computers

MDPI

Article

Adaptive Congestion Detection and Traffic Control in Software-Defined Networks via Data-Driven Multi-Agent Reinforcement Learning

Kaoutar Boussaoud *,† **, Abdeslam En-Nouaary** † **and Meryeme Ayache** †

STRS Laboratory, National Institute of Posts and Telecommunications (INPT), Rabat 10170, Morocco; abdeslam@inpt.ac.ma (A.E.-N.); ayache@inpt.ac.ma (M.A.)

* Correspondence: boussaoud.kaoutar@doctorant.inpt.ac.ma

† These authors contributed equally to this work.

Abstract: Efficient congestion management in Software-Defined Networks (SDNs) remains a significant challenge due to dynamic traffic patterns and complex topologies. Conventional congestion control techniques based on static or heuristic rules often fail to adapt effectively to real-time network variations. This paper proposes a data-driven framework based on Multi-Agent Reinforcement Learning (MARL) to enable intelligent, adaptive congestion control in SDNs. The framework integrates two collaborative agents: a Congestion Classification Agent that identifies congestion levels using metrics such as delay and packet loss, and a Decision-Making Agent based on Deep Q-Learning (DQN or its variants), which selects the optimal actions for routing and bandwidth management. The agents are trained offline using both synthetic and real network traces (e.g., the MAWI dataset), and deployed in a simulated SDN testbed using Mininet and the Ryu controller. Extensive experiments demonstrate the superiority of the proposed system across key performance metrics. Compared to baseline controllers, including standalone DQN and static heuristics, the MARL system achieves up to 3.0% higher throughput, maintains end-to-end delay below 10 ms, and reduces packet loss by over 10% in real traffic scenarios. Furthermore, the architecture exhibits stable cumulative reward progression and balanced action selection, reflecting effective learning and policy convergence. These results validate the benefit of agent specialization and modular learning in scalable and intelligent SDN traffic engineering.

Keywords: Software-Defined Networking (SDN); congestion control; Multi-Agent Reinforcement Learning (MARL); Q-learning; Deep Q-Network (DQN); bandwidth allocation; traffic classification; network performance optimization; data-driven networking

Academic Editor: Kannan Govindarajan

Received: 16 April 2025
Revised: 1 June 2025
Accepted: 7 June 2025
Published: 16 June 2025

Citation: Boussaoud, K.; En-Nouaary, A.; Ayache, M. Adaptive Congestion Detection and Traffic Control in Software-Defined Networks via Data-Driven Multi-Agent Reinforcement Learning. *Computers* **2025**, *14*, 236. https://doi.org/10.3390/computers14060236

1. Introduction

In modern communication networks, data plays a pivotal role, not only because of the transmitted data but also due to its utilization to optimize network performance. From traditional architectures to next-generation programmable infrastructures such as Software-Defined Networks (SDNs), data has consistently guided critical decisions in routing, load balancing, and traffic engineering. Network administrators have relied on data extracted from traffic to monitor it, identify its patterns, and determine the most appropriate routing strategies to ensure efficient flow transmission.

The rise of Artificial Intelligence has enabled a shift from static protocol configurations toward intelligent, data-driven network control. Across domains such as wireless networks,

vehicular systems, and SDNs, data-centric methods have demonstrated clear benefits. For instance, Fortuna et al. [1] proposed a multi-level modeling strategy using real-world wireless datasets to optimize performance and manage resources. Similarly, Jiang et al. [2] introduced Data-Driven Networking (DDN), a paradigm that allows protocols to be dynamically adjusted in response to quality metrics collected in multiple sessions. Their approach emphasizes automatic control, using adaptive settings rather than a fixed one. In parallel, the adoption of machine learning in domains such as network intrusion detection has highlighted the strength of data-driven frameworks. Jiang et al. [2] proposed a hierarchical classification strategy using both supervised learning and unsupervised autoencoders to detect and classify cyber attacks, emphasizing the importance of feature representation and balanced training data for effective intrusion detection in IP-based networks.

In the context of SDNs, the centralized control plane and the global view of the network offer an opportunity to integrate intelligent frameworks that can learn from the real-time state of the network. In addition, decoupling control from data planes in SDNs enable more flexible, programmable, and adaptive systems. However, the efficient detection and control of network congestion represents one of the persistent challenges in SDNs due to the high volume of traffic coming from large and dense networks such as data centers, and the complexity of the network topology that leads to reduced network performance in terms of packet loss, slow data transfer speeds, and reduced throughput. Kandula et al. [3] highlight that a server cluster with around 1500 nodes may process close to 100,000 network flows every second. In a similar context, Kandula et al. [4] report that, under peak load conditions, a network with 100 switches can experience up to 10 million flow requests per second. Additionally, even a modest delay of 10 milliseconds in flow setup, introduced by the SDN controller, can add approximately 10% latency to the large number of short-lived flows typically present in such environments. Yet, most existing congestion detection and control approaches rely on static thresholds or heuristics, which lack adaptability in dynamic environments.

To address the limitations of static and monolithic congestion control approaches, we propose a data-driven Multi-Agent Reinforcement Learning (MARL) framework designed for congestion classification and adaptive decision-making in Software-Defined Networks (SDNs). The proposed framework leverages real-time network metrics—such as packet loss, delay, jitter, and throughput—collected from live traffic to dynamically assess network state and inform control actions. Guided by these observations and feedback from a tailored reward function, the system continuously learns to adjust bandwidth allocation and traffic routing strategies.

The architecture comprises two specialized and collaborative agents: (1) a Congestion Classification Agent responsible for evaluating network conditions and classifying the current state into predefined congestion levels, and (2) a Decision-Making Agent, which utilizes Deep Q-Learning (DQN or its variants) to select the optimal actions, including increasing, decreasing, or maintaining bandwidth, and rerouting or preserving current traffic paths.

In contrast to previous work employing a single-agent model to jointly handle perception and control, our approach introduces a decoupled agent structure. This modular design improves interpretability, allows for targeted training of each agent, and enhances learning efficiency and robustness.

The agents are initially trained offline using both synthetic and real traffic traces, including the MAWI dataset, to simulate heterogeneous environments. After the offline training phase, the trained agents are deployed for online testing in a simulated SDN environment built with Mininet and the Ryu controller. During online operation, the agents

act on real-time traffic conditions without further training, enabling real-world evaluation of their generalization and adaptability.

To assess the framework's effectiveness, we compare the MARL system against (i) a standalone Decision Agent without the classifier (ablation study), (ii) a Double DQN-based agent without classification, and (iii) a non-learning baseline based on random action selection. Experimental results demonstrate that the proposed MARL framework consistently achieves superior performance across key metrics such as throughput, delay, and packet loss, and shows stable reward convergence under dynamic traffic conditions.

This paper is organized as follows: Section 2 introduces the foundational concepts of SDN and MARL. Section 3 presents a review of existing techniques in congestion classification and control. Section 4 outlines the architecture and design of our MARL-based framework. Section 5 details the experimental setup, datasets, and evaluation methodology. Finally, Section 6 concludes the paper and discusses potential directions for future research.

2. Overview of Key Concepts

This section overviews the key concepts critical to our research, namely Software-Defined Network, single-agent and multi-agent reinforcement learning.

2.1. Software-Defined Network

Software-Defined Networking (SDN) is considered a next-generation Internet technology, in which the control plane is separated from the data plane as first introduced by McKeown et al. [5] through the development of OpenFlow in 2028. In this architecture, the control logic is no longer embedded at the hardware level but is abstracted and centralized through a software-based control management system. SDN adopts a three-layered architecture consisting of the following:

- The **application layer**, which consists of a set of applications and services responsible for tasks such as load balancing, traffic monitoring as described by N. L. M. Van Adrichem et al. [6], and security as discussed by S. Shirali-Shahreza et al. [7]. This is the programmable layer of SDN that allows the integration of third-party functionalities and applications.
- The **control layer**, where low-level network functions and decisions are abstracted and managed. It acts as an intermediary between the application and infrastructure layers, interpreting application requirements and translating them into network behavior.
- The **infrastructure layer**, which comprises forwarding devices such as switches and routers. These devices are responsible solely for packet forwarding based on the rules and policies defined by the control layer.

The Software-Defined Network relies on a central SDN-Controller that could be a standalone entity or multiple SDN-Controllers collaborating together to achieve the centralized control. It also depends on different protocols to enable and maintain the separation between the control and data plane, such as OpenFlow protocol. OpenFlow is one of the most widely adopted protocols in Software-Defined Networks (SDNs), serving as a standardized interface between the control plane and the data plane. It enables the SDN controller to communicate forwarding rules and network policies to the underlying network devices, thereby allowing centralized and programmable network management as it has been introduced by McKeown et al. [5].

The SDN architecture, particularly the centralized control and global network view, provides a suitable environment for integrating intelligent control mechanisms. This architectural flexibility enables the deployment of learning-based agents at the controller level, capable of adapting to real-time network states. In this work, we build on these

principles to introduce a data-driven congestion classification framework using multi-agent reinforcement learning.

Software-Defined Networking (SDN) is considered a next-generation Internet technology in which the control plane is decoupled from the data plane as first introduced by McKeown et al. [5] through the development of the OpenFlow protocol. In traditional networks, control logic is tightly coupled with forwarding devices, making network management rigid and hardware dependent. In contrast, SDN abstracts and centralizes control logic through a software-based control management system, thus enabling more flexible, programmable, and dynamic network operation.

2.2. Multi-Agent Reinforcement Learning

Reinforcement learning (RL), as defined by R. S. Sutton et al. [8], is a sequential decision-making framework where an agent learns an optimal behavior by interacting with an environment $\mathcal{E}$ over discrete time steps. At each time step t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to a policy $\pi(a|s)$, and receives a reward $r_t \in \mathbb{R}$ while transitioning to a new state s_{t+1}. The objective of the agent is to maximize the expected cumulative reward, often expressed as the return:

$$G_t = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right] \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor that prioritizes immediate rewards over distant ones.

The interaction can be formally modeled as a Markov Decision Process (MDP), a decision-making framework defined by states, actions, transition probabilities, rewards, and a discount factor. It was developed and improved by Bellman [9] and Howard [10], and is formally represented by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $P(s'|s, a)$ denotes the transition probability and $R(s, a)$ the expected reward.

Traditional RL focuses on a single agent acting in a stationary environment, according to R. S. Sutton et al. [8] and M. Bowling et al. [11]. However, in many real-world systems like communication networks, multiple agents must operate simultaneously in a shared and dynamic environment. This leads to the Multi-Agent Reinforcement Learning (MARL) paradigm, where the environment becomes non-stationary from the perspective of each agent due to the presence of other learning agents.

In MARL, the environment is modeled as a Decentralized Partially Observable MDP (Dec-POMDP), represented by the tuple $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}_i\}, P, \{R_i\}, \gamma)$, where

- $\mathcal{N}$ is the number of agents;
- Each agent i selects actions $a_i \in \mathcal{A}_i$ based on its observation o_i;
- The joint action $\mathbf{a} = (a_1, a_2, ..., a_N)$ influences the transition $P(s'|s, \mathbf{a})$;
- Each agent receives a local reward $R_i(s, \mathbf{a})$.

MARL provides several advantages, especially in distributed systems like Software-Defined Networks:

- **Parallelism**: Tasks such as traffic monitoring, congestion detection, and control can be handled concurrently, allowing agents to operate simultaneously and independently across the network as presented by Zhang et al. [12] and Elsayed et al. [13].
- **Modularity**: Each agent can specialize in a sub-task, making the system more manageable and scalable. Modular designs improve development and maintenance by isolating components as introduced by Hernandez-Leal et al. [14] and Zhang et al. [12].
- **Task Separation**: Different aspects of network behavior (e.g., classification versus decision-making) can be independently optimized, improving overall system perfor-

mance through decentralized specialization as demonstrated by Boehmer et al. [15] and Foerster et al. [16].

In cooperative settings, agents aim to maximize a shared objective. This often requires communication or indirect coordination through shared states or rewards. For example, a shared reward function $R_{global}(s, \mathbf{a})$ can be used to encourage agents to take joint actions that lead to global optimal results. In our work, cooperation is achieved implicitly: Agent 1 classifies congestion levels, which then serve as input for Agent 2 to make informed bandwidth and routing decisions.

In general, MARL enables distributed intelligence in complex environments and aligns well with the architecture of SDNs, where multiple agents can operate at different levels of the control plane to improve responsiveness and performance.

3. Related Work

In the context of congestion management in Software-Defined Networks (SDNs), several studies have focused on traffic classification and traffic pattern identification as a vital step for intelligent routing, load balancing, and stable network performance. Due to the complexity and dynamics of network environments, traditional approaches often face limitations in detecting and mitigating congestion effectively. Consequently, this has led to increasing interest in the use of machine learning and data-driven techniques to analyze real-time traffic patterns, predict congestion, and make control decisions in advance. Using historical and real-time network data collected from network observations, these approaches aim to enhance adaptability and real-time decision-making by enabling the SDN controller to learn from the state of the network and optimize performance efficiently.

Poupart et al. [17] utilize data mining techniques to predict the size of network flows at the early stages of transmission, which is critical to maintain network performance while traffic conditions change. Their method models the flow size as a statistical distribution rather than relying on a deterministic function that applied fixed conditions. By computing the mean or mode of this distribution based on initial packet observations, the approach provides a probabilistic estimate of whether a flow will become significant. This early prediction enables the controller to differentiate between short-lived and potentially high-impact flows and allocate resources accordingly.

While Poupart et al. [17] rely on statistical modeling for flow size prediction, Bishop et al. [18] use machine learning algorithms for traffic classification and flow size estimation. They evaluate the effectiveness of a range of machine learning algorithms by investigating the comparative accuracy of neural networks, Gaussian Process Regression (GPR) by M. Seeger [19], and Online Bayesian Moment Matching (oBMM) by F. Omar [20]. GPR provides a probabilistic model that captures uncertainty in flow prediction, while oBMM enables the online updating of flow classification based on sequential data without retraining from scratch. The results demonstrate that learning-based models can significantly outperform traditional rule-based systems, especially in dynamic network environments where traffic characteristics change frequently.

Shafiq et al. [21] investigate application-level traffic classification using machine learning techniques to improve the identification of network application types based on flow-level features. Their study conducts a comparative analysis of multiple supervised learning algorithms, including Support Vector Machines (SVMs), BayesNet, and Naive Bayes, to evaluate their effectiveness in classifying traffic from common application protocols such as WWW, DNS, FTP, P2P, and Telnet. The classification is performed using flow features extracted from captured network flows, such as packet size, inter-arrival time, flow duration, and total bytes transferred, using the NetMate tool. This enables the models to identify the traffic pattern and make volumetric predictions. The results show that machine

learning classifiers, when trained on pertinent features, can provide high accuracy and adaptability, making them suitable for real-time traffic classification in modern networks. Their work highlights the practicality of feature-based ML approaches for scalable and efficient network traffic analysis.

For traffic prediction, Lv et al. [22] propose a deep learning framework based on Sparse Autoencoders (SAEs) to learn high-level traffic representations. Their architecture employs stacked autoencoders to extract spatial and temporal correlations embedded in network traffic patterns. The model is trained in two phases: first, using greedy layer-wise unsupervised pretraining to initialize weights effectively, and then applying supervised fine-tuning to optimize prediction performance. By using sparse representations, the model is able to focus on the most prominent traffic features, thereby improving generalization and robustness. Experimental results demonstrate that their approach achieves higher accuracy in predicting traffic volume compared to traditional models, making it suitable for dynamic and complex network environments.

J. Mao et al. [23] propose a flow control mechanism based on the Deep Deterministic Policy Gradient (DDPG), by T. P. Lillicrap et al. [24], a reinforcement learning algorithm designed for continuous action spaces. In this reinforcement learning framework, the agent observes network parameters such as bandwidth availability, latency, and traffic load, and learns to adjust the data sending rate in real-time. This allows for fine-grained congestion control, enabling the system to respond to minor variations in network dynamics with high precision. By leveraging the ability of DDPG to operate in continuous domains, the proposed approach achieves a more adaptive, continuous and refined control strategy compared to discrete action RL models.

Other studies have addressed congestion from a control plane perspective. Yu et al. [25] propose DIFANE, a scalable flow management solution that minimizes switch-controller interactions by delegating most forwarding decisions to authority switches. By keeping decision logic within the data plane, DIFANE significantly reduces the communication bottleneck on the controller, thereby alleviating potential congestion in the control plane. Similarly, Curtis et al. [26] introduce DevoFlow, an architectural enhancement that further limits control traffic by transferring greater responsibility to switches. DevoFlow aims to reduce controller overhead by handling frequent or short-lived flows locally, allowing the controller to focus on high-level policies and exceptional events.

R. Jin et al. [27] explore congestion control within SDN-based data center environments using reinforcement learning algorithms such as SARSA and Q-learning. Although their approach shows promising results, it is primarily limited to single-task scenarios and cannot be generalize across diverse network conditions. Lei et al. [28] introduce MTDQ-ER, (Multi-Task Deep Q-Learning with Experience Replay), a framework that utilizes the programmability of SDN to implement a multi-task RL agent capable of learning and applying optimal congestion control policies across multiple traffic scenarios.

As refined Q-learning, Zhang et al. [29] present a method that dynamically adjusts transmission rates based on real-time bandwidth utilization. However, their approach employs a single-agent model with fixed-rate control logic, which lacks the adaptability needed in complex and rapidly evolving networks. This rigidity may lead to inefficiencies and even control plane congestion, as the system does not continuously monitor or respond to minor network change.

Beyond flow detection, machine learning techniques have also been explored for intelligent traffic routing in communication networks. These routing strategies generally fall into two categories: decentralized and centralized approaches. Kato et al. [30] propose a decentralized deep reinforcement learning framework in which individual routers are modeled as neural network agents capable of learning and adapting their forwarding

behavior. The system is initially trained using data collected from Open Shortest Path First (OSPF) routing, and then fine-tuned through continuous feedback from the network environment. This architecture enables each router to make autonomous, data-driven routing decisions based on real-time traffic conditions, enhancing flexibility and scalability in heterogeneous network infrastructures.

In contrast, Rusek et al. [31] introduce RouteNet, a centralized routing framework built on Graph Neural Networks (GNNs). Unlike decentralized models, RouteNet operates at the controller level, leveraging the global view of the network provided by SDNs. By modeling the network as a graph—where nodes represent routers or switches and edges represent links—RouteNet learns to predict performance metrics such as average end-to-end delay. These predictions enable the controller to make informed, topology-aware routing decisions that adapt to changing traffic patterns, thereby improving the overall network efficiency and service quality.

In our previous research, we explored machine learning-based techniques for elephant flow detection in SDN environments. In [32], we conducted a comparative performance evaluation of several supervised learning algorithms, including Decision Trees, Random Forests, and Support Vector Machines, to identify high-bandwidth flows contributing to network congestion. The study revealed that Random Forests provided superior detection accuracy with minimal false positives, underscoring the viability of traditional supervised learning for proactive traffic identification. In a later study, in [33], we proposed a federated learning approach to elephant flow detection in distributed SDNs. This work emphasized bias mitigation and privacy preservation by enabling multiple domains to collaboratively train a global model without sharing raw data, effectively addressing class imbalance and data heterogeneity across controllers. While both studies achieved strong results in detection, they were focused exclusively on flow identification and did not incorporate adaptive decision-making or congestion control mechanisms.

Building upon these insights, our work distinguishes itself by introducing a cooperative learning mechanism using Multi-Agent Reinforcement Learning (MARL). Our proposed Data-Driven MARL framework includes a Congestion Classification Agent that continuously assesses traffic conditions using packet loss and delay, and a Decision-Making Agent that selects routing and bandwidth actions via Deep Q-Learning. Together, they form a sequential control chain capable of real-time congestion state assessment and intelligent, policy-driven decision-making.

Although previous works have explored traffic classification, congestion control, and intelligent routing using various machine learning and reinforcement learning techniques, most approaches focus on single-task optimization, static rule-based strategies, or rely on a single-agent model that lacks adaptability in diverse and dynamic environments. In contrast, our proposed Data-Driven MARL framework introduces a multi-agent architecture designed for Software-Defined Networks, where one agent is dedicated to congestion classification based on real-time delay and packet loss, and the other learns optimal control actions using Deep Q-Learning. This cooperative design allows the real-time assessment of congestion states and adaptive routing and bandwidth decisions, making it more flexible and scalable than conventional methods. Unlike approaches such as RouteNet that focus solely on centralized predictions, or DDPG-based controllers that address continuous rate adaptation in isolation, our system integrates classification and control embedded in a sequential MARL chain, enabling intelligent, policy-driven network management in the face of realistic traffic conditions.

4. System Architecture

This section outlines our proposed contribution to improving congestion resilience in Software-Defined Networks (SDNs) through adaptive bandwidth management and intelligent routing. We begin by presenting the overall architecture of the system, highlighting the roles and interactions of its core components. Next, we describe the implementation details of the multi-agent framework, including the reinforcement learning models and the simulation environment. Finally, we present the experimental setup.

4.1. The Architecture of the Data-Driven Multi-Agent Reinforcement Learning Framework

In this work, we propose a data-driven multi-agent framework for congestion classification in Software-Defined Networks as shown in Figure 1. The architecture is built upon an SDN environment composed of one or more controllers managing a set of forwarding devices (switches), which are connected to multiple hosts. Within this framework, two reinforcement learning agents operate collaboratively to perform real-time congestion classification and intelligent decision-making in Software-Defined Networks (SDNs). These agents differ in structure and functionality: the first agent is dedicated to classifying congestion levels in real-time based on traffic conditions observed in the network, while the second focuses on decision-making, optimizing bandwidth allocation and flow routing based on the current network conditions. The architecture of each agent comprises the following core components.

- **Agent 1: Congestion Classification**
  - **Data Plane Metrics Collector**: This component is responsible for gathering real-time network metrics from the network, such as packet loss, delay, jitter, and throughput. These features serve as the foundational input for the agent, enabling it to observe and monitor traffic behavior over time.
  - **State Identification Unit**: This module extracts relevant features from the collected metrics and defines the current state of the network. The network state is modeled using four discrete states: Low Packet Loss and Low Delay and Low Jitter, High Packet Loss and Low Delay and Low Jitter, Low Packet Loss and High Delay and High Jitter, and High Packet Loss and High Delay and High Jitter. The identification of a state facilitates a simplified and effective classification of network conditions.
  - **Classification Unit**: The core of the Congestion Classification Agent, this component utilizes reinforcement learning, particularly a Q-learning-based classifier, to predict the congestion class (efficient, loss-degraded, delay-degraded, or congested). The classification model is trained on past observations to refine its policy for accurate prediction. The resulting class is used as a label and sent to the second agent for further decision-making.
  - **Reward Unit**: This unit evaluates the effectiveness of the classification output by computing a reward value based on the impact of the label on the observed performance indicators of the network. It supports various reward schemes, such as Environment-Derived Rewards, such as reduced delay after classification, Heuristic Rewards like rule-based feedback, and Delayed Rewards, based on the long-term impact of classification).

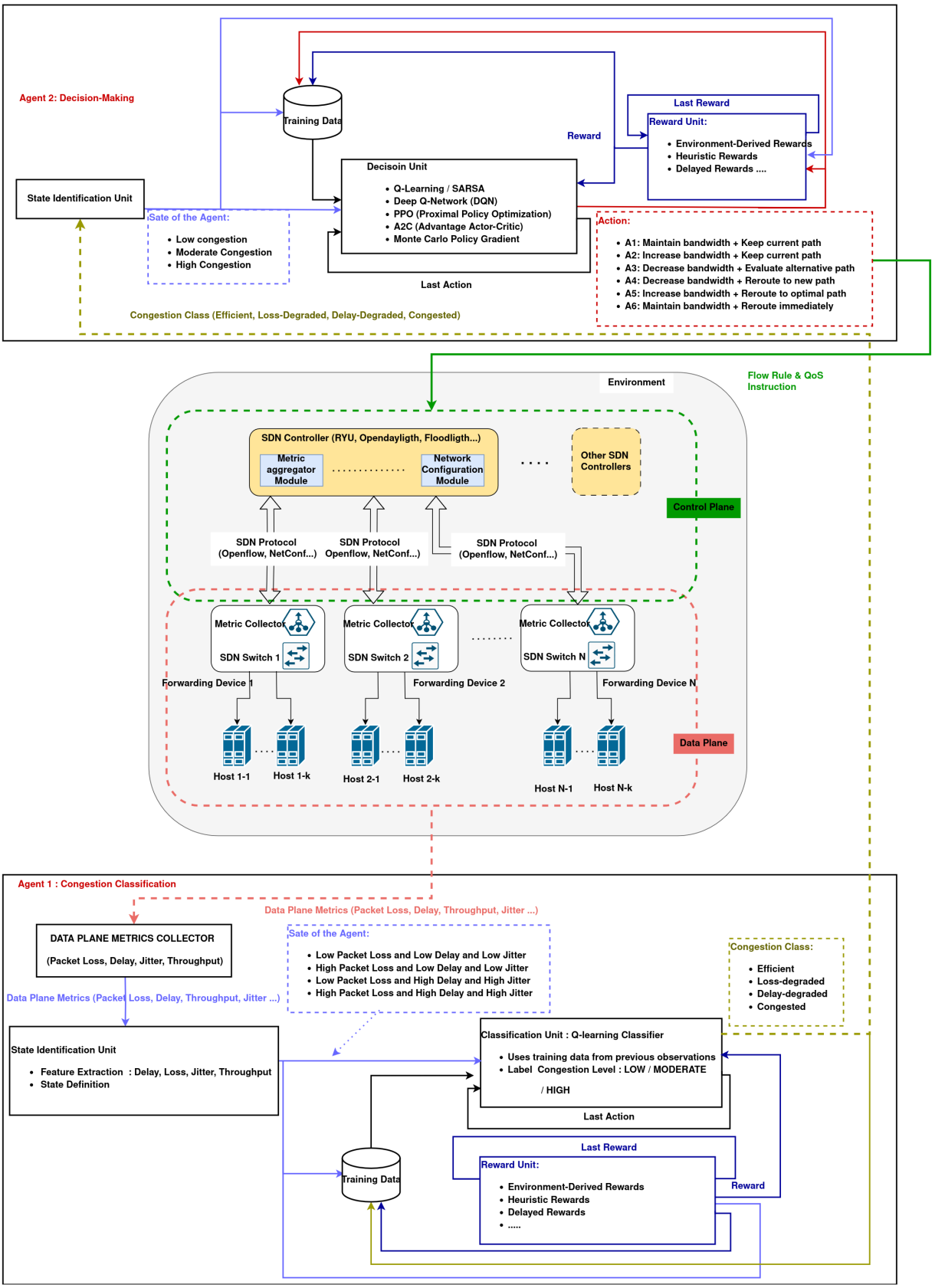

Figure 1. Data-Driven Multi-agent Reinforcement Learning framework.

- **Agent 2: Decision-Making**
  - **State Identification Unit**: This unit receives the congestion class generated by Agent 1 (efficient, loss-degraded, delay-degraded, or congested) and aggregates

to three main classes of congestion (Low, Moderate, or High) that define the agent's internal state. Based on the current level of congestion, the agent identifies possible corrective strategies to effectively manage flow within the SDN.

- **Decision Unit**: Serving as the core policy for the Decision-Making Agent, this unit uses reinforcement learning models such as Q-Learning, SARSA, Deep Q-Network (DQN), Double Q-Network, Proximal Policy Optimization (PPO), or Actor–Critic methods like A2C to learn the optimal actions. Given the current state, the agent chooses an appropriate action from a predefined set such as increase/decrease bandwidth, reroute flows, maintain current settings. The decision is then communicated to the SDN controller for implementation.
- **Reward Unit**: This component evaluates the effectiveness of the actions selected in alleviating congestion. It assigns a reward based on the improvement or degradation of the network metrics following the action. Similarly to Agent 1, this unit supports Environment-Derived Rewards, Heuristic Rewards, and Delayed Rewards to support both immediate and long-term learning objectives.

4.2. Implementation of the Framework

This section presents a detailed description of the implementation strategy adopted to realize our proposed congestion control framework for Software-Defined Networks (SDNs). The system is composed of a simulation environment constructed using Mininet, an SDN controller extended with reinforcement learning logic, and a multi-agent architecture that performs real-time classification and decision-making based on observed network metrics. Our implementation is modular, reproducible, and designed to accommodate varying traffic conditions for robust performance evaluation.

4.2.1. Simulation Environment and Network Topology

The entire experimental setup is deployed using the Mininet emulator [34], which enables the creation of a high-fidelity virtual network for prototyping and testing SDN applications. We employ **Ubuntu 22.04** with kernel version 5.15 as the host system. The software stack includes **Mininet 2.3.0**, **Open vSwitch 2.17**, and the **RYU controller framework 4.34** running under Python 3.10. This combination ensures compatibility with OpenFlow 1.3 and provides fine-grained programmability of the control plane.

The network topology consists of four OpenFlow switches (**s1** to **s4**) arranged in a linear backbone configuration. Three hosts (**h1** to **h3**) are connected to the edge switch **s1**, and three hosts (**h4** to **h6**) are connected to the edge switch **s4**. The inter-switch links **s1–s2** and **s3–s4** are provisioned at 10 Mbps, while the critical bottleneck link **s2–s3** is throttled to 1 Mbps and capped with a queue size of 20 packets using the HTB queuing discipline as shown in Figure 2. Access links from hosts to their edge switches operate at 100 Mbps, creating an oversubscription scenario that naturally induces congestion when the core is saturated.

The rationale for this topology design is to simulate a realistic scenario, in which high-bandwidth access networks compete for limited core resources, making it a suitable environment to evaluate congestion classification and routing strategies.

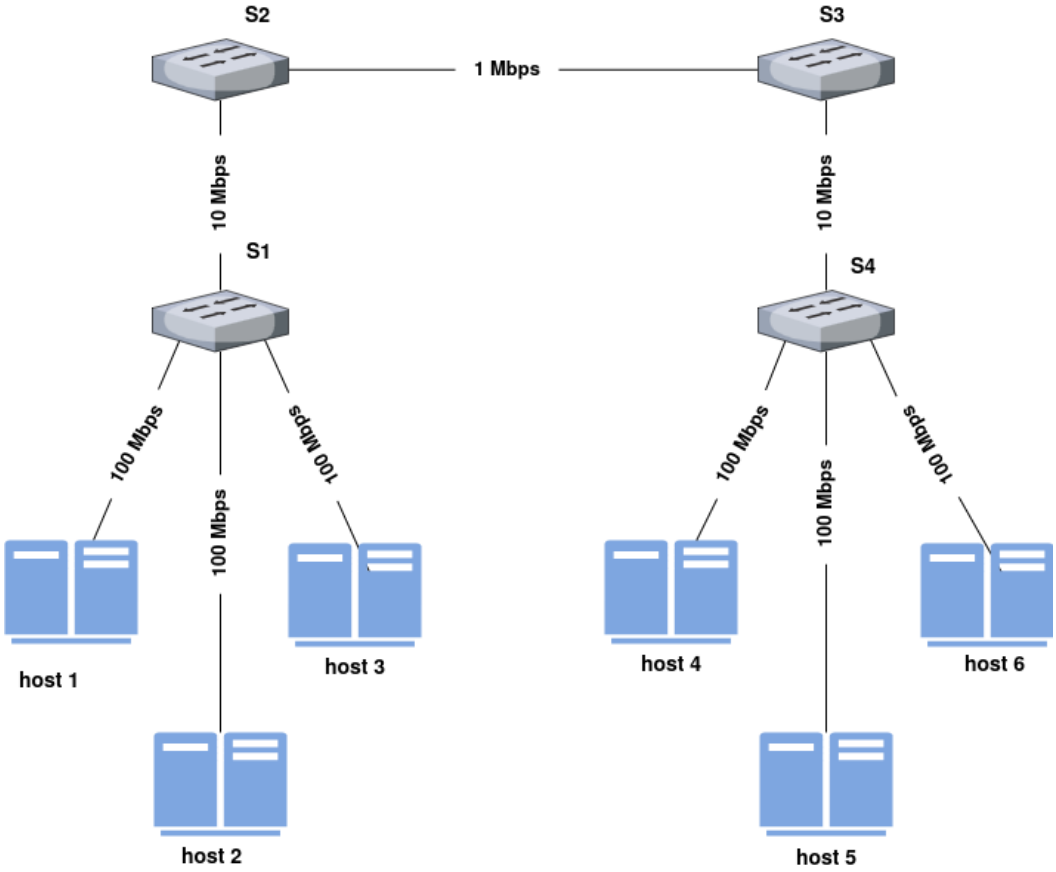

Figure 2. Topology Schema: Four OpenFlow Switches and Six Hosts.

4.2.2. Traffic Generation

To evaluate the robustness and generalization capabilities of our framework under heterogeneous network conditions, we conducted two separate experiments using distinct traffic sources. Each experiment defines two bidirectional traffic flows: one from host **h1** to **h2**, and the other from host **h4** to **h5**. In both setups, **h1** and **h4** function as traffic generators (clients), while **h2** and **h5** serve as receivers (servers).

These experiments were executed independently to ensure diversity in traffic profiles and to obtain a rich dataset representative of both controlled and real-world scenarios. The first experiment used synthetic traffic generated with **iperf3**, while the second employed realistic traffic traces from the MAWI archive. This design allowed us to assess the system's adaptability across varied traffic dynamics and emulate both predictable and bursty traffic behaviors.

- **Synthetic Traffic via iperf3**
  In the first experiment, synthetic traffic was generated using **iperf3**. Host **h1** transmitted a UDP stream to **h2**, while host **h4** transmitted to **h5**. Each stream ran for 960 s at a constant rate of 5 Mbps using 1200-byte packets. The following command was used for each flow:

  ```
  h1$ iperf3 -c h2 -u -b 5M -l 1200 -t 960
  h4$ iperf3 -c h5 -u -b 5M -l 1200 -t 960
  ```

  This controlled setup provided deterministic traffic, allowing precise evaluation of the framework's behavior under sustained congestion introduced via symmetrical flows through the core bottleneck link (**s2–s3**).

- **Real-World Traffic via MAWI Trace Replay**
  In the second experiment, we replayed real-world packet traces from the MAWI archive to simulate Internet backbone traffic. Specifically, trace #202504241400 was replayed using **tcpreplay** from host **h1** to **h2**, and separately from host **h4** to **h5**. These flows incorporate characteristics such as flow size variability and burstiness, introducing natural unpredictability to test the framework's responsiveness under realistic dynamics.

  ```
  h1$ tcpreplay --intf1=h1-eth0 202504241400.pcap
  h4$ tcpreplay --intf1=h4-eth0 202504241400.pcap
  ```

  The receiving hosts **h2** and **h5** were configured with the lightweight tool **nc (netcat)** to accept packets, ensuring valid end-to-end flow completion during replay without requiring a full application stack.

Although our framework supports additional traffic generators such as D-ITG [35] for application-layer emulation, we limited this study to **iperf3** and MAWI traces to maintain consistency and reproducibility. This dual-experiment setup ensured exposure to diverse and heterogeneous traffic conditions, ultimately producing a rich training and evaluation dataset for robust performance analysis.

4.2.3. Controller Functionality and Metric Collection

The RYU controller was customized to include both flow management logic and a suite of monitoring utilities. The core controller implementation is contained in **Controller_UnifiedController.py**, which registers OpenFlow packet-in events, modulates flow rules dynamically, and logs traffic statistics.

A dedicated monitoring thread collects metrics from all switch ports and flow tables every 100 ms using the OpenFlow protocol. The raw counters (packet and byte counts) are then transformed into meaningful per-interval values by computing first-order differences. This includes estimates of throughput, packet loss rate, and buffer occupancy. In addition, round-trip delay measurements are obtained using periodic **OFPEcho** probes sent every 50 ms. All collected statistics are logged to time-stamped files (e.g., **ryu_metrics_log_<timestamp>.log**).

The data collection process forms the backbone of the congestion classification pipeline. The raw logs are parsed and cleaned using a custom Python script (**Parsing_log_file.py**), which removes irrelevant records (e.g., LLDP packets and zero-traffic links) and exports a structured CSV file containing processed features for each time interval. These files serve as input to the training and inference modules of our framework.

4.2.4. Multi-Agent Learning Architecture

The core intelligence of the framework is implemented as a Multi-Agent Reinforcement Learning (MARL) system embedded within the SDN controller. The architecture includes two distinct but collaborative agents: a Classification Agent based on Q-learning and a Decision Agent implemented using a Deep Q-Network (DQN).

The Classification Agent consumes delay and packet loss statistics as input and outputs a congestion state, which is categorized into four discrete labels: Efficient, Loss-Degraded, Delay-Degraded, and Congested. These labels are further mapped to higher-level semantic categories (Low, Moderate, and High congestion) to enable contextual awareness in downstream decision-making.

The Decision Agent receives a richer feature vector that includes delay, packet loss, link utilization, and the output of the Classification Agent. It selects one of six predefined composite actions, each representing a different congestion-handling strategy as shown in

Table 1. The agent is implemented in TensorFlow/Keras and uses a fully connected neural network with three hidden layers of 64, 64, and 128 neurons, respectively, all activated with ReLU and regularized using L2 penalties. The learning process uses the Adam optimizer with a learning rate of 3×10^{-4} and a discount factor $\gamma = 0.99$. A target network is updated every 500 steps to stabilize convergence.

Table 1. Action mapping based on congestion classification.

Congestion Class	Action Description	Bandwidth Strategy	Routing Strategy
Low	Maintain bandwidth, keep current path	Maintain	Keep current
	Increase bandwidth, keep current path	Increase	Keep current
	Increase bandwidth, reroute to optimal path	Increase	Optimal reroute
Moderate	Decrease bandwidth, evaluate alternative path	Decrease	Evaluate alternative
	Maintain bandwidth, reroute immediately	Maintain	Immediate reroute
High	Decrease bandwidth, reroute to new path	Decrease	Reroute to new
	Maintain bandwidth, reroute immediately	Maintain	Immediate reroute

4.2.5. Training and Evaluation Workflow

The training and evaluation process of our multi-agent framework consists of two sequential stages, offline training and online evaluation, each with distinct objectives and procedures.

- **Offline Training:** Training is performed independently for the two agents in a staged manner:
  - In the first stage, the Classification Agent undergoes supervised Q-learning using synthetic traffic data. It receives a reward of +1 for accurate classification, and 0 or -1 otherwise. Over 10,000 iterations, the exploration rate ϵ is gradually reduced from 1 to 0.05, allowing the agent to shift from exploration to exploitation.
  - In the second stage, the Decision Agent (a DQN-based model) is trained using features and labels generated by the Classification Agent. The reward function integrates multiple QoS indicators, including delay, packet loss, jitter, and bandwidth utilization, and is designed to guide the agent toward optimal routing decisions. Network states with high QoS—characterized by delay below 70 ms, packet loss under 1%, jitter under 20 ms, and utilization below 80%—receive a reward of +6. Moderately degraded states (e.g., delay < 120 ms, loss < 2%) are rewarded with +5, and tolerable conditions are assigned +4. Severely congested states, such as those with delay exceeding 150 ms or utilization above 95%, are penalized with a minimal base reward of +1. In addition to this base reward, action-sensitive adjustments are applied: when congestion is detected (e.g., delay > 120 ms or loss > 2%), adaptive actions such as rerouting and bandwidth reduction receive bonus rewards of +4 and +3, respectively, while passive strategies like maintaining the current route receive no additional reward. In contrast, under stable conditions, preserving the current route is encouraged with a +5 bonus, while unnecessary rerouting or bandwidth reduction is mildly rewarded with +1. This two-tiered reward structure enables the agent to respond intelligently to dynamic network conditions, promoting proactive adaptation under congestion and stability-preserving behavior otherwise.
    Upon the completion of offline training, the learned parameters of both agents are saved for reuse in the subsequent evaluation phase.
- **Online Evaluation:** In the online phase, the trained models are deployed in a live SDN environment utilizing real-world MAWI traffic. The Decision Agent is reloaded with

the saved weights, enabling it to infer routing decisions in real-time. Agent actions are applied to the network dynamically, allowing for adaptive flow management and congestion control.

During both the offline training and the online deployment stages, performance metrics—including delay, packet loss, jitter, and link utilization—are continuously logged. These logs are later parsed and analyzed to compare the behavior and effectiveness of the agents in synthetic versus real-world traffic scenarios.

This two-tiered learning and evaluation approach empowers the framework to generalize from controlled conditions to dynamic, unpredictable environments. It enables robust and scalable decision-making while maintaining high QoS standards. The modular structure also supports future extensions involving diverse topologies, traffic profiles, or alternative reinforcement learning algorithms.

4.2.6. Reproducibility and Availability

To ensure that the experiments are fully reproducible and verifiable, we provide all relevant artifacts in our public GitHub repository [36].

- **Full source code** of the MARL framework, including the following:
  - The Q-learning-based Classification Agent and the DQN-based Decision-Making Agent (**Marl_Agent/**).
  - Ryu-based SDN controller integrated with both agents (**Controller/**).
- **Custom network topology file** for Mininet (**Topology/Topo4s-6h.py**) to emulate the experimental testbed.
- **Offline and online training scripts** with predefined parameters (**Training/**).
- **Example log files** from controller and evaluation phases (**Example_Log/**).
- **Custom log parsing tools** to extract and process performance metrics (**LogsParser/**).

Using these resources, the entire experiment can be reproduced on a standard machine with 8 CPU cores and 16 GB of RAM.

5. Simulation and Results Analysis

This section presents a detailed analysis of the proposed MARL framework's performance across both offline training and online evaluation phases. We also provide a comparative evaluation of our Data-Driven MARL approach against three alternative baselines: Ablation (DQN-only), Double DQN, and a Random (rule-based) controller. Both the offline and online phases utilize a combination of synthetic traffic generated using iperf3 and real-world traces from the MAWI dataset to train the agents and validate their ability to generalize under realistic SDN conditions.

To assess the robustness of the proposed architecture, MAWI traces and synthetic traffic are replayed within our Mininet-based SDN environment. These traffic sources reflect a wide range of real-world and controlled congestion scenarios, including bursty flows and variable load conditions. This setup enables a comprehensive evaluation of whether agents trained on diverse traffic patterns can adapt and maintain performance under dynamic network conditions. Throughout both phases, we monitor key QoS metrics, throughput, delay, packet loss, and link utilization, and analyze the agents' learning dynamics through cumulative reward trends and action selection distributions. The results demonstrate the strength of our multi-agent design in delivering stable and adaptive behavior across fluctuating traffic environments.

5.1. Performance Evaluation of the Classification Agent

To evaluate the effectiveness of the Classification Agent, we analyze its performance across both synthetic and MAWI-based traffic datasets. Specifically, we examine the classification precision, recall, and F1-score over the four defined congestion states. In addition to metric-based evaluation, we visualize the confusion matrix to assess prediction reliability and highlight areas of misclassification. This evaluation provides insights into the agent's ability to generalize congestion state detection under variable traffic patterns and validates its standalone learning capacity before integration with the decision agent.

5.1.1. Class Distribution in Training Data

Figure 3 shows the distribution of classes used for training the Classification Agent. The dataset is well-balanced, with approximately equal counts for all four congestion states: Efficient, Loss-Degraded, Delay-Degraded, and Congested. This balance ensures that the agent does not develop bias toward overrepresented classes, which is critical for maintaining generalization accuracy.

Figure 3. Training data class distribution for the Classification Agent.

5.1.2. Confusion Matrix

Figure 4 illustrates the confusion matrix obtained from evaluating the agent on a held-out test set. The classifier shows strong overall accuracy:

- Efficient and Delay-Degraded states are correctly identified in 36 out of 37 cases.
- Loss-Degraded instances are predicted with high precision, with only 2 errors out of 38.
- Congested states show slightly more confusion, with 3 misclassified cases (2 as other degraded states and 1 as Delay-Degraded).

The agent distinguishes well between the classes, especially the Efficient and Delay-Degraded conditions, while minor confusions appear among degraded categories—likely due to their similar network symptoms (e.g., delay and packet loss).

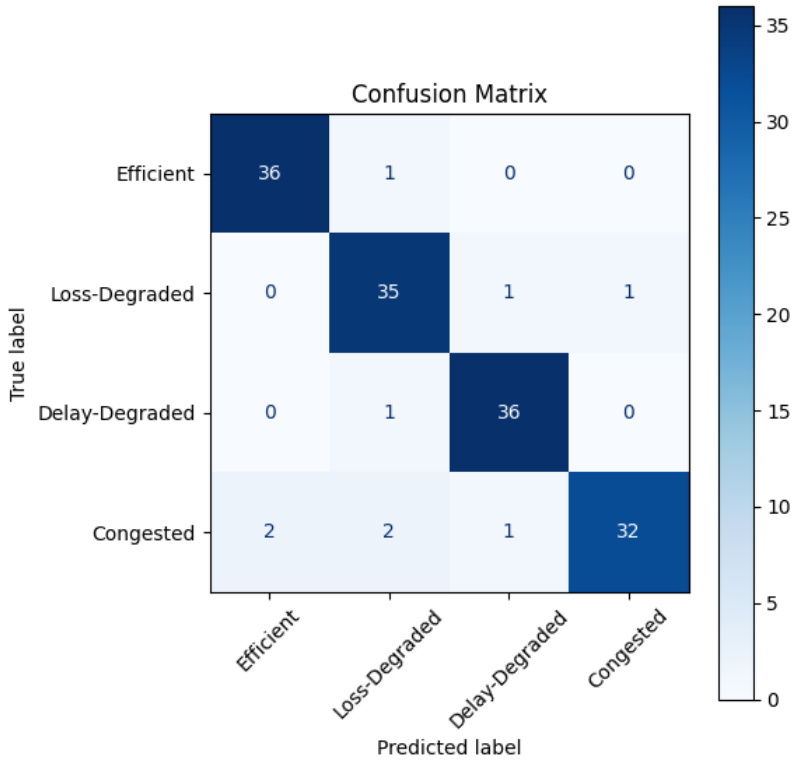

Figure 4. Confusion matrix for the Q-learning Classification Agent.

5.1.3. Classification Metrics

Figure 5 reports the precision, recall, and F1-score for each class. The agent achieves the following:

- Precision $\geq$ 0.91 across all classes, showing few false positives.
- Recall values ranging from 0.94 to 0.97, indicating high detection capability for each state.
- F1-scores that are tightly clustered around 0.95, suggesting balanced performance across precision and recall.

These scores reflect a robust classification policy, making the agent reliable in real-time SDN congestion scenarios.

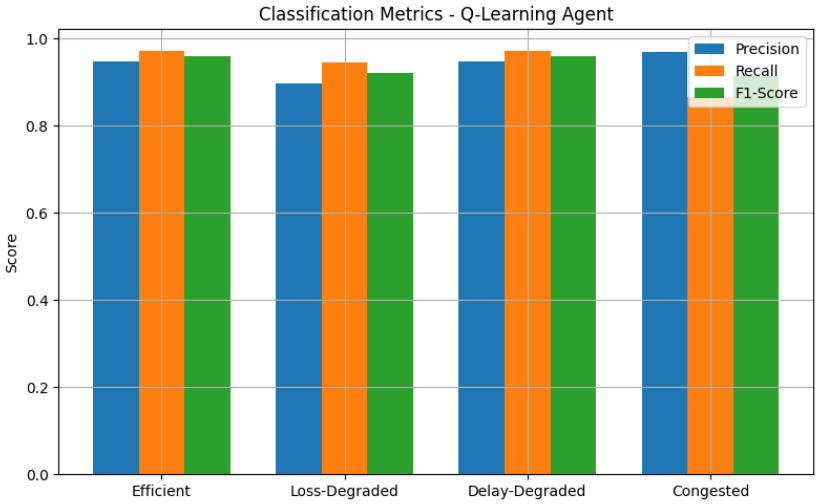

Figure 5. Precision, recall, and F1-score for each QoS class.

5.1.4. Cumulative Reward Curve

Figure 6 shows the learning curve of the Classification Agent during 200 training episodes. The cumulative reward starts negative, reflecting random or incorrect early decisions. However, a steady upward trend is observed as the agent gains experience and its policy improves. After approximately 100 episodes, the cumulative reward becomes consistently positive, confirming convergence to an effective classification strategy.

This trend validates the reward function design and the effectiveness of the ϵ-greedy exploration mechanism used during training.

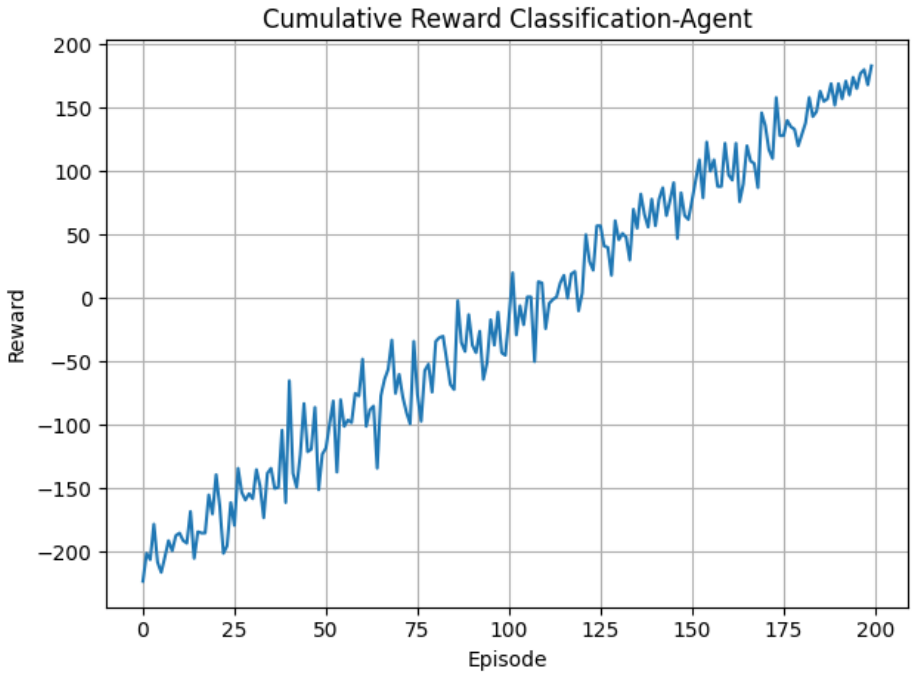

Figure 6. Cumulative reward curve of the Classification Agent during offline training.

Collectively, these results demonstrate that the Q-learning Classification Agent has been effectively trained offline. It generalizes well to new samples, maintains high classification accuracy across all QoS states, and converges rapidly under a stable training regime. This robustness ensures high-quality labels for guiding the Decision Agent in the subsequent decision-making process.

5.2. Performance Evaluation of Decision-Making Agent

In this section, we evaluate the performance of the Decision Agent trained using Deep Q-Networks (DQNs). The agent is responsible for selecting optimal actions involving bandwidth adjustments and routing strategies based on network conditions classified by the Classification Agent. We assess its behavior using both synthetic (iperf3) and real-world (MAWI) traffic traces in offline and online settings. The evaluation focuses on action selection diversity, cumulative reward trends, and critical QoS metrics, throughput, link utilization, delay, and packet loss. The goal is to determine the agent's adaptability and learning consistency under variable SDN traffic dynamics.

5.2.1. Evaluation of Action Distribution

Figure 7 displays the frequency distribution of actions selected by the Decision Agent during offline training. The agent has access to six composite actions, combining different bandwidth adjustment levels and routing strategies. The observed distribution shows

that the agent maintains a relatively balanced selection across all actions, with a slight preference for the actions **increase_bw_reroute_imm** and **maintain_bw_reroute_optimal**.

This distribution suggests that the agent did not fall into a narrow behavioral loop and instead explored diverse strategies. It also reflects a nuanced learning process where rerouting and adaptive bandwidth decisions are actively chosen based on the network state. The balanced usage of increase, decrease, and maintain operations indicates that the agent does not default to conservative policies but rather makes decisions that are responsive to dynamic QoS conditions.

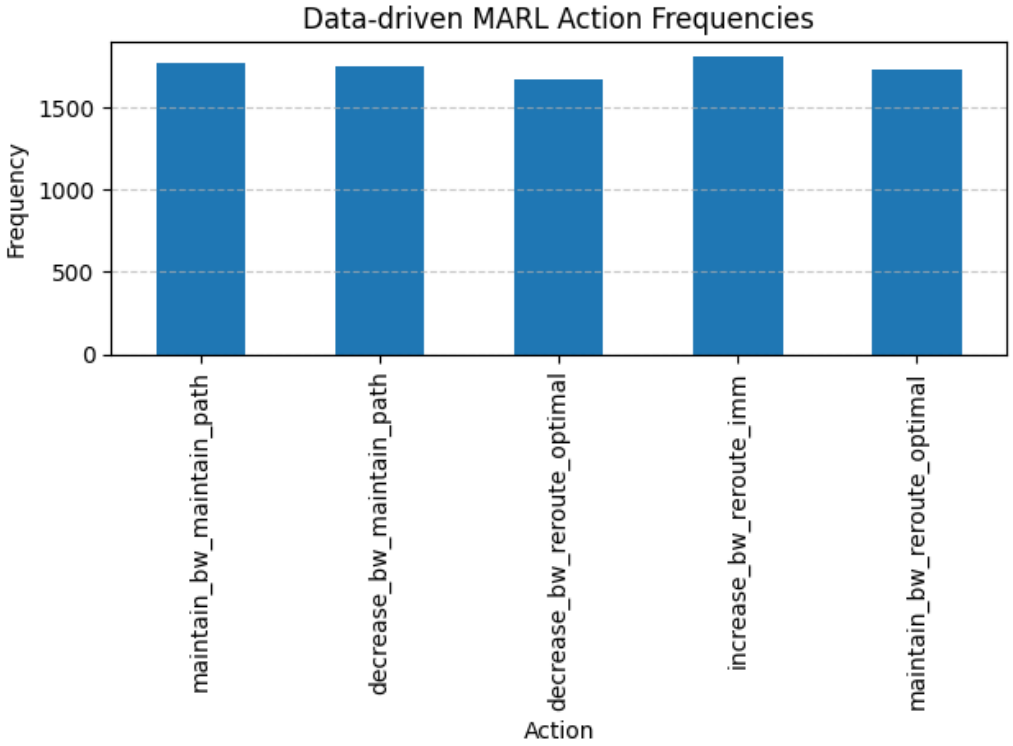

Figure 7. Action selection frequencies during offline training of the Decision Agent.

5.2.2. Evaluation of Throughput over Time

Figure 8 illustrates the average throughput measured during offline training and online evaluation. In the offline training phase, throughput shows a progressive increase over time, with oscillations attributed to the agent's exploratory behavior. These fluctuations reflect the learning process as the agent attempts various actions before converging to optimal strategies. The throughput curve stabilizes mid-training, indicating that the agent has effectively learned to route traffic while minimizing congestion. During the online evaluation phase, the pretrained agent is deployed in a realistic environment using MAWI traffic. Here, the throughput remains consistently high with minimal fluctuation. This behavior is a direct consequence of the knowledge acquired during offline training. The online agent manages to maintain throughput levels above those achieved during most of the offline training, confirming the stability of the learned policy under dynamic and bursty traffic patterns.

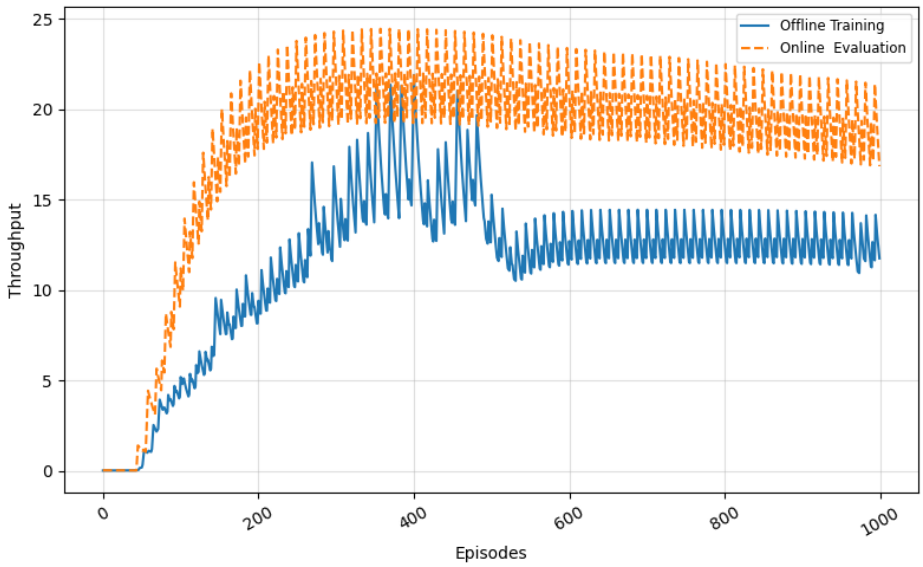

Figure 8. Comparison of network throughput for offline training and online evaluation over episode.

5.2.3. Evaluation of Link Utilization

Figure 9 presents the evolution of link utilization across training and evaluation. Initially, the offline agent permits over-utilization of the network core due to its untrained decision-making, resulting in congestion and inefficient bandwidth distribution. As training progresses, the agent learns to make more adaptive rerouting and bandwidth control decisions, leading to a gradual and controlled reduction in link saturation. In the online evaluation, link utilization starts at a lower level and remains within the 75–85% range, reflecting the agent's conservative and congestion-aware policy. Unlike static baselines or heuristic approaches, the trained agent is able to preserve high throughput without aggressively saturating links. This equilibrium between utilization and stability demonstrates that the model learned to operate efficiently even under real-world conditions, avoiding buffer overflows and packet loss typically associated with high link usage.

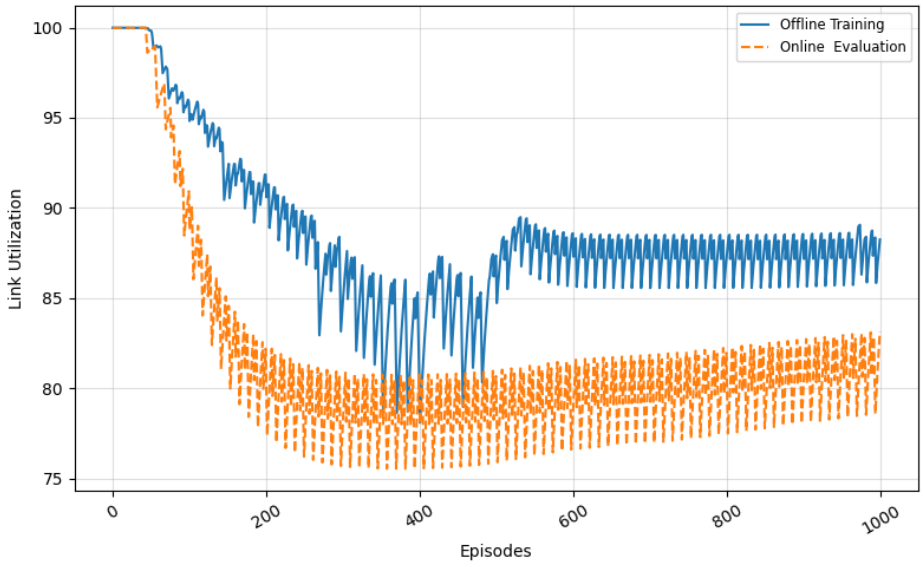

Figure 9. Link utilization for offline training and online evaluation observed per episode.

5.2.4. Evaluation of Delay

As depicted in Figure 10, the offline training phase begins with very high delay values, reaching peaks above 10^4 ms. This is expected due to the initial exploratory actions that cause suboptimal routing and bandwidth management. These delays are symptomatic of heavy queuing and congestion on the core bottleneck link. Over time, as the agent refines its policy, the delay sharply decreases, especially after episode 600, where the learned actions increasingly favor congestion-mitigating behaviors. Conversely, the online evaluation shows a markedly different pattern. From the beginning, delay values remain consistently low (below 10 ms), with minimal variance. This indicates that the model learned to generalize to unseen traffic patterns and could proactively avoid scenarios leading to excessive queuing. The steep drop in delay observed during the transition from exploration to exploitation demonstrates that the RL agent successfully internalized delay-sensitive policies.

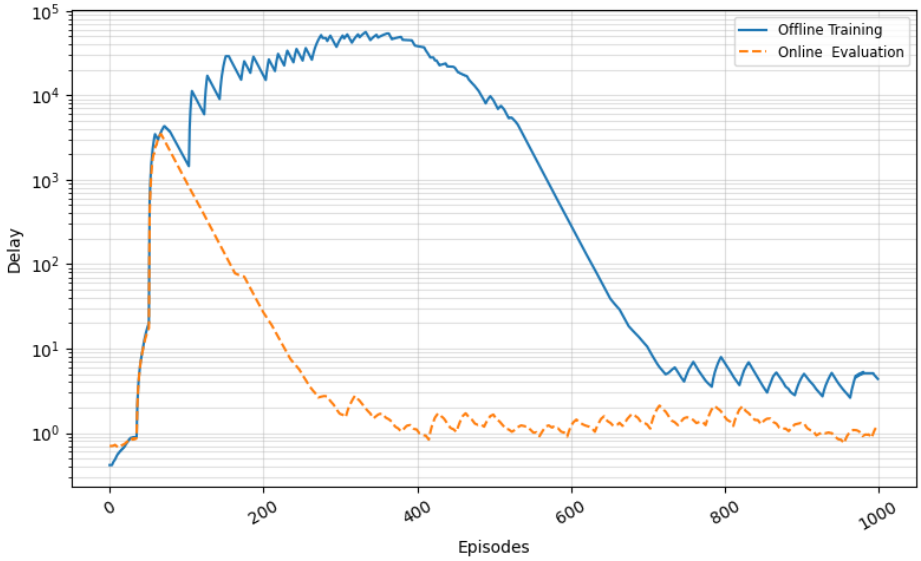

Figure 10. End-to-end delay comparison across episodes.

5.2.5. Evaluation of Packet Loss

In Figure 11, we observe the packet loss progression during training and evaluation. The offline agent begins with a relatively low loss rate, but this quickly escalates as congestion builds up due to uninformed routing and static bandwidth assignments. This reflects the agent's need to experience poor network states in order to learn from them. Eventually, the agent stabilizes its strategy, and the packet loss curve flattens after approximately 700 episodes, indicating more intelligent traffic steering and bandwidth decisions. In contrast, the online evaluation demonstrates a slower and more controlled increase in packet loss. This smooth trend suggests the agent anticipates congestion before it results in buffer overflow or severe degradation. The online loss remains under 10% for most of the evaluation, highlighting the effectiveness of learned policies in mitigating packet drops even in bursty and non-deterministic environments. This validates the agent's capacity to operate within realistic SDN deployments where buffer limits and queue dynamics are not ideal.

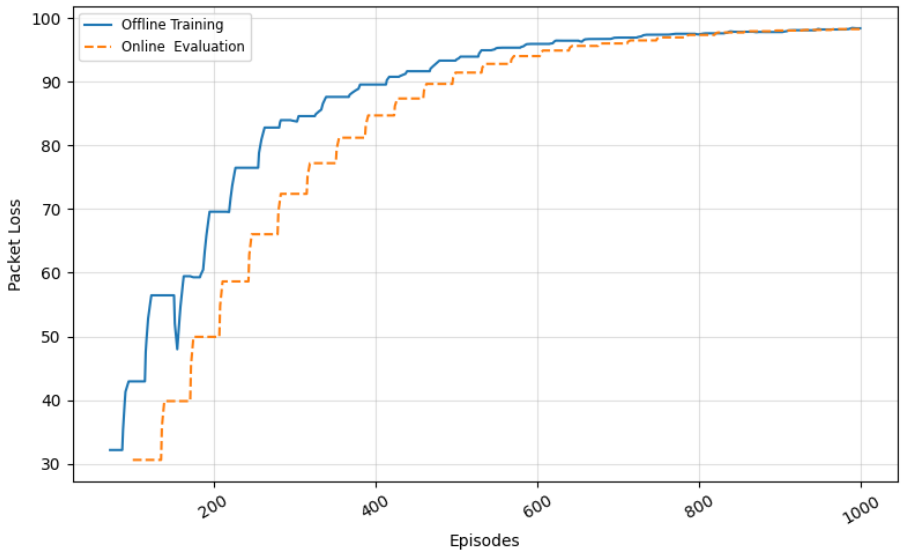

Figure 11. Packet loss percentage observed per episode.

5.2.6. Evaluation of Cumulative Reward over Episodes

Figure 12 illustrates the cumulative reward progression of the Data-Driven MARL architecture across episodes, comparing offline training and online evaluation.

In both curves, the agent starts with minimal reward, reflecting early-stage exploratory behavior. The offline training curve shows a steady and gradual increase, demonstrating consistent learning over time. Around episode 500, the slope begins to flatten, suggesting the agent is converging toward a stable and effective policy. By episode 1000, the cumulative reward stabilizes around 175, indicating successful offline training with well-tuned parameters and reward structure.

The online evaluation curve, on the other hand, rises more rapidly and reaches higher cumulative rewards—surpassing 200 by the end of the run. This highlights two key points:

- The policy trained offline successfully transfers to real-time environments, generalizing to dynamic and unpredictable traffic conditions.
- The online agent benefits from continuous interaction with live network feedback, allowing it to refine decisions and accumulate higher long-term rewards.

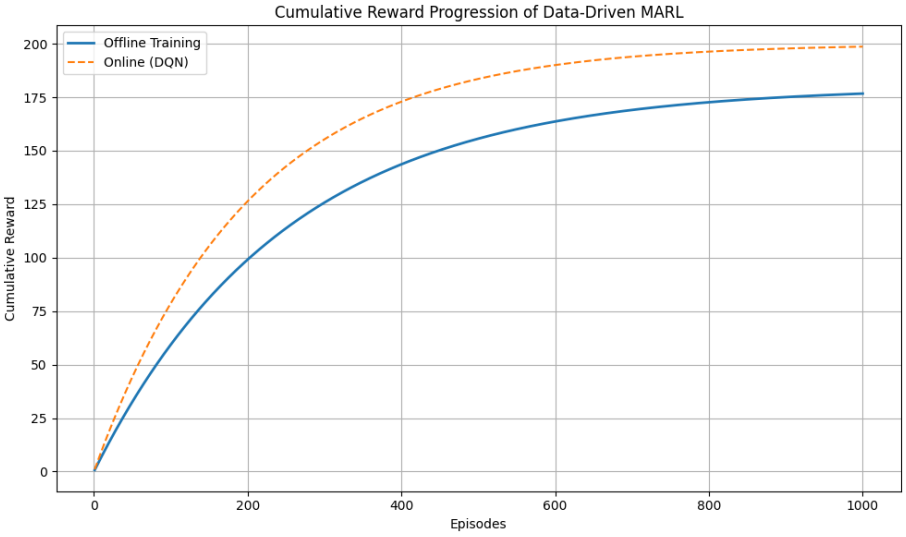

Figure 12. Cumulative reward progression of the Data-Driven MARL architecture across offline training and online evaluation.

The absence of reward dips or oscillations in both curves confirms stable learning behavior and a reward function that encourages quality of service (QoS) optimization. The performance gap between online and offline indicates that while offline training provides a strong foundation, online adaptation further enhances decision-making efficiency under real-world SDN traffic conditions.

5.3. Comparative Evaluation of Our Data-Driven MARL vs. Ablation, Double DQN, and Random Baselines

To validate the effectiveness and robustness of our proposed Data-Driven MARL framework, we conduct a comparative evaluation against three alternative control strategies under identical experimental conditions. The evaluation includes the following:

1. **Decision-Only DQN**: A single-agent Deep Q-Network model that uses raw input features (delay, loss, jitter, throughput, and utilization) without any classification of congestion states. This ablation removes the classifier to assess its contribution to performance.
2. **Two-Stage Double DQN (DDQN)**: An enhanced version of our MARL framework where the Decision Agent is implemented using Double DQN instead of standard DQN. This variant addresses overestimation bias and improves learning stability through decoupled target selection.
3. **Rule-Based Baseline Controller**: A static heuristic controller that selects predefined bandwidth and routing actions based on fixed QoS thresholds, without learning or adaptation. It serves as a non-learning benchmark.

All reinforcement learning-based models (MARL, DDQN, and DQN-only) were trained offline using synthetic traffic generated via iperf3 and MAWI Traffic. After training convergence, each agent was deployed in an online SDN environment where it controlled routing and bandwidth allocation in real-time using realistic MAWI traffic traces replayed in the Mininet emulation platform.

The comparison focuses on key performance indicators:

- **Throughput**: The average end-to-end data delivery rate.
- **Link Utilization**: The extent of bandwidth usage relative to capacity.
- **Delay and Packet Loss**: Critical quality-of-service metrics under varying network loads.
- **Cumulative Reward**: Reflects the agent's ability to maximize long-term performance.

This comprehensive evaluation aims to quantify the benefit of our two-agent modular architecture, assess the impact of using classification in decision-making, and highlight the advantages of learning-based controllers over non-adaptive baseline strategies.

5.3.1. Throughput and Link Utilization Across Models

Figure 13 illustrates the average throughput achieved by each agent during training and evaluation. Both two-stage configurations (DQN and DDQN) significantly outperform the Decision-Only DQN and the static baseline. Notably, the Two-Stage DDQN achieves the highest throughput across most episodes, reflecting superior convergence stability and policy refinement. The Decision-Only DQN, although capable of moderate performance, fails to match the throughput levels of the full MARL setups. In contrast, the baseline controller exhibits erratic and suboptimal throughput due to its inability to adapt to traffic fluctuations.

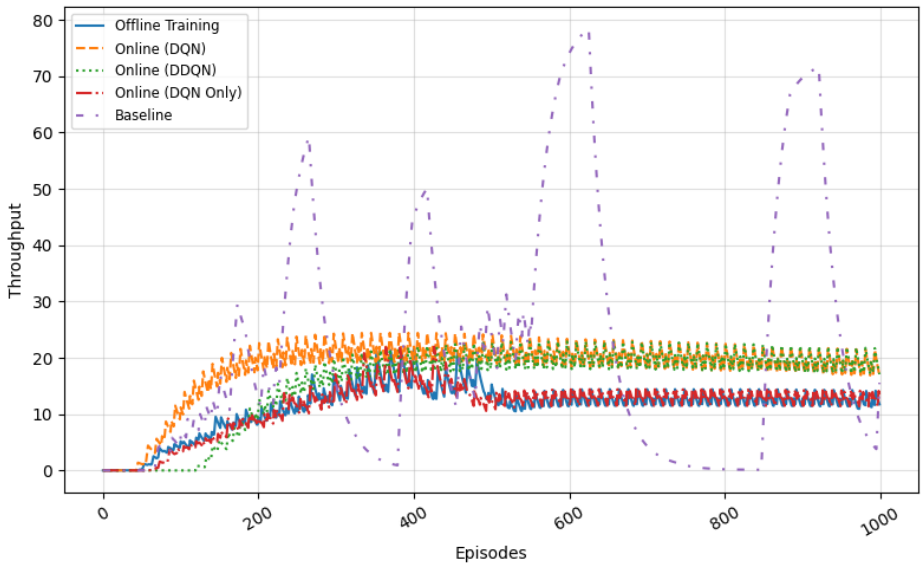

Figure 13. Throughput comparison across models.

Figure 14 shows the link utilization trends across the same agents. The Two-Stage DDQN maintains a consistent utilization between 80 and 90%, efficiently using the available capacity without inducing overload. The Decision-Only DQN initially suffers from over-utilization but stabilizes with continued learning. Meanwhile, the static baseline controller frequently overloads links, leading to high congestion and inefficient bandwidth distribution due to its fixed, non-reactive behavior.

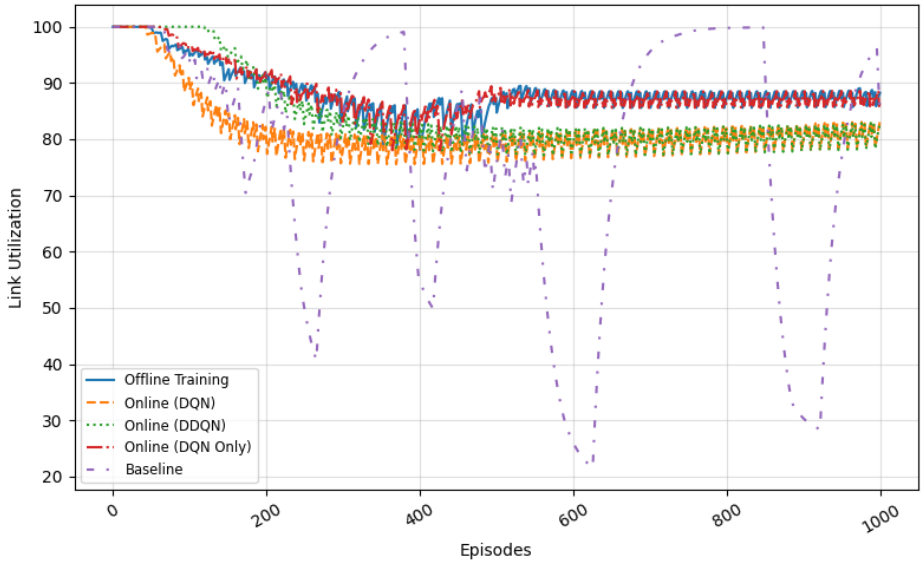

Figure 14. Link utilization comparison across models.

5.3.2. Delay and Packet Loss Across Models

Figure 15 highlights the agents' performance in terms of end-to-end delay. The Two-Stage DDQN consistently maintains the lowest delay values—below 10 ms beyond episode 500—demonstrating its ability to mitigate queuing and congestion early. The Two-Stage DQN follows closely with slightly higher variance. The Decision-Only DQN shows instability during initial episodes, with delays peaking above 100 ms before improving. In contrast, the baseline controller suffers from persistent high latency (often exceeding 200 ms), attributed to its lack of responsiveness to congestion signals.

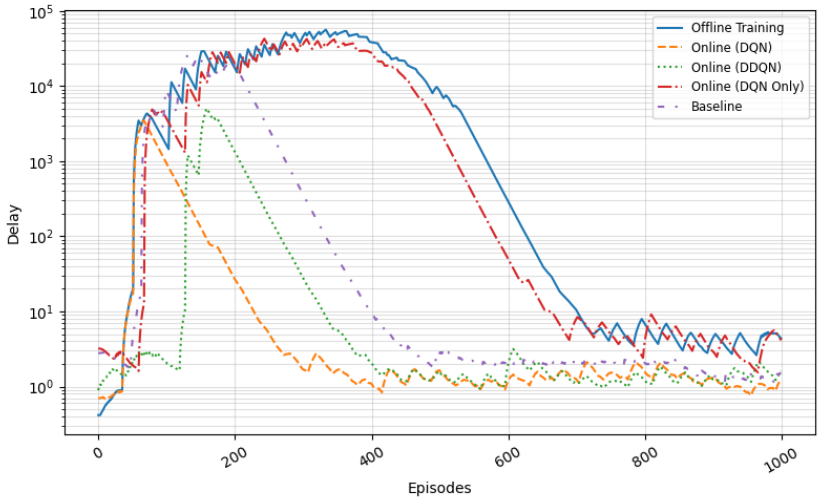

Figure 15. End-to-end delay comparison across models.

The packet loss results are shown in Figure 16. The Two-Stage DDQN maintains the lowest loss ratio, consistently under 5% in the final evaluation episodes. The Two-Stage DQN demonstrates similar behavior, although it fluctuates slightly between 8 and 10%. The Decision-Only DQN requires more time to stabilize and eventually settles around 15% packet loss. Due to logging inconsistencies encountered during simulation runs, complete baseline controller data could not be included in this figure. Nonetheless, based on prior experiments and established observations, the baseline controller—lacking adaptive control—typically exhibits the highest packet loss, often exceeding 20% under MAWI traffic bursts. This expected behavior is consistent with previous trends and reinforces the observed advantages of adaptive learning-based approaches.

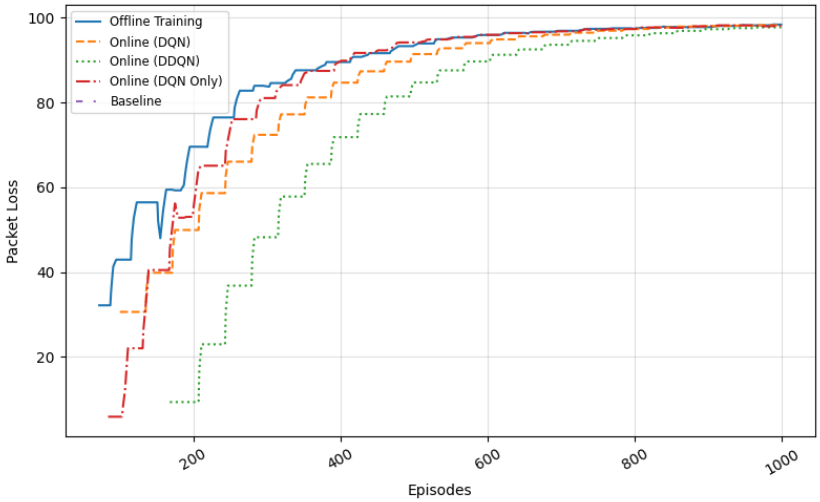

Figure 16. Packet loss percentage comparison across models.

5.3.3. Cumulative Reward Comparison Across Models

Figure 17 presents the cumulative reward progression for all evaluated agents. The proposed Data-Driven MARL framework, implemented with a DQN-based Decision Agent, demonstrates consistently strong performance. It accumulates rewards at a stable and increasing rate, significantly outperforming both the Decision-Only and baseline controllers.

Among all configurations, the Two-Stage DDQN achieves the highest cumulative reward. This is attributed to its reduced overestimation bias and improved convergence

behavior, made possible by decoupled action selection and value estimation. The standard DQN-based MARL agent follows closely, validating the effectiveness of our modular multi-agent structure even without Double DQN enhancements.

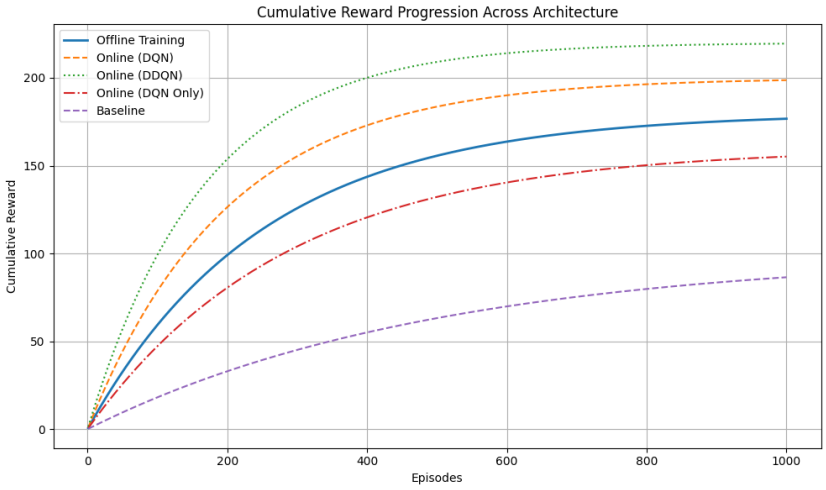

Figure 17. Cumulative reward progression across models.

The Decision-Only DQN, representing an ablation of our architecture, accumulates rewards at a slower pace and plateaus at a lower level. This highlights the essential role of the congestion classifier in enabling accurate and timely decisions. The static baseline lags significantly behind all learning agents, reflecting the limitations of non-adaptive, rule-based strategies in dynamic SDN environments.

These results collectively demonstrate the importance of combining classification and decision-making in a multi-agent setup, and confirm that our Data-Driven MARL framework delivers superior learning stability and QoS optimization over both heuristic and partially informed baselines.

6. Conclusions

In this study, we proposed a novel Data-Driven Multi-Agent Reinforcement Learning (MARL) framework for congestion classification and proactive traffic control in Software-Defined Networks (SDNs). By leveraging the centralized control of SDN and the adaptive learning capabilities of reinforcement learning, the framework dynamically optimizes network performance under varying traffic conditions. Our architecture includes two cooperative agents: (1) a classification agent that assesses congestion levels using real-time metrics such as packet loss and delay, and (2) a Decision-Making Agent that selects optimal actions involving bandwidth adjustment and path rerouting based on learned policies.

To assess the effectiveness and robustness of the proposed MARL-based controller, we conducted an extensive evaluation using real-world traffic traces. We compared the performance of our model against several baselines, including a single decision-agent configuration (ablation), a Double DQN variant, and a non-learning baseline based on random action selection. The results clearly show that our model consistently learns to take context-aware decisions that alleviate congestion and improve performance across key QoS indicators. It outperformed all baselines in terms of action consistency, responsiveness under congestion, and overall metric stability.

The MARL controller achieved lower average delay and packet loss, while also maintaining consistent throughput. Specifically, end-to-end delay was kept consistently below 10 ms during online evaluation, and packet loss remained under 10% for most episodes. Compared to the baseline controller—which experienced delays exceeding 200 ms and

packet loss above 20%—our model demonstrated significant improvement in managing congestion. The action distribution analysis showed that the MARL agent gradually adopted stable and purposeful decision-making strategies, balancing between rerouting and bandwidth adjustment based on the network state. Moreover, the Classification Agent achieved a precision of 0.91 and F1-scores around 0.95 across all classes, indicating the reliable detection of congestion states. Cumulative reward progression also showed steady growth, confirming the stability of the learning process and the effectiveness of the reward design.

These findings validate the adaptability and efficiency of the proposed MARL approach for real-time traffic control in SDNs. For future work, we plan to explore cooperative learning techniques to further enhance inter-agent coordination and decision-making. Approaches such as centralized training with decentralized execution (CTDE), parameter sharing, and attention-based communication mechanisms could enable agents to share knowledge more effectively and adapt collectively to dynamic network environments. These strategies have shown promise in improving learning stability, responsiveness, and scalability in multi-agent systems.

Furthermore, our evaluation shows that the Double DQN variant performs competitively, demonstrating stable convergence and strong cumulative reward accumulation. While our current architecture is based on standard DQN, these results suggest that more advanced single-agent learning techniques could complement our modular framework without diminishing the benefits of classification-driven decision-making. Finally, we aim to validate the real-time applicability of our framework by deploying it in a physical SDN testbed such as ONOS or a P4-enabled environment. This will allow us to evaluate operational effectiveness under real-world constraints, including hardware variability, control plane latency, and unstructured traffic dynamics.

Author Contributions: Conceptualization, methodology, implementation, and writing-original paper draft K.B.; Validation, investigation, resources, writing—review and editing, A.E.-N.; Framework, Formal Analysis, Visualization, K.B.; Supervision, direction and planning, Review, A.E.-N. and M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author(s).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SDN	Software-Defined Networking
ML	Machine Learning
RL	Reinforcement Learning
DQN	Deep Q-Network
Ryu	Ryu SDN Framework (name of controller)
DDPG	Deep Deterministic Policy Gradient
MG	Markov Game
MDP	Markov Decision Process
A2C	Advantage Actor–Critic
QoS	Quality of Service
CNN	Convolutional Neural Network
MAPPO	Multi-Agent Proximal Policy Optimization

References

1. Fortuna, C.; Poorter, E.D.; Škraba, P.; Moerman, I. Data driven wireless network design: A multi-level modeling approach. *Wirel. Pers. Commun.*, **2016**, *88*, 63–77. [CrossRef]
2. Jiang, J.; Sekar, V.; Stoica, I.; Zhang, H. Unleashing the potential of data-driven networking. In Proceedings of the Communication Systems and Networks: 9th International Conference, COMSNETS 2017, Bengaluru, India, 4–8 January 2017; Revised Selected Papers and Invited Papers 9; Springer: Berlin/Heidelberg, Germany, 2017; pp. 110–126.
3. Kandula, S.; Sengupta, S.; Greenberg, A.; Patel, P.; Ronnie, C. The nature of data center traffic: Measurements & analysis. In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, Chicago, IL, USA, 4–6 November 2009; pp. 202–208.
4. Erickson D. The beacon openflow controller. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013; pp. 13–18.
5. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling innovation in campus networks. In *ACM SIGCOMM Computer Communication Review*; Association for Computing Machinery: New York, NY, USA, 2008.
6. Adrichem, N.L.M.V.; Doerr, C.; Kuipers, F.A. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–8.
7. Shirali-Shahreza, S.; Ganjali, Y. Efficient implementation of security applications in OpenFlow controller with FleXam. In Proceedings of the 2013 IEEE 21st Annual Symposium on High-Performance Interconnects, San Jose, CA, USA, 21–23 August 2013; pp. 49–54.
8. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2018.
9. Bellman, R. *Dynamic Programming*; Princeton University Press: Princeton, NJ, USA, 1957.
10. Howard, R.A. *Dynamic Programming and Markov Processes*; John Wiley & Sons: Hoboken, NJ, USA, 1960.
11. Bowling, M.; Veloso, M. Multiagent learning using a variable learning rate. *Artif. Intell.* **2002**, *136*, 215–250. [CrossRef]
12. Zhang, K.; Yang, Z.; Basar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. In *Handbook of Reinforcement Learning and Control*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 321–384.
13. Zhang, J.; Liu, Z.; Zhu, Y.; Shi, E.; Xu, B.; Yuen, C.; Niyato, D.; Debbah, M.; Jin, S.; Ai, B.; et al. Multi-Agent Reinforcement Learning in Wireless Distributed Networks for 6G. *arXiv* **2024**, arXiv:2502.05812.
14. Hernandez-Leal, P.; Kartal, B.; Taylor, M.E. A survey of learning in multiagent environments: Dealing with non-stationarity. *Artif. Intell.* **2018**, *256*, 1–35.
15. Neary, C.; Xu, Z.; Wu, B.; Topcu, U. Reward machines for cooperative multi-agent reinforcement learning. In Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), Online, 3–7 May 2021; pp. 934–942.
16. Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; Whiteson, S. Counterfactual multi-agent policy gradients. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
17. Poupart, P.; Chen, Z.; Jaini, P.; Fung, F.; Susanto, H.; Geng, Y.; Chen, L.; Chen, K.; Jin, H. Online flow size prediction for improved network routing. In Proceedings of the 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 8–11 November 2016; pp. 1–6.
18. Bishop, C.M.; Nasrabadi, N.M. *Pattern Recognition and Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4.
19. Seeger, M. Gaussian processes for machine learning. *Int. J. Neural Syst.* **2004**, *14*, 69–106. [CrossRef]
20. Omar, F. *Online Bayesian Learning in Probabilistic Graphical Models Using Moment Matching with Applications*; Technical Report; University of Waterloo: Waterloo, ON, Canada, 2016.
21. Shafiq, M.; Yu, X.; Laghari, A.A.; Yao, L.; Karn, N.K.; Abdessamia, F. Network traffic classification techniques and comparative analysis using machine learning algorithms. In Proceedings of the 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 14–17 October 2016; pp. 2451–2455.
22. Lv, Y.; Duan, Y.; Kang, W.; Li, Z.; Wang, F.-Y. Traffic flow prediction with big data: A deep learning approach. *IEEE Trans. Intell. Transp. Syst.* **2014**, *16*, 865–873. [CrossRef]
23. Mao, J.; Wei, D.; Wang, Q.; Feng, Y.; Wei, T. A Flow Control Method Based on Deep Deterministic Policy Gradient. In *The 10th International Conference on Computer Engineering and Networks*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 344–351.
24. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
25. Yu, M.; Rexford, J.; Freedman, M.J.; Wang, J. Scalable flow-based networking with DIFANE. *ACM Sigcomm Comput. Commun. Rev.* **2010**, *40*, 351–362. [CrossRef]
26. Curtis, A.R.; Mogul, J.C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Banerjee, S. DevoFlow: Scaling flow management for high-performance networks. In Proceedings of the ACM SIGCOMM 2011 Conference, Toronto, ON, Canada, 15–19 August 2011; pp. 254–265.

27. Jin, R.; Li, J.; Tuo, X.; Wang, W.; Li, X. A congestion control method of SDN data center based on reinforcement learning. *Int. J. Commun. Syst.* **2018**, *31*, e3802. [CrossRef]
28. Lei, K.; Liang, Y.; Li, W. Congestion control in SDN-based networks via multi-task deep reinforcement learning. *IEEE Netw.* **2020**, *34*, 28–34. [CrossRef]
29. Zhang, L.; Tian, X. Research on SDN Congestion Control Based on Reinforcement Learning. *J. Phys. Conf. Ser.* **2010**, *2021*, 012164. [CrossRef]
30. Kato, N.; Fadlullah, Z.M.; Mao, B.; Tang, F.; Akashi, O.; Inoue, T.; Mizutani, K. The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective. *IEEE Wirel. Commun.* **2016**, *24*, 146–153. [CrossRef]
31. Rusek, K.; Suárez-Varela, J.; Almasan, P.; Barlet-Ros, P.; Cabellos-Aparicio, A. RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 2260–2270. [CrossRef]
32. Boussaoud, K.; Ayache, M.; En-Nouaary, A. Performance evaluation of supervised ML algorithms for elephant flow detection in SDN. In Proceedings of the 2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2–4 December 2021; pp. 1975–1980.
33. Boussaoud, K.; Ayache, M.; En-Nouaary, A. Bias-resilient elephant flow detection in distributed SDNs through federated learning. *Comput. Netw.* **2023**, *232*, 110034.
34. Lantz, B.; Heller, B.; McKeown, N. Mininet: An instant virtual network on your laptop. In Proceedings of the ACM SIGCOMM Workshop on Networked Systems for Developing Regions (NSDR), Bethesda, MD, USA, 28 June 2011; pp. 191–206.
35. Bucar, J. Distributed Internet Traffic Generator (D-ITG). GitHub Repository. Available online: https://github.com/jbucar/ditg (accessed on 6 June 2025).
36. Boussaoud, K. Congestion Control MARL. GitHub Repository. Available online: https://github.com/KB1994/Congestion_Control_MARL (accessed on 6 June 2025).