

Research Article

QoS-Aware Task Scheduling Using Reinforcement Learning in Long Range Wide Area Network IoT Application

Ermias Melku Tadesse^{1,*} , Haimanot Edmealem¹, Tesfaye Belay², Abubeker Girma³

¹Information Technology Department, Kombolcha Institute of Technology, Wollo University, Kombolcha, Ethiopia

²Department of Computer Science, Institute of Technology, Wollo University, Kombolcha, Ethiopia

³Software Engineering Department, Kombolcha Institute of Technology, Wollo University, Kombolcha, Ethiopia

Abstract

In order to solve the problems of effective resource allocation in low-power wide-area networks, this thesis investigates the scheduling of end devices in Internet of Things applications using LoRaWAN technology. The main goal of this research is to use RL to improve QoS measures including energy efficiency, throughput, latency, and dependability. This was accomplished by using a simulation-based approach that evaluated the effectiveness of the RL-based scheduling algorithm using NS3 simulations. The main findings show that, in comparison to current scheduling practices, the RL agent greatly improves data transmission reliability and improves network throughput. Furthermore, the suggested approach efficiently lowers average system latency and overall energy usage, improving network resource utilization. These findings imply that using reinforcement learning (RL) for job scheduling in LoRaWAN networks can offer a reliable and expandable solution to present problems, resulting in more intelligent and environmentally friendly IoT systems. In the end, this study finds that using RL-based techniques can help improve resource management in contexts that are dynamic and resource-constrained.

Keywords

IoT, LoRaWAN, Reinforcement Learning, Task Scheduling, QoS

1. 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 high-

lighted as a significant enabler for 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 which is LPWAN technology can connect battery-powered devices at

*Corresponding author: ermiasmelku3400@gmail.com (Ermias Melku Tadesse)

Received: 10 March 2025; **Accepted:** 26 March 2025; **Published:** 19 April 2025



Copyright: © The Author(s), 2025. Published by Science Publishing Group. This is an **Open Access** article, distributed under the terms of the Creative Commons Attribution 4.0 License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited.

very long distances while consuming minimum power, hence making it affordable [4].

LoRaWAN operates in the unlicensed ISM bands, which vary according to region [5]. It employs chirp spread spectrum modulation technique to attain long-distance communication with low power [6]. One of the main advantages of LoRaWAN is its remarkable coverage. For this reason, it can transmit data within several kilometers in open settings such as rural areas or large industrial facilities without the need for cellular towers and other infrastructure items. Consequently, LoRaWAN is best suitable for applications that need a wider coverage area, such as smart agriculture, asset tracking, environmental monitoring, and smart city deployments [1]. Hence, LoRaWAN has become an attractive technology for IoT applications due to its unique combination of long-range capability, low power consumption, and cost-effective deployment [7] and LoRaWAN relies on four key components [8].

Figure 1 depicts 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, illustrating 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, where it is processed and routed to the appropriate application server for further analysis or action, showcasing the hierarchical structure and functionality of the LoRaWAN ecosystem.

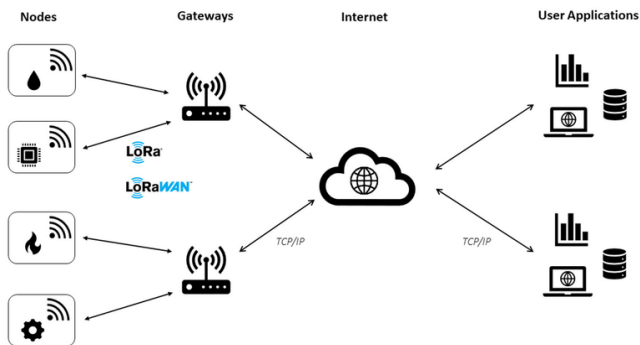


Figure 1. LoRaWAN Network Architecture [8].

In LoRaWAN IoT applications, maintaining Quality of Service (QoS) is crucial due to challenges like limited resources, channel congestion, and varying QoS requirements, which can lead to issues such as high latency and packet loss. Reinforcement Learning (RL) is identified as the most 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].

The study proposes the use of reinforcement learning (RL) techniques to develop a scheduling algorithm that can adapt to dynamic network conditions, optimize energy consumption, and enhance overall system performance. By leveraging 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].

Contributions of the Article

- 1) *Development of an RL-Based Scheduling Algorithm:* The 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:* It 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 the article offer valuable insights and a foundation for future research in the field of IoT and LoRaWAN, encouraging further exploration of adaptive and intelligent scheduling techniques to address the evolving challenges in network management.

2. Related Work

LPWANs like LoRaWAN have revolutionized the IoT by enabling long-range communication with battery-powered devices. However, IoT applications within the IoT domain demand reliable and expedited data delivery, posing challenges for LoRaWAN due to inherent limitations in range, latency, and energy constraints [12]. This review explores existing research related to Task Scheduling in LoRaWAN.

The paper [13] proposes a dynamic transmission Priority Scheduling Technique (PST) based on an unsupervised learning clustering algorithm for dense LoRaWAN networks. 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 approach aims to improve transmission delay, and decrease energy consumption. Simulation results suggest that the proposed work outperforms conventional LoRaWAN and recent clustering and scheduling schemes, making it potentially well suited for dense LoRaWAN deployments.

In [14] a Real-Time LoRa (RT-LoRa) communication protocol for industrial Internet of things applications is introduced. The real-time flow is processed by the RT-LoRa using a medium access strategy. Static and movable nodes are used to build the entire network. The QoS level is regarded as being the same for every static node. Three classes-normal, dependable, and most reliable-are used to categorize the QoS

level for flows produced by mobile nodes. The technique distributes SF and CF based on the QoS level. A star topology is used to arrange and connect the mobile and static nodes to the gateway. The following are the important points raised in this paper: For a single gateway network using single hop communication, the general process is described. Even within 180 meters, this results in a significant transmission delay of up to 28 seconds for the majority of dependable flows. This study has not addressed the need for greater coverage and reduced time delay for industrial data in real-time. There are limitations in QoS provisioning because the QoS level is only assigned to mobile nodes and all static node flows are given the same priority level. All nodes need a lot of energy to connect with the central gateway, and nodes farther from the gateway use even more energy.

The paper [15] proposes a method to optimize the performance of LoRaWAN networks through dynamically assigning values for the Spreading Factor and Carrier Frequency radio parameters. This assignment is formulated as a Mixed Integer Linear Programming problem to maximize network metrics like Data Extraction Rate and minimize packet collisions. An approximation algorithm is also developed to solve the problem more efficiently at scale. The results show improved performance for metrics like DER and average 6-13% fewer packet collisions compared to baseline policies. The performance evaluation of the proposed optimization algorithms is done through simulation using the LoRaSim simulator. The optimization focuses on optimizing just the SF and CF parameters of the LoRa radio configuration. Considering additional parameters could lead to even better performance. The simulations assume a single gateway setup. Therefore, in summary, the key limitations are limited configuration parameters, static network assumptions and evaluation based on few metrics.

In [16], the authors explore the viability of real-time communication within LoRaWAN-based IoT systems. Leveraging an integer linear programming (ILP) model, they assess the feasibility of real-time communication during the network design stage. This model not only determines feasibility but also optimizes the number and placement of gateways necessary to achieve real-time requirements. The paper further validates the model's performance through various scenarios, offering valuable insights into LoRaWAN's scalability and real-time support limitations. However, it is important to note that the model primarily focuses on static network design at deployment. This may not fully capture the dynamic nature of real-world networks, where factors like interference, congestion, and gateway availability can significantly affect real-time QoS performance.

In [17], the authors present a low-overhead synchronization and scheduling concept implemented on top of LoRaWAN Class A. They design and deploy an end-to-end architecture on STM32L0 microcontrollers (MCUs), where a central entity provides synchronization metrics and allocate transmission slots. By measuring clock drift in devices, the system

defines slot lengths within the network. This approach achieves 10-millisecond accuracy and demonstrates significant improvements in packet delivery ratios compared to Aloha-based setups, especially under high network loads. Notably, the paper addresses the gap in the literature regarding experimental approaches to LoRaWAN scheduling and demonstrates the feasibility of the proposed concept. However, the paper does not delve into the energy consumption impact of the implemented scheduling algorithms.

A number of existing studies have proposed different methods to reduce the retransmissions, including adaptive retry limits and error correction mechanisms. However, most of these methods fundamentally fail to adapt dynamically with changing network conditions, and this issue is addressed in the proposed reinforcement learning-based scheduling algorithm.

3. Research Methodology

3.1. Proposed Method

The research methodology focuses on designing and implementing a reinforcement learning (RL)-based scheduling algorithm for reliable data delivery in LoRaWAN networks. It adopts a design science research (DSR) approach, which emphasizes 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, which emphasizes the need for a task scheduling algorithm that can effectively manage resources in dynamic environments. The study identifies the limitations of existing scheduling methods in LoRaWAN networks, particularly their inability to meet the QoS demands of modern IoT applications. To address these challenges, the research proposes a reinforcement learning (RL) based algorithm that can adapt to varying network conditions and optimize resource allocation.

3.2. Research Design

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

3.3. Algorithm Design and Implementation

3.3.1. 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.

1. **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.
2. **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.
3. **Reward Function:** The reward function is designed to provide 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.
4. **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.
5. **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.

Figure 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 can include various features relevant to the task at hand. 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 (in this case, 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 about which action to take.
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.

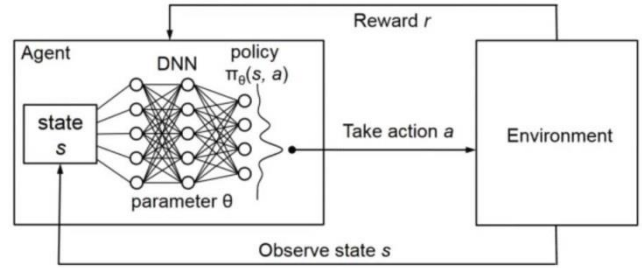


Figure 2. DQN Architecture [18].

The diagram shows 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 allows the agent to learn an optimal policy for maximizing rewards in the environment.

Here's a breakdown of the diagram's elements:

- 1) **Agent:** This is 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 neural network (DNN) with parameters θ .
- 2) **Environment:** This is the external world 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 encourage behaviors, while negative rewards discourage them.
- 6) **Policy (π):** A function that maps states to actions. In DRL, it's often represented as a neural network.

a). Training Phase of the Proposed Scheduling Algorithm

Figure 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 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 exploration of new actions

and 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 re-

play 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 the performance converges to an acceptable level.

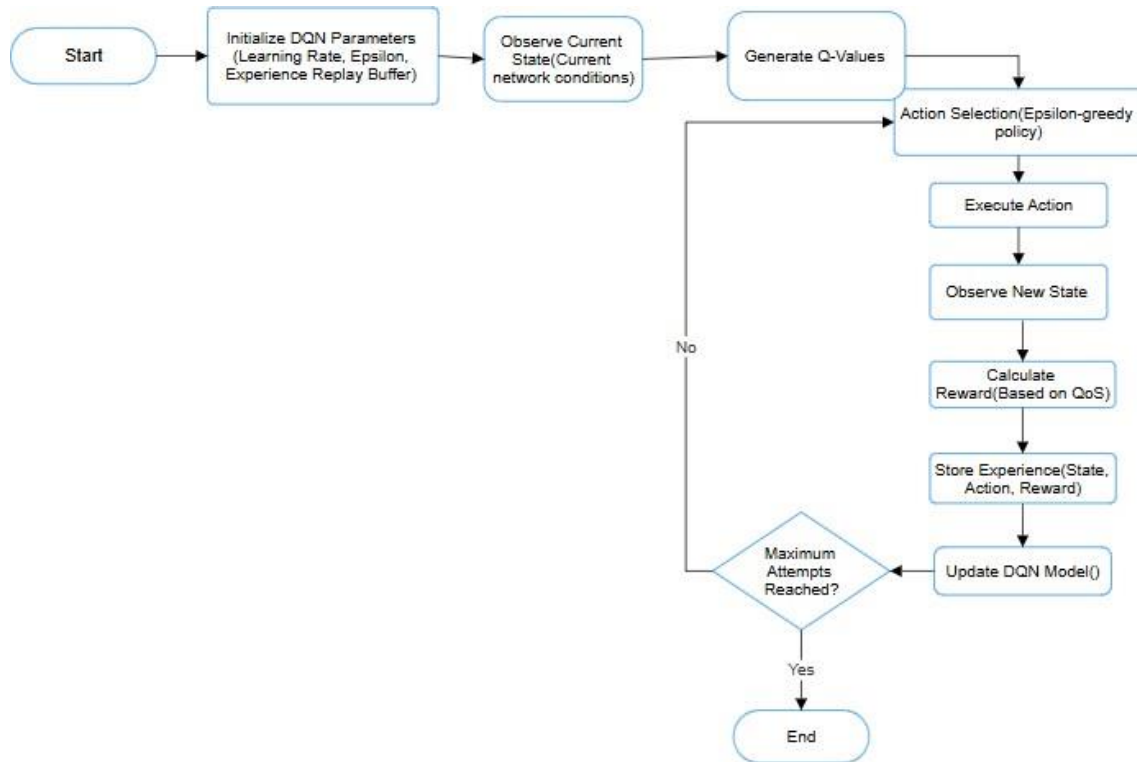


Figure 3. Training Phase of the Proposed Scheduling Algorithm.

b). The Trained Proposed Scheduling Algorithm Diagram

Figure 4 presents a diagram of the trained proposed 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 like 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:** In cases where the initial schedule is infeasible, the RL agent recalibrates the task assignments to meet the 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, Figure 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.

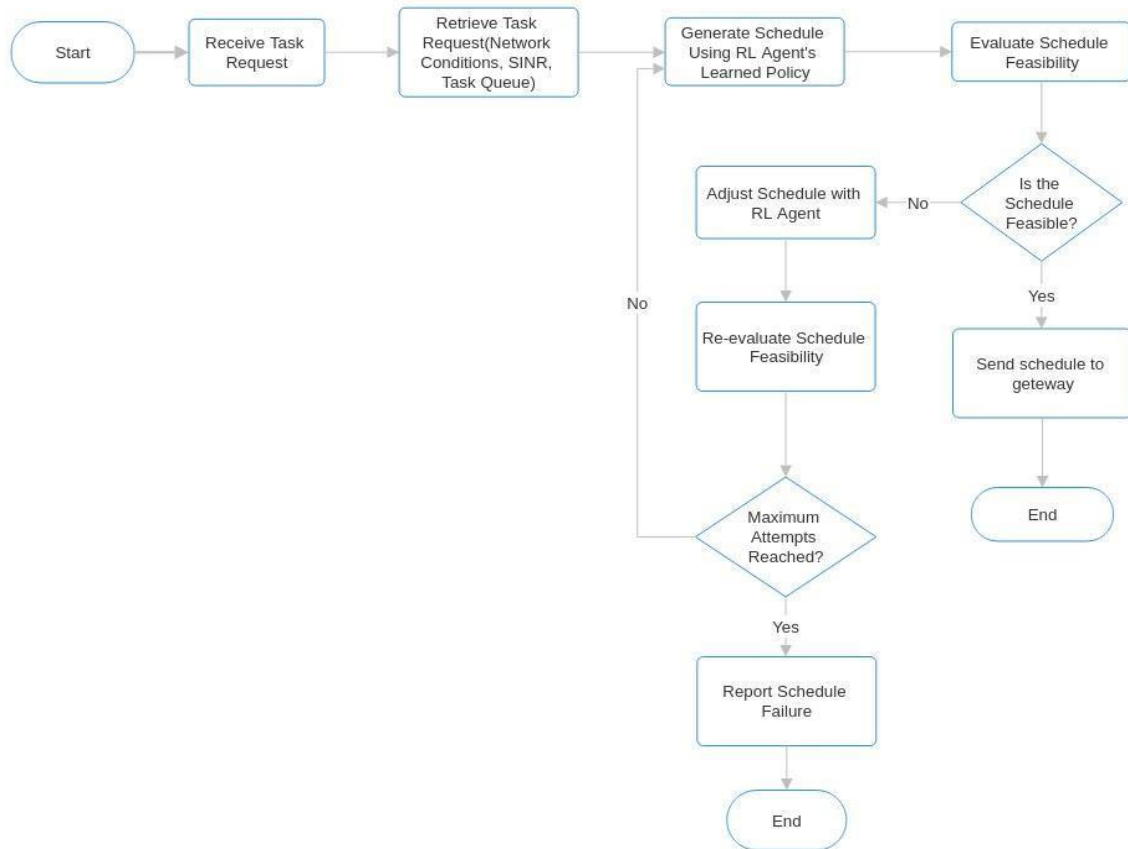


Figure 4. The trained proposed scheduling algorithm diagram.

3.3.2. 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, 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, where 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), allowing for real-time interaction with the simulated LoRaWAN

network. This integration enables 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 working model of the algorithm that can learn and adapt to improve network performance in real-time scenarios.

3.4. Pseudocode for Task Scheduling Algorithm

The LoRaWAN networks improved task scheduling algorithm focuses on channel selection, task priority, and adaptive gateway placement in order to achieve better QoS parameters. The RL agent interacts with the LoRaWAN environment,

observes network states, selects actions based on policy, receives rewards, and updates its knowledge to optimize QoS metrics like delay, reliability, throughput, and energy efficiency.

1. 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

2. Algorithm 1 Initialization

- 1) Initialize Q -network with random weights
- 2) Initialize target Q -network with the same weights as Q -network
- 3) Initialize Replay Memory D with capacity N
- 4) Set ϵ for ϵ -greedy policy
- 5) Set learning rate α , discount factor γ , and batch size
- 6) Define action space $A = \{\text{channel selection, task prioritization, gateway allocation}\}$
- 7) Define state space $S = \{\text{channel status, signal strength, gateway congestion, task deadlines}\}$
- 8) Define reward function $R(s, a)$ based on QoS metrics
- 9) Periodically synchronize target Q -network with Q -network weights every K episodes

3. Algorithm 2 State Observation

- 1) Function $\text{ObserveState}()$
- 2) Initialize state as an empty list
- 3) Normalize current channel status, signal strength (SINR), gateway congestion, and task deadlines
- 4) Append normalized values to state
- 5) return state

4. Algorithm 3 Action Selection using ϵ -Greedy Policy

- 1) Function $\text{SelectAction}(\text{state}, \epsilon)$
- 2) Generate a random number $\text{rand} \in [0, 1]$
- 3) if $\text{rand} < \epsilon$ then
- 4) Choose a random action from action space A
- 5) else
- 6) Compute Q -values for all actions using Q -network
- 7) Choose action $\text{argmax}(Q\text{-values})$ // Select action with the highest Q -value
- 8) end if
- 9) return action

5. Algorithm 4 Environment Interaction

- 1) Function $\text{PerformAction}(\text{action})$
- 2) Initialize the OpenAI Gym environment
- 3) if $\text{action} == \text{"channel selection"}$ then
- 4) Select channel with lowest interference and load
- 5) else if $\text{action} == \text{"task prioritization"}$ then
- 6) Prioritize tasks based on deadlines
- 7) else if $\text{action} == \text{"gateway allocation"}$ then
- 8) Assign tasks to gateways with optimal load balancing and signal quality

9) end if

10) Execute the selected action in the LoRaWAN environment via OpenAI Gym

11) Observe the resulting state, reward, and whether the episode is done using $\text{GetEnvironmentFeedback}()$ from Gym environment

12) return new state, reward, done

6. Algorithm 5 Reward Calculation

- 1) Function $\text{CalculateReward}(\text{state}, \text{action})$
- 2) Initialize $\text{reward} = 0$
- 3) if QoS metrics are improved then
- 4) $\text{reward} += k$ // Positive reward for improved QoS metrics
- 5) else
- 6) $\text{reward} -= k$ // Negative reward for decreased QoS metrics
- 7) end if
- 8) return reward

7. Algorithm 6 Q-Value Update (Learning)

- 1) Function $\text{UpdateQNetwork}()$
- 2) Sample a random minibatch of transitions (state, action, reward, next state) from Replay Memory D
- 3) for each transition in minibatch do
- 4) $\text{target} = \text{reward}$
- 5) if not done then
- 6) $\text{target} += \gamma \times \max(\text{target } Q\text{-network.predict}(\text{next state}))$
- 7) end if
- 8) Compute loss as Mean Squared Error (MSE) between target and Q -network.predict(state, action)
- 9) Perform gradient descent step to minimize loss
- 10) end for
- 11) Periodically synchronize target Q -network with Q -network weights

8. Algorithm 7 Training Loop

- 1) for episode in range(total_episodes) do
- 2) $\text{state} = \text{ObserveState}()$
- 3) $\text{done} = \text{False}$
- 4) while not done do
- 5) $\text{action} = \text{SelectAction}(\text{state}, \epsilon)$
- 6) $\text{new_state}, \text{reward}, \text{done} = \text{PerformAction}(\text{action})$
- 7) Store transition (state, action, reward, new_state, done) in Replay Memory D
- 8) if $\text{len}(\text{Replay Memory}) > \text{batch size}$ then
- 9) $\text{UpdateQNetwork}()$
- 10) end if
- 11) $\text{state} = \text{new_state}$
- 12) end while
- 13) if $\epsilon > \epsilon_{\text{min}}$ then
- 14) $\epsilon *= \text{epsilon_decay}$ // Decay exploration rate
- 15) end if
- 16) if $\text{episode \% evaluation_interval} == 0$ then
- 17) $\text{EvaluatePolicyPerformance}()$
- 18) end if
- 19) end for

9. Algorithm 8 Policy Improvement and Execution
 - 1) *Function EvaluatePolicyPerformance()*
 - 2) *Initialize performance metrics*
 - 3) *for test_episode in range(test episodes) do*
 - 4) *state = ObserveState()*
 - 5) *done = False*
 - 6) *while not done do*
 - 7) *action = SelectAction(state, $\epsilon = 0$) // Greedy action selection during evaluation*
 - 8) *new state, reward, done = PerformAction(action)*
 - 9) *Update performance metrics based on reward and QoS metrics*
 - 10) *state = new_state*
 - 11) *end while*
 - 12) *end for*
 - 13) *Return metrics*

3.5. 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, making it suitable for real-time scheduling tasks in LoRaWAN networks and 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.

3.6. Reward Function Design

The reward function design is a critical component of the proposed scheduling algorithm, as it directly influences the

learning process of the reinforcement learning (RL) agent 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.

1. **QoS Metrics:** The design incorporates positive rewards for actions that improve QoS metrics, such as reducing task completion times, increasing successful packet delivery rates, and enhancing overall network throughput. Conversely, negative rewards are assigned for actions that lead to excessive delays, packet losses, or increased network congestion.
2. **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.
3. **Implementation in Learning:** The reward function is integrated into the learning process, guiding the agent's actions based on the observed outcomes and facilitating the 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.

4. Result and Analysis

4.1. Simulation Setup and Scenarios

The simulation setup and scenarios 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 allows for realistic simulations of long-range, low-power communication in an unlicensed spectrum, with a defined area of 200m x 200m and a maximum distance of 200m to the gateway.
 2. **Parameters:** Key simulation parameters include three gateways, 100 IoT devices, one network server, a LoRa Log Normal Shadowing propagation model, a frequency band of 868MHz, and a maximum of five retransmissions. 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 the choice of parameters, such as learning rate, batch size, and discount factor. The section discusses the importance of optimizing these parameters to enhance the convergence rate and overall effectiveness of the algorithm, ensuring it can adapt to varying network conditions and QoS requirements.
- Overall, this section emphasizes the careful design of the simu-

lation environment and parameters to facilitate a comprehensive analysis of the proposed scheduling algorithm's performance in realistic scenarios.

Table 1 outlines the key parameters used in the simulation of the LoRaWAN network to evaluate the proposed scheduling algorithm. The parameters include:

- (1) Number of Gateways: Set to 3, indicating the infrastructure available for communication within the network.
- (2) Number of IoT Devices: A total of 100 devices are simulated, representing the end-user devices that will communicate through the gateways.
- (3) Network Server: There is 1 network server managing the communication and data processing for the IoT devices.
- (4) Environment Size: The simulation area is defined as 200m x 200m, providing a controlled space for the network operations.
- (5) Maximum Distance to Gateway: The maximum communication distance for devices to the gateway is set at 200m, reflecting the range capabilities of LoRaWAN technology.
- (6) Propagation Model: The LoRa Log Normal Shadowing Model is used to simulate realistic signal propagation conditions, accounting for environmental factors.
- (7) Number of Retransmissions: A maximum of 5 retransmissions is allowed for packet delivery attempts, enhancing reliability.
- (8) Frequency Band: The simulation operates on the 868MHz frequency band, commonly used for LoRaWAN communications.
- (9) Spreading Factor: Set to SF7, which determines the data rate and range of communication.

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

Table 1. Simulation parameters.

Parameter	Value
Number of Gateways	3
Number of IoT Devices	100
Network Server	1
Environment Size	200m x 200m
Maximum Distance to Gateway	200m
Propagation Model	LoRa Log Normal Shadowing Model
Number of Retransmissions	5(Max)
Frequency Band	868MHz
Spreading Factor	SF7, SF8, SF9, SF10, SF11, SF12
Number of Rounds	1000

Parameter	Value
Voltage	3.3v
Bandwidth	125kHz
Payload Length	10 bytes
Timeslot Technique	CSMA10
Data Rate (Max)	250kbps
Number of Channels	5
Simulation Time	600 Seconds

The above parameters have been chosen in order to perform a realistic LoRaWAN environment and investigate the QoS metrics in IoT applications. The selected parameters aim to simulate a realistic medium-scale LoRaWAN IoT network that offers a good balance between the complexity of the network and communication reliability and computational efficiency for reinforcement learning. They rely on widely adopted real-world LoRaWAN configurations but provide the flexibility needed to effectively test a range of QoS and scheduling algorithms.

4.2. Parameters and Tuning Strategies

The selection of the algorithm's parameters generally affects a sizable portion of the outcomes of the RL-based scheduling method. The ensuing sections outline the recommended practices for modifying the primary parameters as well as how the modifications affect algorithms.

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 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, it affects 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 associated with larger or smaller sizes.

Table 2 presents the key parameters utilized in the reinforcement learning-based scheduling algorithm, which are crucial for its performance and effectiveness. The parameters include:

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, which influences the network's ca-

capacity 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 the network weights during training, impacting convergence speed and stability.
4. Discount Factor (Gamma): Set to 0.95, this factor balances the importance of immediate rewards versus future rewards, guiding the agent's long-term decision-making.
5. Exploration Rate (Epsilon): Initialized at 1.0, this rate determines the likelihood of the agent exploring new actions versus exploiting known actions, promoting exploration in the early training stages.
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, this ensures that the agent retains a small chance of exploring new actions even after extensive training.
8. Replay Buffer Size: Set to 30,000, this parameter defines the capacity of the experience replay buffer, which stores past experiences for training stability.
9. Batch Size: Fixed at 64, this parameter determines the number of experiences sampled for each training iteration, balancing convergence speed and generalization.
10. Target Network Update Frequency: Set to every 500 steps, this parameter specifies how often the target network's weights are synchronized with the main Q-network, aiding in stable learning.

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

Table 2. 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

Parameter	Value
Optimizer	Adam
Loss Function	Mean Squared Error

4.3. Performance Metrics Analysis

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

4.3.1. Network Delay

Figure 5 illustrates the relationship between network delay and the number of nodes in a LoRaWAN environment.

1. Trend Analysis: The graph typically shows that as the number of nodes increases, the delay experienced in the network also increases. This trend is indicative of the 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 versus traditional methods like LoRa+ and RT-LoRa. The RL-based algorithm is expected to demonstrate significantly lower delays, showcasing its effectiveness in optimizing resource allocation and scheduling tasks.
3. Implications for QoS: The results presented in this figure highlight the importance of efficient scheduling in maintaining low latency, especially in scenarios with a high density of nodes. This is crucial for applications requiring real-time data transmission, emphasizing the need for advanced algorithms to manage network performance effectively.

Overall, Figure 5 provides valuable insights into how network delay is affected by node density and the performance advantages of the proposed scheduling approach.

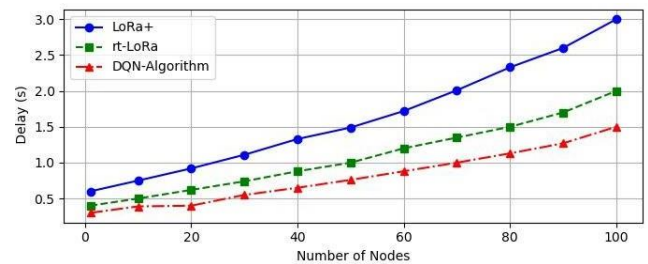


Figure 5. Delay vs number of nodes.

4.3.2. Packet Delivery Ratio (PDR)

Figure 6 depicts 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 plateaus or declines. This behavior indicates that while more nodes can enhance network coverage, increased contention and potential collisions can negatively impact the successful delivery of packets.
2. **Algorithm Comparison:** The figure highlights the performance of the proposed RL-Based Algorithm (DQN) in achieving the highest PDR compared to other algorithms like RT-LoRa and LoRa+. This superiority suggests that the RL-based approach effectively manages scheduling and resource allocation, minimizing packet losses.
3. **Significance for Network Performance:** The PDR is a critical metric for assessing the reliability of communication in IoT networks. A higher PDR indicates better performance and reliability, which is essential for applications that require consistent data transmission, reinforcing the importance of advanced scheduling techniques in optimizing network performance.

Overall, Figure 6 emphasizes the impact of node density on packet delivery success and showcases the advantages of the proposed algorithm in maintaining high delivery ratios.

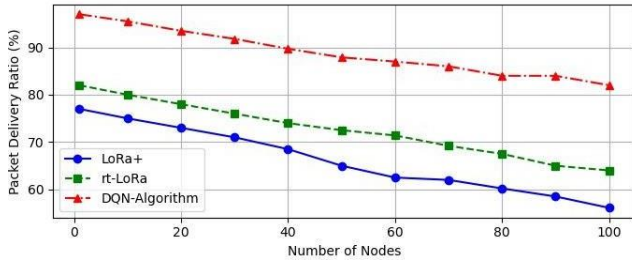


Figure 6. Packet delivery ratio (PDR) vs Number of Nodes.

4.3.3. Packet Error Rate (PER)

Figure 7 illustrates the relationship between 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 tends to rise, 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 like RT-LoRa and LoRa+. This lower PER is attributed to the dynamic optimization

of scheduling and resource allocation performed by the RL-based approach, which effectively reduces packet collisions and transmission errors.

3. **Implications for Network Reliability:** A lower PER is crucial for ensuring reliable communication in IoT applications, as it directly impacts the overall performance and efficiency of the network. The results presented in this figure underscore the importance of employing advanced scheduling algorithms to enhance network reliability and minimize transmission errors, especially in scenarios with a high number of nodes.

Overall, Figure 7 emphasizes the correlation between node density and packet error rates, showcasing the effectiveness of the proposed RL-based algorithm in maintaining low error rates in a congested network.

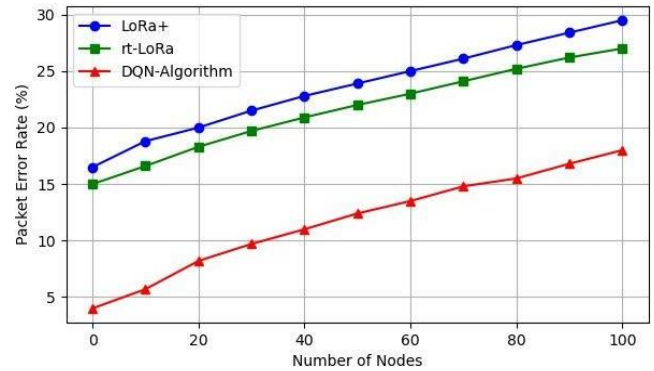


Figure 7. Packet Error Rate (PER) vs Number of Nodes.

4.3.4. Throughput

Figure 8 illustrates the relationship between throughput and the number of nodes in a LoRaWAN network. It shows that as the number of nodes increases, the throughput achieved by the RL-Based Algorithm (DQN) remains significantly higher compared to other algorithms like 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.

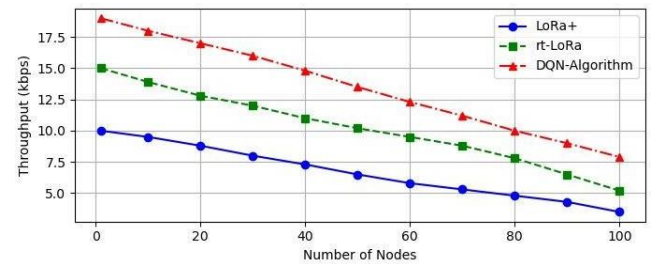


Figure 8. Throughput vs Number of Nodes.

5. Conclusion and Recommendations

5.1. Conclusion

This section concludes our investigation into creating a reinforcement learning-based task-scheduling algorithm for LoRaWAN IoT applications that is QoS-aware. Researchers investigated how current scheduling techniques fall short of satisfying the various QoS needs of contemporary IoT applications, especially in dynamic and expansive LoRaWAN networks. In order to overcome this difficulty, we developed and assessed a task scheduling algorithm based on reinforcement learning with the goal of optimizing important QoS metrics including throughput, latency, and reliability. Our study showed that reinforcement learning has a great deal of promise for improving task scheduling in LoRaWAN networks. We compared the performance of our RL-based scheduler to baseline techniques, such as RT-LoRa and LoRa+, utilizing a thorough simulation-based analysis with NS-3. Our research showed that the RL-based method has significant benefits in a number of situations. Especially in situations with dynamic traffic, the RL scheduler continuously beat baseline algorithms. Especially during periods of high network traffic, it successfully adjusted and obtained a noticeably decreased average delay. In comparison to baselines, our RL scheduler continuously showed lower packet error rates and greater packet delivery ratios, indicating increased reliability. Higher average throughput was also attained by the RL-based method, demonstrating its efficiency in managing heavy network loads and optimizing data transfer rates.

We have demonstrated that the difficulties of current LoRaWAN scheduling techniques may be successfully addressed by using reinforcement learning, especially when it comes to meeting QoS requirements like throughput, latency, and reliability. For real-world LoRaWAN deployments to optimize QoS, our RL-based scheduler must be able to learn and adjust to dynamically changing network conditions, traffic patterns, and device attributes. For the development of intelligent scheduling systems for LoRaWAN networks across multiple sectors, our study offers a solid basis. Numerous applications, such as industrial automation, smart cities, and remote healthcare monitoring, may perform better thanks to the optimized scheduling method.

5.2. Prospects for Further Research

Even though our study offers insightful information about RL's potential for LoRaWAN scheduling, there are still a number of aspects that require further study and advancement:

- 1) Further enhancements in scheduling performance may result by evaluating additional reinforcement learning algorithms, such as deep reinforcement learning architectures, policy gradient methods, or other cutting-edge techniques.

- 2) Deploying dependable and secure IoT applications requires addressing security and privacy issues in LoRaWAN scheduling. How to incorporate security measures into the RL-based algorithm can be investigated in future studies.
- 3) Extensive testing and assessment in realistic contexts are necessary for the proposed scheduler's real-world implementation in order to validate its performance in the context of network dynamics, communication latency, and device heterogeneity.

Abbreviations

AI	Artificial Intelligence
CF	Carrier Frequency
DER	Data Extraction Rate
DNN	Deep Neural Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DSR	Design Science Research
ILP	Integer Linear Programming
IoT	Internet of Things
ISM	Industrial, Scientific, and Medical
LoRa	Long Range
LoRaSim	LoRa Network Simulator
LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network
MAC	Medium Access Control
MCUs	Microcontrollers
MILP	Mixed Integer Linear Programming
NS-3	Network Simulator-3
PDR	Packet Delivery Ratio
PER	Packet Error Rate
PST	Priority Scheduling Technique
QoS	Quality of Service
ReLU	Rectified Linear Unit
RL	Reinforcement learning
RT-LoRa	Real-Time LoRa
SF	Spreading Factor
SINR	Signal-to-Interference-plus-Noise Ratio
TCP/IP	Transmission Control Protocol/Internet Protocol

Author Contributions

Ermias Melku Tadesse: Conceptualization, Investigation, Methodology, Project administration, Writing – original draft

Haimanot Edmealem: Funding acquisition, Resources, Writing – review & editing

Tesfaye Belay: Data curation, Validation, Visualization

Abubeker Girma: Formal Analysis, Software, Supervision

Conflicts of Interest

The authors declare no conflicts of interest.

References

- [1] N. H. Mahmood, N. Marchenko, M. Gidlund, and P. Popovski, *Wireless Networks and Industrial IoT: Applications, Challenges and Enablers*. 2020.
<https://doi.org/10.1007/978-3-030-51473-0>
- [2] L. R. de Oliveira, P. de Moraes, L. P. S. Neto, and A. F. da Conceição, "Review of LoRaWAN Applications," 2020, [Online]. Available: <http://arxiv.org/abs/2004.05871>
- [3] J. M. Marais, R. Malekian, and A. M. Abu-Mahfouz, "LoRa and LoRaWAN testbeds: A review," *2017 IEEE AFRICON Sci. Technol. Innov. Africa, AFRICON 2017*, pp. 1496–1501, 2017, <https://doi.org/10.1109/AFRCON.2017.8095703>
- [4] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT," *2018 IEEE Int. Conf. Pervasive Comput. Commun. Work. PerCom Work. 2018*, pp. 197–202, 2018, <https://doi.org/10.1109/PERCOMW.2018.8480255>
- [5] T. Bouguera, J. F. Diouris, J. J. Chaillout, R. Jaouadi, and G. Andrieux, "Energy consumption model for sensor nodes based on LoRa and LoRaWAN," *Sensors (Switzerland)*, vol. 18, no. 7, pp. 1–23, 2018, <https://doi.org/10.3390/s18072104>
- [6] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A study of Lora: Long range & low power networks for the internet of things," *Sensors (Switzerland)*, vol. 16, no. 9, pp. 1–18, 2016, <https://doi.org/10.3390/s16091466>
- [7] M. Ragnoli, G. Barile, A. Leoni, G. Ferri, and V. Stornelli, "An autonomous low-power lora-based flood-monitoring system," *J. Low Power Electron. Appl.*, vol. 10, no. 2, 2020, <https://doi.org/10.3390/jlpea10020015>
- [8] J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Low overhead scheduling of LoRa transmissions for improved scalability," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3097–3109, 2019, <https://doi.org/10.1109/JIOT.2018.2878942>
- [9] R. S. Sutton and A. G. Barto, "Reinforcement learning_Peter," *Learning*, vol. 3, no. 9, 2012, [Online]. Available: <http://incompleteideas.net/sutton/book/the-book.html%5Cn>
- [10] J. Petäjäjärvi, K. Mikhaylov, M. Pettissalo, J. Janhunen, and J. Linatti, "Performance of a low-power wide-area network based on lora technology: Doppler robustness, scalability, and coverage," *Int. J. Distrib. Sens. Networks*, vol. 13, no. 3, 2017, <https://doi.org/10.1177/1550147717699412>
- [11] T. Polonelli, D. Brunelli, A. Marzocchi, and L. Benini, "Slotted ALOHA on LoRaWAN-design, analysis, and deployment," *Sensors (Switzerland)*, vol. 19, no. 4, 2019, <https://doi.org/10.3390/s19040838>
- [12] M. Alenezi, K. K. Chai, A. S. Alam, Y. Chen, and S. Jimaa, "Unsupervised learning clustering and dynamic transmission scheduling for efficient dense LoRaWAN networks," *IEEE Access*, vol. 8, pp. 191495–191509, 2020, <https://doi.org/10.1109/ACCESS.2020.3031974>
- [13] L. Leonardi, F. Battaglia, and L. Lo Bello, "RT-LoRa: A Medium Access Strategy to Support Real-Time Flows Over LoRa-Based Networks for Industrial IoT Applications," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10812–10823, 2019, <https://doi.org/10.1109/JIOT.2019.2942776>
- [14] E. Sallum, N. Pereira, M. Alves, and M. Santos, "Improving quality-of-service in Lora low-power wide-area networks through optimized radio resource management," *J. Sens. Actuator Networks*, vol. 9, no. 1, pp. 1–26, 2020, <https://doi.org/10.3390/jsan9010010>
- [15] M. Micheletto, P. Zabala, S. F. Ochoa, R. Meseguer, and R. Santos, "Determining Real-Time Communication Feasibility in IoT Systems Supported by LoRaWAN," *Sensors*, vol. 23, no. 9, pp. 1–27, 2023, <https://doi.org/10.3390/s23094281>
- [16] C. Garrido-Hidalgo *et al.*, "LoRaWAN Scheduling: From Concept to Implementation," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12919–12933, 2021, <https://doi.org/10.1109/JIOT.2021.3064430>
- [17] U. F. Siddiqi, S. M. Sait, and M. Uysal, "Deep Reinforcement Based Power Allocation for the Max-Min Optimization in Non-Orthogonal Multiple Access," *IEEE Access*, vol. 8, pp. 211235–211247, 2020, <https://doi.org/10.1109/ACCESS.2020.3038923>