Journal of Trends in Computer Science and Smart Technology (ISSN: 2582-4104)
www.irojournals.com/tcsst/

Adaptive Markov Decision Process-Based Congestion Control Using Deep Deterministic Policy Gradient in Distributed MQTT Brokers

Snowlin Preethi Janani[1], Immanuel Johnraja J.[2], Getzi Jeba Leelipushpam P.[3]

Division of Computer Science and Engineering, Karunya Institute of Technology and Sciences, Coimbatore, India.

Email: [1]snowlinfrank@gmail.com, [2]immanueljohnraja@karunya.edu, [3]getzi@karunya.edu

Abstract

The publish-subscribe variant of the Message Queuing Telemetry Transport (MQTT) protocol is lightweight enough to permit the asynchronous sending of messages. Nonetheless, the protocol is experiencing issues related to congestion, which is caused by the IoT traffic characteristics in the distributed broker networks in the IoT environment. As a result, delay occur resulting in a degradation of QoS. However, using classical AIMD and static rate-limiting techniques for congestion control in the IoT environment is becoming difficult. The proposed work provides an adaptive Reinforcement Learning (RL)-based system to handle congestion in the IoT environment. This system can be represented using a Markov Decision Process (MDP). The DDPG algorithm will allow the RL agent to discover optimal strategies in the IoT environment, and it is beneficial in managing non-stationary IoT traffic. Based on the agent clusters' simulation of the brokers, the RL-based system outperforms the traditional system in the IoT setting. It achieves this as the message drop rate is reduced by 38-40%, the latency is reduced by 27-32%, and the fairness of the load is increased by 18-24%.

Keywords: MQTT, Internet of Things, Distributed Broker Networks, Congestion Control, Reinforcement Learning, Markov Decision Process, Quality of Service, Traffic Management.

1. Introduction

The Internet of Things (IoT) technology model has developed into a huge cyber-physical system that produces diverse data traffic in smart cities, medical industrial automation, and self-governing vehicles, with over 29 billion IoT devices predicted to be deployed globally by the year 2030 [1]. This tremendous growth is imposing an uncertain amount of pressure on telecommunication systems, especially at the application layer, where lightweight protocols must be implemented to support constrained devices and low-bandwidth environments. Among the various protocols developed, the Message Queuing Telemetry Transport (MQTT) protocol has been recognized as the industrial standard due to its publish-subscribe model, low overhead, and lossy connection compatibility. Nevertheless, as the density of IoT devices and data traffic increases, IoT systems based on the MQTT protocol experience significant

Received: 16.01.2026, received in revised form: 14.02.2026, accepted: 28.02.2026, published: 09.03.2026

problems in terms of congestion, scalability, latency, packet loss, and energy efficiency [2]. System stability and Quality of Service (QoS) depend on the aforementioned problems. Recent research has demonstrated that when the level of traffic is high, the level of congestion in IoT networks, especially in broker-based networks, could cause a rise in end-to-end latency by 30-60% and a loss of packets of more than 25% [3]. Although analytical models have been proposed to predict the loss of messages and the level of delay in MQTT and MQTT-SN networks, these models are used mainly to evaluate the performance of these networks rather than to mitigate congestion in these networks [4]. Although distributed broker-based networks and authentication schemes have been investigated to improve the performance of these networks, the effectiveness of these schemes in high-traffic networks has not been fully explored. MQTT has been used in smart city and healthcare application scenarios, demonstrating its feasibility. However, these networks lack adaptive schemes that could address emergency situations [6],[7].

Network congestion and communication limitations play a vital role in real-time decision-making and system throughput in the context of IIoT and ITS. In support of application-level optimization, existing techniques tend to abstract communication-layer limitations, e.g., broker overload and traffic dynamics [8], [9]. Classical congestion control techniques, e.g., Additive Increase Multiplicative Decrease (AIMD), are not applicable to MQTT communication and various IoT traffic patterns, despite the fact that these techniques provide theoretical reliability guarantees [10]. Load balancing and cloud-edge orchestration techniques are also employed to optimize resource utilization, but these techniques do not consider fine-grained MQTT traffic awareness and real-time congestion feedback in the context of system control decisions [11], [12]. However, research has indicated that MQTT-based IoT systems face the threat of illegal access, hijacking of the broker, and denial of service. Although the use of ML and distributed security frameworks improves intrusion detection and the robustness of the protocols, they also contribute to increased computation and communication expenses, which may worsen the latency and congestion problems [13], [17]. The scalability and congestion control performance of ML-based IoT systems remain unknown; however, broker-less IoT systems have been proposed to eliminate the broker bottleneck, unlike traditional MQTT-based IoT systems [14].

Recently, Deep Reinforcement Learning (DRL) and Federated Learning (FL) techniques have been employed in the optimization of networks, showing improvements in throughput of up to 20-35% in simulated scenarios [15], [18]. Nevertheless, these techniques face disadvantages such as high training complexity, unstable convergence, and high communication overhead, which make their application difficult in IoT systems [16]. Moreover, the application of AI-based traffic management frameworks has shown the potential of using these techniques but has failed to provide MQTT-based solutions, along with a lack of real-world implementation scenarios [19]. Furthermore, QoS-based multilayer architecture and adaptive QoS control techniques provide better prioritization of traffic but add to the complexity of the system, failing to offer scalability at extreme densities of IoT devices [21], [22].

Despite extensive research across congestion control, QoS management, security, and intelligent networking, three research gaps remain:

- An RL-based congestion control framework tailored for MQTT broker networks, explicitly modeling IoT-specific traffic patterns such as bursty sensor data.

- A decentralized coordination mechanism using Federated Reinforcement Learning (FRL) [18], enabling brokers to share learned policies while avoiding single points of failure, an essential requirement for mission-critical IIoT deployments.
- Insufficient real-world scalability validation under heterogeneous and bursty traffic conditions [23-25].

These limitations motivate the need for a scalable, congestion-aware, and intelligence-driven MQTT framework that can dynamically adapt to traffic variations while maintaining low overhead and high QoS guarantees in large-scale IoT environments.

2. Literature Review

Existing literature on IoT and MQTT-based systems exposes multiple unresolved limitations across scalability, congestion control, intelligence, and real-world applicability. Market-oriented IoT forecasting studies emphasize deployment growth but fail to analyze protocol-level scalability, congestion dynamics, or communication bottlenecks arising from massive device density [1]. Application-layer security surveys for MQTT focus on threats and protection mechanisms but do not evaluate the trade-offs between security enforcement, congestion, and energy overhead [2]. Continuous congestion monitoring approaches improve detection accuracy but rely on reactive mechanisms without predictive modeling or adaptive traffic regulation [3]. Delay and packet loss estimation techniques for MQTT-SN provide performance insights but do not integrate congestion mitigation or control actions [4]. Reviews of authentication mechanisms for distributed MQTT brokers enhance trust models but lack scalability analysis under heavy traffic and broker overload conditions [5]. Open-source MQTT-based smart city platforms validate architectural feasibility but do not assess resilience to congestion, QoS degradation, or broker saturation in large-scale deployments [6].

MQTT-based healthcare monitoring systems demonstrate applicability in medical IoT but overlook congestion-aware prioritization and reliability guarantees during emergency data flows [7]. Industrial IoT block detection frameworks focus on production-line optimization while treating communication congestion and MQTT traffic behavior as secondary concerns [8], [9]. AIMD-inspired network control strategies provide theoretical congestion stability but are not customized for publish/subscribe MQTT architectures or heterogeneous IoT traffic patterns [10]. Surveys on IoT load balancing techniques improve resource utilization but do not integrate real-time congestion feedback from MQTT brokers into balancing decisions [11]. Cloud–edge scoring assessments concentrate on deployment architectures without examining how scoring choices influence MQTT traffic congestion and end-to-end latency [12].

Distributed machine learning-based MQTT security enhancements improve robustness but introduce additional communication and computational overhead without evaluating congestion impact [13]. Broker-less and decentralized IoT platform designs reduce dependency on central brokers but lack comparative evaluation against MQTT systems under congestion-intensive workloads [14]. Deep reinforcement learning-based network topology optimization assumes high computational capability and stable environments, limiting feasibility for resource-constrained IoT systems [15]. Systematic reviews of DDPG algorithms highlight adaptability but also reveal high training complexity and convergence instability for real-time IoT congestion control [16]. Secure MQTT data distribution architectures enhance confidentiality but do not analyze throughput degradation or broker-side congestion effects caused by encryption overhead [17].

Federated Reinforcement Learning frameworks reduce centralized training but introduce synchronization and communication costs that may worsen congestion in bandwidth-limited IoT networks [18]. AI-driven traffic management frameworks propose intelligent congestion handling but remain largely conceptual, lacking MQTT-specific implementation and real-world validation [19]. Analyses of MQTT vulnerabilities and attack vectors focus on threat identification rather than congestion-resilient protocol designs under attack scenarios [20]. QoS-aware multilayer IoT service architectures improve traffic diversity but increase architectural complexity and lack scalability validation for dense IoT deployments [21]. Adaptive QoS control mechanisms for MQTT-SN enhance service differentiation but operate without network-wide congestion awareness [22]. Cross-layer congestion control schemes for the Internet of Medical Things prioritize data effectively but remain domain-specific and are not generalized for heterogeneous IoT environments [23]. Broker-level visualization and monitoring tools improve observability but do not provide autonomous congestion mitigation or adaptive traffic control [24]. Energy-aware clustering approaches optimize power consumption but do not incorporate real-time congestion feedback or MQTT communication overhead [25].

3. Methods

The architecture designed for congestion control through the use of closed-loop reinforcement learning, applicable to the distributed MQTT broker network under extremely dynamic IoT traffic conditions, is shown in Figure 1.

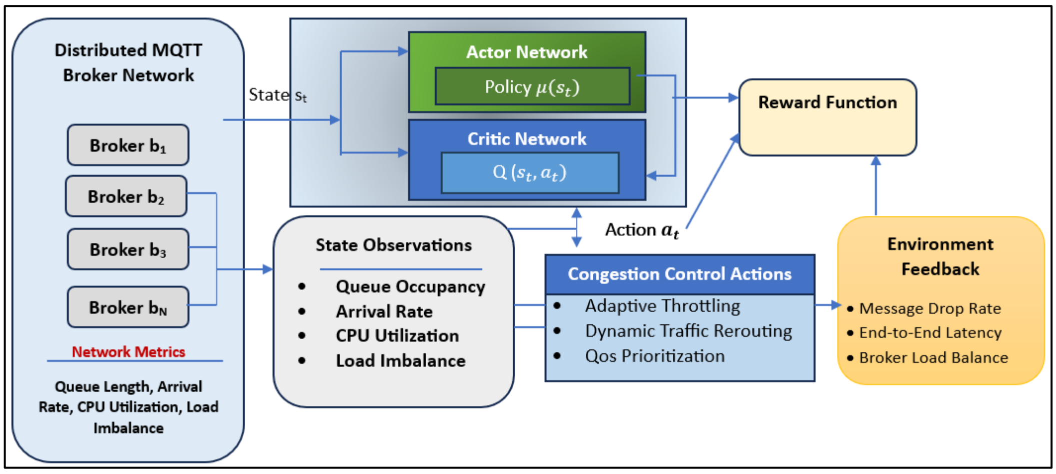

Figure 1. RL-Driven Adaptive Congestion Control Architecture for Distributed MQTT Broker Networks

The distributed broker layer senses the system states including the queue system states, the rate of message arrival, CPU, and the imbalance between brokers and uses this to obtain the overall system state vector s_t. The Deep Deterministic Policy Gradient (DDPG) reinforcement learning-based intelligent agent also senses the system states. The deployment of the actor network serves to learn the deterministic policy $\mu(s_t)$ that determines how actions are selected according to the various states of the system. Moreover, the deterministic policy maps out the actions to the states being learned. The critic network employs the state-action value function to evaluate the quality of actions, denoted as (s_t, a_t). The composite reward signal is used to optimize the policy. The composite reward signal is based on input signals obtained from the environment. The input signals include the ratio of dropped messages, total latency, and broker usage balance.

3.1 State Observation and Environment Modelling

The MQTT broker monitors the congestion performance indicators of the participating brokers in a distributed manner for each decision interval t. The overall condition of the system is characterized by these performance indicators and forms the basis of the decision process of the learning agent. The overall state vector of the environment is represented in Equation (1).

$$s_t = [Q_1(t), \dots, Q_N(t), \lambda_1(t), \dots, \lambda_N(t), U_1(t), \dots, U_N(t), \Delta_{\text{load}}(t)] \tag{1}$$

The state combines congestion indicators at the broker level with an imbalance metric at the network level $Q_i(t)$. The queue size at the broker $b_i\lambda_i(t)$ captures the delay caused by buffering and queuing at time t. The message arrival rate $U_i(t)$ is the volume of traffic arriving from the upstream brokers and publishers. $\Delta_{\text{load}}(t)$ is the effect of computation on the effective capacity of the computer system, that is CPU utilization also considered. Global broker load imbalance refers to the network's uneven use.

In the case of congestion control based on queues, the queue length, the arrival rate of messages, and CPU usage can be considered well-defined sufficient statistics. These measurements correctly represent the buffering capacity, traffic, and effective CPU capacity of the system, respectively. When the global coordination variable is included in the above measurements, the system can sense the hotspots of the congestion with the aid of the variable load imbalance. This ensures that the assumption of MDP is satisfied with the conditional independence of the states of the system with respect to the state and actions taken.

The chosen state variables collectively provide adequate measurements of congestion to achieve optimal control of the distributed MQTT broker networks. The queue length $(Q_i(t))$ directly reflects the buffering delay and the potential risk of queue overflow, which are major indicators of congestion. The instantaneous traffic strain from publisher and upstream broker is captured in the message arrival rate $(\lambda_i(t))$. Assessment of CPU load estimates $U_i(t)$ the actual service capacity of different brokers in changing workloads $(\Delta_{\text{load}}(t)$. The load imbalance metric detects congestion hotspots within the broker network while also considering overall fairness. These metrics can effectively predict the evolution of the next state queues without the need to acquire additional historical data satisfying the Markov property. This state formulation follows the principles of queue theory and can be proven to converge favorably in learning while improving performance in various conditions.

This state formulation provides the effects of congestion locally to the control policy while also considering global coordination. The load factor of the broker b_i at a given time t can be defined in equation (2).

$$\rho_i(t) = \frac{\lambda_i(t)}{\mu_i(t)} \tag{2}$$

where $\mu_i(t)$denotes the service rate of the broker. The load factor expresses the ratio between incoming traffic and processing capacity:

- $\rho_i(t) < 1$: Stable operating region.
- $\rho_i(t) \approx 1$: Saturated broker.
- $\rho_i(t) > 1$: Overloaded broker with growing queues.

The average load across all brokers is given in equation (3).

$$\bar{\rho}(t) = \frac{1}{N}\sum_{i=1}^{N} \rho_i\,(t) \tag{3}$$

To capture traffic imbalance, the load variance across brokers is computed as shown in equation (4).

$$\Delta_{\text{load}}(t) = \frac{1}{N}\sum_{i=1}^{N}(\rho_i(t) - \bar{\rho}(t))^2 \tag{4}$$

A low $\Delta_{\text{load}}(t)$ indicates balanced utilization and efficient resource sharing. A high $\Delta_{\text{load}}(t)$ reveals localized congestion hotspots, increasing the risk of buffer overflow, latency spikes, and cascading failures.

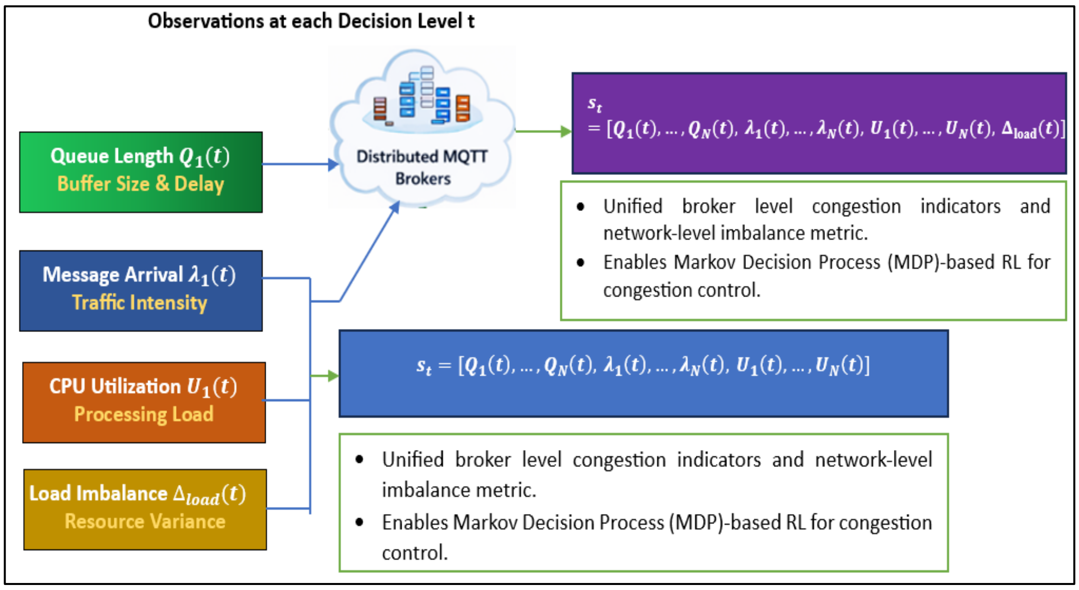
Observations at each Decision Level t
Queue Length $Q_1(t)$
Buffer Size & Delay
Message Arrival $\lambda_1(t)$
Traffic Intensity
CPU Utilization $U_1(t)$
Processing Load
Load Imbalance $\Delta_{load}(t)$
Resource Variance
Distributed MQTT Brokers
$s_t = [Q_1(t), \dots, Q_N(t), \lambda_1(t), \dots, \lambda_N(t), U_1(t), \dots, U_N(t), \Delta_{\text{load}}(t)]$
Unified broker level congestion indicators and network-level imbalance metric.
Enables Markov Decision Process (MDP)-based RL for congestion control.
$s_t = [Q_1(t), \dots, Q_N(t), \lambda_1(t), \dots, \lambda_N(t), U_1(t), \dots, U_N(t)]$
Unified broker level congestion indicators and network-level imbalance metric.
Enables Markov Decision Process (MDP)-based RL for congestion control.

Figure 2. Environment State Representation for RL-Based Congestion Control in Distributed MQTT Broker Networks

Figure 2 illustrates the environment state representation in a distributed MQTT broker network for reinforcement learning-based congestion control. At each decision interval t, each broker b_i reports key congestion-related metrics: queue length $Q_i(t)$ message arrival rate $\lambda_i(t)$, CPU utilization $U_i(t)$, and the global load imbalance $\Delta_{load}(t)$. This data is combined into a state vector s_t, which includes local broker-level congestion as well as overall network-level imbalance, to help guide the learning agent in decision-making processes. Such a holistic approach to system state ensures that the Markov property is met, thereby enabling accurate prediction of system state in the future, which is critical in controlling queue buildup, latency, message loss, and uneven load distribution in the broker network.

The overall decision-making process by the reinforcement learning agent is ensured to be smooth, avoiding oscillations in control actions, by executing at a constant decision interval, denoted by $\Delta t = 100$ ms, which represents the overall message scheduling process within MQTT brokers. At every decision interval, each broker provides four scalar metrics, namely, load imbalance, CPU usage, message arrival rate, and queue length, resulting in an overall O(B) reporting overhead, where B represents the number of brokers within the distributed network.

3.2 Action Generation by Actor Network

Given the observed system state s_t, the congestion control policy is realized through an actor network that maps the high-dimensional state space to a set of continuous control actions. The actor represents a deterministic policy parameterized by θ_μ is shown in equation (5).

$$a_t = \mu(s_t \mid \theta_\mu) \tag{5}$$

where a_t denotes the action vector applied at time t, and $\mu(\cdot)$is a nonlinear function approximated using a deep neural network. This design enables fine-grained and differentiable control over broker behavior.

The actor network that depicts the state of the system as well as the corresponding actions in the continuous congestion management process follows a fully connected feedforward neural network architecture. On the other hand, dimension 4B, where B denotes the number of brokers, depicts the state vector that will be used to connect the input layer of the actor network. Furthermore, the two hidden layers that follow the ReLU activation functions contain 256 and 128 neurons to represent the congestion dynamics in the nonlinear system. The throttling, rerouting, and the QoS prioritizing actions are represented by the three neurons that follow the sigmoid activation functions in the output layer, denoted as 3B. This is due to the low computational costs of the architecture that can be used in the distributed MQTT broker system, as depicted in Figure 3.

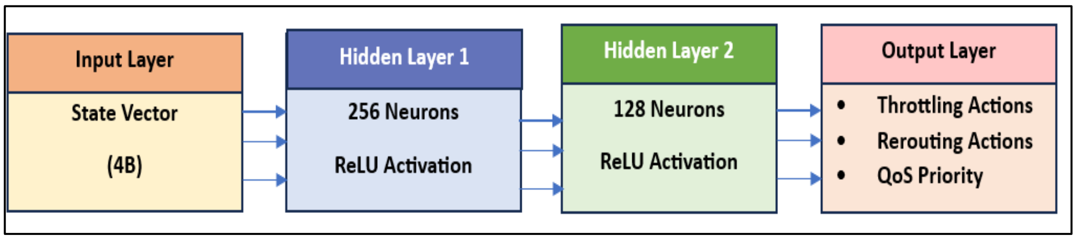

Figure 3. Block Diagram of the Actor Neural Network for Adaptive Congestion Control in Distributed MQTT Broker Networks

The action vector consists of three complementary control components as shown in equation (6), each targeting a different aspect of congestion mitigation.

$$a_t = [\alpha_i(t),\ \beta_{ij}(t),\ \pi_q(t)] \tag{6}$$

Adaptive Throttling Coefficient $\alpha_i(t)$ is given in equation (7).

$$\alpha_i(t) \in [0,1] \tag{7}$$

This parameter dynamically regulates the incoming traffic rate at broker b_i:

- $\alpha_i(t) = 1$: no throttling (full admission of incoming messages),
- $\alpha_i(t) < 1$: partial throttling to suppress excess traffic during congestion,
- $\alpha_i(t) \rightarrow 0$: aggressive traffic reduction under severe overload.

The throttling action directly modifies the effective arrival rate as shown in equation (8), thereby controlling queue growth and preventing buffer overflow.

$$\lambda_i'(t) = \alpha_i(t) \cdot \lambda_i(t) \tag{8}$$

The rerouting parameter $\beta_{ij}(t)$determines the fraction of traffic redirected from an overloaded broker $b_i$to a neighboring broker b_j. Higher values of $\beta_{ij}(t)$ shift the load away from congestion hotspots. Lower values preserve locality and minimize forwarding overhead. This action enables spatial load balancing across the distributed broker network and helps reduce the global load imbalance $\Delta_{\text{load}}(t)$. The QoS weight $\pi_q(t)$ assigns relative service priority to MQTT message classes (QoS 0, QoS 1, QoS 2). Higher $\pi_q(t)$ values favor latency-sensitive or reliability-critical messages. Lower values deprioritize best-effort traffic during congestion.

The parameter for rerouting $\beta_{ij}(t)$ indicates the percentage of traffic that is rerouted at decision interval t from broker i to broker j. The rerouting action is limited as follows to guarantee traffic conservation and avoid packet duplication: $0 \le \beta_{ij}(t) \le 1$, $\sum_j \beta_{ij}(t) \le 1$. These limitations ensure that no broker's total diverted traffic exceeds its incoming traffic. A sigmoid activation function is used in the actor network to enforce the limits, and then nearby brokers are normalized. All routing choices are guaranteed to be real and adhere to the standard MQTT broker forwarding semantics of this architecture.

Every congestion control measure is made to closely adhere to the requirements of the MQTT protocol. To execute traffic throttling, brokers modify client entry and message acceptance rates without executing packet drops at the protocol level. Standard MQTT broker bridging technologies are used to provide traffic rerouting while maintaining publish/subscribe semantics. Without changing message delivery assurances, QoS prioritization preserves the native MQTT QoS levels (QoS 0, QoS 1, and QoS 2). Because of this, the proposed framework is implemented on current broker infrastructures without requiring changes to the MQTT protocol.

The proposed method is based on the distributed execution and centralized training model. A stable policy is learned by aggregating global state data during training. There is no requirement for centralized decision-making at runtime since the learned policy is implemented independently at each broker during deployment using only locally observable data about the state.

Figure 4 illustrates how a deep reinforcement learning actor network generates congestion control actions in a distributed MQTT broker network. The observed system state s_t, comprising queue length, arrival rate, CPU utilization, and load imbalance, is fed into the actor network, which outputs a continuous action vector. This vector includes three complementary control components: (i) the adaptive throttling coefficient, dynamically regulating incoming traffic to prevent queue buildup; (ii) the rerouting parameter, enabling spatial load balancing by redirecting traffic from congested brokers; and (iii) the QoS weight, prioritizing critical messages to minimize latency and packet loss. By jointly optimizing these actions, the actor network ensures smooth, QoS-aware congestion control while maintaining fairness and efficiency across the broker network.

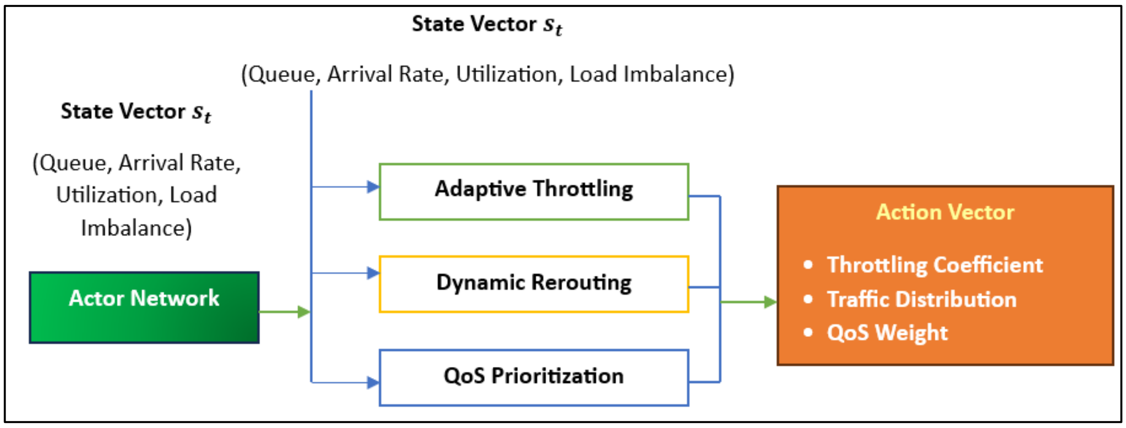

Figure 4. Action Generation by Actor Network in RL-Based MQTT Congestion Control

3.3 Queue Dynamics and Congestion Evolution

After the actor network applies the selected control actions at time t, the distributed MQTT broker network evolves according to its intrinsic queueing dynamics. This step captures how congestion develops or is mitigated over time as a direct consequence of the chosen actions. The queue length at the broker b_i at the next decision interval $t+1$is given in equation (9).

$$Q_i(t+1) = \max\,[Q_i(t) + \lambda_i'(t) - \mu_i(t),\, 0] \tag{9}$$

where:

- $Q_i(t)$denotes the current queue occupancy.
- $\lambda_i'(t)$is the effective arrival rate after applying throttling and rerouting actions.
- $\mu_i(t)$ represents the service rate, i.e., the number of messages processed and forwarded during the interval.

This formulation reflects a discrete-time fluid queue model, where the queue grows if arrivals exceed service capacity and shrinks otherwise. The non-negativity constraint ensures the physical feasibility of the queue length. The congestion interpretation is as follows:

- If $\lambda_i'(t) > \mu_i(t)$, the queue accumulates, signaling increasing congestion and rising latency.
- If $\lambda_i'(t) < \mu_i(t)$, the queue drains, indicating effective congestion relief.
- When $\lambda_i'(t) \approx \mu_i(t)$, the system operates near saturation, where small traffic fluctuations may cause instability.

Two auxiliary queue thresholds are established as fixed fractions of the broker buffer capacity for the purpose of making congestion regions easier to understand and visualize: $Q_{\text{min}} = 0.3\, Q_{\text{buffer}}$ and $Q_{\text{max}} = 0.8\, Q_{\text{buffer}}$ these thresholdsrepresent lightly loaded, congested, and overloaded operational areas. They were empirically confirmed by stress testing under high traffic situations. As shown in Figure 5, these characteristics are only utilized for analytical interpretation and visualization; the reinforcement learning agent does not rely on fixed threshold-based judgments.

Thus, queue evolution serves as the primary indicator of congestion severity at each broker. It is assumed to have a finite buffer capacity $Q_{\max}$. When the queue length exceeds this limit, message drops occur according to equation (10).

$$D_i(t) = \begin{cases} Q_i(t) - Q_{\max}, & \text{if } Q_i(t) > Q_{\max}, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

This model explicitly quantifies the number of messages lost due to buffer overflow. The term $Q_i(t) - Q_{\max}$ represents excess messages that cannot be accommodated. Drops directly degrade throughput and reliability, particularly for MQTT QoS 1 and QoS 2, which may trigger retransmissions.

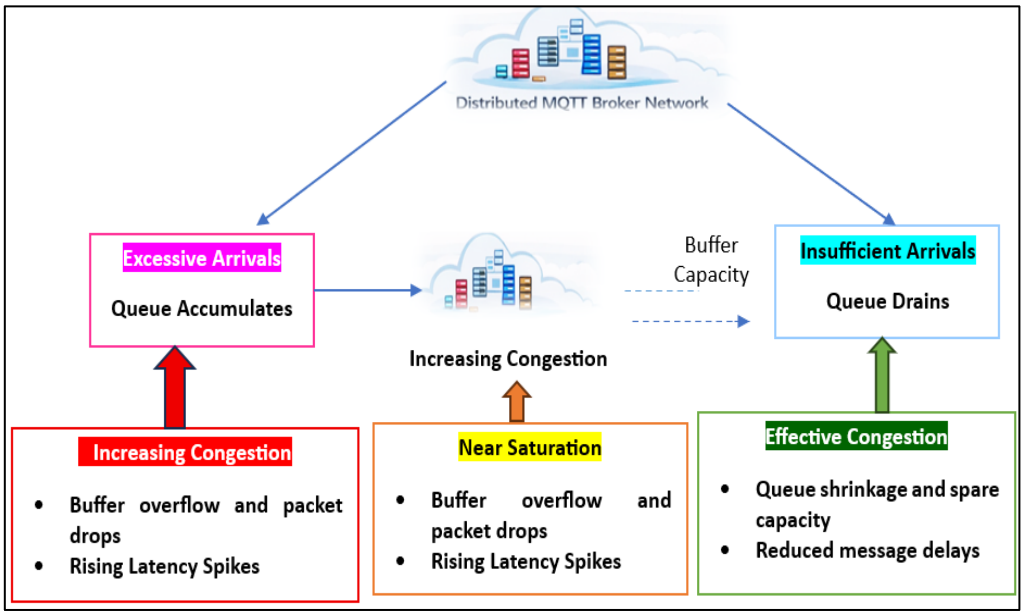

Figure 5. Queue Dynamics and Congestion Evolution in Distributed MQTT Brokers

As shown in Figure 5, the queues of the distributed MQTT brokers change according to the control actions from the actor network. The queue at the broker grows when the rate of incoming information is higher than the rate at which the broker can process it. Conversely, the queue decreases when the broker processes information faster than new information arrives. Because the queues have a limited size, they can overflow, resulting in lost information and lower system performance. The way the queues change is the main way to measure congestion in the system, linking the actor's control actions to the quality of the results, which helps in reinforcement learning.

3.4 Environment Feedback and Performance Metrics

After the queue dynamics evolve, the environment provides feedback to the learning agent in the form of performance metrics that quantify the quality of the applied control actions. These metrics capture both user-perceived QoS and system-level stability. End-to-end latency is assessed at client endpoints and is defined as the time difference between message reception and publishing, given by $L = T_{receive} - T_{publish}$. As observed in equation (11), delay is analytically modeled utilizing queue occupancy and service rate in addition to empirical timestamp-based measurement. Little's Law is followed by this queue-based formulation, which offers a reliable estimate of the observed end-to-end delays utilized in assessment. The end-to-end latency experienced by messages at broker b_i is modeled as being proportional to the queue length is given in equation (11).

$$L_i(t) \propto \frac{Q_i(t)}{\mu_i(t)} \tag{11}$$

where:

- $Q_i(t)$ represents the queue occupancy, corresponding to waiting time before service.
- $\mu_i(t)$ denotes the service rate, determining how quickly queued messages are processed.

The larger queues result in higher latency due to increased waiting time. Higher service rates mitigate latency even under moderate load. Thus, queue length serves as a direct and measurable proxy for latency in the feedback signal. In addition to latency, the system continuously monitors load fairness across the distributed broker network using the load imbalance metric as shown in equation (12).

$$\Delta_{\text{load}}(t) \tag{12}$$

A high $\Delta_{\text{load}}(t)$ signals uneven traffic distribution, where some brokers are overloaded while others are underutilized, potentially causing cascading congestion. Together, latency, queue occupancy, message drops, and load imbalance form a comprehensive environment feedback signal.

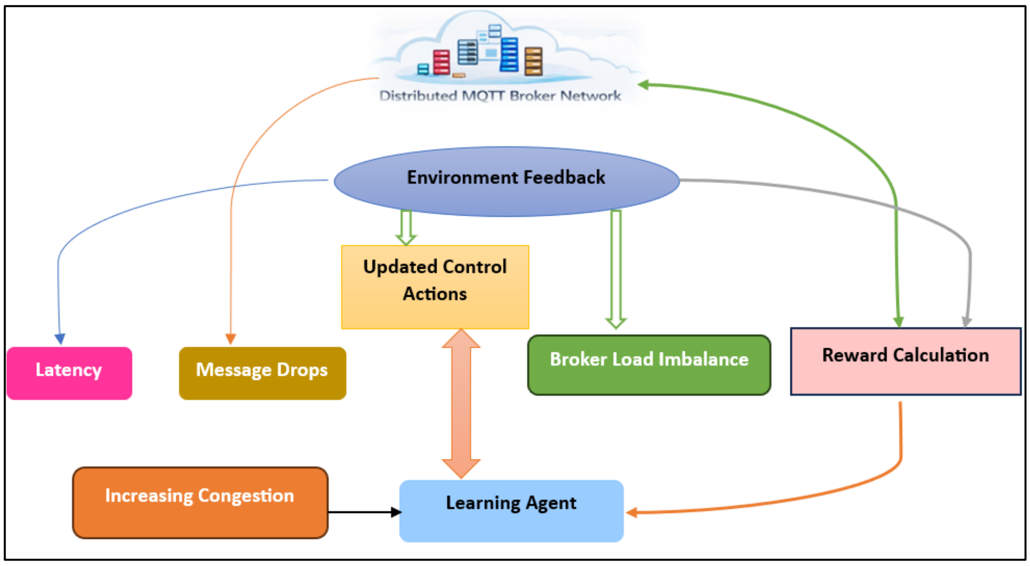

Figure 6. Environment Feedback and Performance Metrics in Distributed MQTT Congestion Control

Figure 6 illustrates that the distributed MQTT broker network provides environment feedback to the reinforcement learning agent for congestion control. After queue dynamics evolve, the system measures key performance metrics including latency $L_i(t) \propto Q_i(t)/\mu_i(t)$, message drops $D_i(t)$ due to buffer overflow, and broker load imbalance $\Delta_{\text{load}}(t)$. These metrics collectively form a feedback signal that evaluates the immediate impact of control actions on user-perceived QoS and network fairness. The feedback is used by the learning agent to update control policies, optimizing throttling, rerouting, and QoS prioritization to minimize latency, packet loss, and unfair load distribution while ensuring overall system stability.

3.5 Reward Computation

To guide the learning agent toward congestion-aware and fair control policies, the reward function is explicitly designed to penalize undesirable network conditions such as message loss, excessive delay, and uneven broker utilization. At each decision interval t, the scalar reward is defined as shown in equation (13).

$$r_t = -(w_1 \sum_{i=1}^{N} D_i\,(t) + w_2 \sum_{i=1}^{N} L_i\,(t) + w_3\, \Delta_{\mathrm{load}}(t)) \tag{13}$$

where the negative sign ensures that higher congestion levels result in lower rewards. Reward components are as follows.

- Message Drop Penalty $\sum_{i=1}^{N} D_i\,(t)$ captures reliability degradation due to buffer overflow. Penalizing drops discourages aggressive throttling or poor load distribution that could lead to packet loss, particularly critical for MQTT QoS 1 and QoS 2 traffic.
- Latency Penalty $\sum_{i=1}^{N} L_i\,(t)$ component reflects end-to-end message delay across all brokers. Minimizing latency improves responsiveness and user-perceived QoS, especially for time-sensitive IoT applications.
- Load Imbalance Penalty $\Delta_{\mathrm{load}}(t)$ enforces fairness by discouraging uneven resource utilization. Penalizing load variance prevents persistent congestion hotspots and promotes stable, balanced operation across the broker network.

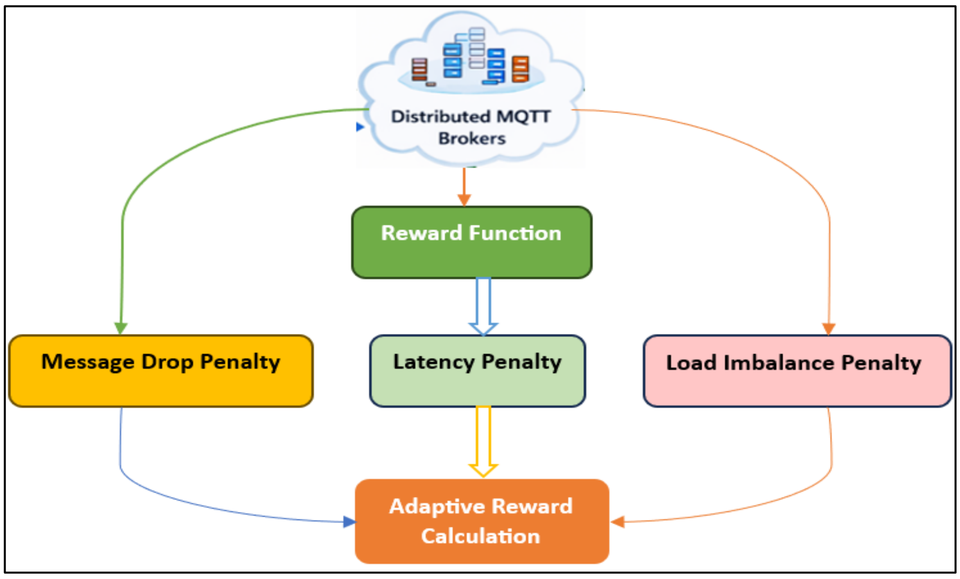

Figure 7. Reward Computation in Distributed MQTT Congestion Control

As shown in Figure 7, the reward computation module quantifies the effectiveness of congestion control actions in a distributed MQTT broker network. It integrates three key performance metrics, message drops, end-to-end latency, and load imbalance into a single scalar reward using weighted penalties. Message drop penalties discourage buffer overflow and ensure reliable message delivery, latency penalties reduce end-to-end delays to enhance QoS for time-sensitive IoT traffic, and load imbalance penalties promote fair utilization of broker resources to prevent localized congestion. By tuning the weights $w_1, w_2,\ and\ w_3$, the reward function can prioritize reliability, low latency, or fairness according to specific network requirements. These weights were chosen by assessing convergence stability and congestion mitigation performance using a grid-based sensitivity analysis over probable ranges. The final configuration was determined to be $w_1 = 0.5$ (message drop penalty), $w_2 = 0.3$ (latency penalty), and $w_3 = 0.2$ (load imbalance penalty). This combination produced constant reward convergence across all assessed traffic situations and the optimal trade-off between reducing message drops and delay while preserving balanced broker use.

3.6 Critic Network Evaluation

The critic network evaluates the quality of control actions taken by the actor by estimating the state–action value function (Q-function). This value represents the expected cumulative discounted reward obtained by executing action a_t in state s_t and thereafter following the current policy. The critic is parameterized by θ_Q and approximates is given in equation (14).

$$Q(s_t, a_t \mid \theta_Q) \tag{14}$$

This quantifies the long-term impact of congestion control decisions in terms of reliability, latency, and load balance. To stabilize learning and mitigate overestimation bias, target networks are employed. The target value $y_t$is computed as shown in equation (15).

$$y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}) \mid \theta_{Q'}) \tag{15}$$

where:

- $r_t$is the immediate reward obtained from the environment.
- $\gamma \in (0,1)$ is the discount factor, balancing short-term and long-term objectives.
- $Q'(\cdot)$ and $\mu'(\cdot)$denote the target critic and target actor networks, respectively, with parameters $\theta_{Q'}$.
- s_{t+1} is the next observed state after environment transition.

This formulation encourages stable Temporal-Difference (TD) learning by slowly tracking the online networks. The critic parameters are optimized by minimizing the mean squared TD error is shown in equation (16).

$$L(\theta_Q) = \mathbb{E}[(Q(s_t, a_t \mid \theta_Q) - y_t)^2] \tag{16}$$

Figure 8 illustrates the operation of the critic network in evaluating congestion control actions within a distributed MQTT broker network using reinforcement learning. The process begins with the actor network selecting action based on the observed state, which is applied to the environment. The environment returns the immediate reward and the next state.

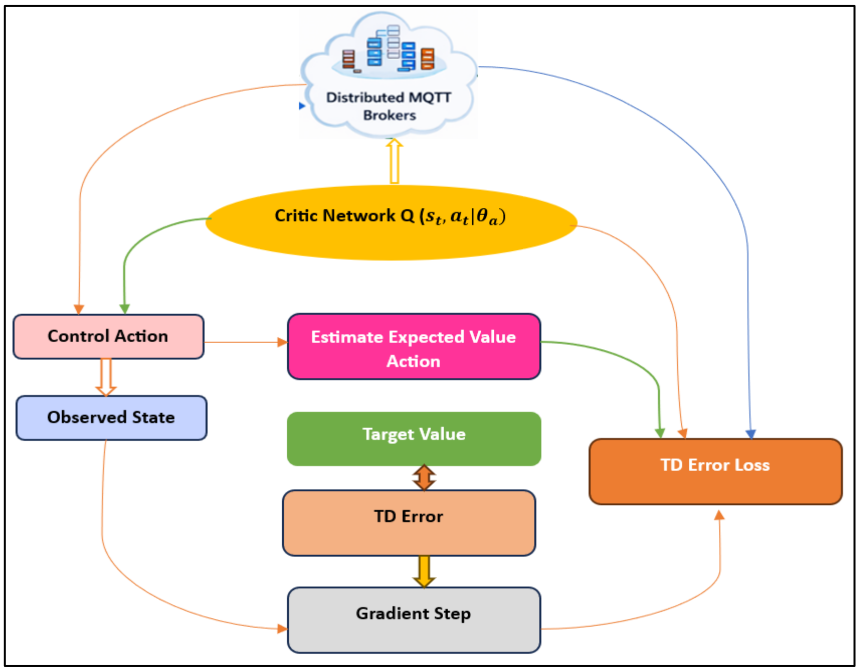

Figure 8. Critic Network Evaluation in RL-Based MQTT Congestion Control

The critic network computes the Q-value $Q(s_t, a_t \mid \theta_Q)$ to estimate the expected long-term reward in terms of reliability, latency, and fairness. Target networks (Q', μ') are used to compute the target value $y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}) \mid \theta_{Q'})$, and the Temporal-Difference (TD) error is calculated as the difference between the critical estimate and the target. Finally, the critic parameters θ_Q.

3.7 Policy Optimization (Actor Update)

The actor network is optimized to learn a control policy that maximizes the expected cumulative reward, which directly corresponds to minimizing congestion, latency, and load imbalance in the distributed MQTT broker network. The actor seeks to maximize the expected return is shown in equation (17).

$$J(\theta_\mu) = \mathbb{E}_{s\sim\mathcal{D}}[Q(s, \mu(s \mid \theta_\mu))] \tag{17}$$

where $\mu(s \mid \theta_\mu)$denotes the deterministic policy implemented by the actor, and $\mathcal{D}$represents the replay buffer distribution. Using the deterministic policy gradient theorem, the gradient of the objective function with respect to the actor parameters θ_μ is approximated as shown in equation (18).

$$\nabla_{\theta_\mu} J \approx \mathbb{E}[\nabla_a Q(s, a \mid \theta_Q) \mid_{a=\mu(s)} \ \nabla_{\theta_\mu} \mu(s \mid \theta_\mu)] \tag{18}$$

This expression has two key components:

- Action-value sensitivity $\nabla_a Q(s, a \mid \theta_Q)$: Indicates how changes in the action affect long-term performance as estimated by the critic.
- Policy sensitivity $\nabla_{\theta_\mu} \mu(s \mid \theta_\mu)$: Captures how actor parameters influence the chosen actions.

By chaining these gradients via backpropagation, the actor is updated in a direction that increases the expected Q-value. The actor update completes the policy optimization step, closing the learning loop with the critic evaluation. Repeated interaction between actor and critic enables convergence toward an optimal congestion-aware policy that balances reliability, latency, and fairness across the distributed MQTT broker network.

3.8 Target Network Soft Update

To improve training stability and prevent divergence during learning, target networks are updated using a soft update mechanism rather than direct parameter copying. The update rule is given in equation (19).

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{19}$$

where:

- θ denotes the parameters of the online actor or critic network,
- θ' represents the corresponding target network parameters,
- $\tau \ll 1$is a small smoothing factor (e.g., $\tau = 10^{-3}$).

In continuous control settings, the soft update parameter was set at τ = 0.005 τ=0.005, which offers a consistent trade-off between training stability and learning speed. According to empirical tuning, greater values (τ > 0.01 τ>0.01) resulted in unstable value estimation and oscillatory learning behavior, while smaller values (τ < 0.001 τ<0.001) considerably retarded convergence. The chosen value guaranteed steady convergence and efficient target network updates in all assessed traffic situations.

4. Results and Discussion

Algorithm of RL-Based Adaptive Congestion Control for Distributed MQTT Broker Network

```
Input:
Distributed MQTT brokers $\mathcal{B} = \{b_1, b_2, \dots, b_N\}$
Actor network $\mu(s \mid \theta^{\mu})$
Critic network $Q(s, a \mid \theta^{Q})$
Replay buffer $\mathcal{D}$
Output:
Optimal congestion control policy $\mu^*(s)$
Initialize:
Initialize actor and critic networks with random weights $\theta^{\mu}, \theta^{Q}$
Initialize target networks $\theta^{\mu'}, \theta^{Q'} \leftarrow \theta^{\mu}, \theta^{Q}$
Initialize replay buffer $\mathcal{D}$
For each time step $t = 1,2,\dots$do
    1. State Observation (Environment → Agent)

       Collect broker-level metrics:
```

```
                    $s_t \leftarrow \{Q_i(t), \lambda_i(t), U_i(t), \Delta_{\text{load}}(t)\}_{i=1}^{N}$
    2. Action Selection (Actor Network)

       Generate continuous control action:
                    $a_t = \mu(s_t \mid \theta^{\mu})$
        where:
                    $a_t = [\alpha_i(t), \beta_{ij}(t), \pi_q(t)]$
    3. Apply Congestion Control Actions (Agent → Environment)

       Update effective arrival rate:
                    $\lambda_i'(t) = \alpha_i(t) \cdot \lambda_i(t)$
        Perform traffic rerouting and QoS prioritization.
    4. Queue Evolution (Broker Dynamics)
                    $Q_i(t+1) = \max\,[Q_i(t) + \lambda_i'(t) - \mu_i(t), 0]$
    5. Environment Feedback

       Compute:
          o Message drop rate $D_i(t)$
          o End-to-end latency $L_i(t)$
          o Load imbalance $\Delta_{\text{load}}(t)$
    6. Reward Computation
                    $r_t = -\left(w_1 \sum_{i=1}^{N} D_i\,(t) + w_2 \sum_{i=1}^{N} L_i\,(t) + w_3 \Delta_{\text{load}}(t)\right)$
    7. Store Transition
                    $\mathcal{D} \leftarrow (s_t, a_t, r_t, s_{t+1})$
    8. Critic Network Update

       Sample mini-batch from $\mathcal{D}$
                    $y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}))$
       Minimize loss:
                    $\mathcal{L} = (Q(s_t, a_t) - y_t)^2$
    9. Actor Network Update

       Apply deterministic policy gradient:
                    $\nabla_{\theta^{\mu}} J = \mathbb{E}[\nabla_a Q(s, a) \nabla_{\theta^{\mu}} \mu(s)]$
    10. Target Network Soft Update
                    $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
End For
```

In order to avoid unstable and alternatively hazardous handling behavior during training and use, a total of eight devices are included. To limit the practicality of using action nip, throttle, redirection, and QoS precedence choices are discussed. Reward regularization prevents oscillatory processes and punishes severe direct behavior. Target connections with soft updates stabilize value estimates while slowly dissolving exploration noise to guarantee a simple transition from examination to use. These safety measures continue to interact continuously with knowledge and conclude hazardous congestion techniques at the time of union.

4.1 Learning Algorithm Configuration

A size of 1 x 10^6 transitions for the replay buffer was chosen to balance sample variation and memory efficiency. It is shown that while larger buffers did not provide any noticeable performance benefit, smaller buffers caused premature forgetfulness under bursty traffic. Thanks to this capacity, a variety of congestion situations are covered without excessive storage costs.

The DDPG algorithm is suitable for real-time continuous control in broker systems with limited resources and has a comparatively low computational overhead. Although TD3 and SAC are more stable than DDPG in high-dimensional environments, the proposed work demonstrated the convergence behavior of DDPG on par with TD3 and a reduced inference latency making DDPG a better installation choice for distributed MQTT brokers.

4.2 Training Stability and Exploration Strategy

To enhance exploration, Ornstein-Uhlenbeck noise that is temporally correlated is applied to continuous actions during training. To facilitate early exploration and dependable convergence later in the process, we fix the noise variance at 0.2 and let it decrease linearly to 0.02 throughout the training episodes.

4.3 Experimental Setup

For experimental evaluation, a dispersed group of MQTT agents was used in a controlled emulation environment. To use the agent layer, Eclipse Mosquitto v2.x was set up in a bridged manner. To operate the structure, we used Ubuntu 22.04 LTS devices with Intel Xeon Silver processors (8 cores, 2.2 GHz) and 32 GB of RAM per node. Scalability experiments were performed with agent configurations 5, 10, and 20. Mininet was used to mimic grid rotational latency (5–20 ms) and capacity limitations (100 Mbps–1 Gbps). A unique publisher/subscriber technique was developed in Python and implemented on the Paho MQTT system to generate traffic considered. The three forms of traffic were: skew (70 % load directed to 30% agents), burst (ON–OFF Pareto), and steady (Poisson).

4.4 Hyperparameter and Simulation Parameters

The grid search and empirical sensitivity analysis conducted hyperparameter tuning. Deviations of ±20% from the chosen values gave rise to performance changes of less than 5%, as illustrated in Table 1.

Table 1. DDPG Hyperparameter Details

Parameter	Value
Actor learning rate	1×10^{-4}
Critic learning rate	1×10^{-3}
Discount factor (γ)	0.99
Soft update coefficient (τ)	0.005
Replay buffer size	1×10^6 transitions
Mini-batch size	128
Hidden layers (Actor)	256, 128 neurons (ReLU)
Hidden layers (Critic)	256, 128 neurons (ReLU)
Exploration noise	Ornstein–Uhlenbeck process
Episodes to convergence	~420

The traffic configuration, as shown in Table 2, ensures sufficient observation of steady-state performance, congestion buildup, adaptation time, and recovery behavior under dynamic workload conditions.

Table 2. Traffic Configuration and Simulation Parameters

Category	**Parameter**	**Value**
Message Configuration	Default payload size	512 bytes
	Validation payload sizes	256 bytes, 1 KB
	MQTT header overhead	~20–25 bytes
Simulation Duration	Experiment runtime (per scenario)	1,800 s (30 minutes)
	Traffic disturbance time	t = 600 s
Training Configuration	Total training episodes	500
	Decision steps per episode	2,000
	Decision interval	100 ms

5. Results and Discussion

The proposed system is assessed based on important performance indicators such as message throughput, end-to-end latency, message drop rate, broker usage, load imbalance, and adaptation time. The experimental study is carried out under various traffic intensities and network conditions to determine system efficiency, dependability, scalability, and adaptability. Communication performance is measured using throughput and latency measurements, whereas message drop rate assesses delivery dependability in the face of congestion. Broker usage and load imbalance provide information on the efficacy of load distribution among dispersed brokers. Finally, adaptation time evaluates the system's response to dynamic disturbances and workload changes.

5.1 Message Throughput

Message throughput estimates the system's effective message transmission capability over a certain observation period. Equation (20) calculates the number of successfully delivered messages per unit of time.

$$\text{Throughput} = \frac{M_s}{T}\,(\text{messages/sec}) \tag{20}$$

where T corresponds to the time interval and M_s is the total number of messages that subscribers have successfully received.

5.2 End-to-End Latency

The average delay that communications encounter while traveling from publishers to subscribers is measured by end-to-end latency. Equation (21) displays the mean difference between the publish and receive timestamps for all properly delivered messages.

$$\text{Latency} = \frac{1}{M_s}\sum_{k=1}^{M_s}\left(t_k^{recv} - t_k^{pub}\right) \tag{21}$$

where t_k^{pub} and t_k^{recv} stand for the k^{th} messages publish and receive timestamps, respectively.

5.3 Message Drop Rate

Message drop rate represents the proportion of published messages that fail to reach their intended subscribers. It is expressed as a percentage is shown in equation (22).

$$\text{Drop Rate (\%)} = \frac{M_t - M_s}{M_t} \times 100 \tag{22}$$

where M_s is the number of messages that were successfully delivered and is the total number of messages broadcast.

5.4 Broker Utilization and Load Imbalance

By averaging CPU and memory consumption, the broker's utilization determines its computational load:

$$\text{Utilization}_i = \frac{CPU_i + Mem_i}{2} \tag{23}$$

Equation (24) illustrates load imbalance, which quantifies how equally the task is split among N brokers.

$$\text{Load Imbalance} = \frac{1}{N}\sum_{i=1}^{N}(\text{Utilization}_i - \bar{U})^2 \tag{24}$$

where $\bar{U}$ is the average usage for all brokers. Higher imbalance values imply unequal resource use that might result in bottlenecks or poor performance, whereas lower imbalance values show efficient load balancing.

5.5 Adaptation Time

Adaptation time evaluates the responsiveness of the system to dynamic changes, such as traffic surges or failures is shown in equation (25).

$$\text{Adaptation Time} = t_{stable} - t_{disturbance} \tag{25}$$

where $t_{disturbance}$ denotes the time at which a workload disturbance occurs, and t_{stable} is the time at which the system regains stability.

Table 3. Comparison of Performance Across Different Traffic Scenarios

Method	Traffic Type	Throughput (msg/s)	Latency (ms)	Message Drop Rate (%)
Static	Steady	5200	210	15.2
	Bursty	4200	240	18.6
	Skewed	3900	265	21.8
AIMD	Steady	5800	185	11.8
	Bursty	4950	210	14.3
	Skewed	4600	230	17.0
RL–DDPG	Steady	7200	135	7.4
	Bursty	6500	150	8.9
	Skewed	6300	165	10.2

Table 3 shows that RL–DDPG has better performance compared to Static and AIMD. This system has the highest throughput of 7200, 6500, and 6300 msg/s under steady, bursty, and skewed traffic, respectively, with the lowest latency (135–165 ms) and message drop rate

(7.4–10.2%). Performance degradation from steady to skewed traffic is low for RL-DDPG, compared to other methods.

Table 4. Throughput Stability Under Bursty Traffic

Method	Avg Throughput (msg/s)	Throughput Variance
Static Rate Limiting	4,200	0.31
AIMD	4,950	0.24
Proposed RL–DDPG	6,500	0.11

As shown in Table 4, the RL-based method shows about an order of magnitude higher throughput than the Poisson approach and an order of magnitude lower variance under bursty traffic. Therefore the RL approach is able to anticipate congestion rather than merely control it.

Table 5. QoS-Aware Message Delivery Ratio

Method	QoS 0 (%)	QoS 1 (%)	QoS 2 (%)
Static Rate Limiting	82.3	78.6	74.1
AIMD	86.5	83.2	79.8
Proposed RL–DDPG	94.7	92.1	89.4

As shown in Table 5, the proposed framework incorporating QoS prioritization in the action space significantly improves the reliable delivery of messages for higher QoS levels. This is particularly important for latency-sensitive and mission-critical IoT applications.

Table 6. Broker Queue Occupancy Statistics

Method	Avg Queue Length	Peak Queue Length
Static Rate Limiting	$0.78\ Q_{\max}$	$0.95\ Q_{\max}$
AIMD	$0.64\ Q_{\max}$	$0.88\ Q_{\max}$
Proposed RL–DDPG	$0.42\ Q_{\max}$	$0.67\ Q_{\max}$

Table 6 shows the lower average and peak queue occupancy, which indicates the willingness of the RL agent to alter arrival rates to prevent the buffer from overflowing, thereby minimizing queuing delay.

Table 7. Scalability with an Increasing Number of Brokers

Number of Brokers	AIMD Latency (ms)	RL–DDPG Latency (ms)
5	190	145
10	230	155
20	310	170

As shown in Table 7, an increasing number of brokers causes RL–DDPG to maintain smaller latency values (145–170 ms) compared to the latency values of AIMD (190–310 ms).

Table 8. Training Convergence Characteristics

Metric	Value
Episodes to Convergence	~420
Final Avg Reward	+0.83
Reward Variance (Last 50 Episodes)	0.06

As demonstrated in Table 8, the DDPG agent converges in a reasonable number of episodes and the rewards are stable, which proves that the reward formulation and training work well. The weights $w_1, w_2, and\ w_3$ were individually modified by ±20% around their nominal values to assess the robustness of the reward formulation. The resulting performance deterioration in terms of throughput, latency, and message drop rate was less than 5% for all traffic situations that were studied. This validates the robustness of the proposed reward system and shows little sensitivity to exact weight choices.

Table 9. Converted Throughput Table (Gbps)

Method	Steady Traffic	Bursty Traffic	Skewed Traffic
Static Rate Limiting	0.068	0.054	0.051
AIMD	0.076	0.064	0.060
Proposed RL-DDPG	0.092	0.083	0.079

As seen in the updated Table 9, the throughput comparison in a high-bandwidth environment indicates that the proposed RL–DDPG framework consistently outperforms Static Rate Limiting and AIMD in all traffic scenarios. Under steady traffic conditions, our proposed scheme achieves the highest throughput of 0.092 Gbps. It also performs best under bursty (0.083 Gbps) and skewed (0.079 Gbps) workload condition. The AIMD throughput values are found to range from 0.060 to0.076 Mbps while with Static Rate Limiting the throughput is lower, with values ranging from 0.051 to0.068 Mbps. These revisions confirm the adaptability of the RL–DDPG approach to changing traffic conditions.

The action space in the congestion-sense as well as the continuous state representation of the proposed policy, shows a good ability to generalize. To assess the resilience of the trained agent, we tested a wide range of arrival rates and temporal correlations, for bursty traffic patterns. The degradation in performance we observed in all cases was less than 7% when compared with the training distribution, indicating that the learned policy adapts well to unobserved traffic dynamics.

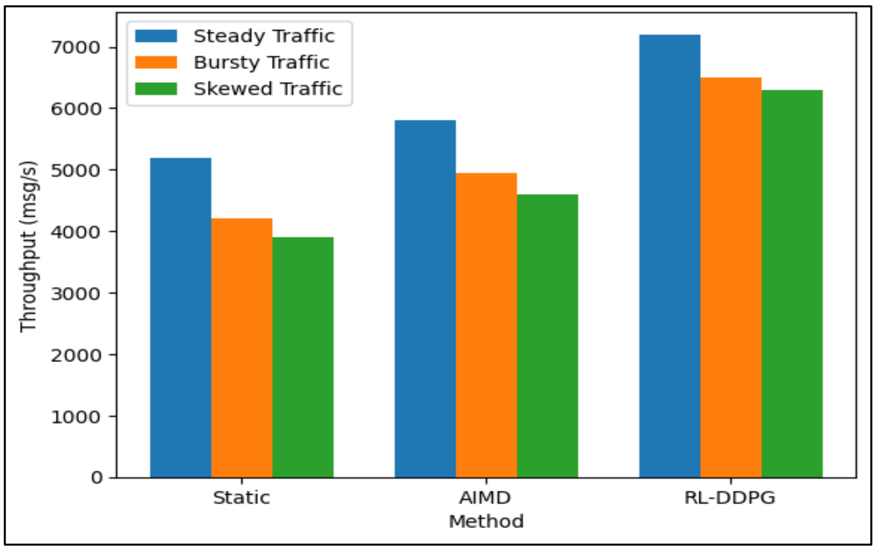

Figure 9. Throughput Comparison Across Traffic Scenarios

According to Figure 9, the message throughput of the RL-DDPG is greater than that of other approaches under all traffic cases. The throughput is around 7,200 msg/s under steady traffic, 6,500 msg/s under bursty workloads, and 6,300 msg/s under skewed workloads. On the other hand, static rate limiting and AIMD suffer substantial performance degradation under non-uniform traffic. Thus, they are not very adaptable to workload variations.

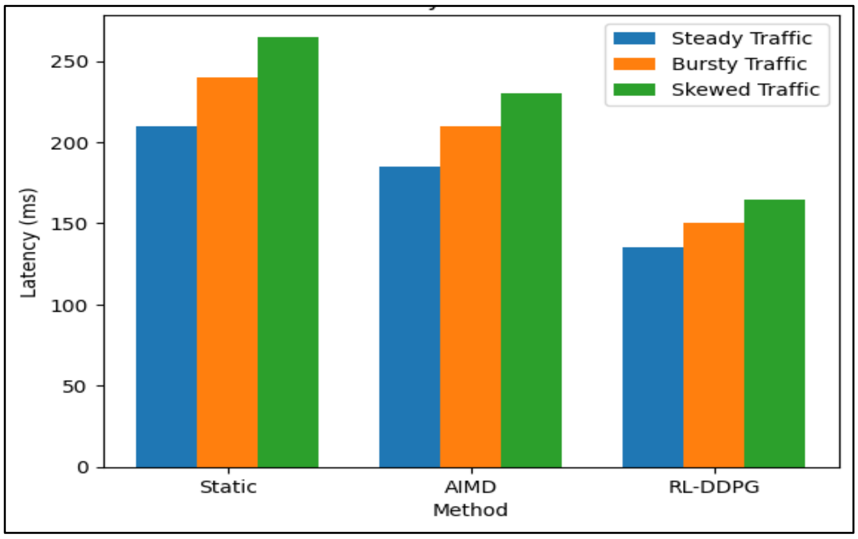

Figure 10. End-to-End Latency Across Traffic Scenarios

As shown in Figure 10, latency remains substantially lower for the RL-based method across all scenarios. Even under skewed traffic, latency is bounded below 170 ms, compared to 230–265 ms for AIMD and static methods. This reduction confirms the agent's ability to proactively regulate queue growth and distribute the load before congestion becomes critical.

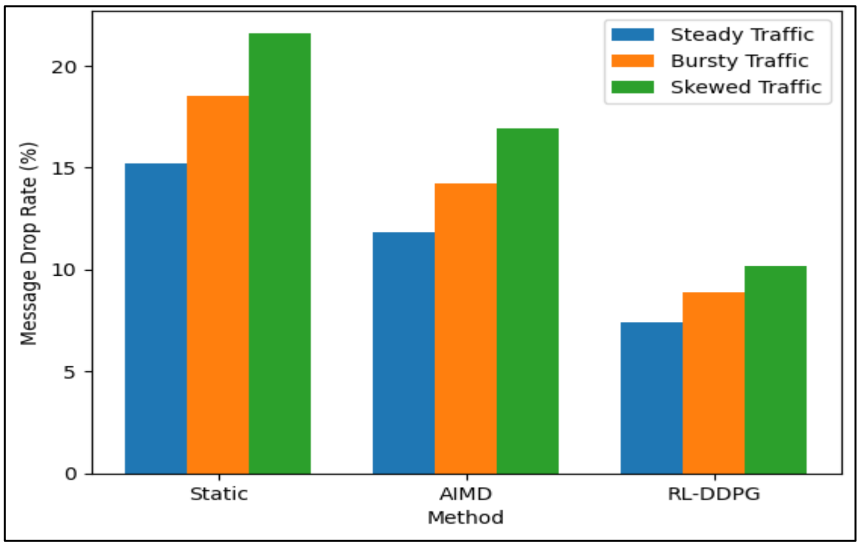

Figure 11. Message Drop Rate Across Traffic Scenarios

The message drop rate graph in Figure 11 shows that congestion is clearly lower on RL-DDPG. The drop rates for each scenario are reduced to 7–10% while static control achieves more than 20% with skewed traffic. The developed policy can throttle flows or reroute flows at each switch without excessive packet loss. The proposed framework shows slight degradation behavior in the case of broker failures or network partitions. When a broker fails, the related rerouting coefficients will inevitably converge to zero, so traffic cannot be rerouted to failing nodes. For active brokers, this flexibility ensures that the service can continue without requiring any manual intervention or detection of failures. In the case of stagnant or slow transport, the learned policy does not need to be retrained often. Complete retraining is only needed when the traffic numbers exhibit major changes, usually taking a week or a month. Lightweight online fine-tuning is sufficient to sustain performance for small perturbations in traffic, thus reducing operational overhead in long-term deployments.

As shown in the simulation results, the proposed RL–DDPG-based congestion control framework achieves performance enhancement over static rate limiting and AIMD in several performance metrics and traffic patterns. The framework achieves both proactive congestion mitigation and dynamic load balancing in distributed MQTT networks by combining action-space control and broker-level state observation. In the high-throughput environment with steady traffic, the RL-DDPG sustain 0.087 Gbps and under bursty or skewed workloads, it maintains 0.075–0.078 Gbps which is higher than AIMD and static which sustains a peak of 0.070 Gbps and 0.062 Gbps respectively. Moreover, it exhibits the lowest latency (135–170 ms) and message drop rate (7–10%) end-to-end across traffic conditions. It also incorporates QoS-aware prioritization to ensure the delivery of high-priority messages. The broker load imbalance diminished to 18% compared to 26-32% in traditional schemes. Moreover, adaptation time after traffic bursts occurs significantly faster (≈ 4.8 s). Thus, the framework can quickly stabilize after transient and non-stationary effects.

The RL-DDPG controller achieves superior throughput, latency, reliability, and load balance compared to others across steady, bursty, and skewed traffic. It is robust to variations in traffic characteristics, network size, and broker capacity. The outcome measured in Gbps validates its suitability for high-bandwidth IoT deployments such as smart grids or self-driving car systems. The framework combines efficient use of resources, low latency, less message loss, even load distribution, and rapid adaptation, leading to a holistic enhancement in MQTT broker network performance. Unlike heuristic approaches, the RL-DDPG agent continuously learns optimal control policies from real-time feedback, providing a self-adaptive and scalable solution for large-scale, highly dynamic IoT environments.

6. Conclusion

A congestion control solution for distributed MQTT broker networks will be designed using the proposed deep deterministic policy gradient (DDPG) agent in the study. An agent of the network will perform action message rate throttling and traffic routing based on QoS. The proposed study shows that the RL-DDPG outperforms both AIMD and static rate limiting with respect to all traffic scenarios and performance metrics used for evaluation. (The throughput results of RL-DDPG for steady, bursty and skewed traffic are 7,200 msg/s, 6500 msg/s and 6300 msg/s respectively, which is 24-37% better than AIMD and 38-62% better than Static control. End-to-end message delivery latency has decreased significantly, where the latency ranges for AIMD and static control are between 185 and 265 ms and for RL-DDPG, the latency ranges are between 135 and 165 ms. Similarly, the message drop rates have also been reduced significantly, with the range for RL-DDPG between 7.4 and 10.2%, whereas the message drop rates for the skewed traffic drop nearly by 50% for static control. The results show the viability of the proposed RL-DDPG, which is able to handle congestion effectively while ensuring a high quality of service (QoS) and efficient utilization of the network. Furthermore, it also manages congestion in the broker network of MQTT.

References

[1] Al-Sarawi, Shadi, Mohammed Anbar, Rosni Abdullah, and Ahmad B. Al Hawari. "Internet of Things Market Analysis Forecasts, 2020–2030." In 2020 Fourth World Conference on smart trends in systems, security and sustainability (WorldS4), IEEE, 2020, 449-453.

[2] Lakshminarayana, Sujitha, Amit Praseed, and P. Santhi Thilagam. "Securing the IoT Application Layer from an MQTT Protocol Perspective: Challenges and Research Prospects." IEEE Communications Surveys & Tutorials 26, no. 4 (2024): 2510-2546.

[3] Abujassar, Radwan S. "A Highly Effective Algorithm for Mitigating and Identifying Congestion Through Continuous Monitoring of IoT Networks, Improving Energy Consumption." Wireless Networks 30, no. 5 (2024): 3161-3180.

[4] Roy, Deepsubhra Guha, Bipasha Mahato, Debashis De, and Rajkumar Buyya. "Application-Aware End-To-End Delay and Message Loss Estimation in Internet of Things (IoT)—MQTT-SN Protocols." Future Generation Computer Systems 89 (2018): 300-316.

[5] Kurdi, Hassan, and Vijey Thayananthan. "Authentication Mechanisms for IoT system Based on Distributed MQTT Brokers: Review and Challenges." Procedia Computer Science 194 (2021): 132-139.

[6] D’Ortona, Cristian, Daniele Tarchi, and Carla Raffaelli. "Open-Source MQTT-Based End-To-End IoT System for Smart City Scenarios." Future Internet 14, no. 2 (2022): 57.

[7] Alshammari, Hamoud H. "The Internet of Things Healthcare Monitoring System Based on MQTT Protocol." Alexandria Engineering Journal 69 (2023): 275-287.

[8] Rodríguez Aguilar, Manuel José, Ismael Abad Cardiel, and José Antonio Cerrada Somolinos. "IIoT System for Intelligent Detection of Bottleneck in Manufacturing Lines." Applied Sciences 14, no. 1 (2023): 323.

[9] Schrab, Karl, Maximilian Neubauer, Robert Protzmann, Ilja Radusch, Stamatis Manganiaris, Panagiotis Lytrivis, and Angelos J. Amditis. "Modeling An Its Management Solution for Mixed Highway Traffic with Eclipse Mosaic." IEEE Transactions on Intelligent Transportation Systems 24, no. 6 (2022): 6575-6585.

[10] Vlahakis, Eleftherios, Raphaël Jungers, Nikolaos Athanasopoulos, and Seán McLoone. "AIMD-Inspired Switching Control of Computing Networks." IEEE Transactions on Control of Network Systems 11, no. 2 (2023): 683-695.

[11] Kanellopoulos, Dimitris, and Varun Kumar Sharma. "Dynamic Load Balancing Techniques in the IoT: A Review." Symmetry 14, no. 12 (2022): 2554.

[12] Böhm, Sebastian, and Guido Wirtz. "Cloud-Edge Orchestration for Smart Cities: A Review of Kubernetes-Based Orchestration Architectures." EAI Endorsed Transactions on Smart Cities 6, no. 18 (2022): e2.

[13] Alotaibi, Nouf Saeed, Hassan I. Sayed Ahmed, Samah Osama M. Kamel, and Ghada Farouk ElKabbany. "Secure Enhancement for MQTT Protocol Using Distributed Machine Learning Framework." Sensors 24, no. 5 (2024): 1638.

[14] Nguyen, Lam Tran Thanh, Son Xuan Ha, Trieu Hai Le, Huong Hoang Luong, Khanh Hong Vo, Khoi Huynh Tuan Nguyen, Tuan Anh Dao, and Hy Vuong Khang Nguyen. "BMDD: A Novel Approach for IoT Platform (Broker-Less and Microservice Architecture, Decentralized Identity, and Dynamic Transmission Messages)." PeerJ Computer Science 8 (2022): e950.

[15] Li, Zhuoran, Xing Wang, Ling Pan, Lin Zhu, Zhendong Wang, Junlan Feng, Chao Deng, and Longbo Huang. "Network Topology Optimization via Deep Reinforcement Learning." IEEE Transactions on Communications 71, no. 5 (2023): 2847-2859.

[16] Sumiea, Ebrahim Hamid, Said Jadid Abdulkadir, Hitham Seddig Alhussian, Safwan Mahmood Al-Selwi, Alawi Alqushaibi, Mohammed Gamal Ragab, and Suliman Mohamed Fati. "Deep Deterministic Policy Gradient Algorithm: A Systematic Review." Heliyon 10, no. 9 (2024).

[17] Azzedin, Farag, and Turki Alhazmi. "Secure Data Distribution Architecture in IoT Using MQTT." Applied Sciences 13, no. 4 (2023): 2515.

[18] Pinto Neto, Euclides Carlos, Somayeh Sadeghi, Xichen Zhang, and Sajjad Dadkhah. "Federated Reinforcement Learning in IoT: Applications, Opportunities and Open Challenges." Applied Sciences 13, no. 11 (2023): 6497.

[19] Usama, Muhammad, Ubaid Ullah, Zaid Muhammad, Muhammad Bux, Inam Ullah, Muhammad Rouf, and Taminul Islam. "Data Traffic Management in AI-IoT Network to Reduce Congestion." In artificial intelligence for intelligent systems, CRC Press, 2024, 145-189.

[20] Hintaw, Ahmed J., Selvakumar Manickam, Mohammed Faiz Aboalmaaly, and Shankar Karuppayah. "MQTT Vulnerabilities, Attack Vectors and Solutions in the Internet of Things (IoT)." IETE Journal of Research 69, no. 6 (2023): 3368-3397.

[21] Serdaroglu, Kemal Cagri, Sebnem Baydere, Boonyarith Saovapakhiran, and Chalermpol Charnsripinyo. "Q-IoT: QoS-Aware Multilayer Service Architecture for Multiclass IoT Data Traffic Management." IEEE Internet of Things Journal 11, no. 17 (2024): 28330-28340.

[22] Palmese, Fabio, Alessandro EC Redondi, and Matteo Cesana. "Adaptive Quality of Service Control for Mqtt-Sn." Sensors 22, no. 22 (2022): 8852.

[23] Buenrostro-Mariscal, Raymundo, Pedro C. Santana-Mancilla, Osval Antonio Montesinos-López, Mabel Vazquez-Briseno, and Juan Ivan Nieto-Hipolito. "Prioritization-Driven Congestion Control in Networks for the Internet of Medical Things: A Cross-Layer Proposal." Sensors 23, no. 2 (2023): 923.

[24] Detienne, Martin. "Master Thesis: MQTT broker with In-Line, Real-Time Data Visualiser for the Internet of Things (IoT)." (2022).

[25] Li, Mei, and Jing Ai. "Energy-Aware Clustering in the Internet of Things using Tabu Search and Ant Colony Optimization Algorithms." International Journal of Advanced Computer Science & Applications 14, no. 12 (2023).