is sent to the CP, which takes a decision and allocates bandwidth to the flow. In contrast, our proposed method allocates bandwidth at the line rate and forwards numerous packets by the time a decision is made by the CP, thereby significantly reducing the packet loss rate, as evident in fig. 4(b). Additionally, despite the fact that TCP traffic requires more processing time (fig. 4(a)), it has a lower packet loss rate than UDP due to its inbuilt flow control mechanism. Overall, our proposed method, on average, achieves approximately 92% and 89.7% reductions in packet loss for UDP and TCP traffic, respectively, when compared to the CP-based approach.

Jitter is typically associated with UDP traffic because UDP does not provide any flow control or error correction mechanisms; therefore, we measure the jitter for UDP traffic only. As illustrated in fig. 4(c), our proposed method helps reduce the jitter by allocating bandwidth at the line rate, eliminating the delays and variability associated with the round-trip time from the DP to the CP. This allows for the swift and precise control of network traffic, minimizing the impact of jitter on real-time applications. Our proposed in-network RL inference improves jitter by 57% for UDP traffic compared to the CP-based approach.

B. *Performance Evaluation*

We evaluate the performance of the in-network bandwidth allocation framework with the TFLAN-based DPPG agent in terms of convergence speed and operational cost. The two alternative agents used to compare with our TFLAN-based DDPG agent are described below:

- **TFLAN-Uniform:** The actor and critic networks are equipped with our proposed TFLAN-based architecture. However, transitions are sampled randomly from the replay buffer to train the agent.
- **DDPG-NN-Uniform:** Existing DDPG agent [19], which uses two fully connected layers for the actor and critic networks and a uniform sampling replay buffer.

The actor and critic networks in all agents are set with a learning rate of 0.0001 and 0.001, respectively, and use a discount factor $\gamma = 0.99$. The loss function is optimized using the Adam optimizer during training. Training consists of 1000 episodes with 1000 learning steps each.

We run the simulation for 100 switches, each comprising 48 ports with 25 Gbps capacity per port. For the performance analysis of our proposed agent, four categories of traffic with varying data rates and different QoS demands are generated that can be anticipated in 5G/6G environment. RT-1 traffic is considered time-sensitive and bandwidth-intensive, usually originating from AR/VR or similar applications, and its data rate varies from 6 Mbps to 8 Mbps. RT-2 traffic is also time-sensitive but not bandwidth-intensive, like that from smart-city sensors, and its data rate varies from 1 Mbps to 2 Mbps. BE-1 traffic is delay-tolerant but bandwidth-intensive, typically generated by users uploading or downloading files, with its data rate ranging from 6 Mbps to 8 Mbps. Lastly, BE-2 traffic is also delay-tolerant, which generates less traffic within the range of 1 Mbps to 2 Mbps and may consist of text, email,

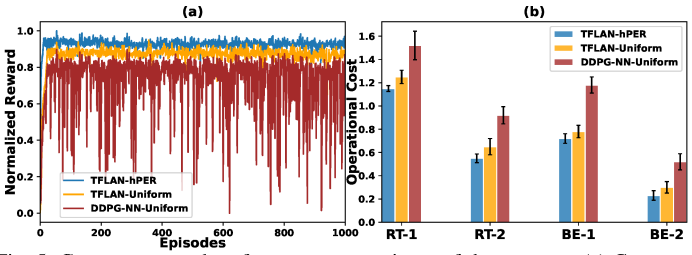

Fig. 5: Convergence and performance comparisons of three agents. (a) Convergence, and (b) Operational Cost.

web traffic, etc. Real-time traffic has a higher weight (ω), which signifies the QoS requirements and helps the algorithm prioritize such traffic over best-effort traffic.

Fig. 5(a) illustrates the convergence curves of all three DDPG-based agents. Overall, the TFLAN-based methods show less oscillations and achieve better episodic reward compared to the DDPG-NN. The proposed approach outperforms both agents in terms of convergence speed, higher reward, and training stability, all due to the virtue of our unique neural network design (i.e., TFLAN) and efficient replay buffer (i.e., hPER). Lastly, the operational cost comparison is depicted in fig. 5(b), which is calculated using Equations (3) and (4). Allocating a smaller amount of bandwidth reduces the cost (Eq. (3)); however, imprecise allocation with respect to the demand (Eq. (2)) incurs an additional penalty (Eq. (4)). Consequently, allocating insufficient bandwidth to high-weighted flows raises the overall cost. Our proposed method learns precise bandwidth allocation policy by minimizing the penalty ratio for all flow types. As a result, our proposed agent (i.e., TFLAN-hPER) achieves up to 23% and 56% reduction in cost compared to TFLAN-Uniform and DDPG-NN-Uniform, respectively.

In summary, our proposed framework successfully achieves in-network RL inference and significantly reduces packet loss rate and jitter at the cost of negligible time overhead. Furthermore, our proposed TFLAN-based DDPG agent prioritizes real-time traffic over best-effort while satisfying the QoS demands of all kinds of traffic. Our agent improves the convergence rate and reduces operational cost when compared with two alternative agents.

V. Conclusion and Future Work

We presented a novel in-network RL inference framework for efficient bandwidth allocation at the line rate using P4 meter and match-action table mapping techniques. Our proposed method achieved a fine-grained control of the network for demand-based bandwidth allocation to individual flows with negligible processing overhead, while reducing packet loss rate and jitter by up to 92% and 57%, respectively. Furthermore, our proposed DDPG agent outperforms two alternative agents in terms of convergence speed, higher reward, and training stability, achieving up to 56% reduction in the operational cost.

The framework was validated using a BMv2 simple switch and Mininet emulator, and future work will extend the implementation to a hardware Intel Tofino-based switch (Netberg Aurora 610) and run extensive performance and scalability tests in a real-world testbed environment. We also plan to explore multi-agent reinforcement learning to address resource management problems in multi-domain networks.

Acknowledgment

This work is supported by NSF PAWR (#1827923) and NSF IRNC (#2029295).

References

[1] P. Kamble *et al.*, "6G Wireless Networks: Vision, Requirements, Applications and Challenges," 2022 5th International Conference on Advances in Science and Technology (ICAST), Mumbai, India, 2022, pp. 577-581, doi: 10.1109/ICAST55766.2022.10039549.

[2] P. N. Srinivasu *et al.*, "6G Driven Fast Computational Networking Framework for Healthcare Applications," in IEEE Access, vol. 10, pp. 94235-94248, 2022, doi: 10.1109/ACCESS.2022.3203061.

[3] G. Liu *et al.*, "The SOLIDS 6G mobile network architecture: Driving forces features and functional topology", Engineering, vol. 8, pp. 42-59, Jan. 2022.

[4] A. Liatifis *et al.*, "Advancing SDN from OpenFlow to P4: A Survey," 2023 ACM Comput. Surv. 55, 9, Article 186 (September 2023), 37 pages, doi: 10.1145/3556973.

[5] 5G PPP Architecture Working Group, "The 6G Architecture Landscape European perspective," Creative Commons Attribution 4.0 International, DOI:10.5281/zenodo.7313232.

[6] W. Quan *et al.*, "AI-Driven Packet Forwarding With Programmable Data Plane: A Survey," in IEEE Communications Surveys & Tutorials, vol. 25, no. 1, pp. 762-790, Firstquarter 2023, doi: 10.1109/COMST.2022.3217613.

[7] C. Busse-Grawitz *et al.*, "pForest: In-Network Inference With Random Forests," 2019, doi: arxiv.org/abs/1909.05680.

[8] G. Siracusano *et al.*, "Running neural networks on the NIC," arXiv:2009.02353, 2020.

[9] C. Zheng *et al.*, "IIsy: Practical In-Network Classification," 2022, doi: 10.48550/ARXIV.2205.08243.

[10] Francesco Musumeci *et al.*, "Machine-Learning-Enabled DDoS Attacks Detection in P4 Programmable Networks", Journal of Network and Systems Management, vol. 30.1, pp. 1-27, 2021.

[11] J. Luo *et al.*, "Binary Neural Network with P4 on Programmable Data Plane," 2022 18th International Conference on Mobility, Sensing and Networking (MSN), Guangzhou, China, 2022, pp. 960-965, doi: 10.1109/MSN57253.2022.00155.

[12] T. Swamy *et al.*, "Taurus: a data plane architecture for per-packet ML," In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '2022). Association for Computing Machinery, New York, NY, USA, 1099–1114, doi: 10.1145/3503222.3507726.

[13] G. Siracusano *et al.*, 2018. "In-Network Neural Networks," arXiv:1801.05731 (2018).

[14] A. Qadeer *et al.*, "Flow-Level Dynamic Bandwidth Allocation in SDN-Enabled Edge Cloud using Heuristic Reinforcement Learning," 2021 8th International Conference on Future Internet of Things and Cloud (FiCloud), Rome, Italy, 2021, pp. 1-10, doi: 10.1109/FiCloud49777.2021.00009.

[15] A. Qadeer *et al.*, "DDPG-Edge-Cloud: A Deep-Deterministic Policy Gradient based Multi-Resource Allocation in Edge-Cloud System," 2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Jeju Island, Korea, Republic of, 2022, pp. 339-344, doi: 10.1109/ICAIIC54071.2022.9722676.

[16] A. Qadeer *et al.*, "Deep-Deterministic Policy Gradient Based Multi-Resource Allocation in Edge-Cloud System: A Distributed Approach," in IEEE Access, vol. 11, pp. 20381-20398, 2023, doi: 10.1109/ACCESS.2023.3249153.

[17] Blake, S. *et al.*, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998.

[18] "$P4_{16}$ Language Specification," *version 1.2.3*, The P4 Language Consortium, 11 July. 2022, Available: [Online] https://p4.org/wp-content/uploads/2022/07/P4-16-spec.html.

[19] Chen, Z. *et al.*, Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach. J Wireless Com Network 2020, 188 (2020). https://doi.org/10.1186/s13638-020-01801-6

[20] P4 Language Consortium, "Behavioral Model (BMv2)," 2023, Available [Online] https://github.com/p4lang/behavioral-model.

[21] Bob Lantz *et al.*, 2010. "A network in a laptop: rapid prototyping for software-defined networks," In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX). Association for Computing Machinery, New York, NY, USA, Article 19, 1–6. DOI:https://doi.org/10.1145/1868447.1868466