Asian Journal of Information Science and Technology
ISSN: 2231-6108 (P); 3049-2661 (E)
Vol.15 No.1, 2025, pp.30-43

DOI: https://doi.org/10.70112/ajist-2025.15.1.4324

Reinforcement Learning-Based Task Scheduling for IoT Applications in Long-Range Wide Area Networks

Ermias Melku Tadesse[1]*, Haimanot Edmealem[2], Tesfaye Belay[3] and Abubeker Girma[4]
[1&2]Department of Information Technology, [4]Department of Software Engineering,
Kombolcha Institute of Technology, Wollo University, Dese, Ethiopia
[3]Department of Computer Science, Institute of Technology, Wollo University, Dese, Ethiopia
E-mail: ermiasmelku3400@gmail.com
*Corresponding Author: Ermias Melku Tadesse
(Received 5 January 2024; Revised 16 February 2025; Accepted 2 March 2025; Available online 11 March 2025)

Abstract - **To address the challenges of effective resource allocation in low-power wide-area networks, this thesis examines the scheduling of end devices in Internet of Things (IoT) applications using LoRaWAN technology. The primary objective of this research is to utilize reinforcement learning (RL) to enhance quality of service (QoS) metrics, including energy efficiency, throughput, latency, and reliability. This objective was achieved through a simulation-based approach that assessed the performance of the RL-based scheduling algorithm using NS-3 simulations. The key findings indicate that, compared to existing scheduling methods, the RL agent significantly enhances data transmission reliability and increases network throughput. Additionally, the proposed approach effectively reduces average system latency and overall energy consumption, leading to improved network resource utilization. These results suggest that applying RL to task scheduling in LoRaWAN networks can provide a scalable and reliable solution to existing challenges, ultimately contributing to more intelligent and sustainable IoT systems. Overall, this study concludes that RL-based techniques can enhance resource management in dynamic and resource-constrained environments.**
Keywords: **Reinforcement learning (RL), LoRaWAN, Quality of service (QoS), Task scheduling, Energy efficiency**

I. INTRODUCTION

The Internet of Things (IoT) encompasses a vast network of interconnected devices that communicate and exchange data over the Internet, impacting various sectors such as smart cities, healthcare, agriculture, and industry. The rapid expansion of IoT applications has created a pressing need for efficient resource allocation and task scheduling mechanisms to optimize resource utilization while meeting quality of service (QoS) requirements [1]. LoRaWAN (Long Range Wide Area Network) is a significant enabler of IoT, designed to provide long-range communication with low power consumption. This wireless communication protocol is particularly optimized for IoT devices, allowing them to transmit small amounts of data over considerable distances.

LoRaWAN's capabilities make it suitable for applications requiring remote monitoring and data acquisition, thus facilitating the expansion of IoT solutions [2, 3]. For example, LoRaWAN, an LPWAN technology, can connect battery-powered devices over very long distances while consuming minimal power, making it a cost-effective solution [4]. LoRaWAN operates in the unlicensed ISM bands, which vary by region [5]. It employs a chirp spread spectrum modulation technique to achieve long-distance communication with low power consumption [6].

One of the main advantages of LoRaWAN is its extensive coverage. It can transmit data over several kilometres in open settings, such as rural areas or large industrial facilities, without the need for cellular towers or other infrastructure. Consequently, LoRaWAN is well suited for applications requiring wide coverage, such as smart agriculture, asset tracking, environmental monitoring, and smart city deployments [1]. Due to its unique combination of long-range capability, low power consumption, and cost-effective deployment, LoRaWAN has become an attractive technology for IoT applications [7]. Figure 1 illustrates the overall architecture of a LoRaWAN network, highlighting its key components and their interactions.

The architecture consists of end devices (sensors), gateways, a network server, and application servers, demonstrating how data flows from the end devices to the application layer. End devices communicate wirelessly with gateways using LoRa technology, which then forward the data to the network server. The network server processes and routes the data to the appropriate application server for further analysis or action, showcasing the hierarchical structure and functionality of the LoRaWAN ecosystem.

In LoRaWAN IoT applications, maintaining quality of service (QoS) is crucial due to challenges such as limited resources, channel congestion, and varying QoS requirements, which can result in high latency and packet loss. Reinforcement learning (RL) is identified as a highly suitable machine learning approach for dynamic task scheduling in LoRaWAN networks, as it can adapt to changing conditions and optimize multiple QoS metrics simultaneously.

By leveraging RL, nodes can self-optimize scheduling performance, enhancing reliability and efficiency in diverse applications such as smart agriculture, industrial IoT, and smart city management [9, 10, 7]. This study proposes the use of RL techniques to develop a scheduling algorithm that adapts to dynamic network conditions, optimizes energy consumption, and enhances overall system performance. By incorporating RL, the proposed solution aims to improve latency, reliability, and efficiency in LoRaWAN networks, ultimately contributing to the sustainability and scalability of IoT deployments [11, 10].

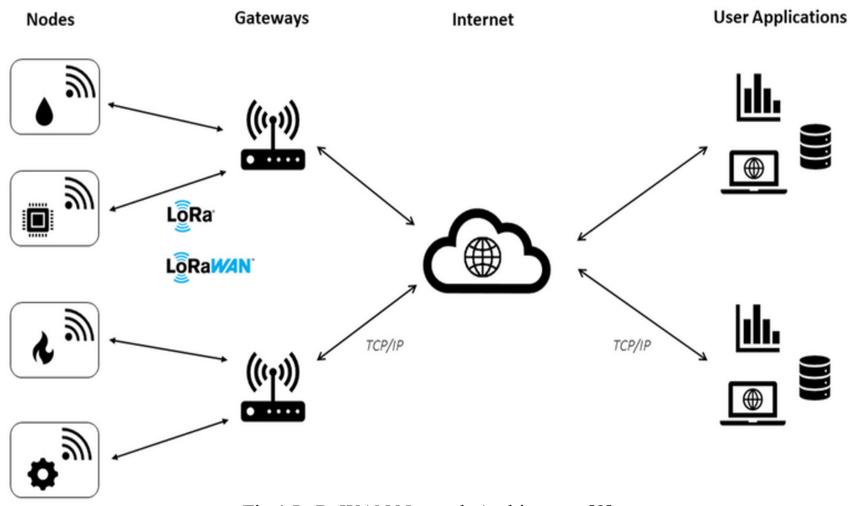

Fig.1 LoRaWAN Network Architecture [8]

A. Contributions of the Article

1. Development of an RL-Based Scheduling Algorithm: This article presents a novel reinforcement learning-based task scheduling algorithm specifically designed for LoRaWAN networks, enhancing resource allocation and optimizing quality of service (QoS) metrics.
2. Performance Evaluation: The study provides a comprehensive performance analysis of the proposed algorithm through simulations, demonstrating its effectiveness in improving throughput, reducing delay, and increasing packet delivery ratios compared to existing scheduling methods.
3. Insights for Future Research: The findings and methodologies outlined in this article offer valuable insights and a foundation for future research in the fields of IoT and LoRaWAN. They encourage further exploration of adaptive and intelligent scheduling techniques to address evolving challenges in network management.

II. LITERATURE REVIEW

Low-power wide-area networks (LPWANs), such as LoRaWAN, have revolutionized the Internet of Things (IoT) by enabling long-range communication with battery-powered devices. However, IoT applications require reliable and expedited data delivery, posing challenges for LoRaWAN due to inherent limitations in range, latency, and energy constraints (Author, Year). This review explores existing research related to task scheduling in LoRaWAN.The study by Author (Year) proposes a dynamic transmission priority scheduling technique (PST) based on an unsupervised learning clustering algorithm for dense LoRaWAN networks. In this approach, the LoRa gateway classifies nodes into different priority clusters, and the dynamic PST allows the gateway to configure transmission intervals based on cluster priorities. This technique aims to improve transmission delay and reduce energy consumption. Simulation results suggest that the proposed method outperforms conventional LoRaWAN and recent clustering and scheduling schemes, making it potentially well suited for dense LoRaWAN deployments.

Author (Year) introduces a real-time LoRa (RT-LoRa) communication protocol for industrial IoT applications. The real-time flow is processed using a medium access strategy, with both static and mobile nodes forming the network. The QoS level remains the same for all static nodes, while mobile node flows are categorized into three classes: normal, dependable, and most reliable. The technique distributes the spreading factor (SF) and carrier frequency (CF) based on QoS levels.

The network follows a star topology, connecting both mobile and static nodes to the gateway. The study highlights several key points: In a single-gateway network using single-hop communication, significant transmission delays of up to 28 seconds occur for most dependable flows, even within a 180-meter range. The study does not address the need for greater coverage or reduced time delay for real-time industrial data. Additionally, limitations exist in QoS provisioning, as only mobile nodes receive differentiated QoS levels, while all static node flows are assigned the same priority. Furthermore, energy consumption remains high, especially for nodes located farther from the gateway. In another study, Author (Year) proposes a method to optimize LoRaWAN network performance by dynamically assigning values for SF and CF radio parameters. The assignment is formulated as a mixed-integer linear programming (MILP) problem to maximize key network metrics, such as the data extraction rate (DER), while minimizing packet collisions. An approximation algorithm is also developed to solve the problem efficiently at scale. The results indicate a 6–13% reduction in packet collisions compared to baseline policies. The performance evaluation is conducted using the LoRaSim simulator. However, the optimization focuses solely on SF and CF parameters, and considering additional parameters could lead to even better performance. The study also assumes a static network with a single-gateway setup, limiting its applicability to more complex real-world deployments.

Author (Year) explores the viability of real-time communication in LoRaWAN-based IoT systems. Using an integer linear programming (ILP) model, the study assesses real-time communication feasibility during the network design phase. The model optimizes the number and placement of gateways required to meet real-time requirements and is validated through multiple scenarios. The findings provide valuable insights into LoRaWAN's scalability and real-time support limitations. However, the model primarily focuses on static network design at deployment, which may not fully account for dynamic network conditions such as interference, congestion, and gateway availability, all of which significantly impact real-time QoS performance.

In another study, Author (Year) presents a low-overhead synchronization and scheduling concept implemented on LoRaWAN Class A devices. The proposed system, deployed on STM32L0 microcontrollers (MCUs), incorporates a central entity that provides synchronization metrics and allocates transmission slots. By measuring clock drift in devices, the system defines precise slot lengths, achieving a 10-millisecond accuracy and significantly improving packet delivery ratios compared to Aloha-based setups, particularly under high network loads. The study addresses a gap in the literature by experimentally demonstrating the feasibility of LoRaWAN scheduling techniques. However, it does not explore the energy consumption impact of the proposed scheduling algorithms. Several existing studies have proposed methods to reduce retransmissions, including adaptive retry limits and error correction mechanisms. However, most of these approaches fail to dynamically adapt to changing network conditions. This issue is specifically addressed by the proposed reinforcement learning-based scheduling algorithm.

III. METHODOLOGY

A. Proposed Method

The research methodology focuses on designing and implementing a reinforcement learning (RL)-based scheduling algorithm for reliable data delivery in LoRaWAN networks. This study adopts a design science research (DSR) approach, which emphasizes the systematic development and evaluation of practical solutions to address inefficiencies in existing task scheduling mechanisms. The methodology begins with a detailed description of the research design, highlighting the need for a task scheduling algorithm capable of effectively managing resources in dynamic environments. Additionally, the study identifies the limitations of existing scheduling methods in LoRaWAN networks, particularly their inability to meet the quality of service (QoS) demands of modern Internet of Things (IoT) applications. To address these challenges, this research proposes an RL-based algorithm that can adapt to varying network conditions and optimize resource allocation.

B. Research Design

The research employs a mixed-methods approach, combining quantitative research with design science to systematically design, develop, and evaluate a QoS-aware task scheduling algorithm. This approach addresses questions regarding the effectiveness of the proposed algorithm in improving QoS in dynamic IoT environments.

C. Algorithm Design and Implementation

1. *Algorithm Design:* The design of the RL-based scheduling algorithm focuses on creating an intelligent agent that optimizes task scheduling in a LoRaWAN environment. Key components include defining the state space, action space, and reward function, which guide the agent's learning process to make optimal scheduling decisions based on network conditions.
2. *State Space:* The state space encompasses various network parameters, such as node status, channel conditions, and traffic patterns, allowing the agent to assess the current environment effectively.
3. *Action Space:* The action space includes possible scheduling actions, such as channel selection, task prioritization, and gateway allocation, enabling the agent to make informed decisions to enhance QoS metrics.
4. *Reward Function:* The reward function provides feedback to the agent based on its actions, encouraging behaviors that lead to improved QoS outcomes, such as reduced delay, increased packet delivery ratio, and minimized packet error rates.
5. *Policy (π):* The policy defines the strategy the agent uses to select actions based on the observed state, enabling it to balance exploration and exploitation during learning.
6. *Learning Algorithm:* A suitable reinforcement learning algorithm, such as Deep Q-Networks (DQN), is employed to enable the agent to learn from its experiences and improve its scheduling decisions over time.

Fig. 2 illustrates the architecture of a Deep Q-Network (DQN), which combines Q-learning with deep neural networks to enable reinforcement learning in complex environments. The architecture typically consists of the following key components:

1. *Input Layer:* This layer receives the state representation of the environment, which includes various features relevant to the task. The input is often a high-dimensional vector that captures the current state of the system.
2. *Hidden Layers:* The DQN architecture includes multiple fully connected hidden layers (e.g., two layers) that process the input data. Each hidden layer consists of a specified number of neurons (e.g., 128), which are responsible for extracting features and learning non-linear relationships between the input state and potential actions. ReLU (Rectified Linear Unit) activation functions are commonly used to introduce non-linearity.
3. *Output Layer*: The output layer generates Q-values for each possible action based on the processed input state. These Q-values represent the expected future rewards for taking specific actions in the given state, allowing the agent to make informed decisions.
4. *Experience Replay*: Although not explicitly shown in the architecture diagram, experience replay is an integral part of the DQN framework. It involves storing past experiences (state, action, reward, next state) in a replay memory, which is sampled during training to improve learning stability and efficiency.

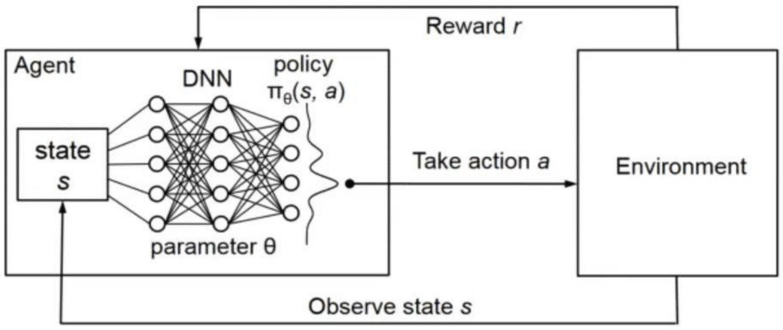

Fig. 2 DQN Architecture [18]

The diagram illustrates the agent taking an action in the environment, receiving a new state and reward, and updating its policy based on the experience. This iterative process enables the agent to learn an optimal policy for maximizing rewards in the environment.

D. Breakdown of the Diagram's Elements

1. *Agent:* The decision-making entity. It receives the current state of the environment (s) and uses its policy (π) to select an action (a). The policy is typically implemented as a deep neural network (DNN) with parameters (θ).
2. *Environment*: The external system the agent interacts with. It receives the agent's action (a) and provides the agent with a new state (s') and a reward (r).
3. *State (s)*: The current situation or observation of the environment.
4. *Action (a)*: The decision or move made by the agent.
5. *Reward (r)*: A scalar value indicating the outcome of the agent's action. Positive rewards reinforce desired behaviors, while negative rewards discourage undesirable ones.
6. *Policy (π)*: A function that maps states to actions. In deep reinforcement learning (DRL), it is often represented as a neural network.

E. Training Phase of the Proposed Scheduling Algorithm

Fig.3 outlines the training phase of the proposed scheduling algorithm, which utilizes a Deep Q-Network (DQN) approach to optimize task scheduling in a LoRaWAN environment. The training phase consists of several key steps:

1. *Initialization of DQN Parameters*: The training process begins with the initialization of essential DQN parameters, including the learning rate, which determines how much the Q-values are updated during training; epsilon, which controls the exploration-exploitation trade-off; and the experience replay buffer, which stores past experiences to enhance training stability.
2. *Observation of the Current State*: The agent interacts with the OpenAI Gym environment to observe the current state of the network. This state includes various parameters, such as network conditions, task queue status, and other relevant metrics that influence scheduling decisions.
3. *Action Selection and Execution*: Based on the observed state, the agent selects an action using an epsilon-greedy policy, balancing the exploration of new actions and the exploitation of known rewarding actions. The selected action is then executed within the environment.
4. *Reward Calculation*: After executing the action, the agent receives feedback in the form of a reward, which quantifies the effectiveness of the action taken in terms of QoS metrics, such as delay, throughput, and packet delivery ratio.
5. *Experience Storage and Learning*: The agent stores the experience (state, action, reward, next state) in the

replay buffer. A mini-batch of experiences is sampled from this buffer to update the Q-values, allowing the agent to learn from past actions and improve its scheduling policy over time.

6. *Iteration and Convergence*: The training process continues iteratively, with the agent observing new states, selecting actions, and updating Q-values until a predefined maximum number of training iterations is reached or performance converges to an acceptable level.

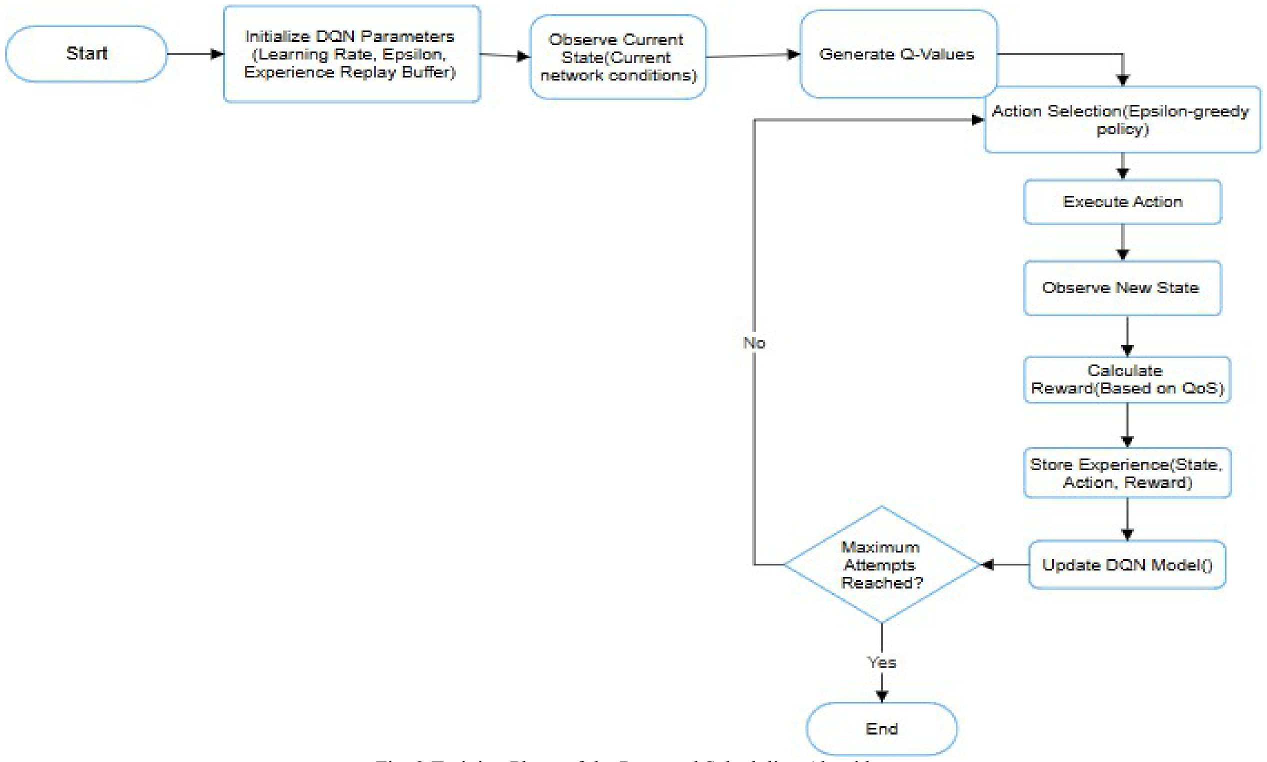
Start
Initialize DQN Parameters (Learning Rate, Epsilon, Experience Replay Buffer)
Observe Current State(Current network conditions)
Generate Q-Values
Action Selection(Epsilon-greedy policy)
Execute Action
Observe New State
Calculate Reward(Based on QoS)
Store Experience(State, Action, Reward)
Update DQN Model()
Maximum Attempts Reached?
No
Yes
End

Fig. 3 Training Phase of the Proposed Scheduling Algorithm

F. The Trained Proposed Scheduling Algorithm Diagram

Fig.4 presents a diagram of the trained scheduling algorithm, illustrating the workflow and key components involved in the task scheduling process within a LoRaWAN environment. The diagram outlines the following steps:

1. *Receive Task Request:* The process begins with the system receiving a new task scheduling request, which includes critical parameters such as deadlines and network context. This initiates the scheduling cycle.
2. *Retrieve Network State:* The algorithm retrieves the current network state, which encompasses various factors such as the Signal-to-Interference-plus-Noise Ratio (SINR), existing task queue, and other relevant network conditions that influence scheduling decisions.
3. *Generate Schedule:* Utilizing the learned policy from the training phase, the reinforcement learning (RL) agent generates a schedule by assigning tasks to specific gateways. This assignment is optimized based on Quality of Service (QoS) metrics and the deadlines specified in the task request.
4. *Evaluate Schedule Feasibility:* The generated schedule is assessed for feasibility, ensuring that it meets all required constraints and QoS criteria. This step is crucial to confirm that tasks can be completed within their deadlines and adhere to the necessary QoS standards.
5. *Feasibility Check:* If the schedule is deemed feasible, it is sent to the relevant gateways for execution. If not, the algorithm enters an adjustment phase to refine the schedule.
6. *Adjust Schedule with RL Agent:* If the initial schedule is infeasible, the RL agent recalibrates the task assignments to meet QoS requirements, iteratively adjusting the schedule until it becomes feasible or the maximum number of attempts is reached.
7. *Re-evaluate Schedule Feasibility:* The adjusted schedule undergoes another feasibility evaluation to ensure compliance with the required constraints.
8. *Final Outcome:* If a feasible schedule is produced, it is transmitted to the gateways for execution. If a feasible schedule cannot be achieved within the maximum attempts, a failure report is generated, indicating that the task scheduling request could not be fulfilled.

Overall, Fig.4 effectively illustrates the structured workflow of the trained scheduling algorithm, highlighting the interaction between task requests, network state retrieval, schedule generation, feasibility evaluation, and adjustments made by the RL agent to optimize task scheduling in a LoRaWAN network.

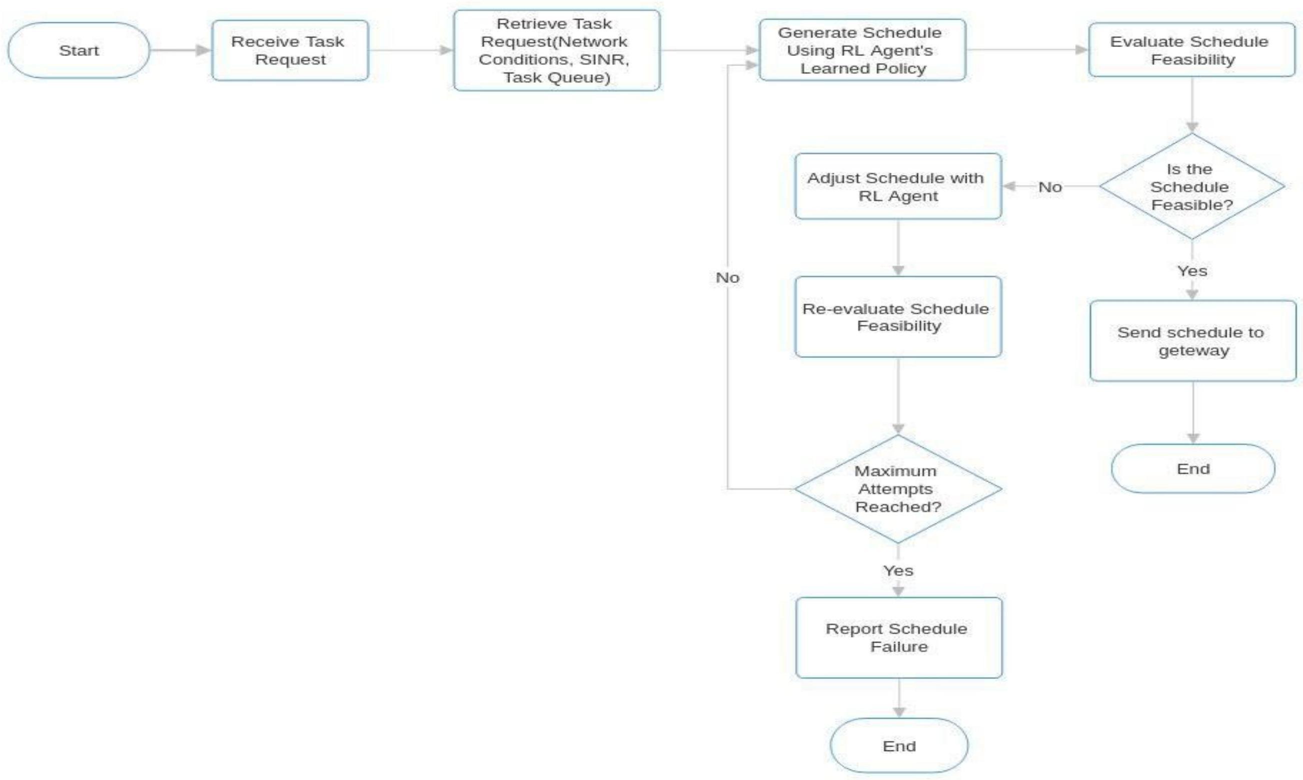
Start
Receive Task Request
Retrieve Task Request(Network Conditions, SINR, Task Queue)
Generate Schedule Using RL Agent's Learned Policy
Evaluate Schedule Feasibility
Is the Schedule Feasible?
No
Adjust Schedule with RL Agent
Re-evaluate Schedule Feasibility
Yes
Send schedule to geteway
End
No
Maximum Attempts Reached?
Yes
Report Schedule Failure
End

Fig. 4 The trained proposed scheduling algorithm diagram

G. Algorithm Implementation

The implementation of the RL-based scheduling algorithm involves translating the designed components into a functional system that operates within the simulated LoRaWAN environment. This process includes several key steps:

1. *Initialization*: The algorithm initializes the RL agent by setting up the state space, action space, and reward structure, along with any necessary parameters for the learning process.
2. *Training Phase*: The agent interacts with the environment through a reinforcement learning loop, in which it observes the current state, selects actions based on its policy, receives rewards, and updates its knowledge (Q-values) to improve future decision-making.
3. *Integration with Simulation*: The algorithm is integrated with the network simulator (NS-3), enabling real-time interaction with the simulated LoRaWAN network. This integration allows the agent to adapt its scheduling decisions based on dynamic network conditions and traffic patterns.
4. *Evaluation*: The performance of the implemented algorithm is evaluated using various QoS metrics, such as delay, packet delivery ratio, and packet error rate, to assess its effectiveness in optimizing task scheduling in the LoRaWAN environment.

Overall, the implementation phase focuses on creating a functional model of the algorithm that can learn and adapt to improve network performance in real-time scenarios.

H. Pseudocode for Task Scheduling Algorithm

The improved task scheduling algorithm for LoRaWAN networks focuses on channel selection, task prioritization, and adaptive gateway placement to enhance QoS parameters. The RL agent interacts with the LoRaWAN environment by observing network states, selecting actions based on policy, receiving rewards, and updating its knowledge to optimize QoS metrics such as delay, reliability, throughput, and energy efficiency.

Pseudocode Structure

1. Initialization
2. State Observation
3. Action Selection
4. Environment Interaction (OpenAI Gym Integration)
5. Reward Calculation
6. Q-Value Update (Learning)
7. Training Loop
8. Policy Improvement and Execution

1. Algorithm 1: Initialization

i. Initialize Q-network with random weights
ii. Initialize target Q-network with the same weights as Q-network
iii. Initialize Replay Memory D with capacity N
iv. Set ϵ for ϵ-greedy policy
v. Set learning rate α, discount factor γ, and batch size
vi. Define action space A = {channel selection, task prioritization, gateway allocation}

```
vii. Define state space S = {channel status, signal strength, gateway congestion, task deadlines}
viii. Define reward function R(s, a) based on QoS metrics
ix. Periodically synchronize target Q-network with Q-network weights every K episodes
```

2. Algorithm 2: State Observation

```
i. Function ObserveState()
ii. Initialize the state as an empty list.
iii. Normalize the current channel status, signal strength (SINR), gateway congestion, and task deadlines.
iv. Append normalized values to the state.
v. Return state.
```

3. Algorithm 3: Action Selection Using ϵ\epsilonϵ-Greedy Policy

```
i. Function SelectAction(state, $\epsilon$)
ii. Generate a random number rand $\in [0, 1]$
iii. if rand < $\epsilon$ then
iv. Choose a random action from action space A
v. else
vi. Compute Q-values for all actions using Q-network
vii. Choose action argmax(Q-values) // Select action with the highest Q-value
viii. end if
ix. return action
```

4. Algorithm 4: Environment Interaction

```
i. Function PerformAction(action)
ii. Initialize the OpenAI Gym environment
iii. if action == "channel selection" then
iv. Select channel with lowest interference and load
v. else if action == "task prioritization" then
vi. Prioritize tasks based on deadlines
vii. else if action == "gateway allocation" then
viii. Assign tasks to gateways with optimal load balancing and signal quality
ix. end if
x. Execute the selected action in the LoRaWAN environment via OpenAI Gym
xi. Observe the resulting state, reward, and whether the episode is done using GetEnvironmentFeedback() from Gym environment
xii. return new state, reward, done
```

5. Algorithm 5: Reward Calculation

```
i. Function CalculateReward(state, action)
ii. Initialize reward = 0
iii. if QoS metrics are improved then
iv. reward += k // Positive reward for improved QoS metrics
v. else
vi. reward -= k // Negative reward for decreased QoS metrics
vii. end if
viii. return reward
```

6. Algorithm 6: Q-Value Update (Learning)

```
i. Function UpdateQNetwork()
ii. Sample a random minibatch of transitions (state, action, reward, next state) from Replay Memory D
iii. for each transition in minibatch do
iv. target = reward
v. if not done then
vi. target += $\gamma \times$ max(target Q-network. predict(next state))
vii. end if
viii. Compute loss as Mean Squared Error (MSE) between target and Q- network. predict(state, action)
ix. Perform gradient descent step to minimize loss
x. end for
xi. Periodically synchronize target Q-network with Q-network weights
```

7. Algorithm 7: Training Loop

```
i. for episode in range(total_episodes) do
ii. state = ObserveState()
iii. done = False
iv. while not done do
v. action = SelectAction(state, $\epsilon$)
vi. new_state, reward, done = PerformAction(action)
vii. Store transition (state, action, reward, new_state, done) in Replay Memory D
viii. if len(Replay Memory) > batch size then
ix. UpdateQNetwork()
x. end if
xi. state = new_state
xii. end while
xiii. if $\epsilon > \epsilon$_min then
xiv. $\epsilon$ *= epsilon_decay // Decay exploration rate
xv. end if
xvi. if episode % evaluation_interval == 0 then
xvii. EvaluatePolicyPerformance()
xviii. end if
xix. end for
```

8. Algorithm 8: Policy Improvement and Execution

```
i. Function EvaluatePolicyPerformance()
ii. Initialize performance metrics
iii. for test episode in range(test episodes) do
iv. state = ObserveState()
v. done = False
vi. while not done do
vii. action = SelectAction(state, $\epsilon$ = 0) // Greedy action selection during evaluation
viii. new state, reward, done = PerformAction(action)
ix. Update performance metrics based on reward and QoS metrics
x. state = new state
xi. end while
xii. end for
xiii. Return metrics
```

I. Algorithm Complexity Analysis

The algorithm complexity analysis encompasses three main aspects: time complexity, space complexity, and scalability and feasibility.

1. Time Complexity

The training time complexity of the RL-based scheduling algorithm is $O(T \times (|S| \times |A| + L \times N^2 + B \log E))$, where T is the number of training episodes, |S| is the number of states, |A| is the number of actions, L is the number of layers, N is the number of neurons per layer, B is the mini-batch size, and E is the total experiences stored. This complexity arises from exploring the state-action space, performing neural network computations, and sampling experiences.

2. Space Complexity

The space complexity is defined as $O(L \times N^2 + E \times M)$, where $L \times N^2$ accounts for the neural network parameters and $E \times M$ represents the memory required for the replay buffer, with M being the memory space per experience tuple. This indicates the memory requirements for both the neural network and the experience replay mechanism.

3. Scalability and Feasibility

The DQN-based algorithm is computationally intensive during the training phase due to the complexity of state-action exploration and neural network computations. However, once trained, the decision-making phase is efficient, requiring only a single forward pass through the neural network. This efficiency makes the algorithm suitable for real-time scheduling tasks in LoRaWAN networks, enabling scalability to handle large numbers of devices. Overall, the analysis highlights the algorithm's computational demands and its potential for effective deployment in resource-constrained environments.

4. Reward Function Design

The reward function design is a critical component of the proposed scheduling algorithm, as it directly influences the reinforcement learning (RL) agent's learning process and the quality of scheduling decisions. The reward function is structured as a weighted sum of various Quality of Service (QoS) metrics, including delay minimization, reliability maximization, and throughput optimization.

5. QoS Metrics

The design incorporates positive rewards for actions that improve QoS metrics, such as:
1. Reducing task completion times,
2. Increasing successful packet delivery rates, and
3. Enhancing overall network throughput.

Conversely, negative rewards are assigned for actions that lead to excessive delays, packet losses, or increased network congestion.

6. Balancing Trade-Offs

The reward function aims to balance trade-offs among different QoS metrics, ensuring that the RL agent can make informed scheduling decisions that optimize overall network performance while adhering to specific constraints.

7. Implementation in Learning

The reward function is integrated into the learning process, guiding the agent's actions based on observed outcomes and facilitating continuous improvement of the scheduling policy through experience replay and Q-value updates. Overall, the reward function design is pivotal in shaping the agent's behavior, promoting effective scheduling strategies that meet the dynamic demands of LoRaWAN networks.

IV. RESULTS AND ANALYSIS

A. Simulation Setup and Scenarios

This section outlines the environment and parameters used to evaluate the proposed scheduling algorithm in a LoRaWAN context.

1. Simulation Environment

The simulations were conducted using the NS-3 simulator, specifically utilizing the ns-3-lora-module to accurately emulate LoRaWAN network characteristics. This environment enables realistic simulations of long-range, low-power communication in an unlicensed spectrum. The simulation area is defined as 200 m × 200 m, with a maximum device-to-gateway communication distance of 200m.

2. Simulation Parameters

Key simulation parameters include:

a. *Number of gateways:* Three, representing the available network infrastructure.
b. *Number of IoT devices:* 100, simulating end-user devices communicating through the gateways.
c. *Network server:* One, responsible for managing communication and data processing.
d. *Environment size:* 200 m × 200 m, defining the controlled simulation area.
e. *Maximum distance to gateway:* 200 m, reflecting LoRaWAN's range capabilities.
f. *Propagation model:* LoRa Log Normal Shadowing, simulating realistic signal propagation conditions.
g. *Number of retransmissions:* Up to five, ensuring packet delivery reliability.
h. *Frequency band:* 868 MHz, commonly used for LoRaWAN communications.

These parameters were selected to create a medium-scale LoRaWAN network that balances complexity, communication reliability, and computational efficiency.

3. Tuning Strategies

The performance of the reinforcement learning-based scheduling algorithm is highly dependent on key parameters, such as the learning rate, batch size, and discount factor. Optimizing these parameters enhances the algorithm's convergence rate and overall effectiveness, ensuring it adapts to varying network conditions and Quality of Service (QoS) requirements. Overall, this section emphasizes the careful design of the simulation environment and parameters to facilitate a comprehensive analysis of the proposed scheduling algorithm's performance in realistic scenarios.

Table I outlines the key parameters used in the LoRaWAN network simulation to evaluate the proposed scheduling algorithm.

1. *Number of gateways:* Set to three, indicating the infrastructure available for communication within the network.
2. *Number of IoT devices:* A total of 100 devices are simulated, representing end-user devices that communicate through the gateways.
3. *Network server:* One network server manages communication and data processing for the IoT devices.
4. *Environment size:* The simulation area is 200 m × 200 m, providing a controlled space for network operations.
5. *Maximum distance to gateway:* The maximum device-to-gateway communication distance is 200 m, reflecting the range capabilities of LoRaWAN technology.
6. *Propagation model:* The LoRaWAN Log Normal Shadowing Model is used to simulate realistic signal propagation conditions, accounting for environmental factors.
7. *Number of retransmissions:* A maximum of five retransmissions is allowed for packet delivery attempts, enhancing reliability.
8. *Frequency band:* The simulation operates on the 868 MHz frequency band, commonly used for LoRaWAN communications.
9. *Spreading factor:* Set to SF7, which determines the data rate and communication range.

These parameters were carefully selected to create a realistic, medium-scale LoRaWAN environment, enabling a detailed investigation of Quality of Service (QoS) metrics and the effectiveness of the scheduling algorithm.

TABLE I SIMULATION PARAMETERS

Parameter	Value
Number of Gateways	3
Number of IoT Devices	100
Network Server	1
Environment Size	200m x200m
Maximum Distance to Gateway	200m
Propagation Model	LoRaLog Normal Shadowing Model
Number of Retransmissions	5(Max)
Frequency Band	868MHz
Spreading Factor	SF7, SF8, SF9, SF10, SF11, SF12
Number of Rounds	1000
Voltage	3.3v
Bandwidth	125kHz
Pay load Length	10 bytes
Time slot Technique	CSMA10
Data Rate(Max)	250kbps
Number of Channels	5
Simulation Time	600 Seconds

The selected parameters were chosen to simulate a realistic, medium-scale LoRaWAN IoT network that balances network complexity, communication reliability, and computational efficiency for reinforcement learning. These parameters are based on widely adopted real-world LoRaWAN configurations while providing the flexibility needed to effectively test a range of QoS metrics and scheduling algorithms in IoT applications.

B. Parameters and Tuning Strategies

The selection of algorithm parameters significantly impacts the performance of the RL-based scheduling method. The following sections outline best practices for adjusting key parameters and explain how these modifications affect the algorithm's performance.

The parameters and tuning strategies for the algorithm are crucial for optimizing performance:

1. *Learning Rate (α):* Set at 0.001, it determines how much new information influences existing knowledge, balancing convergence speed and stability.
2. *Exploration-Exploitation Balance (ε in the ε-greedy strategy):* The exploration rate starts at 1 and decays to 0.1, allowing the agent to explore initially while gradually favoring known actions.
3. *Discount Factor (γ):* Optimized at 0.95, this factor balances the importance of future rewards, promoting a balance between long-term and short-term rewards.
4. *Batch Size for Training:* An optimal batch size of 128 is used to achieve faster convergence and effective generalization, avoiding overfitting or underfitting issues.

Table II presents the key parameters used in the reinforcement learning-based scheduling algorithm, which are critical for its performance and effectiveness:

1. *Number of Hidden Layers:* Set to 2, indicating the depth of the neural network used in the scheduling algorithm.
2. *Number of Neurons per Layer:* Each hidden layer contains 128 neurons, influencing the network's ability to learn complex patterns and relationships in the data.
3. *Learning Rate (α):* Fixed at 0.001, this parameter controls the magnitude of updates to network weights during training, affecting convergence speed and stability.
4. *Discount Factor (γ):* Set to 0.95, this factor balances immediate and future rewards, guiding the agent's long-term decision-making.
5. *Exploration Rate (ε):* Initialized at 1.0, this rate determines the likelihood of the agent exploring new actions versus exploiting known actions, promoting early exploration.
6. *Exploration Decay Rate:* Set at 0.995, this parameter gradually reduces the exploration rate over time, allowing the agent to focus more on exploitation as it learns.
7. *Minimum Exploration Rate:* Fixed at 0.01, ensuring the agent retains a small chance of exploring new actions even after extensive training.
8. *Replay Buffer Size:* Set to 30,000, defining the capacity of the experience replay buffer, which stores past experiences for training stability.
9. *Batch Size:* Fixed at 64, determining the number of experiences sampled per training iteration, balancing convergence speed and generalization.
10. *Target Network Update Frequency:* Set to every 500 steps, specifying how often the target network's weights are synchronized with the main Q-network, aiding in stable learning.

These parameters are essential for optimizing the scheduling algorithm's performance, ensuring effective learning and adaptation to the dynamic conditions of the LoRaWAN network.

TABLE II ALGORITHM PARAMETERS

Parameter	Value
Number of Hidden Layers	2
Number of Neurons per Layer	128
Learning Rate	0.001
Discount Factor (Gamma)	0.95
Exploration Rate (Epsilon)	1.0
Exploration Decay Rate	0.995
Minimum Exploration Rate	0.01
Replay Buffer Size	30,000
Batch Size	64
Target Network Update Frequency	Every 500 steps
Activation Function	ReLU
Optimizer	Adam
Loss Function	Mean Squared Error

E. Performance Metrics Analysis

The Performance Metrics Analysis evaluates the effectiveness of the proposed algorithm using key indicators, including delay, reliability, and throughput. The analysis demonstrates significant improvements in these metrics compared to baseline policies, highlighting the algorithm's ability to optimize Quality of Service (QoS) in LoRaWAN networks. Overall, the results indicate that the reinforcement learning (RL)-based scheduling approach enhances network performance, particularly in managing overlapping QoS requirements.

F. Network Delay

Fig. 5 illustrates the correlation between network delay and the number of nodes in a LoRaWAN environment.

1. *Trend Analysis:* The graph generally shows that as the number of nodes increases, network delay also increases. This trend indicates growing contention for communication resources, leading to longer wait times for packet transmission.
2. *Comparison of Algorithms:* The figure likely compares the delay performance of different scheduling algorithms, such as the proposed RL-based algorithm and traditional methods like LoRa+ and RT-LoRa. The RL-based algorithm is expected to exhibit significantly lower delays, demonstrating its effectiveness in optimizing resource allocation and scheduling tasks.
3. *Implications for QoS:* The results presented in this figure underscore the importance of efficient scheduling in maintaining low latency, particularly in high-density node scenarios. This is crucial for applications requiring real-time data transmission, reinforcing the need for advanced algorithms to effectively manage network performance.

Overall, Fig. 5 provides valuable insights into the impact of node density on network delay and the performance advantages of the proposed scheduling approach.

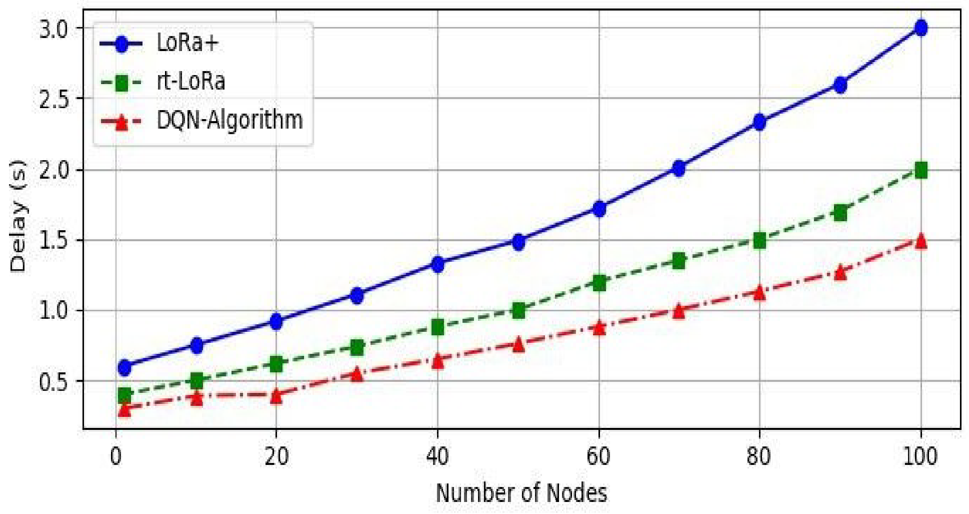

Fig. 5 Delay vs Number of Nodes

G. Packet Delivery Ratio (PDR)

Fig. 6 illustrates the relationship between the Packet Delivery Ratio (PDR) and the number of nodes in a LoRaWAN network.

1. *PDR Trends:* The graph typically shows that as the number of nodes increases, the PDR may initially rise but eventually plateau or decline. This behavior suggests that while a greater number of nodes can enhance network coverage, increased contention and potential collisions may negatively affect packet delivery.
2. *Algorithm Comparison:* The figure highlights the performance of the proposed RL-based algorithm (DQN) in achieving the highest PDR compared to other algorithms, such as RT-LoRa and LoRa+. This finding indicates that the RL-based approach effectively manages scheduling and resource allocation, thereby minimizing packet loss.
3. *Significance for Network Performance:* The PDR is a critical metric for assessing communication reliability in IoT networks. A higher PDR signifies improved performance and reliability, which is essential for applications requiring consistent data transmission. These findings underscore the importance of advanced scheduling techniques in optimizing network performance.

Overall, Fig. 6 highlights the impact of node density on packet delivery success and demonstrates the advantages of the proposed algorithm in maintaining high delivery ratios.

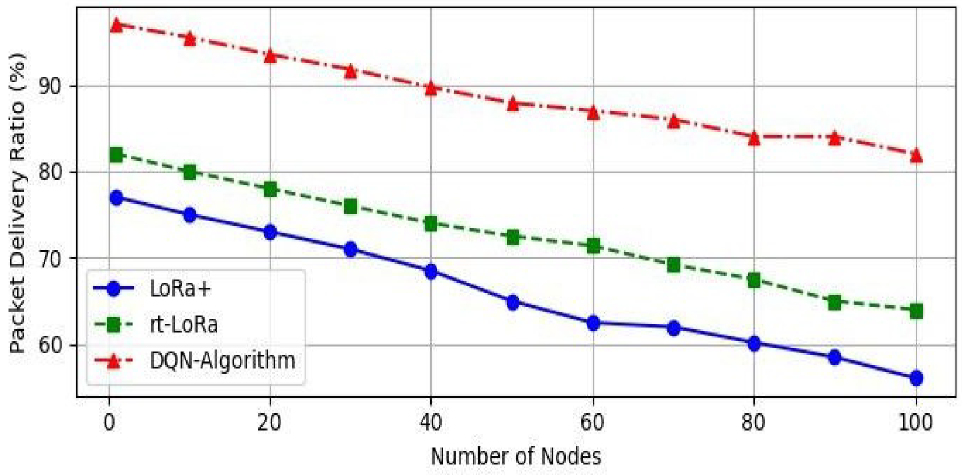

Fig. 6 Packet Delivery Ratio (PDR) vs Number of Nodes

H. Packet Error Rate (PER)

Fig. 7 illustrates the relationship between the Packet Error Rate (PER) and the number of nodes in a LoRaWAN network.

1. *PER Trends:* The graph typically shows that as the number of nodes increases, the PER also rises, indicating a higher percentage of packets experiencing errors during transmission. This trend reflects the increased likelihood of packet collisions and interference in a congested network environment.
2. *Algorithm Performance:* The figure highlights that the RL-based algorithm exhibits the lowest PER compared to other algorithms, such as RT-LoRa and LoRa+. This reduction in PER is attributed to the dynamic optimization of scheduling and resource allocation performed by the RL-based approach, which effectively mitigates packet collisions and transmission errors.
3. *Implications for Network Reliability:* A lower PER is essential for ensuring reliable communication in IoT applications, as it directly impacts overall network performance and efficiency. The results presented in this figure underscore the importance of employing advanced scheduling algorithms to enhance network reliability and minimize transmission errors, particularly in high-node-density scenarios.

Overall, Fig. 7 highlights the correlation between node density and packet error rates, demonstrating the effectiveness of the proposed RL-based algorithm in reducing errors in congested networks.

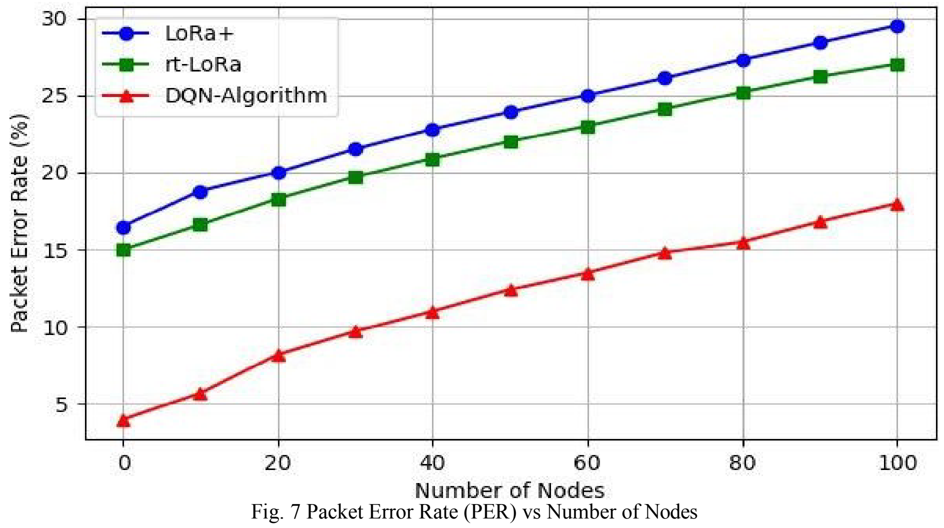

Fig. 7 Packet Error Rate (PER) vs Number of Nodes

I. Throughput

Fig. 8 illustrates the relationship between throughput and the number of nodes in a LoRaWAN network. The figure shows that as the number of nodes increases, the RL-based algorithm (DQN) achieves significantly higher throughput compared to other algorithms, such as RT-LoRa and LoRa+.

This superior performance is attributed to the RL-based algorithm’s dynamic optimization of scheduling decisions, which effectively balances network load and minimizes collisions, resulting in enhanced data transmission rates even as node density increases.

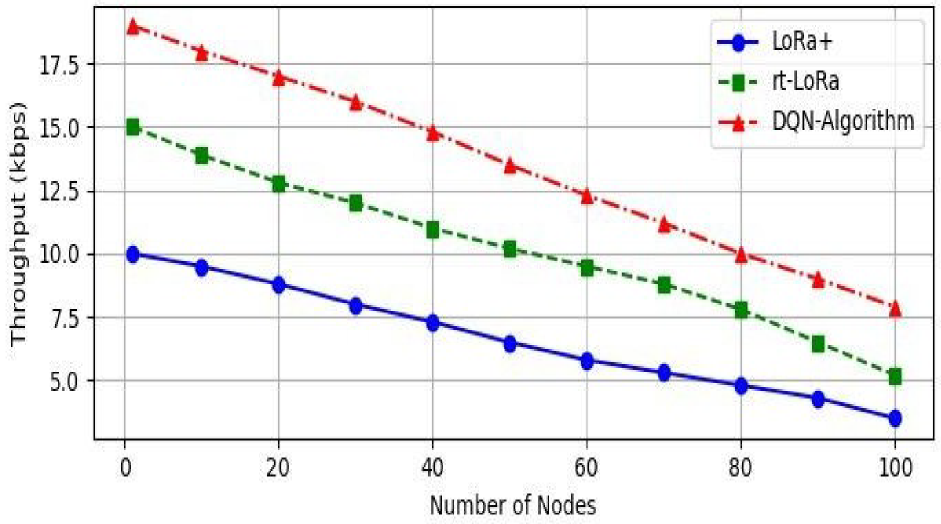

Fig. 8 Throughput vs Number of Nodes

V. CONCLUSION

This section concludes our investigation into developing a reinforcement learning-based, QoS-aware task scheduling algorithm for LoRaWAN IoT applications. Researchers examined the limitations of existing scheduling techniques in meeting the diverse QoS requirements of modern IoT applications, particularly in dynamic and large-scale LoRaWAN networks. To address these challenges, we designed and evaluated a reinforcement learning-based task scheduling algorithm aimed at optimizing key QoS metrics, including throughput, latency, and reliability. Our study demonstrated the significant potential of reinforcement learning in enhancing task scheduling for LoRaWAN networks. We conducted a comprehensive simulation-based analysis using NS-3 to compare the performance of our RL-based scheduler against baseline techniques, such as RT-LoRa and LoRa+. The results consistently showed that the RL-based approach outperformed baseline algorithms, particularly in dynamic traffic scenarios. Specifically, during periods of high network traffic, the RL scheduler effectively adapted, achieving a significantly reduced average delay. Additionally, compared to baseline methods, our RL scheduler consistently exhibited lower packet error rates and higher packet delivery ratios, indicating improved reliability. Furthermore, the RL-based method achieved higher average throughput, demonstrating its efficiency in managing heavy network loads and optimizing data transmission rates.

Our findings confirm that reinforcement learning can effectively address the challenges of existing LoRaWAN scheduling techniques, particularly in meeting QoS requirements such as throughput, latency, and reliability. For real-world LoRaWAN deployments, the ability of an RL-based scheduler to learn and adapt to dynamic network conditions, traffic patterns, and device characteristics is crucial for optimizing QoS. This study provides a strong foundation for the development of intelligent scheduling systems for LoRaWAN networks across various sectors. The optimized scheduling approach has the potential to enhance the performance of numerous applications, including industrial automation, smart cities, and remote healthcare monitoring.

VI. PROSPECTS FOR FURTHER RESEARCH

Although our study provides valuable insights into the potential of reinforcement learning (RL) for LoRaWAN scheduling, several aspects require further investigation and development:

1. *Exploring Alternative RL Approaches:* Further improvements in scheduling performance may be achieved by evaluating additional reinforcement learning algorithms, such as deep reinforcement learning architectures, policy gradient methods, or other advanced techniques.
2. *Addressing Security and Privacy Concerns:* Ensuring the security and privacy of IoT applications in LoRaWAN scheduling is essential. Future research can explore methods for integrating security measures into the RL-based algorithm.
3. *Real-World Validation:* Extensive testing and evaluation in realistic environments are necessary for the real-world implementation of the proposed scheduler. Future studies should assess its performance under varying network dynamics, communication latencies, and device heterogeneity.

Declaration of Conflicting Interests
The authors declare no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding
The authors received no financial support for the research, authorship, and/or publication of this article.

Use of Artificial Intelligence (AI)-Assisted Technology for Manuscript Preparation
The authors confirm that no AI-assisted technologies were used in the preparation or writing of the manuscript, and no images were altered using AI.

REFERENCES

[1] Mahmood, N. H., Marchenko, N., Gidlund, M., & Popovski, P. (2020). *Wireless networks and industrial IoT: Applications, challenges and enablers*. Springer. https://doi.org/10.1007/978-3-030-51473-0

[2] de Oliveira, L. R., de Moraes, P., Neto, L. P. S., & da Conceição, A. F. (2020). Review of LoRaWAN applications. *arXiv preprint*. http://arxiv.org/abs/2004.05871

[3] Marais, J. M., Malekian, R., & Abu-Mahfouz, A. M. (2017). LoRa and LoRaWAN testbeds: A review. *IEEE AFRICON 2017*, 1496–1501. https://doi.org/10.1109/AFRCON.2017.8095703

[4] Mekki, K., Bajic, E., Chaxel, F., & Meyer, F. (2018). Overview of cellular LPWAN technologies for IoT deployment: Sigfox, LoRaWAN, and NB-IoT. *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 197–202. https://doi.org/10.1109/PERCOMW.2018.8480255

[5] Bouguera, T., Diouris, J. F., Chaillout, J. J., Jaouadi, R., & Andrieux, G. (2018). Energy consumption model for sensor nodes based on LoRa and LoRaWAN. *Sensors, 18*(7), 1–23. https://doi.org/10.3390/s18072104

[6] Augustin, A., Yi, J., Clausen, T., & Townsley, W. M. (2016). A study of LoRa: Long range & low power networks for the Internet of Things. *Sensors, 16*(9), 1–18. https://doi.org/10.3390/s16091466

[7] Ragnoli, M., Barile, G., Leoni, A., Ferri, G., & Stornelli, V. (2020). An autonomous low-power LoRa-based flood-monitoring system. *Journal of Low Power Electronics and Applications, 10*(2), 1–15. https://doi.org/10.3390/jlpea10020015

[8] Haxhibeqiri, J., Moerman, I., & Hoebeke, J. (2019). Low overhead scheduling of LoRa transmissions for improved scalability. *IEEE Internet of Things Journal, 6*(2), 3097–3109. https://doi.org/10.1109/JIOT.2018.2878942

[9] Sutton, R. S., & Barto, A. G. (2012). *Reinforcement learning: An introduction* (2nd ed.). MIT Press. http://incompleteideas.net/sutton/book/the-book.html

[10] Petäjäjärvi, J., Mikhaylov, K., Pettissalo, M., Janhunen, J., & Iinatti, J. (2017). Performance of a low-power wide-area network based on LoRa technology: Doppler robustness, scalability, and coverage. *International Journal of Distributed Sensor Networks, 13*(3), 1–14. https://doi.org/10.1177/1550147717699412

[11] Polonelli, T., Brunelli, D., Marzocchi, A., & Benini, L. (2019). Slotted ALOHA on LoRaWAN: Design, analysis, and deployment. *Sensors, 19*(4), 1–15. https://doi.org/10.3390/s19040838

[12] Alenezi, M., Chai, K. K., Alam, A. S., Chen, Y., & Jimaa, S. (2020). Unsupervised learning clustering and dynamic transmission scheduling for efficient dense LoRaWAN networks. *IEEE Access, 8*, 191495–191509. https://doi.org/10.1109/ACCESS.2020.3031974

[13] Leonardi, L., Battaglia, F., & Lo Bello, L. (2019). RT-LoRa: A medium access strategy to support real-time flows over LoRa-based networks for industrial IoT applications. *IEEE Internet of Things Journal, 6*(6), 10812–10823. https://doi.org/10.1109/JIOT.2019.2942776

[14] Sallum, E., Pereira, N., Alves, M., & Santos, M. (2020). Improving quality-of-service in LoRa low-power wide-area networks through optimized radio resource management. *Journal of Sensor and Actuator Networks, 9*(1), 1–26. https://doi.org/10.3390/jsan9010010

[15] Micheletto, M., Zabala, P., Ochoa, S. F., Meseguer, R., & Santos, R. (2023). Determining real-time communication feasibility in IoT systems supported by LoRaWAN. *Sensors, 23*(9), 1–27. https://doi.org/10.3390/s23094281

[16] Garrido-Hidalgo, C., et al. (2021). LoRaWAN scheduling: From concept to implementation. *IEEE Internet of Things Journal, 8*(16), 12919–12933. https://doi.org/10.1109/JIOT.2021.3064430

[17] Siddiqi, U. F., Sait, S. M., & Uysal, M. (2020). Deep reinforcement-based power allocation for the max-min optimization in non-orthogonal multiple access. *IEEE Access, 8*, 211235–211247. https://doi.org/10.1109/ACCESS.2020.3038923.