Received 15 March 2023, accepted 5 April 2023, date of publication 10 April 2023, date of current version 20 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3265717

RESEARCH ARTICLE

Adaptive Trickle Timer for Efficient 6TiSCH Network Formation Using Q-Learning

DZAKY ZAKIYAL FAWWAZ, (Student Member, IEEE), AND SANG-HWA CHUNG, (Member, IEEE)
Department of Information Convergence Engineering, Pusan National University, Busan 46241, South Korea

Corresponding author: Sang-Hwa Chung (shchung@pusan.ac.kr)

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the Grand Information Technology Research Center support program (IITP-2023-2016-0-00318) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation). This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2020-0-01450, Artificial Intelligence Convergence Research Center [Pusan National University]).

ABSTRACT The 6TiSCH (IPv6 over IEEE802.15.4e time-slotted channel hopping mode) wireless sensor network architecture utilizes control packets to construct network formation. These control packets are essential for establishing communication links between nodes and configuring network settings. The trickle timer algorithm is utilized to broadcast the DIO control packet. DIO carries information about the available parent nodes, which is then used to form the routing tree. Sensors transmit control packets in one cell on each TSCH slotframe, called the minimal cell. This leads to the problem that RPL trickle timer algorithm encounters congestion in DIO control packet transmission with other control messages, particularly in dense networks. Moreover, high traffic transmission also leads to high queue usage, which then drops the DIO control packet. Failed DIO transmission can increase network formation time and energy consumption. To address this issue, we propose Q-Trickle, an adaptive trickle timer algorithm based on Q-learning that determines the optimal policy for transmitting or suppressing DIO based on minimal cell and transmission queue conditions. Q-Trickle adaptively selects a redundancy constant value and transmission interval that promotes fair transmission distribution and considers network condition. Additionally, a control scheme over minimal cell transmission is formulated to lower transmission congestion and faster synchronization. The proposed methods were assessed using simulation and actual testing on the FIT IoT-LAB testbed. The results indicated that Q-Trickle performed better than the benchmark methods. Q-Trickle decreases joining time, energy consumption, and number of failed DIO compared with the original algorithm by -13%, -11%, and -43%, respectively.

INDEX TERMS 6TiSCH, MSF, RPL, trickle timer, wireless sensor network, Q-learning.

I. INTRODUCTION

Many latency-sensitive applications have emerged because of the rapid growth of wireless sensor networks (WSNs). A WSN is a network consisting of dedicated sensors that are distributed across a specific area to monitor and gather data on the physical conditions of the environment. The collected data is then transmitted to a central location for further processing and analysis [1]. 6TiSCH is a collection of protocols and an architecture designed to incorporate the IEEE 802.15.4 time-slotted channel hopping (TSCH) protocol into the Internet of Things (IoT). This includes adaptations to various layers of the architecture, such as IPv6 over low-power wireless personal area networks (6LoWPAN), the Routing Protocol for Low-power and Lossy Networks (RPL), and the 6top Minimal Scheduling Function (MSF) scheduling.

In the 6TiSCH network, the sink node serves as the join coordinator (JRC) and initiates network formation by broadcasting network information through an enhanced beacon (EB). To join the network, a new node scans random channels for the EB and sends a join request (JRQ) packet to either the JRC or a preferred parent. The node then waits for a

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamad Afendee Mohamed.

join response (JRS) packet. Then, the node waits for the destination-oriented directed acyclic graph (DODAG) information object (DIO) packet provided by RPL for routing layer control. Upon receiving the DIO, the node calculates its rank and joins RPL. Also, the node can request DIO by sending a DODAG information solicitation (DIS) packet. Finally, the node can send its own EB frame to let other nodes join the network. 6TiSCH network formation by exchanging control packets is depicted in Figure 1. 6TiSCH minimal configuration provides a single shared cell, called a minimal cell, to transmit control packets like EB, DIO, and DIS. This condition makes those control packets collide over reception since multiple transmitters utilize it together. When the nodes are not optimally set control packets transmission rate will also suffer from congestion over the minimal cell.

To establish an upward and downward route and maintain a stable routing topology, RPL utilizes trickle timer algorithm to periodically broadcast DIO packet. It is used to control transmissions that maintain energy efficiency. However, the standard trickle timer carries some challenges. A high DIO transmission frequency setting may collide with other DIO and control packets while consuming more energy. Otherwise, a low transmission frequency may result in a longer network formation and inhibit network performance improvement like finding the optimal route. A high-traffic transmission network also leads to high queue usage that may drop the DIO control packet. Since the trickle timer parameters were static and defined before the network deployed, this condition limited the nodes to adapt to dynamic network conditions and some nodes may transmit more than others.

Reinforcement learning (RL) algorithms are being increasingly employed in various IoT networking applications, including MAC scheduling, congestion control, and routing protocol [2], [3], [4]. Reinforcement learning is a learning strategy in which an agent interacts with the environment to learn what actions to take and how to map situations, as well as give feedback to maximize a long-term objective.

Therefore, this study proposes Q-Trickle, an adaptive tickle-timer algorithm based on RL algorithm called Q-learning (QL), which considers network conditions, including shared-cell congestion, network traffic, density, and stability. Shared-cell congestion occurs when the cell requires more data than it can handle; in this case, more than one control packet. Network traffic and density are depicted by the number of queue utilization and neighbors of a node. Network stability represents how often a node had trickle timer's reset events, such as inconsistent DIO and DODAG loop, that make a trickle state and parameter returns to the initial value. Q-Trickle offers intelligent decision-making for DIO transmission or suppression through TSCH minimal cell and transmission queue observation. Moreover, the transmission frequency is adjusted via dynamic redundancy constant and transmission interval parameter rather than a fixed parameter, leading to a network condition-aware transmission distribution. A control scheme over minimal cell transmission was also formulated to lower transmission congestion and accelerate synchronization. The key findings of this research are outlined below.

- An adaptive Q-learning-based RPL trickle is proposed to improve DIO success transmission based on minimal cell and queue conditions.
- Dynamic redundancy constant and transmission interval are employed considering network conditions to maintain balanced allocation of DIO transmission.
- A transmission control scheme over minimal cell is formulated to reduce congestion of DIO transmission and faster node synchronization on early formation phase.
- The proposed schemes are evaluated over simulation and real testbed on FIT IoT-LAB using OpenWSN OS.

The subsequent sections of this paper are organized as follows: Section II provides an overview of the 6TiSCH architecture, RPL trickle timer, reinforcement learning, and related work. The proposed methods and algorithms are presented in Section III, while Section IV reports on the results of extensive experiments used to evaluate the effectiveness of the proposed approach. Finally, the paper concludes by discussing the main findings and suggestions for future research.

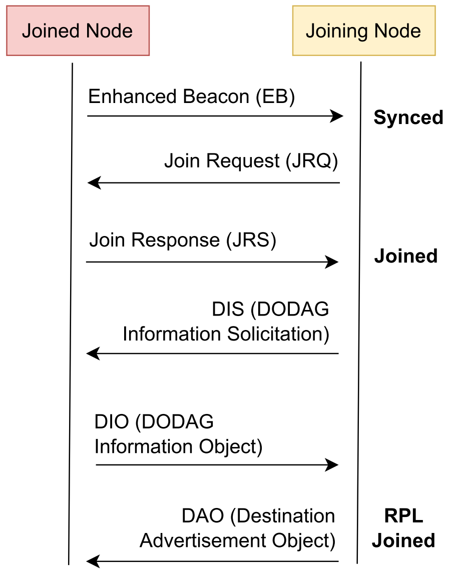

Joined Node
Joining Node
Enhanced Beacon (EB)
Synced
Join Request (JRQ)
Join Response (JRS)
Joined
DIS (DODAG Information Solicitation)
DIO (DODAG Information Object)
DAO (Destination Advertisement Object)
RPL Joined

FIGURE 1. Control packet exchanges on 6TiSCH network formation.

II. BACKGROUND AND RELATED WORK

A. 6TiSCH ARCHITECTURE

The 6TiSCH working group standardizes protocols that enable IPv6 to be used in low-power industrial networks. 6TiSCH utilizes the TSCH mode of the IEEE802.15.4-2015 standard [5]. TSCH MAC layer mode uses two fundamental aspects, namely time division multiple access (TDMA) based medium access layer and channel hopping. The TDMA mechanism enables congestion-free access to the shared medium, ensuring efficient channel access. Time is divided into equal-length time slots, and a slotframe comprises a group of time slots that are repeated over time, with each time slot long enough for a pair of nodes to exchange a packet and optional acknowledgment. 6TiSCH can be used in low-power IoT applications, such as smart industries, buildings, infrastructure, and home applications. 6TiSCH adapts 6LoWPAN [6] to bridge IPv6 over the network by using header compression, packet fragmentation, and reassembly processes to convert IPv6 packets into IEEE 802.15.4 frames. 6TiSCH uses RPL as a network-layer routing protocol, which is discussed in detail in Section II-B. In 6TiSCH, dynamic scheduling is enabled by adding or removing link-layer resources (TSCH schedule cells) based on the communication needs of the applications. This is accomplished through a minimal scheduling function (MSF) that triggers 6top protocol (6P) negotiations to add or delete cells. 6P enables neighboring nodes to negotiate which cells to add or remove from their schedule.

MSF controls a node's actions when joining the network and manages communication schedules in a distributed manner [7]. It extend the minimum scheduling setup and adds child-parent links based on traffic load [8]. MSF uses a single shared cell, referred to as the minimal cell, to send control packets, typically at slot offset 0 and channel offset 0 [9]. An example of a slotframe schedule over a simple 6TiSCH network shows in Figure 2, which illustrates the minimal shared cell with unicast Tx and Rx schedule. Dedicated cells are the remaining cells in a slotframe starting with time slot one. They are used for data transmission using a communication schedule set by an SF. Dedicated cells consist of both autonomous and negotiated cells. Autonomous cells offer connections to neighbors without requiring control of signalling. Then, 6P protocol handled the negotiated cells, which adds or removes them from the schedule depending on a traffic-based reactive policy.

As all linked nodes compete for the same physical channel to broadcast their control packets on minimal cells, congestion between the control packets transmitted by nodes is inevitable [10]. Hence, a higher network density and control packet transmission frequency can increase congestion in the minimal cell. The frequent transmission of control packets can trigger congestion in shared cells. In contrast, infrequent transmission can result in longer network formation, inhibit network QoS improvement (e.g., finding optimal routing), and other network problem resolutions. Thus, it is necessary to control a node's control packet transmission to avoid network congestion.

The challenges of transmitting DIO control packets are illustrated in Figure 3. Since EB and DIO must be sent during the shared cell period when all nodes are active, broadcast packets are continuously received from neighbors. Because broadcast packets are not acknowledged, transmission backoff is not used, making it difficult to detect control packet collisions. Furthermore, since EB has priority over other packets such as DIO, a node may transmit its new EB frame even when other control packets are still in its transmission queue. Therefore, the control packets may undergo reception collision and transmission congestion. DIO control packet collision will result in wasted resources and energy. It also inhibits performance improvement, like faster network formation and finding the optimal route. DIO transmission congestion with other control packets also decreases the node performance. It will fill the queue and wait for the subsequent shared cell transmission, which holds the following packet transmission and may drop upcoming packets to enter the queue. Adjoining DIO A and B and DIO and EB demonstrate those points in the figure. Transmission queue utilization also has a vital role in the success of DIO transmission. The figure illustrates that the 6P request command already fulfills the queue, so the scheduled DIO dropped.

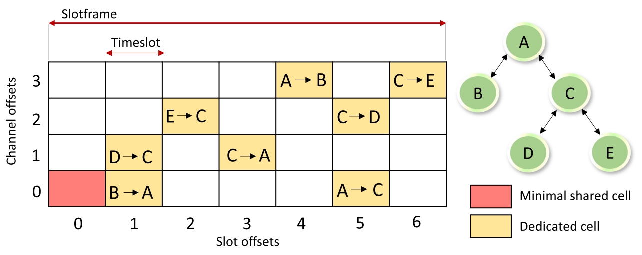


FIGURE 2. Illustration of a slotframe schedule in 6TiSCH network.

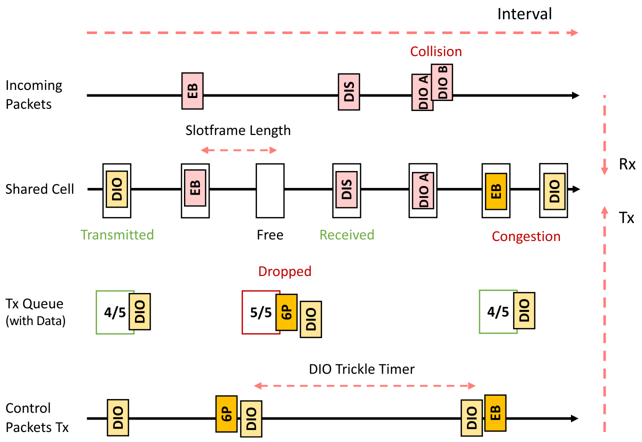


FIGURE 3. DIO control packet transmission challenges.

B. RPL TRICKLE TIMER

RPL is a routing protocol designed for low-power and lossy networks (LLNs) and utilizes an objective function (OF) to construct a route toward a root node or border router [11]. The protocol generates a DODAG to determine the optimal path between nodes. The downward route is constructed using DIO, while the upward route is built using DAO. To join the network, a new node sends DIS, and the rank in the DODAG represents a node's level calculated using the OF.

To maintain a stable routing topology, the nodes broadcast DIO messages regularly. The transmission of DIO is controlled by a trickle timer to efficiently preserve the energy [12]. It has numerous predefined variables such as redundancy constant k, minimum interval size I_{min},

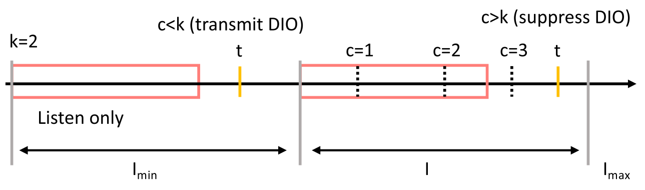


FIGURE 4. RPL trickle timer.

maximum interval size I_{max} determined using Equation 1, current interval I, consistency counter c, and transmission time within interval t. The trickle timer scheme is illustrated in Figure 4. The steps of the trickle timer are as follows:

1) Initially, select I equal to I_{min}.
2) Beginning at each interval, $c = 0$ is initialized, and t is randomly selected based on Equation 2.
3) If a node receives a consistent transmission, it increments counter c.
4) If a node receives an inconsistent transmission or any other reset events, it resets the trickle timer to step 1 to reset trickle state.
5) At time t, if $c < k$ allows its own DIO transmission. Otherwise, the DIO is suppressed.
6) At the end of the interval, I double. If new I is greater than I_{max}, set new I to I_{max} and execute step 2 to do next trickle iteration.

A trickle timer has limitations [13], such as a fixed parameter that cannot adapt to network conditions. Then, the distribution is unfair because some nodes may transfer more than others and increase shared-cell congestion over a dense network. A busy data transmission also increases the chance that DIO control packets to be dropped in the packet queue. The trickle interval can affect network convergence. If it is too long, it can result in hardening network convergence. If it is too small, it may not collect sufficient consistent transmissions and always decide to send DIO more frequently, which is unnecessary. The trickle timer algorithm defines the frequency of DIO transmission. When frequent transmission occurs, DIO packets congest the shared cells, resulting in a higher node joining time. However, infrequent transmissions can release more freely shared cells. This results in non-optimal usage and a higher joining time of the nodes, as the optimal rate control packet helps network convergence. This indicates that an optimal frequency is required to maintain the network performance. This may be addressed through a dynamic trickle timer setting to control the transmission frequency and reduce the control packet congestion in the minimal cell based on the network condition.

$$I_m = 2^m \times I_{\min}$$
$$I_{max} = 2^{m_{max}} \times I_{\min} \tag{1}$$

$$t = [t_{min}, t_{max}] = \left[\frac{I_m}{2}, I_m\right] \tag{2}$$

C. REINFORCEMENT LEARNING AND Q-LEARNING

RL [14] is a technique that aims to learn how to optimize the interaction between actions and states to achieve maximum rewards. It automates decision-making and learning based on goals. This approach involves interactions between agents and their environments, where an agent is responsible for decision-making and learning. The four essential components of RL are the policy, reward, value function, and environment model. The policy is a set of state-action pairs that determine the agent's behavior. It can be a simple function or a lookup table for certain situations. A reward reflects a learner's positive and negative experiences. This is the most important reason for any state to change its policy. The value function of a particular state is the sum of the rewards that a learner can eventually accumulate. We are not looking for rewards, but rather for the value function. The environmental model is optional for RL. It mimics the behavior of the environment or predicts how the environment will operate, which generally enables suggestions regarding how the environment will behave.

The QL algorithm [15] is a popular RL method, where an agent has a set of actions $a \in A$ and states $s \in S$. The agent receives a reward $r(s, a)$ when it performs an action in a state. The agent then transitions to the next state based on the obtained reward and saves the reward as a Q-value for a specific state-action pair $Q(s, a)$. The agent uses this information to select future actions in a given state. QL is an off-policy algorithm where the agent learns from experience that it doesn't necessarily choose the actions that it learns from. Instead, it can learn from any actions taken in the environment, even if they were chosen by a different policy. The action selection are based on the exploration probability ε that set the ratio of exploration and exploitation. The greedy optimal action and ε-greedy policy controls are presented in Equations 3 and 4, respectively.

QL employs a model-free RL approach called the temporal difference (TD). This allowed the agent to learn from each action that it has performed. Instead of updating the agent's knowledge after each episode, the TD updates the agent's experience after each step (action), reaching the goal or the end state. The Q-value update is expressed in Equation 6, which is calculated from the old Q-value and learned Q-value with a learning rate α between 0 and 1. The learning rate α determines how quickly a new value overrides the previous value. When $\alpha = 0$, the agent ignores existing information and does not learn new values. However, when $\alpha = 1$, the agent is required to analyze only the most recent input and ignore prior knowledge. Equation 5 shows the learning estimate or the TD error. It is calculated from the new Q-value given the new states, their possible actions (optimal target value), and the current Q-value (predicted value), with a certain discounted reward factor of β between 0 and 1. The value of β influences the importance of future rewards to an agent compared to current rewards. The agent evaluated the present reward more vigorously when $\beta = 0$. The agent is then prompted to prioritize long-term benefits over short-term rewards when $\beta = 1$.

$$a_{\text{opt}} = \text{argmax}_a\, Q(s, a) \tag{3}$$

$$a_t = \begin{cases} a_{\text{opt}} & 1 - \varepsilon \text{ (exploitation)} \\ a & \varepsilon \text{ (exploration)} \end{cases} \quad (4)$$

$$\Delta Q(s, a) = r(s, a) + \beta \times \max_a Q(s_{next}, a) \quad (5)$$

$$Q(s, a)' = (1 - \alpha) \times Q(s, a) + \alpha \times \Delta Q(s, a) \quad (6)$$

D. RELATED WORK

Kalita et al. [26] adopted a strategy to fully utilize the available channels at time slot zero for each slotframe by assigning multiple shared cells instead of a single shared cell. Additionally, in a subsequent study [27], they proposed a method that dynamically adjusted the priority of control packets to ensure sufficient routing information was transmitted throughout the network. This approach also enabled nodes with urgent packets to transmit immediately. Meanwhile, Vallati et al. [28] increased the number of shared cells in each slotframe based on the number of control packets generated in the network. However, the introduction of additional shared cells in separate time slots of a slotframe affected the default data transmission schedule. In [29] and [30], MSF was modified to suit a particular case problem, such as heavy traffic loads on industry and efficient routing for mobile nodes. Autonomous scheduling such as Alice [31] and Orchestra, [32], uses a MAC address-based hash function. Separated slotframes are used for EB in TSCH, control packets in RPL, and independent data frames. Consequently, the shared cells were eliminated. Their issue is that when traffic rates are high or the network size increases, performance declines drastically because of static allocation. However, MSF does not experience significant changes, owing to its flexibility in adapting to traffic situations. The slotframe length has a significant impact on autonomous performance. With autonomous scheduling, the number of packets buffered in the local queue is typically close to the maximum value. It exhibits a large end-to-end delay compared to MSF, where the queue is always under control [33].

Several techniques have been proposed for improving the efficiency of trickle timers. By enhancing the suppression mechanism, I-trickle [16] eliminated the load-balancing issue. At the end of the current interval, the redundancy was reset to zero, which reduced the amount of energy used. E-trickle [17] solves the load-balancing problem by establishing listen-only intervals based on the number of neighbors. It uses less energy and has a faster convergence time. It has a significant traffic control overhead track the number of neighbors. FL-trickle [18] offers a modified trickle timer technique in which the minimum interval I_{min} is set to a greater value to reduce the overhead. To reduce the transmission latency of the DIO, it fixes the transmission time t at $I/2$, rather than choosing it at random. A larger double interval reduces traffic control packets and saves energy. EAAT [19] presents an adaptive trickle timer technique that conserves energy by regulating DIO transmission based on residual and future energy. It modifies the redundancy constant k according to the energy status of a node. Trickle-D [20] increases global fairness by modifying the redundancy constant based on Jain's fairness index, ensuring that all nodes have the same number of transmissions. FI-Trickle [21] improves trickle timer by eliminating the listen-only period and selecting DIO transmission time t based on the suppression history. LA-Trickle [22] uses a learning automaton to determine the number of times the trickle timer is repeated with the minimum interval to resolve inconsistency. Drizzle [23] ensures fairness in DIO transmission by assigning nodes to different transmission probabilities based on their transmission history. RIATA [24] employs QL to determine DIO transmission, with inconsistent reception of DIO packets as a reward. It assigns nodes that have received an inconsistent control packet in the past with a higher probability of transmitting control packets at intervals, and it selects an adaptive redundancy constant value to prevent unnecessary control packet transmissions. Still, RIATA algorithm did not consider congestion of control packets in minimal cell and queue usage. RIATA's QL also does not adaptive to early network formation since it has fixed configuration such as fixed epsilon. ACPB [25] modifies the trickle timer and proposes a slotframe window (SW)-based adaptive scheme that restricts nodes from transmitting their control packets frequently and further reduces congestion. When it receives a DIS, it resets and continues to the last trickle state. However, ACPB does not explore RL to optimize the network formation, and the proposed control packet scheme cannot adapt in early network formation since it has a harsh limitation of control packet transmission in every interval, which is a maximum of 2. ACPB did not consider queue usage and has not tested in the network with data traffic filling the transmission queue, which may congest DIO transmission. None of these works, except ACPB, study the congestion in a shared cell correlated with DIO transmission.

With the open issues in optimizing the trickle timer and minimizing cell congestion, Q-Trickle is proposed by employing QL to support the decision-making of DIO transmission in a trickle timer considering minimal cell congestion and queue utilization. Q-Trickle also provides an adaptive minimal cell transmission control that reinforce efficient network formation and minimize transmission congestion. Finally, a comparison between previous studies and our proposed method is presented in Table 1.

III. PROPOSED Q-TRICKLE ALGORITHM

The proposed Q-Trickle algorithm, which is an adaptive RPL trickle timer using QL. The method has several improved features compared with the baseline trickle timer algorithm. First, intelligent DIO transmission and suppression decisions using QL that consider minimal cell and queue conditions to promote DIO success transmission. Second, dynamic redundancy constant and transmission interval that considers the local network density and stability to reinforce a balanced distribution and maintain transmission priority among nodes. Third, a transmission control scheme over minimal cell to lower the chance of congestion on DIO transmission interval

TABLE 1. Comparison with related works.

Algorithm	Features	Adaptive redundancy constant	Dynamic interval	Intelligent DIO transmission	Control packet transmission scheme	Real testbed evaluation
Original [12]	Original Trickle Timer	✗	✗	✗	✗	✗
I-trickle [16]	Reset the counter c at random time t	✗	✗	✗	✗	✗
E-trickle [17]	Setting listen-only interval based on the number of neighbors	✗	✓	✗	✗	✗
FL-trickle [18]	Fixed time t and set high I_{min}	✗	✗	✗	✗	✗
EAAT [19]	Set k based on the node energy condition	✓	✗	✗	✗	✗
Trickle-D [20]	Set k based on Jain fairness index	✓	✗	✗	✗	✗
FI-Trickle [21]	Eliminate listen-only period and set t based on suppression history	✗	✓	✗	✗	✗
LA-Trickle [22]	Use learning automaton to resolve the inconsistency	✗	✓	✗	✗	✗
Drizzle [23]	Adaptive t and k based on transmitted DIO	✓	✓	✗	✗	✗
RIATA [24]	Adaptive trickle timer parameter using QL based on inconsistency	✓	✓	–	✗	✗
ACPB [25]	Dynamic trickle timer setting for faster network bootstrapping	✓	✓	✗	–	✓
Q-Trickle	Intelligent DIO transmission based on minimal cell & queue conditions using QL	✓	✓	✓	✓	✓

TABLE 2. List of notations.

Notation	Description
I_{min}	Minimum trickle timer interval
I_{max}	Maximum trickle timer interval
MT_I	Total Tx and Rx on minimal cell from start until interval I
c	Counter of consistent DIO
k	Redundancy constant
m	Trickle timer state
N_{mc}	Number of available minimal cells within an interval
N_{nbr}	Number of neighbors
N_{rs}	Number of resets
N_{it}	Total iterations that has been passed
SL	Slotframe length in timeslots
SD	Slot duration in milliseconds
$s \in S$	States in QL
$a \in A$	Action in QL
$r(s, a)$	Reward of state s and action a
$Q(s, a)$	Q-values of state s and action a
t	Interval of DIO transmission
DIO_{fail}	Number of failed DIO
DIO_{tr}	Number of transmitted DIO

and to promote faster synchronization on early formation phase. The related notations are listed in Table 2 and the proposed schemes outlined in Algorithm 1.

A. INTELLIGENT DIO TRANSMISSION USING Q-LEARNING

The proposed algorithm employs QL to improve the performance of the trickle timer algorithm. Each node acts as an agent. The state s comprises minimal cell and queue usage levels presented in Equation 13. The usage ratio was converted into three categorized levels through Equation 12, namely low, medium, and high. Minimal cell usage or busy ratio is calculated with Equation 9. The queue utilization ratio is calculated with Equation 10. The actions a are between DIO suppression or transmission formulated in Equation 14. The reward is determined using Equation 15. The reward would be positive when the DIO failed transmission (Equation 11) was not increased and negative otherwise. We set a higher positive reward value when the node selects DIO transmission action than DIO suppression, preventing QL from always selecting suppression. Then, QL uses a random number between 0 and 1 to determine the exploration rate. During the exploration phase, if the number of received consistent DIO packet c is less than redundancy constant k, the DIO is broadcast and suppressed otherwise. In the exploitation phase, the Q-learning algorithm determines the best action for a given state-action pair by choosing the one with the highest Q-value, which represents the accumulated reward over past iterations.

$$N_{mc} = I/\mathrm{SL} \times \mathrm{SD} \tag{7}$$

$$\mathrm{MT}_I = \Sigma_0^I \mathrm{Tx} + \mathrm{Rx} + \mathrm{Tx}_{Ack} \tag{8}$$

$$p_{busy} = (\mathrm{MT}_{I_{end}} - \mathrm{MT}_{I_{start}})/N_{mc} \tag{9}$$

$$p_{qu} = \text{Total Packets/Queue Size} \tag{10}$$

$$\mathrm{DIO}_{fail} = \Sigma \mathrm{DIO}_{drop} + \mathrm{DIO}_{congest} \tag{11}$$

$$\text{Level}(p) = \begin{cases} \text{Low}(0) & p = [0, \frac{1}{3}] \\ \text{Medium}(1) & p = (\frac{1}{3}, \frac{2}{3}) \\ \text{High}(2) & p = [\frac{2}{3}, 1] \end{cases} \tag{12}$$

$$S = \{\text{Level}(p_{busy}^{prev}), \text{Level}(p_{qu}^{prev})\}$$

$$S_{next} = \{\text{Level}(p_{busy}), \text{Level}(p_{qu})\} \tag{13}$$

$$A = \begin{cases} 0 & \text{Suppress DIO} \\ 1 & \text{Transmit DIO} \end{cases} \tag{14}$$

$$r(s,a) = \begin{cases} 2 & a = 1 \ \& \ \text{DIO}_{fail} = \text{DIO}_{fail}^{prev} \\ 1 & a = 0 \ \& \ \text{DIO}_{fail} = \text{DIO}_{fail}^{prev} \\ -1 & \text{DIO}_{fail} > \text{DIO}_{fail}^{prev} \end{cases} \tag{15}$$

B. DYNAMIC REDUNDANCY CONSTANT AND TRANSMISSION INTERVAL

The performance of a trickle timer also relies on the appropriate selection of the transmission interval t and redundancy constant k. The improper selection of these values may result in an imbalanced transmission load that can drain the energy of nodes in a low-power network. The proposed Q-Trickle algorithm adjusts its redundancy constant k and transmission interval t dynamically, which are set differently at every beginning of the trickle iteration. Q-Trickle set redundancy constant k by considering network density which depicts the number of neighbors, and network stability is related to node stability from receiving any trickle timer's reset event.

Reset events create a trickle state m and the parameter returns to its initial value. This allows the frequent transmission to handle unstable or inconsistent conditions. Several reset events were established. First, at preferred parent changes when a node finds a better parent and constructs a new rank. Second, global and local repairs require a node to rejoin DODAG. Third, a DIS message is received from a node that wants to join DODAG. Fourth, inconsistent DIO, such as infinite rank or different DODAG versions, were obtained. Last, a looping packet is detected when a node receives an upward packet from its parent node. Thus, reset events are counted and formulated into network reset and stable probabilities, which are expressed in Equations 16. The redundancy constant is expressed using Equation 19. The minimum k_{min} is set to 1, and the maximum for k_{max} is 10 as specified by RPL standard. The k value will increase when p_{reset} tends to 1, which represents an unstable network and otherwise. It allows for the quick resolution of unstable or inconsistent conditions around the node. The k value will also increase when having a denser environment.

The transmission interval t is expressed using Equation 18. The proposed algorithm assigns higher DIO transmission probabilities to nodes with less DIO transmission and network stability as determined by the frequency of the trickle reset. These were calculated using Equations 17 and 16. The trickle timer selects a random time within the transmission interval lower bound t_{min} and upper bound t_{max} to execute the DIO transmission or suppression. We modified it to vary the transmission probability based on the network conditions. t_{min} will be set sooner when the node has less DIO transmission ratio and longer when the node already transmits more DIO. t_{max} will be set sooner when the node has an unstable condition and longer when the node is in a stable condition.

$$p_{reset} = N_{rs}/N_{it}$$

$$p_{stable} = 1 - p_{reset} \tag{16}$$

$$p_{transmit} = \text{DIO}_{tr}/N_{it} \tag{17}$$

$$t = \left[\frac{I_m}{2} \times p_{transmit}, \frac{I_m}{2} \times (1 + p_{stable})\right] \tag{18}$$

$$k = 1 + \lceil \min(N_{nbr}, k_{max} - 1) \times p_{reset} \rceil \tag{19}$$

C. TRANSMISSION CONTROL SCHEME

The transmission control scheme consists of two methods. First, we formulate an adaptive EB rate p_{eb} based on the number of neighbors following Equation 20. p_{eb} will double at the beginning of the formation phase to allow not-synced neighbor nodes to sync faster. Later, p_{eb} will drop to its original value when passing the early phase and having more joined neighbors. Further, p_{eb} will decrease gradually when the node reaches a certain number of neighbors, called pivot neighbors N_{nbr}^{pivot}. Since p_{eb} is decreasing, it will lower the congestion to DIO transmission. Second, we limit the transmission of other control packets on the minimal cell based on p_{busy} when passing through the DIO transmission interval. The transmission limit is demonstrated in Lines 40-45 of Algorithm 1. This function is called on every packet transmission over the minimal cell. It will lower the chance of DIO encountering congestion with other control packets and set priority to transmit DIO, giving queue space and reinforcing faster network formation.

$$p'_{eb} = \begin{cases} p_{eb} + \dfrac{1 - p_{eb}}{2^{N_{nbr}}} & N_{nbr} < N_{nbr}^{pivot} \\ \dfrac{p_{eb}}{1 + N_{nbr} - N_{nbr}^{pivot}} & N_{nbr}^{pivot} \leq N_{nbr} \end{cases} \tag{20}$$

IV. PERFORMANCE EVALUATION

We conducted experiments using 6TiSCH simulator [34] developed by the 6TiSCH working group on top of Python language. 6TiSCH simulator uses a piston hack loss model to construct the node link quality [35]. We deployed the proposed Q-Trickle on the trickle timer module. Then, minimal cell and queue observation and transmission scheme on the TSCH module. The parameters are listed in Table 3. Q-Trickle was compared to related benchmark algorithms, namely RIATA [24] and ACPB [25]. RIATA was selected because of a similar approach in proposing RL for the trickle timer, whereas ACPB considers 6TiSCH minimal cell condition to improve the trickle timer.

Several evaluation metrics were used in this study. First, the average joining time of nodes to RPL DODAG which expressed in minutes. Second, energy consumption is based

Algorithm 1 Q-Trickle Algorithm

```
    Input $I_{min}$, $m_{max}$, $k_{max}$, $\alpha$, $\beta$, $\varepsilon$
 1: function StartInterval($I$)
 2:     $N_{it}++$, $c = 0$
 3:     $DIO^{prev}_{fail} = DIO_{fail}$, $p^{prev}_{busy} = p_{busy}$, $p^{prev}_{qu} = p_{qu}$
 4:     Calculate $p_{reset}$ and $p_{stable}$ using Eq. 16
 5:     Calculate $t$ and $k$ using Eq. (18, 19)
 6:     for $i < I$ do                          ▷ Looping over times
 7:         Receiving()
 8:         if $t == i$ then ReachTimer()
 9:         if $I == i$ then EndInterval()
10:         $i++$
11:
12: function Reset
13:     $N_{rs}++$
14:     StartInterval($I_{min}$)
15:
16: function Receiving
17:     if Consistent DIO then $c++$
18:     else if Inconsistent DIO or Reset events then Reset()
19:     else if DIS then ReceivingDIS()
20:
21: function ReachTimer
22:     if $rand < \varepsilon$ then            ▷ Exploration
23:         if $c < k$ then $a = 1$
24:         else $a = 0$
25:     else $a = a_{\text{opt}}$               ▷ Exploitation
26:     if $a == 1$ then Transmit DIO, $DIO_{tr}++$
27:     else Suppress DIO
28:     Calculate $p_{transmit}$ using Eq. 17
29:
30: function EndInterval
31:     Calculate $p_{busy}$, $p_{qu}$, $DIO_{fail}$ using Eq. (9, 10, 11)
32:     Calculate $Q(s, a)$ and $r(s, a)$ using Eq. (6, 15)
33:     $I = min(I_{max}, I \times 2)$
34:     StartInterval($I$)
35:
36: function ReceivingDIS(packet)
37:     Transmit DIO
38:     if packet == Multicast DIS then Reset()
39:
40: function MCTxControl(packet)
41:      ▷ Triggered on each packet transmit in minimal cell
42:     if $t_{min} <= i <= t_{max}$ and $rand < p_{busy}$ and packet $\neq$
    DIO then
43:         Suppress packet
44:         Send if any DIO in queue
45:     else Transmit packet
46:
47: $N_{it} = DIO_{tr} = Q(S, A) = 0$           ▷ Initialization
48: Calculate $I_{max}$ based on $m_{max}$ using Eq. 1
49: StartInterval($I_{min}$)                    ▷ Start Trickle Timer
```

TABLE 3. Experiment parameters.

Parameter	Value
Simulation platform	6TiSCH Simulator
Scheduling and RPL OF	MSF and OF0
Topology (nodes)	2x5 (10), 5x10 (50), 10x10 (100)
Propagation model	Pister-Hack
Number of channels	16
SL and SD	101 slots and 10 ms
m_{max}	8
Runs x Duration	3 x 60 minutes
α, β and ε rate	0.1, 0.3, 0.5, 0.7, 0.9
I_{min}	1 s, 5 s, 10 s
k_{max}	10
p_{eb}	0.5, 0.25, 0.1
N^{pivot}_{nbr}	8
Data size and interval	20 bytes and 1 s
Root position	Bottom-Left
Testbed settings	
Site	Strasbourg, FIT IoT-Lab
Operating System	OpenWSN
Board	Arm Cortex M3
Topology (nodes)	1x10 (10), 3x10 (30)

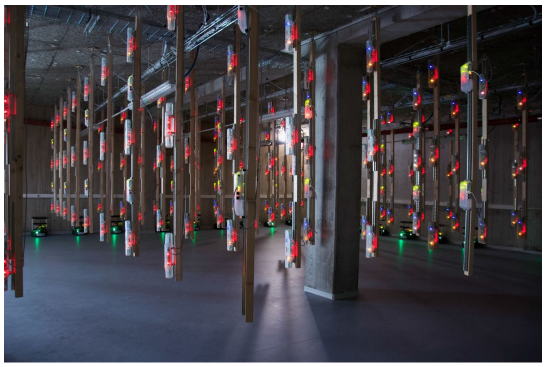


FIGURE 5. FIT IoT-Lab testbed deployment in Strasbourg site.

on the 6TiSCH energy model [36], expressed in milliampere per hour (mAh). The simulation sensor node has a battery capacity of 2821.5 mAh. Last is the number of DIO failed transmission represented as DIO_{fail}. We set data transmission as one packet per second with the size of 20 bytes based on agriculture monitoring application [37].

There are several scenarios in our experiment. First, we determine the QL variables of ε, α, and β rates for Q-Trickle. Then, we observe Q-Trickle convergence through trickle iterations and how Q-Trickle behaved over nodes addition and omission. Last, the benchmarks and proposed method were evaluated over different EB rate p_{eb}, minimum

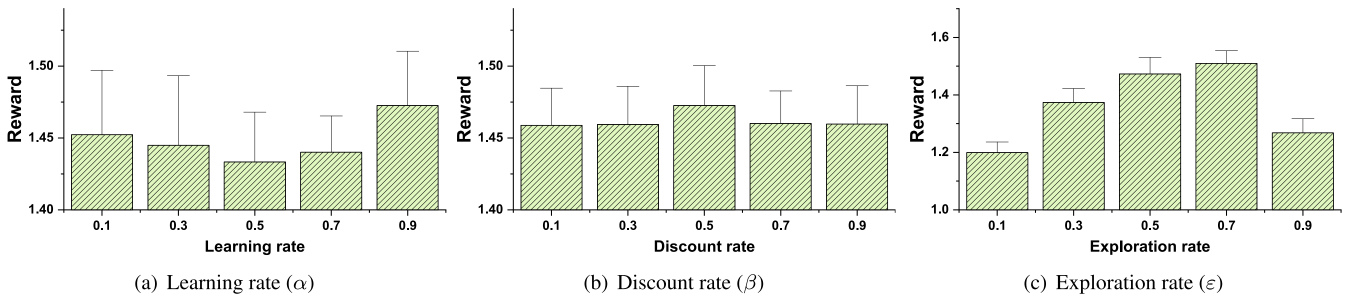


FIGURE 6. **Evaluation on Q-learning parameter.**

interval I_{min}, and the number of nodes. We test them over the simulation and real testbed on Strasbourg, FIT IoT-LAB [38] that implemented using OpenWSN OS [39]. The Strasbourg testbed environment is shown in Figure 5. Tested data for both simulation and testbed evaluation are represented as the mean and standard deviation for each metric (N=3).

A. Q-LEARNING PARAMETER SELECTION

First, we observe the parameter of the Q-learning algorithm for Q-Trickle to achieve a convergence result. We evaluated learning rate α, discount rate β, and exploration rate ε by varying the rate from 0.1 to 0.9. We did the evaluation sequentially and set the other rate with a value of 0.5. The number of nodes was 50, and the minimum interval was 5 s.

The initial parameter analyzed was the learning rate α. This parameter determines the extent to which the algorithm accepts the new value in relation to the old value. When the learning rate is 0, no new information is learned, while a learning rate of 1 results in the complete dismissal of the old value. Figure 6 (a) shows the evaluation of learning rate α and indicates that a value of 0.9 gives the highest reward. This number considers enough portion on old value while reinforcing to learn on new value. We use it in the following experiment of discount rate β.

The discount rate parameter, denoted as β, controls how much weight is given to future rewards. When β is set to 0, only the immediate rewards are considered, while a value of 1 means that future rewards are fully taken into account. Figure 6 (b) shows the evaluation of discount rate β, which result in a pretty similar result within 0.1 to 0.7. We pick 0.5 since it gives the leading result. We use this discount rate value in the following evaluation of exploration rate ε.

Exploration rate ε determines the ratio between choosing exploration and exploitation. Exploration will use the default action from the trickle timer that compares the current counter with the redundancy constant. Exploitation will use the optimal action (transmit or suppress DIO) based on the Q-value. Figure 6 (c) shows the evaluation of exploration rate ε that indicate 0.7 give the best result. This number points out that the agents require significant exploration to converge and learn effectively. Finally, we selected $\alpha = 0.9$, $\beta = 0.5$, and $\varepsilon = 0.7$ to use in the next experiment scenario.

B. Q-LEARNING CONVERGENCE

We observe the change of related variables of Q-Trickle to see their convergence over trickle iterations. We randomly selected one node to be observed. Figure 7 (a) shows that the number of neighbors N_{nbr} increases over iterations since more nodes are joined RPL DODAG. Redundancy constant k adapt based on N_{nbr} and p_{reset}. When k is high, the network is on a high p_{reset} or the unstable condition caused by reset events. Otherwise, k could be low when the network maintains a stable condition. This condition allows the agents to quickly resolve the inconsistency or reset events by setting k adaptively. Transmission time I also indicates it keeps resetting when reset events on the network occur. Lately, it can constantly increase without any more reset, which indicates the network starts having a stable condition. p_{reset} shows an increment and plateau value that indicates the network consistently has reset events like parent change, then decreases when the network becomes stable. $p_{transmit}$ was high at first, reflecting the high chances of DIO transmission in the early phase on the trickle timer. Then, it turns low in the following iteration since Q-Trickle adapts to network conditions and has more DIO suppression.

Figure 7 (b) shows that in the beginning, the reward fluctuates since it undergoes congestion from many nodes trying to join. This congestion in the early phase was observed by high usage of transmission queue represented in p_{qu} and high minimal cell busy ratio in p_{busy}. DIO_{fail} also gives the same signal that it keeps increasing. However, right after that phase, the node has a better condition represented by DIO_{fail} plateau value. p_{eb} was also decreasing as more nodes joined, and it reduced to lower network congestion.

C. NODES ADDITION AND OMISSION

We experiment on nodes addition and omission by randomly selecting 10% of the nodes in the network to simulate the scenario, which is 5 from 50 nodes. To simulate addition, we stop those nodes from booting, which prevents them from joining the network. Then after half time of the simulation runtime, we boot and allow them to join the network. Otherwise, for the omission scenario, we boot all 50 nodes, then shut down 5 target nodes after a half time of simulation runtime. We select one of the neighboring nodes to observe how the variable changes.

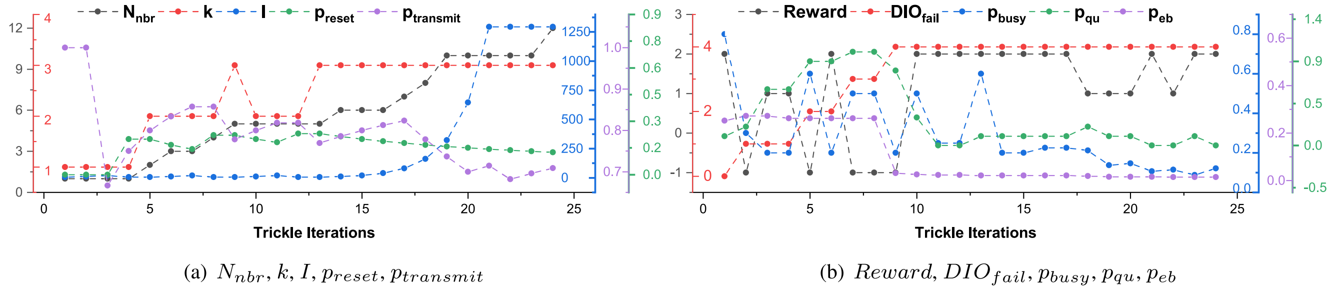

(a) $N_{nbr}, k, I, p_{preset}, p_{transmit}$ (b) $Reward, DIO_{fail}, p_{busy}, p_{qu}, p_{eb}$

FIGURE 7. Evaluation on Q-Trickle variable convergence.

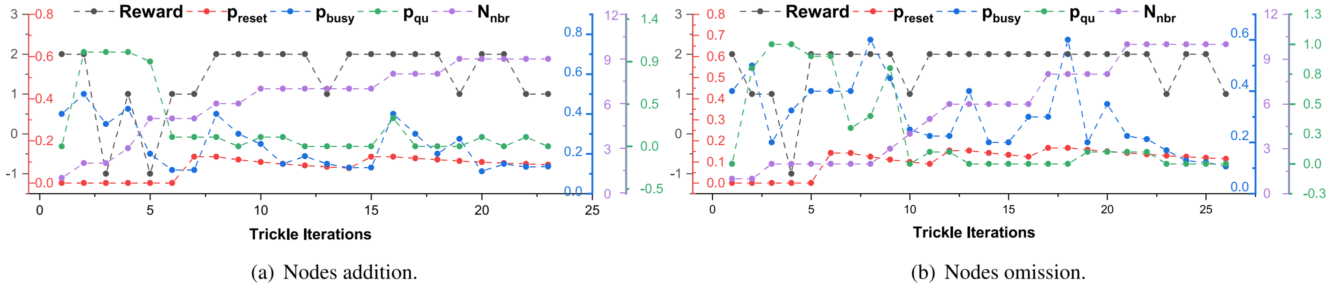

(a) Nodes addition. (b) Nodes omission.

FIGURE 8. Q-Trickle performance on nodes addition and omission.

Figure 8 (a) shows the result of the addition scenario. Reward fluctuation at early and late iterations. It drops at the beginning of learning iterations and has a stable positive reward value henceforward, although having changing network conditions. p_{pqu} and p_{busy} undergo the same thing. They spike at early and late iterations. Those phenomenons indicate the dynamic network changes in the addition scenario. p_{preset} have the same pattern as it increases after having the stable condition. Figure 8 (b) shows the same changes over the omission scenario. The reward, p_{reset}, p_{qu}, and p_{busy}, experience a pretty similar pattern like in the addition scenario that they fluctuate in early and near-late iterations. $p_{b}usy$ and N_{nbr} show an increasing number because omission nodes can lead to leaving orphan child nodes. Thus they send signals to rejoin with other nodes. The observation indicates that our proposed method can adapt to both scenarios.

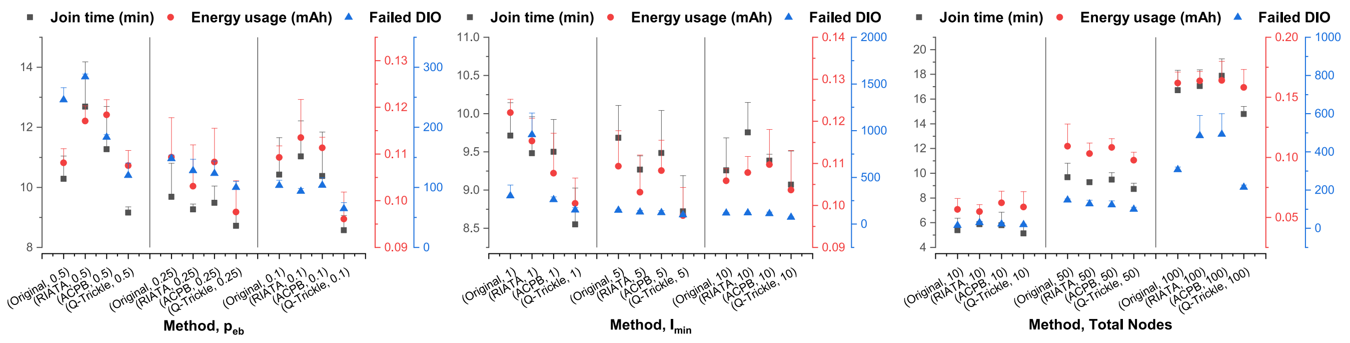
D. BENCHMARK ALGORITHMS

We test our proposed method, Q-Trickle, with the original trickle timer, RIATA, and ACPB. The test done under three control variables, namely EB transmission rate p_{eb}, Trickle timer minimum interval I_{min}, and network size or total nodes. On each variable test, we set the others parameters with the middle value.

1) EB RATE p_{eb}

As we can see in Figure 9 (a), DIO_{fail} tends to have smaller values over a lower EB rate. We observed that higher EB does not always result in faster joining time. Otherwise, it may congest the network and reduce performance on subsequent formation steps. 0.5 rate means that EB will be transmitted on every 2 slotframe intervals, taking too much load of minimal cell usage. A low rate like 0.1 also gets a bad result. Nodes have difficulty matching the EB packet channel since they randomly select it. Thus it cannot fulfill faster network synchronization and result in transferring more EB and consuming more energy. Q-Trickle outperforms other algorithms since it reinforces to decrease DIO_{fail} using the minimal cell transmission control scheme and intelligent DIO transmission by considering cell and queue conditions. Q-Trickle applied adaptive p_{eb} that was set higher initially to allow faster synchronization and gradually decrease it to lower congestion.

RIATA and ACPB have a worse result in several cases than the original trickle timer. ACPB has a control packet transmission scheme that considers cell conditions. However, it sets the harsh setting for minimal cell transmission, which only sends two control packets within the slotframe window. ACBP does not consider data traffic conditions, which may fulfill the queue and withhold other packets, including the control packet. RIATA suffers from training the QL algorithm to achieve better performance even than the original trickle timer. They used QL to handle inconsistencies or reset events but did not handle minimal cell conditions where DIO could be congested with EB and other packets. The original method got a lower result than Q-Trickle because it did not consider those network conditions or employ any adaptive parameters.



(a) EB rate (p_{eb}) (b) Minimum interval (seconds) (c) Network size

FIGURE 9. Evaluation on Benchmark Algorithms.

2) MINIMUM INTERVAL I_{min}

Figure 9 (b) shows the result on the different minimum intervals of the trickle timer. It can harden network formation convergence if I_{min} is too long. Suppose I_{min} is too small. In that case, it may not collect sufficient consistent transmissions and always decide to send DIO more frequently, which is unnecessary and consume more energy and lead to higher packet collision. A lower minimal interval of 1 second returned a more significant difference than the others, indicated by diverse joining time and energy usage. While DIO_{fail} maintains a relatively similar result over a different minimum interval. Overall, Q-Trickle gives a maximum result than the benchmarks. Q-Trickle uses adaptive transmission time t and redundancy constant k that reinforces to solve inconsistency or reset events quickly. Q-learning-based DIO transmission decisions support on higher successful transmission that results in efficient network formation. While the others suffers from the reason we mentioned before.

3) NETWORK SIZE

Figure 9 (c) shows the result of network size evaluation. All of the metrics are increased on a larger network size. In the small network of 10, the result is not much different. Q-Trickle did faster formation but consumed more energy since it doubled the EB transmission rate in the early formation phase to have faster synchronization. Q-Trickle is suitable for larger network sizes like 50, 100, and more nodes. Q-Trickle gives the best performance. Since Q-Trickle relies on learning from dynamic network conditions, more nodes could preserve such conditions. Q-Trickle can maintain lower DIO_{fail}, resulting in less energy wasted from transmission failure. Q-Trickle also has adaptive transmission time t and redundancy constant k that quickly resolve reset events that may waste much energy from that condition. RIATA and ACPB suffer from a higher number of failed DIO. It is caused by higher congestion happening in the network due to a large number of nodes. They received worse results than the original trickle timer, caused the reason we mentioned before. The improvement percentage of Q-Trickle compared to the benchmark algorithms is presented in Table 4.

TABLE 4. Q-Trickle improvement compared to benchmarks.

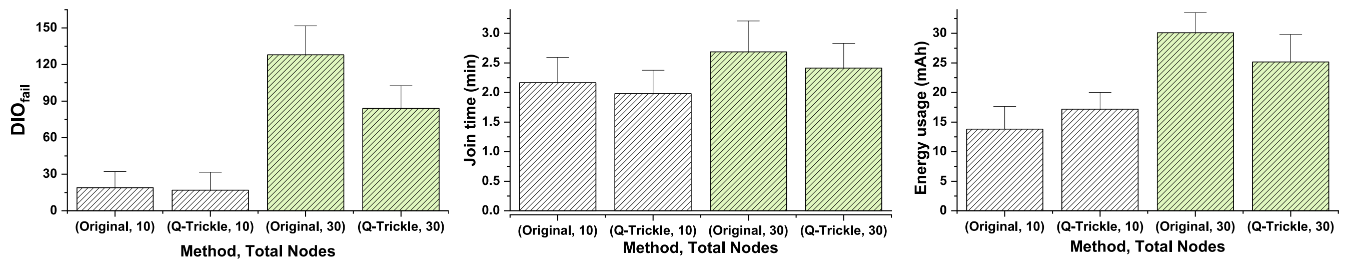
Method	Join time	Energy usage	DIO_{fail}
Original	-13%	-11%	-43%
RIATA	-20%	-10%	-48%
ACPB	-15%	-11%	-47%

E. TESTBED EVALUATION

We tested our algorithm on real sensor nodes on a remote testbed site at Strasbourg, FIT IoT-Lab. Our method was compared with the original trickle timer over 2 different network sizes, 10 and 30 nodes. The parameters were set as the same as the result of the previous experiment in the simulation. The result is shown in Figure 10. Increasing nodes will increase DIO_{fail}, joining time, and energy consumption. Although DIO_{fail} are relatively similar in small size network of 10 nodes, Q-Trickle get slightly faster network formation and more energy usage. It is caused by Q-Trickle adaptively setting a high p_{eb} at the beginning of the formation phase to have faster synchronization and gradually decrease it. To optimize it, we need to consider suitable pivot neighbors N_{nbr}^{pivot}, so that it may not degrade energy resources in a small network. Q-Trickle has a leading performance in the network with 30 nodes compared to the original trickle timer. The original method experienced a higher number of failed DIO transmissions, slower synchronization time, and higher energy usage. Q-Trickle can overcome those problems since it maintains intelligent DIO transmission that reinforces a high DIO success transmission, thus resulting in lower energy usage and faster RPL formation. Dynamic transmission time t and redundancy constant k also play roles in quickly resolving inconsistency or reset events. Adaptive p_{eb} for faster formation in the early formation phase and minimal cell transmission control scheme to prevent control packet congestion.

F. COMPUTATIONAL COMPLEXITY

The computational complexity of Q-Trickle is determined by the number of states and actions, denoted as s and a, respectively, and has a complexity of $O(s)(a)$. The number



(a) DIO failed transmission (DIO_{fail}) (b) Joining time (min) (c) Energy usage (mAh)

FIGURE 10. Evaluation on FIT IoT-Lab Testbed.

TABLE 5. Comparison of memory usage.

Method	ROM (bytes)	RAM (bytes)
Original	103,916	65,408
Q-Trickle	108,176	65,504

of state-action pairs is not dependent on network size as the QL policy is distributed and applied locally on each node. Sensor nodes have limited resources in terms of random-access memory (RAM) and read only memory (ROM). Q-Trickle algorithm requires extra memory due to its QL algorithm implementation. Table 5 shows that Q-Trickle uses an additional 4260 bytes of ROM and 96 bytes of RAM compared to the standard trickle-timer. This is because Q-Trickle needs to maintain more variables and functions to create the Q-value table, transmission control scheme, and adaptive methods for p_{eb}, t, and k.

V. CONCLUSION

The challenge accompanied by a trickle timer over a 6TiSCH network is DIO control packet transmission congestion over minimal cell and transmission queue. It led to a longer network formation time, higher energy consumption and lower network performance, like finding the optimal route. The trickle timer must also consider the network conditions to avoid an imbalanced transmission distribution owing to the static setting. Hence, Q-Trickle algorithm is proposed, which observes minimal cells and queue utilization to decide DIO transmission that avoids congestion using QL. Furthermore, Q-Trickle formulates adaptive redundancy constant, transmission interval and a transmission control scheme over minimal cells that reinforce faster synchronization and lower congestion. We evaluated Q-Trickle using 6TiSCH simulator and a real testbed from FIT IoT-Lab with OpenWSN OS. The evaluation was performed with existing trickle timer algorithms. The results showed that Q-Trickle could reduce network joining time, energy consumption and DIO failed transmission. Therefore, Q-Trickle can be used as an optimized algorithm for efficient 6TiSCH network formation.

Although this research showed the optimized result in using Q-Trickle, there is a limitation that our QL algorithm was employed locally distributed on each sensor node. There will be imbalanced learning on sensors that experience more dynamic conditions to train better QL policy. In future work, we will explore multi-agent reinforcement learning for our subject, enabling the integration of learning outcomes from multiple interacting RL agents. Subsequently, enhancing our adaptive EB rate approach is essential, so it may not degrade energy resources in a small network. We will also consider using other shared cells to improve control packet transmission performance.

REFERENCES

[1] M. Frohberg, S. Weidling, and P. Langendoerfer, "Challenges in developing a wireless sensor network for an agricultural monitoring and decision system," in *Proc. Int. Netw. Conf.*, 2020, pp. 224–240.

[2] D. P. Kumar, A. Tarachand, and C. S. R. Annavarapu, "Machine learning algorithms for wireless sensor networks: A survey," *Inf. Fusion*, vol. 49, pp. 1–25, Sep. 2019.

[3] R. Ali, N. Shahin, Y. B. Zikria, B.-S. Kim, and S. W. Kim, "Deep reinforcement learning paradigm for performance optimization of channel observation–based MAC protocols in dense WLANs," *IEEE Access*, vol. 7, pp. 3500–3511, 2018.

[4] V. Di Valerio, F. L. Presti, C. Petrioli, L. Picari, D. Spaccini, and S. Basagni, "CARMA: Channel-aware reinforcement learning-based multi-path adaptive routing for underwater wireless sensor networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 11, pp. 2634–2647, Nov. 2019.

[5] P. Thubert, *An Architecture for IPv6 Over the Time-Slotted Channel Hopping Mode of IEEE 802.15.4 (6TiSCH)*, document RFC 9030, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9030

[6] C. Yibo, K.-M. Hou, H. Zhou, H.-L. Shi, X. Liu, X. Diao, H. Ding, J.-J. Li, and C. de Vaulx, "6LoWPAN stacks: A survey," in *Proc. 7th Int. Conf. Wireless Commun., Netw. Mobile Comput.*, Sep. 2011, pp. 1–4.

[7] T. Chang, M. Vucinic, X. Vilajosana, S. Duquennoy, and D. R. Dujovne, *6TiSCH Minimal Scheduling Function (MSF)*, document RFC 9033, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9033

[8] X. Vilajosana, K. Pister, and T. Watteyne, *Minimal IPv6 Over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration*, document RFC 8180, May 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8180

[9] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 595–615, 1st Quart., 2020.

[10] R. T. Hermeto, A. Gallais, and F. Theoleyre, "Scheduling for IEEE802.15.4-TSCH and slow channel hopping MAC in low power industrial wireless networks: A survey," *Comput. Commun.*, vol. 114, pp. 84–105, Dec. 2017.

[11] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, document RFC 6550, Mar. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6550

[12] P. Levis, T. H. Clausen, O. Gnawali, J. Hui, and J. Ko, *The Trickle Algorithm*, document RFC 6206, Mar. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6206

[13] T. M. M. Meyfroyt, "An analytic evaluation of the trickle algorithm: Towards efficient, fair, fast and reliable data dissemination," in *Proc. IEEE 16th Int. Symp. A World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2015, pp. 1–2.
[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.
[15] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[16] S. Goyal and T. Chand, "Improved trickle algorithm for routing protocol for low power and lossy networks," *IEEE Sensors J.*, vol. 18, no. 5, pp. 2178–2183, Mar. 2018.
[17] M. B. Yassein, S. Aljawarneh, and E. Masa'deh, "A new elastic trickle timer algorithm for Internet of Things," *J. Netw. Comput. Appl.*, vol. 89, pp. 38–47, Jul. 2017.
[18] H. Lamaazi and N. Benamar, "RPL enhancement based FL-trickle: A novel flexible trickle algorithm for low power and lossy networks," *Wireless Pers. Commun.*, vol. 110, no. 3, pp. 1403–1428, Feb. 2020.
[19] A. Musaddiq, Y. B. Zikria, and S. W. Kim, "Energy-aware adaptive trickle timer algorithm for RPL-based routing in the Internet of Things," in *Proc. 28th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2018, pp. 1–6.
[20] M. Vucinic, M. Krol, B. Jonglez, T. Coladon, and B. Tourancheau, "Trickle-D: High fairness and low transmission load with dynamic redundancy," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1477–1488, Oct. 2017.
[21] S.-T. Liu and S.-D. Wang, "Improved trickle algorithm toward low power and better route for the RPL routing protocol," *IEEE Access*, vol. 10, pp. 83322–83335, 2022.
[22] A. Aghaei, J. Akbari Torkestani, H. Kermajani, and A. Karimi, "LA-trickle: A novel algorithm to reduce the convergence time of the wireless sensor networks," *Comput. Netw.*, vol. 196, Sep. 2021, Art. no. 108241.
[23] B. Ghaleb, A. Y. Al-Dubai, E. Ekonomou, I. Romdhani, Y. Nasser, and A. Boukerche, "A novel adaptive and efficient routing update scheme for low-power lossy networks in IoT," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 5177–5189, Dec. 2018.
[24] Z. Nain, A. Musaddiq, Y. A. Qadri, A. Nauman, M. K. Afzal, and S. W. Kim, "RIATA: A reinforcement learning-based intelligent routing update scheme for future generation IoT networks," *IEEE Access*, vol. 9, pp. 81161–81172, 2021.
[25] A. Kalita and M. Khatua, "Adaptive control packet broadcasting scheme for faster 6TiSCH network bootstrapping," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17395–17402, Dec. 2021.
[26] A. Kalita and M. Khatua, "Autonomous allocation and scheduling of minimal cell in 6TiSCH network," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 12242–12250, Aug. 2021.
[27] A. Kalita and M. Khatua, "Opportunistic transmission of control packets for faster formation of 6TiSCH network," *ACM Trans. Internet Things*, vol. 2, no. 1, pp. 1–29, Jan. 2021.
[28] C. Vallati, S. Brienza, G. Anastasi, and S. K. Das, "Improving network formation in 6TiSCH networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 1, pp. 98–110, Jan. 2019.
[29] Y. Ha and S.-H. Chung, "Enhanced 6P transaction methods for industrial 6TiSCH wireless networks," *IEEE Access*, vol. 8, pp. 174115–174131, 2020.
[30] M.-J. Kim and S.-H. Chung, "Efficient route management method for mobile nodes in 6TiSCH network," *Sensors*, vol. 21, no. 9, p. 3074, Apr. 2021.
[31] S. Kim, H.-S. Kim, and C. Kim, "ALICE: Autonomous link-based cell scheduling for TSCH," in *Proc. 18th Int. Conf. Inf. Process. Sensor Netw.*, 2019, pp. 121–132.
[32] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *Proc. 13th ACM Conf. Embedded Netw. Sensor Syst.*, 2015, pp. 337–350.
[33] F. Righetti, C. Vallati, S. K. Das, and G. Anastasi, "Analysis of distributed and autonomous scheduling functions for 6TiSCH networks," *IEEE Access*, vol. 8, pp. 158243–158262, 2020.
[34] E. Municio, G. Daneels, M. Vučinić, S. Latré, J. Famaey, Y. Tanaka, K. Brun, K. Muraoka, X. Vilajosana, and T. Watteyne, "Simulating 6TiSCH networks," *Trans. Emerg. Telecommun. Technol.*, vol. 30, no. 3, p. e3494, Mar. 2019.
[35] H.-P. Le, M. John, and K. Pister, "Energy-aware routing in wireless sensor networks with adaptive energy-slope control," in *Proc. EE290Q-2*, pp. 1–6, Spring 2009.
[36] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. Pister, "A realistic energy consumption model for tsch networks," *IEEE Sensors J.*, vol. 14, no. 2, pp. 482–489, 2013.
[37] K. S. Burman, S. Schmidt, D. E. Houssaini, and O. Kanoun, "Design and evaluation of a low energy Bluetooth sensor node for animal monitoring," in *Proc. 18th Int. Multi-Conf. Syst., Signals Devices (SSD)*, Mar. 2021, pp. 971–978.
[38] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Proc. IEEE 2nd World Forum Internet Things (WF-IoT)*, Dec. 2015, pp. 459–464.
[39] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "OpenWSN: A standards-based low-power wireless development environment," *Trans. Emerg. Telecommun. Technol.*, vol. 23, no. 5, pp. 480–493, 2012.

DZAKY ZAKIYAL FAWWAZ (Student Member, IEEE) received the B.E. degree from the Department of Computer Engineering, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia, in 2018. He is currently pursuing the integrated M.S.–Ph.D. degree with the Department of Information Convergence Engineering, Pusan National University, Busan, South Korea. His research interests include wireless sensor networks, software-defined networking, edge computing, and artificial intelligence.

SANG-HWA CHUNG (Member, IEEE) was born in Busan, South Korea, in 1960. He received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 1985, the M.S. degree in computer engineering from Iowa State University, Ames, IA, USA, in 1988, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1993.

From 1993 to 1994, he was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, USA. Currently, he is a Professor with the Department of Information Convergence Engineering, Pusan National University (PNU), Busan. Since 2016, he has been the Director of the Dong-Nam Grand ICT Research Center. He has authored over 240 articles and holds 70 patents. His research interests include embedded systems, wireless networks, software-defined networking, and smart factories. He received the Best Paper Award from the *ETRI Journal*, in 2010, and the Engineering Paper Award from PNU, in 2011. In 2017, he was selected as an Excellent Research Professor of the Computer Engineering Faculty at PNU.