MobiRPL: Adaptive, Robust, and RSSI-based Mobile Routing in Low Power and Lossy Networks

Hongchan Kim, Hyung-Sin Kim, and Saewoong Bahk

***Abstract*—This paper tackles the mobile routing issues in low-power and lossy networks (LLNs). The IPv6 standard routing protocol for LLNs, termed IPv6 routing protocol for low-power and lossy networks (RPL), has mostly been investigated in static LLNs and it has no explicit mechanism to support mobility. In addition, there is no mobile routing protocol that works well in mobile LLNs. Considering the importance of mobility support in many LLN applications, this work designs and implements *MobiRPL*, an adaptive, robust, and received signal strength indicator (RSSI)-based mobile routing scheme based on the RPL standard. To cope with network dynamics, the *MobiRPL* design focuses more on maintaining reliable routing topology than on minimizing energy consumption. This design choice significantly improves reliability while maintaining the acceptable energy consumption of mobile LLNs. We implement *MobiRPL* on Contiki OS, and evaluate its effectiveness extensively through Cooja simulation and testbed experiments. Our results from the testbed show that *MobiRPL* improves mobile nodes' packet delivery ratio by 11.3% compared to RPL and reduces the energy consumption of mobile nodes by 73.3% compared to the baseline scheme, i.e.,the lightweight on-demand ad-hoc distance-vector routing protocol - next generation (LOADng).**

***Index Terms*—IEEE 802.15.4, low power lossy network (LLN), mobility, routing.**

I. INTRODUCTION

RPL, the IPv6 routing protocol for low-power and lossy networks (LLNs) standardized in 2012, has been considered as a building block of Internet of Things (IoT) and received great attention from LLN researchers [1]–[4]. RPL reliably and energy-efficiently forms a quasi-forest routing topology to provide IPv6 connectivity to a large number of resource-constrained embedded devices through a few border routers. Motivated by LLN application scenarios, such as home, industrial, urban, and building applications [5]–[8], RPL was designed under the assumption that most devices are static, having no mechanism to explicitly support mobile devices. Not surprisingly, most researches on RPL have considered only static nodes [2].

At the same time, however, there are *non-zero* mobile devices in the above application scenarios: Remote controllers and wearable devices at home [7], cranes, forklifts, and workers in an industrial environment [6], and occupants and movable assets in a building [8]. In addition, clinical applications include mobile medical staffs and patients [9]. To minimize network dynamics with these mobile nodes, there is a routing design guideline in [8]: "*Mobile devices, while in motion, should not be allowed to act as forwarding devices.*" Following this, a number of studies have investigated RPL in hybrid settings: Static router nodes and mobile leaf[1] nodes (walking speed) [10]–[17]. They showed that RPL is problematic even in hybrid environments and attempted to alleviate the problem.

However, depriving mobile nodes of routing/forwarding capability gives a nontrivial constraint: Every mobile node must be within the (unpredictable) coverage of at least one static router node. Strictly speaking, this constraint requires predicting all possible travel areas of mobile nodes and deploying static router nodes sufficiently to cover all the areas, which is hard or impractical in many cases. Even if static router nodes are sufficiently deployed, problems can still arise. Communication environments may change while the network operates. For example, an obstacle added to the network (e.g., a person passing through the network) can make the connection between nodes lost. If the static router nodes are battery-powered, they may fail to provide connectivity due to low battery. In these situations, the remaining static router nodes may not be able to provide connectivity throughout the network [18], [19].

This work considers *more general mobility scenarios* (i.e., non-hybrid settings) where both static and mobile nodes (still having walking speed) participate in packet routing and forwarding. Our intuition is that allowing routing/forwarding of mobile nodes can significantly improve connectivity, reduce the deployment burden, and improve their ability to cope with network dynamics. There have been some studies that explored RPL in *more general mobility scenarios* [20]–[27]. However, some of these studies have limitations arising from inefficient operation designs, and some studies require several assumptions or the support of external mechanisms to operate correctly. Therefore, there is a need for a mobile RPL that

Manuscript received January 16, 2022; approved for publication, January 18, 2022. This paper is specially handled by EIC and Division Editor with the help of three anonymous reviewers in a fast manner.

This work was supported in part by Korea Environment Industry & Technology Institute (KEITI) through Exotic Invasive Species Management Program, funded by Korea Ministry of Environment (MOE) (2021002280002), in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-0-02048) supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation), and in part by the New Faculty Startup Fund from Seoul National University.

H. Kim and S. Bahk are with the Department of ECE and INMC, Seoul National University, Seoul, Republic of Korea, email: hckim@netlab.snu.ac.kr, sbahk@snu.ac.kr.

H.-S. Kim is with the Graduate School of Data Science, Seoul National University, Seoul, Republic of Korea, email: hyungkim@snu.ac.kr.

S. Bahk and H.-S. Kim are corresponding authors.

Digital Object Identifier: 10.23919/JCN.2022.000004

[1]Nodes that do not have sub-nodes and do not perform routing/forwarding for other nodes.

considers the fundamental problems of RPL in mobile LLNs and works with minimal assumptions.

On the other hand, given that the RPL design does not aim to support the general mobility scenario above, *why don't we use or improve another routing protocol originally designed for mobile networks, such as the lightweight on-demand ad-hoc distance-vector routing protocol - next generation (LOADng)* [28], *rather than RPL?* LOADng is a lightweight routing protocol designed for mobile ad-hoc networks (MANETs). Considering that LLN mostly generates upward traffic (i.e., data collection), it is not clear to say that LOADng is well suited for LLN. Indeed, there have been various studies comparing RPL and LOADng [29]–[33]. Because the existing routing protocols designed for data collection do not cope with mobility well, it is worth testing LOADng under various scenarios.

Although LOADng is designed for mobile networks, we found that LOADng has scalability issues as traffic increases due to its flooding-based reactive nature. Besides, to the best of our knowledge, RPL, in terms of both protocol design and implementation, has been most extensively investigated and tested in LLN environments considering resource constraints and link dynamics. In this regard, studying mobile routing in the context of RPL builds on two decades of LLN research and makes our work well-grounded.

Challenges. In a scenario where mobile nodes perform forwarding, challenges are on the protocol design phase rather than the deployment phase. Given that including mobile router nodes increases network dynamics, *how can we design RPL to be reliable and energy-efficient under such dynamics?* The following two requirements need to be fulfilled.

(1) Parent[2] table management: A node should keep track of each neighbor node's link quality and Rank information *reasonably fast* to maintain the parent table freshly. At the same time, information tracking should not impose too much control overhead on the network.

(2) Parent selection[3]: To cope with dynamics, a parent selection mechanism should focus on stability rather than efficiency (e.g., shortest path). Selecting an efficient but barely connected node as the preferred parent may cause confusion in a dynamic mobile network. Although efficiency is not a primary concern in parent selection, it should not be overlooked either.

Approach. RPL fails to meet the above requirements and provide an appropriate routing topology in a mobile network scenario. We introduce *MobiRPL*, which addresses the challenges without violating the RPL standard. Compared to previous work regarding RPL, *MobiRPL design puts more weight on reliability than energy efficiency.* The idea is that low energy consumption makes sense only when a reliable routing topology is given. To this end, we design *MobiRPL* with three main components.

(1) Mobility detection: *MobiRPL* Allows both mobile and static nodes to participate in packet forwarding. However, the characteristics of mobile and static nodes in the routing process are very different. To ensure good routing performance, *MobiRPL* allows each node to detect its mobility from routing information and makes routing decisions based on the detected mobility.

(2) Connectivity Management: Each node manages the connectivity with parent nodes according to its mobility. To this end, *MobiRPL* performs timeout and probing in which the period is adaptively adjusted. If a parent node is determined as disconnected, *MobiRPL* excludes the parent node from the parent table. If there are not enough valid parent nodes in the parent table, *MobiRPL* discovers new parent nodes through proactive discovery.

(3) RSSI and hop distance-based objective function: In order to select a suitable preferred parent for stable routing in mobility scenarios, *MobiRPL* uses received signal strength indicator (RSSI) and hop distance as routing metrics instead of the popular expected transmission count (ETX). *MobiRPL* makes parent selection more stable under dynamic environments by considering node mobility along with the RSSI and hop distance based routing metric.

Contributions. We summarize the contributions of this work as follows.

- We perform a measurement study of RPL and LOADng, verifying that LOADng is not a good choice for LLNs by showing its scalability issues.
- We give routing/forwarding capability to mobile LLN nodes and show the feasibility of dynamic scenarios by resolving challenges on the RPL protocol design.
- We design *MobiRPL*, including the above three components, implement it on Contiki operating system. We make our prototype implementation publicly available.[4]
- We verify the effectiveness of *MobiRPL* on Cooja simulation [34] and a 34-node testbed. On the testbed, *MobiRPL* shows an 11.3% increase in packet delivery ratio compared to RPL and a 73.3% decrease in energy consumption compared to LOADng at mobile nodes, outperforming RPL and LOADng.

The remainder of this paper is organized as follows. We first discuss the brief background and related work in Section II. In Section III, we present the preliminary study results. We summarize the requirements for our proposed scheme in Section IV. We introduce our proposed scheme, *MobiRPL*, and elaborate on its main functional blocks in Section V. We discuss the implementation details and the evaluation results in Section VI. We conclude the paper in Section VIII.

II. Background and Related Work

There have been two types of research efforts to support mobile LLN: (1) Extending RPL [1] to cover mobile nodes and (2) modifying routing protocols for MANETs to support LLNs. The latter results in LOADng [28], a lightweight version of ad-hoc on-demand distance vector (AODV) (a standard MANET routing protocol) [35]. This section reviews the two

[2]In RPL, a candidate node for the next-hop node in the upward route is called a parent. The currently selected parent node is called a preferred parent. The parent table stores all the parent nodes (both preferred and non-preferred).

[3]Parent selection means choosing a preferred parent among the parents.

[4]https://github.com/Hongchan-Kim/MobiRPL

representative protocols, RPL and LOADng, and their related work.

A. RPL and Mobility

RPL Design. Given that traffic mostly goes upwards in LLN (i.e., data collection), RPL forms a destination-oriented directed acyclic graph (DODAG) rooted at a root node, generally an LLN border router to external networks. Each RPL node in a DODAG, including the root node, propagates routing information by broadcasting a control message named DODAG information objective (DIO). An RPL node obtains RPL configurations by receiving DIO and participates in a DODAG by choosing a preferred parent. The DIO transmission interval follows TrickleTimer [36], which doubles the DIO interval after each DIO transmission to minimize control overhead while resetting it to the minimum upon inconsistency detection for fast route recovery. In addition, a node triggers a DIO transmission from its neighbor nodes *on demand* by sending a DODAG information solicitation (DIS) message.

An RPL node selects the best preferred parent from many candidates by using the path metric called Rank. The definition of Rank and rules for parent selection, called objective function (OF), are decoupled from the main RPL standard. The most widely used OF is minimum rank with hysteresis objective function (MRHOF) [37], which uses ETX as the link quality metric and accumulated ETX over a node's upward path as its Rank. The use of ETX is to minimize upward transmission overhead. Lastly, each node transmits destination advertisement object (DAO) messages to the root node along the upward route, which sets its downward route from the root as the reverse of the upward route.

Mobility Support with RPL. Several studies have investigated RPL operation in mobile scenarios, showing that RPL suffers significant performance degradation due to lack of consideration for mobile nodes [2], [38]–[40].

A number of studies have tried to improve RPL to support mobile nodes, most of which use mobile nodes *only as leaf nodes* (i.e., hybrid setting). For example, KP-RPL supports a parent handover for mobile leaf nodes [10]. A mobile node estimates its location by applying Kalman filter to the RSSI from adjacent static nodes. Then it calculates the expected ETX from RSSI and handovers to the best static node. The authors in [13] proposed mobility-aware parent selection RPL, which considers hop distance and RSSI value together to detect the moving direction of a mobile leaf node and then handover to the parent located on the path of movement.

Under the hybrid settings, some studies have attempted to support mobility in an energy-efficient way. EMA-RPL [14] delegates most of their mobility support operations to static nodes to reduce the overhead of mobile nodes. When a static node detects the mobility of a mobile node from RSSI changes, it triggers the mobile node to start burst DIS broadcasting. Neighboring static nodes measure the average RSSI of DIS messages, append it to DIO, and reply to the mobile node. Then the static node that firstly detected the mobility compares RSSI values in DIO messages and informs the mobile node of the best static node. Finally, the mobile node connects to the next static node. EKF-MRPL [15] further improves EMA-RPL by introducing the Extended Kalman filter (EKF). EKF-MRPL assumes that mobile nodes know the exact location of static nodes. Static nodes detect mobility and trigger mobile nodes, similar to EMA-RPL. However, in EKF-MRPL, a mobile node broadcasts DIS once and receives a unicast DIO response from each DIS recipient. By applying EKF to the RSSI values from DIO responses, the mobile node estimates its location and direction, then determines the best static node for connection.

However, depriving mobile nodes of routing/forwarding capability significantly increases the deployment burden; static nodes should be deployed in large enough numbers to cover all areas of the network. Whenever an area where static nodes cannot provide connectivity appears, mobile nodes will not be able to communicate with other nodes without additional static nodes deployed.

Assuming hybrid environments, some other studies have improved timer operations of RPL to better support mobility. The authors in [11] designed a node having a mobile leaf node as a child to exploit a *reverse* TrickleTimer (i.e., halving the DIO interval after each DIO transmission). The intuition is that the link quality between a mobile leaf node and its preferred parent is likely to be degraded as time goes by, and reducing the parent's DIO interval enables the mobile leaf node to update routing information quickly. The authors in [12] proposed DIS interval adaptation for mobile nodes to get DIO quickly when needed. A mobile node calculates the time taken to leave the communication range of the currently connected static node by using RSSI and the Doppler effect. Then it schedules DIS broadcast before it departs the static node, thus enabling timely DIO reception and handover.

However, the modified timer mechanisms designed for mobile nodes to receive DIO faster or more often do not necessarily help mobile nodes. No matter how fast or how often DIO is sent, if RPL does not manage newly updated routing information appropriately, this information will be outdated and incorrect soon. Such incorrect routing information may lead mobile nodes to choose unreachable parents and lose connectivity repetitively. Therefore, the improvement in parent table management should precede the enhancement in timer operations.

Some studies have considered more general mobility (i.e., non-hybrid setting) scenarios where mobile nodes can perform routing or forwarding. The authors in [20] proposed ME-RPL, which gives mobile nodes routing capability in a limited way. When selecting its preferred parent, a ME-RPL node prefers static nodes to mobile nodes since static nodes are more likely to provide robust connectivity. mRPL [21] and mRPL+ [22] support parent handover for mobile nodes. When a mobile node detects that the RSSI from its preferred parent is low, it broadcasts a batch of DISes. Each DIS recipient measures the average RSSI of the DIS batch and includes it when sending a DIO as a reply. Then the mobile node selects a node reporting the best RSSI as its preferred parent.

To cope with dynamics in mobile LLNs, MoMoRo [23] detects route disconnection from link loss and obtains neighbor information quickly by requesting a unicast response. It introduces a fuzzy estimator that considers multiple link metrics to find a neighbor connected through a good link. Co-RPL [24] defines corona ID (minimum hop distance) to indicate how close a node is to the root node and uses it for parent selection. RRD [25] utilizes the RSSI of a broadcast message and an ACK message to detect a mobile node's moving direction. RRD updates the Rank and assigns an appropriate length of lifetime to a route based on this direction and the RSSI value. GTM-RPL [26] introduces game theory into the RPL to support mobility. It finds an optimal parameter setup to minimize mobile nodes' disconnected period. However, GTM-RPL did not investigate the problems arising from the protocol design of RPL.

Although these prior studies extend RPL for mobility scenarios, their Rank (i.e., ETX) and parent selection are from mobility-unfriendly MRHOF, which is their fundamental limitation. MRHOF was designed without considering mobility. For example, even between links that show the best ETX, some other difference may exist, such as the physical distance gap between nodes [41], [42]. This distance gap can be an essential parameter in choosing a long-lasting routing path in mobility scenarios, but ETX cannot provide such significant information. Moreover, MRHOF applies an exponentially weighted moving average (EWMA) filter to the ETX. This filtering lowers the responsiveness of the link metric to the link quality change and makes it challenging for mobile nodes to cope with mobility. Therefore, ETX and MRHOF should be reconsidered together to better support mobility.

Several recent studies have attempted to improve the performance of RPL in mobile scenarios based on specific assumptions or external mechanisms. The authors in [16] proposed Coral software-defined networks (SDN), which cooperates with performance-limited IoT devices. Coral SDN can change RPL parameters to suit mobile scenarios even during runtime. However, the authors only naively controlled parameters related to DIO interval. SDMob [17] is another example of SDN-based mobility support for RPL in a hybrid setting. In SDMob, mobile nodes are equipped with inertial measurement unit (IMU) sensors. The SDN controller knows the precise location of static nodes. Each mobile node periodically broadcasts a beacon, including velocity information from IMU sensors. Static nodes receiving the beacon append the measured RSSI and relay it toward the SDN controller. Then the SDN controller calculates the best next static node for the mobile node based on information contained in the beacon and notifies the result to the mobile node.

ARMOR [27] aims at a mobile RPL for a non-hybrid setting. In ARMOR, all nodes are assumed to know their velocity and location using IMU sensors or external localization mechanisms. Then, each node calculates the time-to-reside (TTR) within the communication range of each neighboring node. ARMOR chooses the parent with the longest connection time using the TTR as a new routing metric.

As aforementioned, many studies have tried to improve RPL to support mobile nodes. However, despite their various attempts, each has its own limitations. Therefore, it is necessary to fundamentally investigate why RPL does not work well on mobile nodes and design a mobile RPL that comprehensively considers the problems observed in RPL. Besides, the assumptions or mechanisms introduced to better support mobility may limit the usability of mobile RPL protocols. For example, it will not be able to add IMU-free devices to the network where a routing protocol is assumed to use IMU sensors. On the other hand, routing protocols that require localization or SDN will not function properly unless the conditions are met. Therefore, we need a mobile RPL protocol that operates with minimal assumptions.

B. LOADng, a MANET Protocol for LLN

Many routing protocols have been developed for MANETs [43], such as destination sequenced distance vector (DSDV) [44], optimized link-stated routing (OLSR) [45], AODV [35], and dynamic source routing (DSR) [46]. In LLN, however, these protocols cause significant control overhead and/or slow recovery [47]. To alleviate the problems, LOADng [28], [48], [49], a lightweight version of AODV, was proposed as a routing solution for mobile LLNs.

As in AODV, a source node in LOADng broadcasts a route request (RREQ) message to discover a path toward its destination node. Different from AODV, however, LOADng allows only the destination node to send a route reply (RREP) message back to the source as a response to the RREQ; an intermediate node only relays the RREQ, which simplifies protocol operation. When detecting a route failure, a LOADng node sends a route error (RERR) message only to the source node, which removes memory overhead for maintaining a precursor list. LOADng also exploits a random jitter when sending an RREQ to resolve congestion due to RREQ flooding. Despite its optimization for LLNs, LOADng is fundamentally not free from AODV's RREQ flooding overhead, which increases with the number of end-to-end sessions [47] and worsens in a duty-cycled network.

LOADng has not been investigated extensively in academia, much less than RPL. Several studies have revealed that in static scenarios, LOADng provides similar performance as RPL only in sparse LLN deployments [29]–[33]. Otherwise, LOADng underperforms RPL. As an improvement, LOADng-CTP [49], [50] tweaks LOADng to better support data collection, building a tree-shaped routing structure rooted at a sink node as RPL and CTP do. Nevertheless, without any experimental evaluation, it is still unclear if LOADng is really an effective solution for mobile LLNs. One of our contributions is to provide the measurement study of LOADng on a mobile LLN testbed.

III. PRELIMINARY STUDY

The previous sections qualitatively showed why it makes sense to design a new mobile routing protocol for LLNs. Building on this intuition, this section presents an experimental, quantitative study of RPL and LOADng on an LLN testbed in static scenarios and a simulation in mobile scenarios.

A. Static Scenario: RPL vs. LOADng

We configure an indoor testbed with 31 *static* TelosB-clone nodes where one node acts as the root, as depicted in Fig. 1. Each node uses -10 dBm transmission power and an

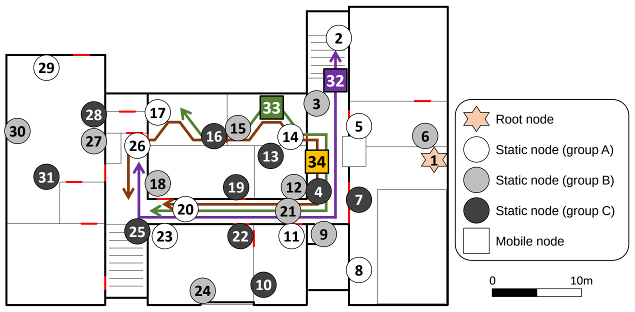


Fig. 1. An indoor testbed with 31 static nodes and 3 mobile nodes. The mobile nodes move along the path shown in the figure.

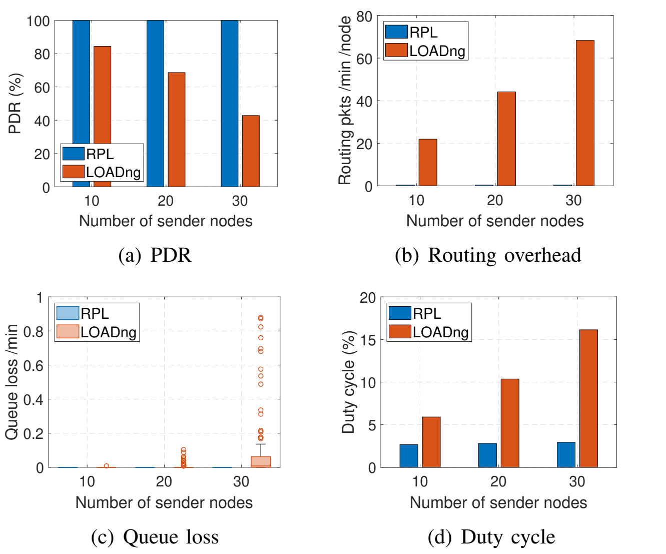


Fig. 2. Performance of RPL and LOADng on a testbed with 31 static nodes (Fig. 1) according to the number of end-to-end sessions.

antenna of 5 dB gain. For the routing layer, RPL and LOADng implementations on Contiki OS [51] version 3.0 are used. The underlying link layer is ContikiMAC [52], the asynchronous duty-cycling MAC of Contiki OS. We set the sleep interval of ContikiMAC as 31.25 ms (32 Hz channel check rate).

To evaluate the scalability of RPL and LOADng according to the number of end-to-end sessions, we divide 30 nodes (except the root) into three groups (group A, B, and C), each of which has ten nodes. Then, we observe the performance of RPL and LOADng with varying the number of data senders. We consider a bidirectional traffic scenario. Each sender node transmits an upward packet every 60 seconds. At the same time, the root node generates the same amount of downward traffic as the upward traffic by sequentially sending downward packets to the sender nodes over 60 seconds. The total number of upward and downward packets generated for each sender is 120 each. Fig. 2 plots various performance metrics in the scenario (the average of five repetitive experiments).

As seen in Fig. 2(a), the end-to-end packet delivery ratio (PDR) of LOADng decreases sharply as the number of sender nodes increases, while RPL maintains high PDR. This performance difference comes from the different routing mechanisms of RPL and LOADng; while RPL manages a

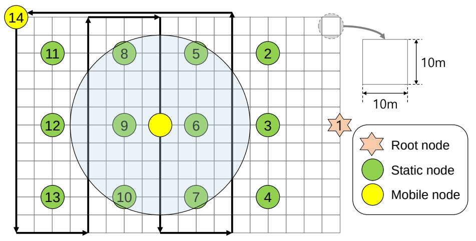


Fig. 3. A mobile LLN scenario on the Cooja simulator with 12 static nodes, a mobile node, and a root. All the nodes have the same transmission range (50 m), and one example is indicated by a large light blue circle.

single DODAG topology for all nodes, LOADng builds a separate end-to-end route per sender-destination pair, resulting in up to 30 independent routes in our setting. Given that LOADng floods RREQ messages to build a route for a sender-destination pair, its routing overhead increases with the number of data senders (i.e., the number of routes). This is verified in Fig. 2(b), which shows the routing overhead of RPL and LOADng. While RPL maintains low routing overhead regardless of the number of senders, LOADng incurs much higher routing overhead than RPL, and the amount increases with the number of senders. As a result, LOADng incurs ∼*150 times* more routing overhead than RPL. If we add more senders to the network, LOADng causes more routing overhead, while RPL would maintain a similar level of routing overhead.

With an asynchronous duty-cycling MAC, such as ContikiMAC, the large flooding overhead causes severe congestion and contention problems. Fig. 2(c) shows that in LOADng cases, several nodes suffer severe queue loss when the number of senders is large. This confirms that the congestion level caused by LOADng is above the threshold which resource-constrained nodes can cope with. According to Fig. 2(d), the large routing overhead decreases PDR due to congestion and increases duty cycle since each node frequently turns on the radio to exchange more routing control packets. Given the relationship between the number of nodes and flooding overhead of LOADng, duty cycle also increases in proportion to the number of senders. Duty cycle, calculated as a percentage of the time the radio is turned on during the entire operating time, is directly related to energy consumption. If the network's size becomes very huge, LOADng will not be able to avoid large energy consumption.

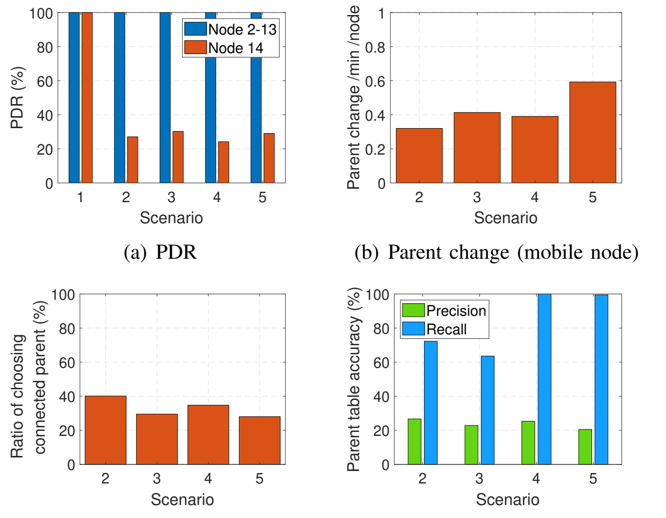
Overall, LOADng's performance in a static LLN is very bad. Although it is designed to support mobility, it is not an alternative to RPL as it cannot support many data senders. Its performance degrades as the number of data senders increases. Given that RPL performs well on static LLNs, it is worth improving RPL to support mobile LLNs.

B. Mobile Scenario: RPL's Problems

We now focus on RPL and investigate how it behaves in a mobile LLN. To do this, we perform simulations by using the Cooja simulator with the 14-node topology depicted in Fig. 3. To focus on the routing layer's behavior, we use NullRDC, an always-on link layer implementation on Contiki OS. Only upward packets are transmitted to facilitate analysis.

TABLE I
SIMULATION SCENARIOS FOR EVALUATING THE PERFORMANCE OF RPL IN MOBILE SCENARIOS.

Scenario	Node 14	Data interval (s)	DIO interval (s)
1	Static	30	4-1048
2	Mobile	30	4-1048
3	Mobile	6	4-1048
4	Mobile	30	1-4
5	Mobile	6	1-4



(a) PDR (b) Parent change (mobile node)

(c) Accuracy of routing decision (mobile node) (d) Accuracy of parent table (mobile node)

Fig. 4. Performance of MRHOF-based RPL in the simulation scenarios described in Fig. 3 and Table I.

We perform simulations on five different scenarios, as shown in Table I. Scenario 1 sets all nodes to be static, which serves as a ground result. In the other four scenarios, a mobile node (node 14) moves along the line illustrated in Fig. 3 at a speed of 1 m/s. We conducted the same simulation for the speeds of 0.5 m/s, 2 m/s, and 5 m/s, but the experimental results were similar to that for 1 m/s. In all the scenarios, all the 13 nodes except the root send upward packets. All the scenarios allow the mobile node to participate in packet forwarding. Both data and DIO intervals affect the routing performance of RPL. Therefore, we conduct a simulation with various parameter values. Scenarios 1 and 2 have the default configuration. Scenarios 3 and 4 decrease data and DIO intervals of not only the mobile node but also static nodes, respectively. Scenario 4 decreases both data and DIO intervals of static and mobile nodes.

Fig. 4 plots the simulation results (the average of five repetitive experiments with different random seeds). As Fig. 4(a) shows, all 14 static nodes show the PDR of nearly 100% in Scenario 1, meaning that RPL operates well in static LLNs. However, in all the other scenarios, while the static nodes have PDR close to 100%, the PDR of the mobile node plunges down to below 40% regardless of DIO and data intervals. This implies that RPL's problem in mobile scenarios is not simply about parameter settings but something more fundamental.

For a mobile node to communicate well, it is necessary to change the (communication) route as it moves. As seen in Fig. 4(b), however, reducing data or DIO interval does not sufficiently increase the number of parent changes[5]. This means that RPL does not detect its need for parent change. Even in scenario 4, where the number of parent changes increases the most, PDR is not improved significantly. RPL's efforts to update routes do not end up working as intended.

We further analyze the simulation results to reveal why RPL's path update mechanisms do not operate well in mobile LLNs. We examine the ratio at which the mobile node selects a new preferred parent that is actually connected when changing its preferred parent. Fig. 4(c) shows that the ratio is below 40%. The mobile node significantly misunderstands its environment, identifying a disconnected neighbor as a valid parent candidate. This implies that the mobile node's parent table may not be managed timely.

To confirm this, we measure the average precision (the ratio of actually connected nodes among the nodes regarded to be connected in the parent table) and recall (the ratio of nodes considered to be connected in the parent table among the actually connected parents) of the mobile node's parent table measured at the moment the parent node changes. Fig. 4(d) shows that recall becomes high when DIO interval is low (i.e., scenarios 4 and 5), meaning that the mobile node fast discovers new parents. On the other hand, precision is lower than 30% in all the cases regardless of parameter settings, confirming that the mobile node misunderstands that a disconnected parent is connected. The low precision incurs faulty parent changes, resulting in performance degradation.

Problem Analysis: Looking into the RPL design together with the experimental results, we have found the three problems in RPL as below, which motivates us to design *MobiRPL*.

- **Slow Link Quality Update:** ETX in RPL is easily outdated, which is not suitable for mobile LLNs: (1) ETX is a *statistical* metric, which is slowly updated and cannot detect quick changes in link connectivity in mobile environments. (2) A node updates ETX for a neighbor only after sending a unicast packet to the neighbor. Given that RPL sends upward data traffic only to the preferred parent, ETX for a *non-preferred parent* becomes outdated. (3) Even ETX for the preferred parent can be outdated depending on upward packet interval. When upward packet interval is too long compared to mobility, a mobile node can misunderstand that its outdated preferred parent is still valid, losing many upward data packets in the air.
- **Rough Link Quality Representation:** Even when ETX is completely up-to-date, it has an inherent limit in design: Representing link quality in terms of packet transmission reliability (i.e., link PDR). Given that PDR does not decrease linearly with RSSI but suddenly drops from $>$ 90 percent to $<$ 10 percent at a certain RSSI threshold (e.g., –87 dBm) [53], ETX can finely distinguish link quality around that threshold. However, *it cannot distinguish a very robust link from possibly fragile links*. For example, when choosing an upward route, two candidate links may have the current ETX of 1 (best quality), but one link has an RSSI of –60 dBm and the other has –85 dBm. Although

[5]We will briefly call a change of a preferred parent a parent change.

the –85 dBm candidate link currently has good reliability, it is likely to become bad (below –87 dBm) in the near future due to mobility. The –60 dBm candidate link is more robust to mobility, which cannot be identified by ETX.
- **Lack of Connectivity Management:** RPL, as a routing protocol instead of a neighbor management protocol, does not have an explicit mechanism to manage connectivity with neighbors. Although a disconnected neighbor may have a bad ETX value, it can be still in the parent table as a *valid* parent candidate and selected as the preferred parent. For example, under MRHOF, a node with a high ETX value can be selected as the preferred parent if it has a very low Rank.

IV. Design Requirements

We summarize the requirements for *MobiRPL* to support mobility as follows.

- *MobiRPL* should update the link quality of the preferred parent frequently enough to timely detect its disconnection, regardless of the data interval.
- *MobiRPL* should determine connectivity with all known parents and avoid choosing a disconnected parent as its preferred parent.
- *MobiRPL* should discover new potential parents fast and efficiently when needed.
- *MobiRPL* should have a new objective function that is more suitable for mobility support than ETX-based MRHOF. It should give preference to robust links over possibly fragile links, regardless of current packet delivery performance.

In addition to meeting these requirements to improve reliability in mobile LLNs, energy efficiency should be also considered since *MobiRPL* runs on energy-constrained embedded devices. In the considered scenario, since many nodes are still *static*, it can be overkill to put a significant effort into quickly updating the link quality for all nodes. For example, if a static node has a static preferred parent, it does not have to frequently update the link quality of the preferred parent. In this case, saving energy would be a better choice. To provide energy-efficient operation for static nodes while improving reliability for mobile nodes, *MobiRPL* should detect the mobility of each node and treat mobile nodes differently from static nodes. Therefore, we add one more requirement for *MobiRPL* design:

- *MobiRPL* should detect the mobility of each node and differentiate between static and mobile nodes.

V. MobiRPL Design

In this section, we design *MobiRPL* to satisfy the above five requirements of RPL in detail. *MobiRPL* introduces three new mechanisms: (1) *Mobility detection*, (2) *connectivity management*, and (3) *RSSI and hop distance-based objective function*, as shown in Fig. 5.

A. Mobility Detection

Our first mechanism, *mobility detection*, enables *MobiRPL* to determine the node's mobility.[6] We let *MobiRPL* detect mobility from the average interval of parent changes: Mobility increases as the parent change interval decreases. Our intuition is that mobile nodes should be able to change their preferred parent more often than static nodes. Given that RPL is designed to make static nodes rarely change preferred parents, this interval of parent change would be significantly large for static nodes while small for mobile nodes. Therefore, by setting a threshold ($t_{c,\mathrm{thr}}$) large enough, we can distinguish most mobile nodes from static nodes.

Specifically, we use the exponentially weighted moving average (EWMA) of the parent change interval to avoid misjudgment from the temporary parent change caused by network fluctuations other than mobility. Let t_c and $\overline{t_c}$ denote the parent change interval and EWMA value of t_c, respectively, and α be a coefficient between 0 and 1. When the i-th parent change occurs, *MobiRPL* calculates the i-th EWMA value ($\overline{t_{c,i}}$) from the previous EWMA value ($\overline{t_{c,i-1}}$) and the newly measured parent change interval ($t_{c,i}$) as

$$\overline{t_{c,i}} = \alpha \cdot \overline{t_{c,i-1}} + (1-\alpha) \cdot t_{c,i}. \tag{1}$$

It is then intuitive for *MobiRPL* to classify a node as a static node if its $\overline{t_{c,i}}$ is greater than or equal to a threshold ($t_{c,\mathrm{thr}}$), or as a mobile node otherwise.

For static nodes, however, $\overline{t_c}$ may not be updated timely. Specifically, when $t_{c,i}$ is very long due to stable link quality (e.g., more than an hour), an update from $\overline{t_{c,i-1}}$ to $\overline{t_{c,i}}$ is significantly delayed. For example, a newly installed static node may frequently change its preferred parent i times (i.e., low $\overline{t_{c,i-1}}$ value) and settle in one preferred parent (i.e., very long $t_{c,i}$). In this case, $\overline{t_c}$ remains small for a long time, resulting in misclassification of the static node as a mobile node. To alleviate such a problem, we define another average value, $\overline{t_m}$, and use it as the *mobility metric* for *MobiRPL*, rather than $\overline{t_c}$.

MobiRPL calculates $\overline{t_m}$ in two cases. As exemplified in Fig. 6, when the i-th parent change occurs, *MobiRPL* calculates $\overline{t^0_{m,i}}$ as (using the newly calculated $\overline{t_{c,i}}$)

$$\overline{t^0_{m,i}} = \overline{t_{c,i}}. \tag{2}$$

Even without the parent change, every moment when $\overline{t^j_{m,i}}$ has passed since the last calculation of $\overline{t^j_{m,i}}$, *MobiRPL* calculates $\overline{t^{j+1}_{m,i}}$ as

$$\overline{t^{j+1}_{m,i}} = \alpha \cdot \overline{t_{c,i}} + (1-\alpha) \cdot \sum_{k=0}^{j} \overline{t^k_{m,i}}, \tag{3}$$

and uses $\overline{t^{j+1}_{m,i}}$ as $\overline{t_m}$ for the moment. Then, *MobiRPL* compares $\overline{t_m}$ with a predetermined threshold $t_{c,\mathrm{thr}}$ to determine mobility. In this way, static nodes can fast increase $\overline{t_m}$ even without their parent changes, avoiding misclassification.

[6]If additional hardware components, such as accelerometer, are allowed, this mechanism can be replaced. This mechanism is to enable mobility support regardless of such external components.

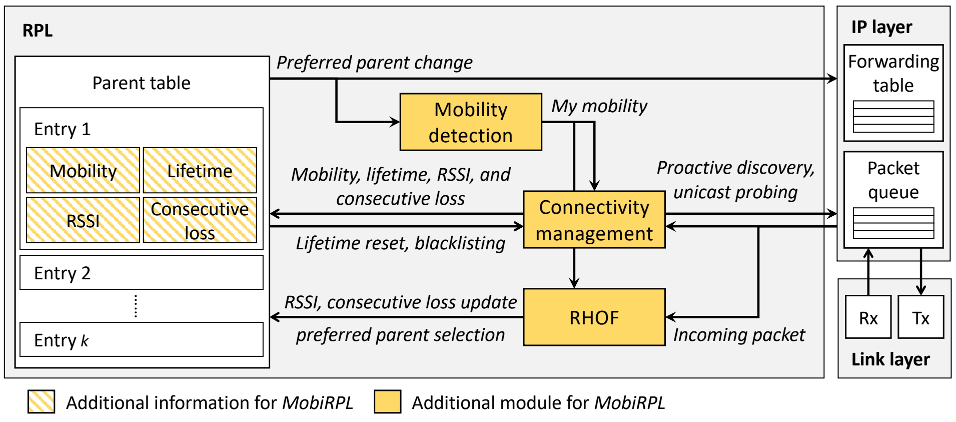

RPL
Parent table
Entry 1
Mobility
Lifetime
RSSI
Consecutive loss
Entry 2
Entry k
Preferred parent change
Mobility detection
My mobility
Mobility, lifetime, RSSI, and consecutive loss
Proactive discovery, unicast probing
Connectivity management
Lifetime reset, blacklisting
RSSI, consecutive loss update
preferred parent selection
RHOF
Incoming packet
IP layer
Forwarding table
Packet queue
Rx
Tx
Link layer
Additional information for MobiRPL
Additional module for MobiRPL

Fig. 5. Overview of *MobiRPL* Design. We propose three new mechanisms (mobility detection, connectivity management, and RHOF) as a part of RPL. Our new mechanisms better cope with mobility by interacting with existing RPL operations and IP layer.

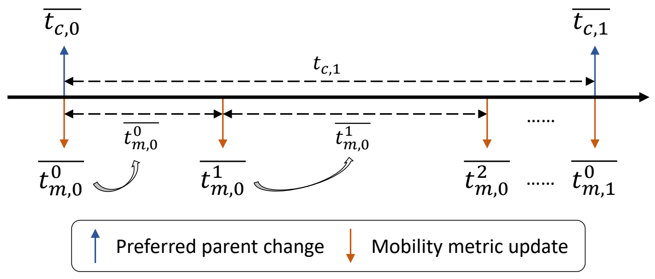


Fig. 6. *MobiRPL*'s mobility detection based on the parent change interval.

MobiRPL piggybacks the detected mobility in the reserved bits of the RPL control messages (e.g., DIO message) and advertises it on the neighboring nodes. This piggybacking enables each node to know the mobility of its neighboring nodes. Overall, the *mobility detection* mechanism enables a *MobiRPL* node to detect the mobility of itself and its neighbors. The other two mechanisms of *MobiRPL* utilize this mobility information for routing and energy saving in mobile LLNs.

Our mobility detection relies on the average interval of parent changes ($\overline{t_m}$). Therefore, even if we set the threshold ($t_{c,\text{thr}}$) large enough, the speed or mobility patterns of mobile nodes can affect detection accuracy. For example, a mobile node that moves extremely slowly and does not change its preferred parent frequently may consider itself a static node. However, the other two mechanisms help *MobiRPL* resolve such exceptional cases. We will discuss more details later, but RHOF makes static nodes prefer static nodes in parent selection and rarely change their preferred parents. Therefore, the parent change interval of static nodes gradually increases, making it possible to distinguish between slow mobile nodes and static nodes. The connectivity management mechanism also helps *MobiRPL* find a valid routing path, even if mobility detection is temporarily inaccurate.

B. Connectivity Management

An explicit connectivity management mechanism is necessary for a *MobiRPL* node to timely include (or exclude) a new (or disconnected) parent in (or from) the parent table and to change the preferred parent accurately in mobile LLNs. At the same time, the connectivity management mechanism should balance between energy efficiency and timely operation. To this end, our *connectivity management* includes *adaptive timeout-based connectivity detection*, *adaptive probing*, and *proactive discovery*, which operate adaptively based on the result of *mobility detection*.

*1) **Adaptive Timeout:*** *MobiRPL* utilizes timeout-based connectivity detection. We let $t_{l,0}$ denote the timeout period (to be used as the initial value of the lifetime). Then, each *MobiRPL* node detects that its neighbor is disconnected if it cannot receive any packet from the neighbor during a timeout period $t_{l,0}$. *MobiRPL* excludes disconnected parents from the valid parent candidate set. The timeout period $t_{l,0}$ is a key parameter for timely and accurate connectivity management. If $t_{l,0}$ is too long, a *MobiRPL* node will consider a disconnected node as a valid parent, resulting in faulty parent changes due to the outdated information. On the other hand, if $t_{l,0}$ is too short (i.e., shorter than packet interval), a *MobiRPL* node will misunderstand that a connected node with a long packet interval is disconnected.

Then, what value should *MobiRPL* use as $t_{l,0}$? Considering that an RPL node periodically transmits DIO messages, the DIO interval can be used to configure the timeout period. If the timeout period is shorter than the DIO interval, *MobiRPL* may hastily mark a connected node as disconnected. Another factor is mobility since a mobile node should fast update its connectivity with each neighbor, which a static node does not have to do.

Considering these two factors, we define $t_{l,0}$ as

$$t_{l,0} = T_{DIO,\max} \cdot 2^{-m}, \quad 0 \le m \le M, \tag{4}$$

where $T_{DIO,\max}$ is the maximum DIO interval, and the exponent m is a control parameter. We design $t_{l,0}$ to be exponentially adjusted, given that the DIO interval is also exponentially adjusted by TrickleTimer. Specifically, a *MobiRPL* node updates the control parameter m when its *mobility detection* mechanism updates $\overline{t_m}$. Suppose the updated $\overline{t_m}$ classifies that the node is mobile. In that case, the node sets its parameter m to M (maximum), which minimizes the timeout period $t_{l,0}$ to the minimum timeout period $T_{l,\min}$ and enables the fastest connectivity updates right away. Otherwise, if the node is determined to be static, it decreases m by 1 (i.e., doubles $t_{l,0}$), gradually increasing $t_{l,0}$ toward $T_{DIO,\max}$.

Overall, this timeout period adaptation enables both mobile and static nodes to update their neighbors' connectivity, removing disconnected nodes from the parent table. Note that this packet reception-based connectivity detection does not incur any additional communication overhead. A caveat is that reducing the timeout period $t_{l,0}$ at a mobile node does not trigger any action at its neighbor nodes: The neighbors still send DIO with a long interval. Therefore, this adaptive timeout mechanism classifies a number of connected nodes as disconnected ones, reducing the number of valid parent candidates.

One way to resolve this tendency is to reduce the mobile node's neighbor nodes' DIO interval like [11]. However, reducing the DIO interval violates the default TrickleTimer operation of RPL. Furthermore, considering that the mobile node does not stay near a particular node continuously, reducing the DIO interval will benefit the mobile node only for a very short period of time. Reduced DIO interval may instead cause frequent unnecessary DIO transmissions in most cases. Therefore, instead of reducing the DIO interval, we propose adaptive probing and proactive discovery mechanisms.

*2) **Adaptive Probing**:* We design the adaptive probing mechanism to compensate for the adaptive timeout mechanism's defects described above. It would be good for a node to actively probe all the neighbors at least once within its timeout period in terms of accuracy. However, this aggressive approach incurs not only network congestion but also significant energy consumption at mobile nodes as the timeout period decreases. The adaptive probing mechanism in *MobiRPL* actively probes *only the preferred parent* to balance between energy efficiency and accuracy. Our intuition behind this design choice is that hastily determining the preferred parent as a disconnected node triggers unnecessary parent changes, degrading network performance. In contrast, misclassifying connected *non-preferred* parents as disconnected ones would be relatively fine.

Specifically, our probing mechanism sends N unicast packets to the preferred parent within the timeout period $t_{l,0}$ to check connectivity. This means that a *MobiRPL* node detects the preferred parent's disconnection when it fails to send N packets consecutively. To this end, the probing interval, t_p, is calculated as

$$t_p = t_{l,0}/(N+1). \tag{5}$$

If a data transmission has been recently performed, the next probing is skipped to reduce control overhead. Fig. 7 exemplifies this adaptive probing operation along with the adaptive timeout operation introduced above.

MobiRPL can use any unicast RPL control messages (e.g., DIS, DIO, and DAO) for adaptive probing. In the current

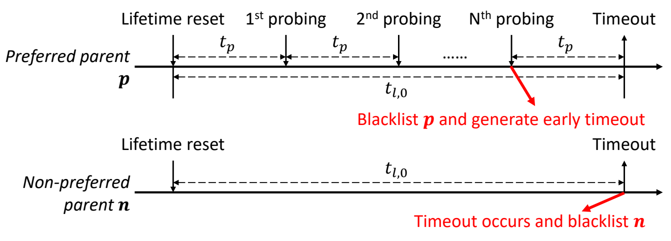


Fig. 7. *MobiRPL* 's connectivity management including adaptive timeout and adaptive probing.

implementation, *MobiRPL* uses unicast DIS messages for probing. Given that a unicast DIS is replied by both link-layer acknowledgment (ACK) and unicast DIO, it is possible to measure connectivity (link-layer information) and get the preferred parent's latest Rank (routing-layer information). However, it is also possible to limit DIO replies to reduce overhead. The most important thing is to get new RSSI information.

*3) **Proactive Discovery**:* The two mechanisms above, adaptive timeout and adaptive probing, check whether valid parent nodes in the parent table are still connected. This is to exclude a parent node from the parent table as soon as it is disconnected. However, these two mechanisms do not discover potential new parents that are not in the parent table yet. Discovering new parents timely is necessary for a mobile node to change its preferred parent accurately. To this end, we design *proactive discovery* as the last piece of connection management.

Given that static nodes operate well with the standard RPL, *proactive discovery* is triggered only at a mobile node (indicated by *mobility detection*). Specifically, a mobile node triggers proactive discovery when its parent table does not have any parent with a robust link (white zone defined by RHOF in Section V-C). In such a situation, with the current parent table entries, the mobile node would suffer fragile link connectivity no matter which parent it selects as the preferred parent. Instead of selecting the best (fragile) node as the preferred parent, discovering if there are new parent candidates will help accurate parent change.

When *proactive discovery* is triggered, *MobiRPL* broadcasts a single DIS message to request DIO messages from neighboring nodes. *MobiRPL* uses the reserved 1 bit of DIS message to indicate whether the DIS is for proactive discovery or not. This flagged DIS triggers two different operations at a DIS receiver compared to a normal DIS. (1) When a node receives a DIS sent for proactive discovery, it *selectively* responds to the DIS, only when it can be a parent of the DIS sender (i.e., its Rank is lower than or equal to the DIS sender's Rank). Note that the proactive discovery is for finding potential *parents*. (2) If the DIS receiver decides to respond, it immediately sends a DIO message but does *not reset the TrickleTimer*. This is because sending multiple DIOs to the mobile node does not add much value to its parent table. The mobile node keeps moving, and the information given by multiple DIOs will be outdated soon anyway. These two unique actions at a DIS receiver reduce communication overhead without sacrificing the accuracy of proactive discovery.

Overall, with the three components described so far, our *connectivity management* mechanism timely manages connectivity with low overhead in mobile LLNs. This mechanism maintains connectable routing paths more effectively, using the RSSI and hop distance-based objective function which will be described next.

C. RSSI and Hop Distance-based Objective Function

ETX is slowly updated and cannot distinguish a robust link from a potentially fragile link. To alleviate the problems, we introduce *RSSI and hop distance-based objective function* (RHOF) that utilizes the hop distance from the root for Rank and RSSI for the link quality metric.

*1) **Metric Choice**:* Hop distance, as pure routing-layer information, does not include link quality information at all. This is why end-to-end ETX, which includes *multi-hop* link cost from the preferred parent to the root, is used for Rank instead of hop distance in static LLNs. In mobile LLNs, however, link cost information in ETX is not reliable, even more so in the case of *multi-hop* link cost in end-to-end ETX (accumulated from the root). Thus in mobile LLNs, simply using hop distance without link quality information is more reliable than using inaccurate link quality information in end-to-end ETX. Therefore RHOF calculates Rank from hop distance as follows.

$$Rank(N) = Rank(P) + MinHopRankInc, \qquad (6)$$

where $MinHopRankInc$ is the minimum unit of Rank increase defined in RPL standard.

Although *MobiRPL* does not allow a node to know multi-hop link cost from its parent to the root, it tries to accurately identify one-hop link cost from the node to its parent, which is necessary for mobile LLNs. To this end, RHOF utilizes RSSI as the link quality metric. Although RSSI is a highly fluctuating metric, there have been a number of recent attempts to use RSSI as a link quality metric by taking advantage of its simplicity [25], [54], [55]. As a signal strength metric related to physical distance, RSSI can distinguish a robust link (e.g., -50 dBm) from a possibly fragile link (e.g., -85 dBm) regardless of the current transmission performance. Note that PDR can be 100% for both links. Moreover, updating RSSI does not require any unicast transmission; it is easily updated from receiving any packets (data, DIS, ACK, DIO, etc.). Since RSSI is not a statistical metric, it can be measured from a *single* packet reception. Thus, using RSSI enables to update link cost fast in mobile LLNs. In addition, we address the challenge of using RSSI, high fluctuation, as below.

*2) **Link Quality Classification**:* To utilize RSSI while mitigating its fluctuation, RHOF does not use raw RSSI values but link quality *zones*. Specifically, RHOF classifies neighboring nodes into three zones according to the lastly measured RSSI as shown in Table II. Inspired by Thread [54], a network protocol currently being actively used in the IoT domain, RHOF classifies nodes with RSSI above $RSSI_{\text{thr}}$ as a white zone. The nodes connected by the link with RSSI lower than the threshold are classified as grey zone. To handle RSSI fluctuation, RHOF considers hysteresis in comparing RSSI values. RHOF classifies the nodes that seem to be disconnected into a black zone. RHOF regards the link with N-consecutive packet losses as disconnected. The nodes determined as disconnected by *connectivity management* mechanism are also classified as the black zone. RHOF selects the best preferred parent among the parents in the parent table based on the Rank and the zone determined by RSSI.

TABLE II
RHOF WITH A CONTROLLABLE THRESHOLD AND A FIXED HYSTERESIS (4 DB).

Condition	Zone
Parents with higher RSSI than $RSSI_{\text{thr}}$	White zone
Parents with lower RSSI than $RSSI_{\text{thr}}$	Gray zone
Blacklisted parents	Black zone

TABLE III
RHOF PRIORITY CALCULATION FROM THE PERSPECTIVE OF A STATIC NODE AND A MOBILE NODE.

Static node perspective			Mobile node perspective		
Zone	Static	Mobile	Zone	Static	Mobile
White	1	3	White	1	2
Gray	2	4	Gray	3	4

*3) **Parent Selection**:* As discussed before, *MobiRPL* allows mobile nodes to participate in packet forwarding. However, static nodes can provide a more stable routing path than mobile nodes. Therefore, it is reasonable to prefer static nodes rather than mobile nodes in the best parent selection. We propose a simple yet effective method that makes static node be preferred in parent selection. This method does not explicitly distinguish the roles of the mobile and static nodes.

We use the mobility information broadcasted by *mobility detection* mechanism. RHOF calculates priority for each parent based on the mobility information and the measured RSSI. RHOF prefers parents with higher priority as preferred parent. Using this priority, RHOF can choose a preferred parent that is suitable for itself, taking into account the mobility of neighboring nodes as well as the Rank and RSSI. Table III shows how this priority is calculated. We note that the priority is differently calculated in static nodes and mobile nodes. This is because the preference according to the RSSI and the mobility is different between static nodes and mobile nodes. Static nodes can select static preferred parent that can reduce Rank even if RSSI is lower than the threshold. On the other hand, maintaining connectivity is essential for mobile nodes, so that mobile nodes must choose high RSSI preferred parent first.

RHOF never considers the nodes in the black zone as a preferred parent candidate. The blacklisted nodes are excluded from the parent change process until connectivity is confirmed through packet reception. Between the nodes not in the black zone, RHOF prioritizes nodes with a higher priority over nodes with a lower priority. The node with a smaller Rank is preferred among the nodes with the same priority. For nodes with the same priority and Rank, RHOF chooses the node with higher RSSI. Suppose the current preferred parent and the newly selected best parent have the same priority and Rank, and the difference between their RSSI values is smaller than the hysteresis. In that case, RHOF does not change the preferred parent and reduce routing overhead.

Because RHOF considers RSSI and Rank together in the preferred parent selection, sometimes child or descendant nodes can be seen as an attractive parent candidate. For example, if there is a very close child node that is classified as the white zone and all other nodes are in the gray zone, the child node might have a higher priority in preferred parent selection. However, choosing such a child node could make a routing loop occur. To prevent this, we add a "Rank filter" as a part of our RHOF. Rank filter allows a node to exclude the neighbor nodes with a Rank greater than or equal to itself from parent candidates. This filter makes RHOF consider only nodes that are not expected to be children or descendants.

Overall, RHOF makes *MobiRPL* choose the best preferred parent to maintain connectivity in a mobile scenario. The synergy between the three mechanisms introduced and the basic RPL operations enables *MobiRPL* to effectively perform data delivery even in mobile LLNs.

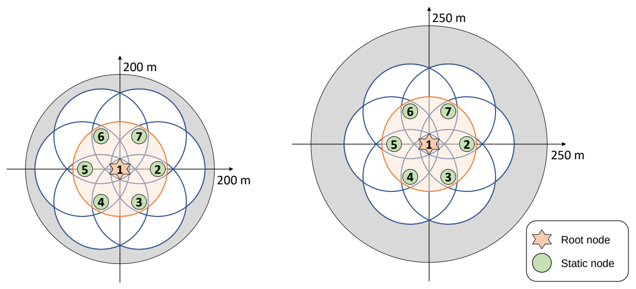
VI. Performance Evaluation

In this section, we evaluate *MobiRPL* on Cooja simulator and a real-world testbed. *MobiRPL* aims to improve the overall performance of RPL to support mobile nodes in non-hybrid LLNs. We, therefore, compare *MobiRPL* against the default RPL. In addition, we show that appropriately improved RPL may be more suitable for mobile LLNs than MANET routing protocols (e.g., LOADng). Therefore, we compare *MobiRPL* with LOADng in terms of various performance perspectives. We discuss the details of how well *MobiRPL* adapts to mobile LLN environments and achieves improved routing performance. We first examine the impact of *MobiRPL*'s mechanisms and parameters on performance. We then verify the performance of *MobiRPL* in more complex and diverse scenarios. In the following sections, if static nodes achieve PDRs of nearly 100%, we omit the plots for the PDRs of the static nodes.

A. Implementation and Evaluation Environments

We implement *MobiRPL* on Contiki OS version 3.0. Our implementation supports a TelosB-clone mote. As an underlying link layer, we use both always-on link layer (NullRDC) and ContikiMAC provided by Contiki OS. When applying ContikiMAC, we set the sleep interval as 31.25 ms (32 Hz channel check rate), to provide enough transmission chances in mobile scenarios. All the evaluation results are averaged over five repetitive experiments.

For Cooja simulation-based evaluations, we use three scenarios. The first scenario is identical to the scenario 2 in Fig. 3 and Table I. We will call this scenario *Cooja-1*. Fig. 8 shows the second and third scenarios of Cooja simulation-based evaluation. In these two scenarios of multiple mobile nodes, the root node is at the center, and six static nodes are located around the root in a regular hexagonal shape. The distance between two adjacent nodes is 40 m. We deploy up to eighteen mobile nodes around the root and static nodes.



(a) Simulation topology with a radius of 200 m (*Cooja-2*) (b) Simulation topology with a radius of 250 m (*Cooja-3*)

Fig. 8. Mobile LLN scenarios on the Cooja simulator with one root, 6 static nodes, and up to 18 mobile nodes. A small circle indicates each node's transmission range. The mobile nodes move within the outer circle following Random way-point model [56].

Mobile nodes move inside a circle around the root node. We set the radius of the circle where mobile nodes can move as 200 m and 250 m. All the nodes have the same transmission range (50 m).

There are shaded areas where mobile nodes cannot be connected to static nodes, which is represented as a gray area in Fig. 8. Mobile nodes independently move following Random way-point model [56] with the minimum and maximum speeds of 0.5 m/s and 2.0 m/s, respectively. We will name the scenarios with the radius of 200 m and 250 m as *Cooja-2* and *Cooja-3*, respectively. Using the *Cooja-2* and *Cooja-3* scenarios, we examine the performance of *MobiRPL* in situations where the number of mobile nodes is large. We also investigate whether the mobile node's participation in packet forwarding helps other mobile nodes maintain connectivity to the network when the static nodes cannot provide connectivity.

For the testbed-based evaluation, we use the testbed shown in Fig. 1. For evaluations including mobility, we add three mobile nodes to the testbed. We configure a mobile node using a model train, Raspberry-pi[7], portable battery[8], and TelosB-clone mote, as illustrated in Fig. 9(a). On the line drawn with arrows in Fig. 1, model train rails are installed. We installed model rails in various places such as corridors, classrooms, laboratories, supply rooms, warehouses. Many obstacles exist there, such as desks, chairs, PCs, wooden or steel shelves, trash cans, paper boxes, etc, which can disrupt communication or cause RSSI fluctuations. Fig. 9(b) is a picture of our mobility-augmented testbed taken at the corner where node 24 is located. Three mobile nodes travel back and forth along their corresponding lines at a speed of about 0.4 m/s.

Table IV summarizes the default experimental parameters. Unless explicitly stated in each experiment, we apply the default parameters in Table IV.

B. Impact of MobiRPL Mechanisms

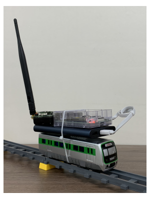
We first investigate the impact of *MobiRPL*'s mechanisms in a simple scenario, *Cooja-1*. The mobile node (node 14) moves

[7]We used Raspberry-pi 2 model B for logging real-time evaluation results.

[8]We used a portable battery named PLM09ZM, made by Xiaomi, whose capacity is 10,000 mAh. In our evaluation, the battery lasted about 26 hours.

TABLE IV
EVALUATION SETTINGS AND PARAMETERS.

Parameters	Testing environments	Values
$t_{c,\text{thr}}$	All	120 s
$T_{l,\text{min}}$	All	16 s
N	All	2
$RSSI_{\text{thr}}$	All	-83 dBm
$T_{DIO,\text{min}}$	All	4.096 s
$T_{DIO,\text{max}}$	All	1048.576 s
ContikiMAC channel check rate	All	32 Hz
Traffic pattern	Cooja-1 Cooja-2 and 3 Testbed	120 upward packets (1 packet / 30 seconds) 100 upward/downward packets (1 packet / 60 seconds) 120 upward/downward packets (1 packet / 60 seconds)
Mobile node speed	Cooja-1 Cooja-2 and 3 Testbed	1 m/s 0.5–2.0 m/s (Random way-point model) 0.4 m/s
Transmission power	Cooja-1, 2, and 3 Testbed	0 dBm -10 dBm with 5 dB antenna


(a) Mobile node

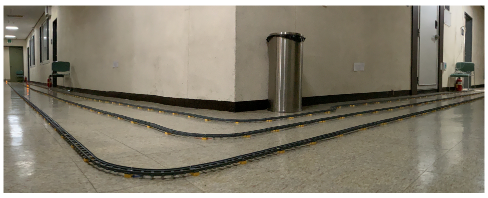

(b) Mobile node's travel path consisting of model rails

Fig. 9. Mobile node configuration and the travel path of mobile nodes. The model rail is installed in the path shown in Fig. 1, and the mobile node travels back and forth along the model rail.

TABLE V
COMBINATION CASES OF *MobiRPL* MECHANISMS FOR EVALUATING THE IMPACT OF *MobiRPL* MECHANISMS.

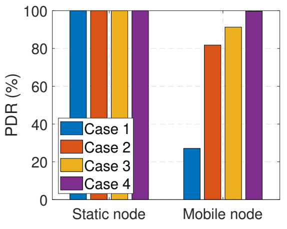
Case	Connectivity Management	RHOF	Proactive discovery
1	X	X	X
2	O	X	X
3	O	O	X
4	O	O	O

along the line illustrated in Fig. 3 at a speed of 1 m/s. We apply the always-on link layer (NullRDC) to this evaluation to concentrate on the behavior of mechanisms in mobile LLNs. We consider upward traffic only for ease of analysis. For each node, a total of 120 upward packets are transmitted to the root node every 30 seconds. The mobile node is allowed to participate in packet forwarding.

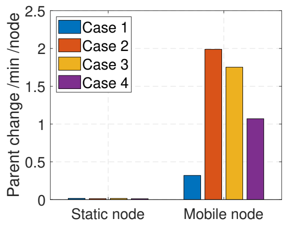
We measure various performance metrics while changing the combination of mechanisms applied. As shown in Table V, we examine four cases: The default RPL (case 1), RPL with connectivity management without proactive discovery (case 2), RPL with RHOF and connectivity management without proactive discovery (case 3), and *MobiRPL* (case 4). We apply the mechanism of mobility detection in all cases.

The minimum timeout period ($T_{l,\text{min}}$) is 16 s, and the number of probing (N) for adaptive probing is 2 (twice). The threshold of parent change interval for mobility detection ($t_{c,\text{thr}}$) is 120 s. The RSSI threshold ($RSSI_{\text{thr}}$) of RHOF is -83 dBm.

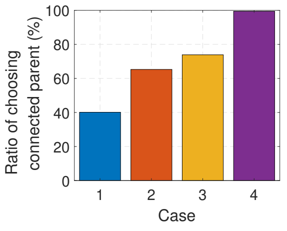
Fig. 10(a) shows the average upward PDR. While RPL



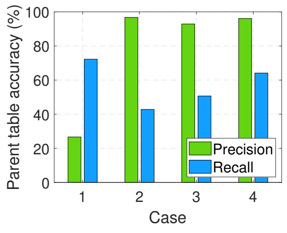
(a) PDR



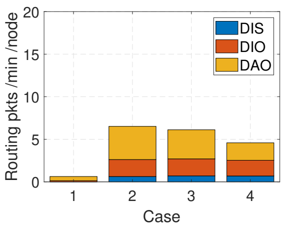
(b) Parent change (mobile node)



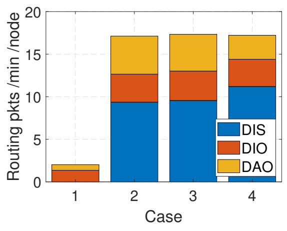
(c) Accuracy of routing decision (mobile node)



(d) Accuracy of parent table (mobile node)



(e) Routing overhead (static node)



(f) Routing overhead (mobile node)

Fig. 10. Performance of *MobiRPL* depending on the type of mechanisms applied. We test four cases described in Table V.

shows PDR lower than 30%, the connectivity management mechanism raises the PDR of mobile nodes to 80%. Other mechanisms, such as RHOF and proactive discovery, further increase the PDR of mobile nodes. The performance improvement comes from accurate routing decisions achieved by our proposed mechanisms. As presented in Fig. 10(c), *MobiRPL*'s mechanisms successfully improve the accuracy of routing decisions made by mobile nodes.

Fig. 10(d), which shows the average precision and recall of the mobile node's parent table measured when parent changes occur, accounts for this improved routing decision accuracy. The connectivity management mechanism dramatically improves the inferior precision of RPL by eliminating outdated parent entries from the parent table. Although RHOF shows a slightly lower precision than MRHOF, considering that the connectivity management mechanism aggressively removes the parent entries and the number of parents believed to be connected is reduced, the precision does not drop significantly. As proactive discovery is applied, the precision rises again.

Since the connectivity management mechanism deletes the parent entries aggressively, we can observe that the recall also drops. However, our RHOF and proactive discovery complement this decrease in recall. MRHOF changes the preferred parent when the link quality with the current preferred parent becomes very poor due to the nature of ETX. On the other hand, RHOF changes the preferred parent if a better preferred parent candidate exists, even if the link quality with the current preferred parent is not very bad. In other words, RHOF reacts more actively to the information added to the routing table than MRHOF. Hence, the recall measured at the moment of parent change also increases. Proactive discovery increases recall by adding new parents to the parent table.

This improved routing accuracy allows the mobile node to change its preferred parent more frequently and maintain connectivity. As seen in Fig. 10(b), *MobiRPL*'s mechanisms increase the number of parent changes made by the mobile node. We note that RHOF and proactive discovery reduces the number of parent changes while they increase PDR. This result confirms that RHOF and proactive discovery help the mobile node select the preferred parent with more robust connectivity, thus lowering the need to change the preferred parent.

Figs. 10(e) and 10(f) show the average routing overhead created by static nodes and mobile nodes, respectively. Compared to RPL, *MobiRPL*'s mechanisms generate more routing overhead. However, this increased routing overhead is not wasted, and it makes mobile nodes successfully increase PDR by maintaining connectivity. Applying RHOF and proactive discovery slightly decreases static nodes' routing overhead because mobile nodes operate well with fewer parent changes reducing the burden on static nodes.

C. Impact of MobiRPL Parameters

We now perform experiments with various *MobiRPL* parameters in *Cooja-1* (one mobile node) with the same evaluation settings used in Section VI-B. The most important two parameters in *MobiRPL* are the minimum timeout period ($T_{l,\min}$) and the number of times (N) probing is performed within the timeout period. The mobile node sets the timeout period of neighboring nodes to ($T_{l,\min}$). Therefore, $T_{l,\min}$ is directly related to how *MobiRPL* aggressively blacklists neighbor nodes. *MobiRPL* determines the link with N-consecutive packet losses as disconnected. Therefore, increasing N improves the accuracy of connectivity examination, but a larger N causes a greater delay in connectivity judgment.

We evaluate *MobiRPL* with different parameter settings as described in Table VI and plot the results in Fig. 11. Fig. 11(a) shows the average end-to-end PDR of the mobile node. When $T_{l,\min}$ is 16 s, except for the case where N is 1, the mobile node achieves the PDR close to 100%. Even when N is 1, the mobile node has a PDR of higher than 95%. If $T_{l,\min}$ is 32 s, the PDR is not close to 100% in all cases, but it approaches 100% as N increases. However, the PDR decreases as N increases when $T_{l,\min}$ is set to 65 s, and it never reaches 100%. As shown in Fig. 11(b), the routing decision accuracy accounts for these two opposite tendencies of PDR. Increasing N makes routing decisions accurate when $T_{l,\min}$ is 16 s or 32 s, but it degrades the accuracy when $T_{l,\min}$ is 65 s.

We then discuss why N affects routing accuracy and PDR differently according to $T_{l,\min}$. If $T_{l,\min}$ is set large, *MobiRPL* generates timeouts for non-preferred parents slowly. Considering that the probing interval for the preferred parent (t_p) is set proportionally to $T_{l,\min}$, using large $T_{l,\min}$ causes *MobiRPL* to take a longer time performing probing N times. Such a delay in timeout and probing can make the parent table of *MobiRPL* full of outdated entries. In this situation, increasing N causes probing operation to take more time, and *MobiRPL* cannot avoid wrong routing decisions. In short, if $T_{l,\min}$ is not set short enough, increasing N does not have any advantage other than accurately determining connectivity to the preferred parent. From the discussion so far, we can derive a design guideline for *MobiRPL* to set $T_{l,\min}$ and N. *MobiRPL* should have $T_{l,\min}$ value small enough to cope with mobility. If $T_{l,\min}$ is appropriately set, the PDR should increase as N increases.

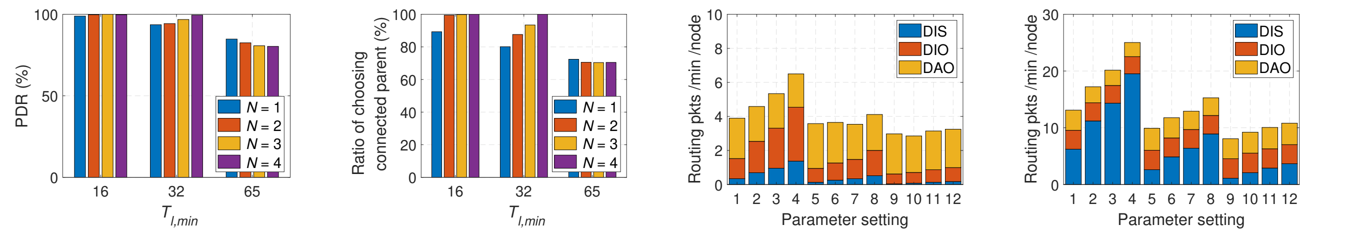
Figs. 11(c) and 11(d) show the routing overhead of *MobiRPL*. Using small $T_{l,\min}$ induces more routing overhead because it makes timeout and probing occur more frequently and *MobiRPL* perform more routing operations. Increasing N also causes more routing overhead because it makes *MobiRPL* perform more probings. Overall, $T_{l,\min}$ and N affect the PDR and the amount of routing overhead, which is directly related to the duty cycle of *MobiRPL* node and contention in the network. Therefore, $T_{l,\min}$ and N should be set appropriately to guarantee a high PDR with acceptable overhead.

Combining the discussions so far, although these two parameters' appropriate values are environment-dependent, it is possible to set universally applicable parameters in various environments. Without restrictions on network capacity and energy consumption, it is sufficient to set $T_{l,\min}$ very small (e.g., 0.1 seconds) and then set N to an appropriately large value (e.g., 4). If there are limitations to network capacity and energy consumption in the real world, we cannot set $T_{l,\min}$ and N this way. The device's transmission range is generally predetermined in the deployment phase. The mobile node's speed is given within a specific range (e.g., a person's walking speed is about 1 m/s). Given this, we can derive appropriate values for $T_{l,\min}$ and N in advance, which can meet the desired reliability and amount of overhead.

Considering all these, we set $T_{l,\min}$ to 16 s and N to 2 in the following evaluations. With these parameters, mobile nodes can determine non-preferred parents that do not have communication history for the last 16 seconds as disconnected. By setting N to 2, mobile nodes can examine connectivity with the preferred parent at least once every 5 seconds via probing. Furthermore, due to the additional use of messages such as DIO, the interval for verifying connectivity with the

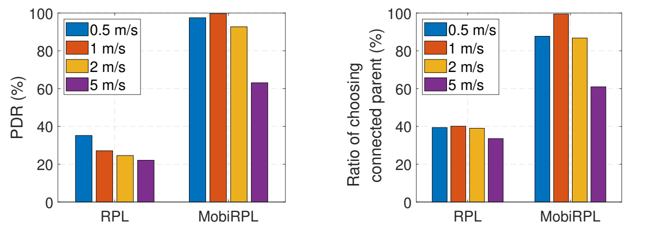
TABLE VI
PARAMETER SETTINGS FOR EVALUATING THE IMPACT OF *MobiRPL* PARAMETERS.

Parameter setting	$T_{l,\min}$ (s)	N	Parameter setting	$T_{l,\min}$ (s)	N	Parameter setting	$T_{l,\min}$ (s)	N
1	16	1	5	32	1	9	65	1
2	16	2	6	32	2	10	65	2
3	16	3	7	32	3	11	65	3
4	16	4	8	32	4	12	65	4



(a) PDR (mobile node) (b) Accuracy of routing decision (mobile node) (c) Routing overhead (static node) (d) Routing overhead (mobile node)

Fig. 11. Performance of *MobiRPL* depending on various parameter settings described in Table VI. For three different minimum timeout period values ($T_{l,\min}$) of 16, 32, 65 seconds, we evaluate *MobiRPL* while varying the number of times probing is performed (N) from 1 to 4.



(a) PDR (mobile node) (b) Accuracy of routing decision (mobile node)

Fig. 12. Performance of *MobiRPL* depending on the speed of the mobile node. For four different speeds of 0.5, 1, 2, 5 m/s, we evaluate *MobiRPL*.

preferred parent node becomes less than 5 seconds. Thus, even if a mobile node selects a disconnected node as a new preferred parent, it can quickly check real connectivity, enabling connecting with other parents again.

D. Impact of Circumstance Parameters

We now set $T_{l,\min}$ and N to 16 s and 2, respectively. In *Cooja-1*, and with the same evaluation settings applied in Section VI-B, we perform experiments while varying circumstance parameters, i.e., the speed of the mobile node to 0.5, 1, 2, and 5 m/s. Fig. 12 plots the result. As shown in Fig. 12(a), at all speeds, *MobiRPL* outperforms RPL in PDR. When the speed is 0.5, 1, and 2 m/s, *MobiRPL* achieves a PDR above 90%. At a speed of 5 m/s, which is much faster than the speed we consider, the PDR of *MobiRPL* drops to around 60%, but it is still 40% higher than RPL.

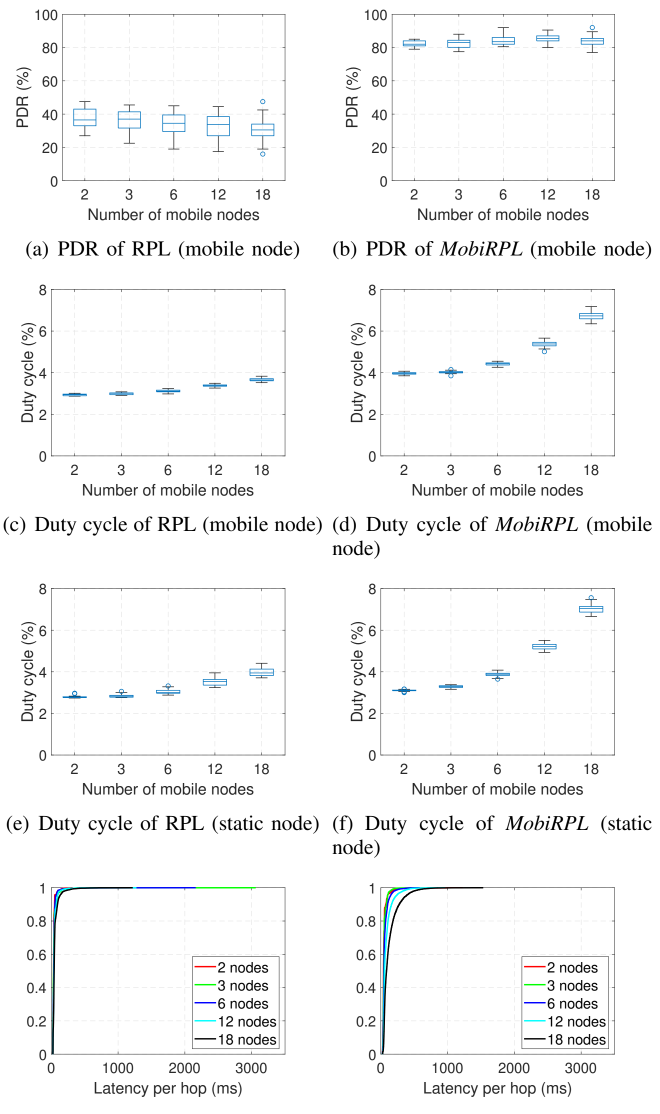
As shown in Fig. 12(b), the PDR improvement comes from accurate routing decisions achieved by *MobiRPL*. At the speed of 1 m/s, *MobiRPL* shows the best routing decision accuracy and PDR. At 0.5 m/s, *MobiRPL*'s routing decision accuracy decreases slightly. The topology setting of *Cooja-1* accounts for this difference. The threshold of the parent change interval for mobility detection ($t_{c,\text{thr}}$) is 120 s. If the mobile node moves at 0.5 m/s in the current topology, changing the preferred parent sometimes takes longer than 120 s, leading to inaccurate mobility detection and some decline in routing decision accuracy.

When the mobile node does not move too slowly (e.g., faster than 0.5 m/s), it correctly detects its mobility most of the time. However, if the mobile node moves quickly (e.g., 2 m/s), the routing decision accuracy can degrade due to the proactive nature of *MobiRPL*. Nevertheless, as shown in Fig. 12(a), *MobiRPL* overcomes deterioration in routing decision accuracy and achieve high PDR through its connectivity management mechanism. *MobiRPL* still surpasses RPL even if the mobile node moves very fast (e.g., 5 m/s), but more appropriate parameters may be required for *MobiRPL* to perform better at such a rapid speed.

RPL shows poor PDR and routing decision accuracy regardless of the mobile node's speed. We found that no matter the mobile node's speed, once outdated routing information fills the mobile node's parent table, RPL begins to make wrong routing decisions repeatedly. RPL shows low PDR at all speeds because it cannot make accurate routing decisions.

E. Performance of MobiRPL in Complicated Scenarios

From now on, we evaluate *MobiRPL* in more complicated scenarios, *Cooja-2* and *Cooja-3*. The underlying link layer is ContikiMAC (with a channel check rate of 32 Hz) in both scenarios. We consider a bidirectional traffic scenario. All the nodes, including static and mobile nodes, transmit one upward packet and one downward packet every 60 s (100 upward packets and 100 downward packets in total). In these two scenarios, static nodes cannot cover all the areas; thus, there is a shaded area. We first examine the impact of the number of mobile nodes, in *Cooja-2*. We then evaluate the impact of allowing mobile nodes to participate in routing, in both *Cooja-2* and *Cooja-3* scenarios. We use boxplots to plot and analyze the performance of all individual nodes.



(a) PDR of RPL (mobile node) (b) PDR of *MobiRPL* (mobile node)

(c) Duty cycle of RPL (mobile node) (d) Duty cycle of *MobiRPL* (mobile node)

(e) Duty cycle of RPL (static node) (f) Duty cycle of *MobiRPL* (static node)

(g) Latency of RPL (mobile node) (h) Latency of *MobiRPL* (mobile node)

Fig. 13. Performance of *MobiRPL* compared to RPL varying the number of mobile nodes in *Cooja-2*.

*1) **Impact of the Number of Mobile Nodes:*** We now test the impact of the number of mobile nodes. To this end, in *Cooja-2*, we evaluate the performance of RPL and *MobiRPL*, changing the number of mobile nodes to 2, 3, 6, 12, and 18. There are six static nodes (excluding the root node) in *Cooja-2*; thus, the ratio of mobile nodes to static nodes varies by 1/3, 1/2, 1, 2, and 3, respectively. In addition, we measure duty cycle to compare energy consumption. We also examine per-hop latency.

Fig. 13 plots the performance of RPL and *MobiRPL*. Figs. 13(a) and 13(b) show the PDR of mobile nodes in RPL and *MobiRPL*, respectively. While RPL always shows PDR lower than 50%, *MobiRPL* achieves PDR around 80%. Interestingly, *MobiRPL*'s PDR even increases with the number of mobile nodes. The positive impact of mobile nodes in *MobiRPL* is because *MobiRPL* makes mobile nodes participate in packet forwarding timely and effectively. Given that a node in the shaded area can deliver its packets only through other mobile nodes, more mobile nodes in *MobiRPL* cause more potential forwarders for the nodes in the shaded area. In contrast, RPL cannot timely update routes with mobile nodes, resulting in lower PDR in the presence of more mobile nodes; mobile nodes cause nothing but chaos in RPL.

Figs. 13(c), 13(d), 13(e) and 13(f) show the duty cycle of static and mobile nodes in RPL and *MobiRPL*. In all the cases, *MobiRPL* shows a higher duty cycle than RPL since it generates more control packets to maintain connectivity. However, in exchange for increased energy consumption, *MobiRPL* achieves much higher PDR compared to RPL.

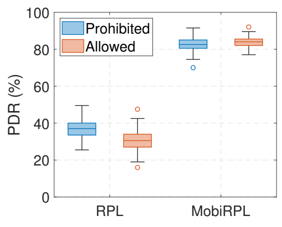
Lastly, Figs. 13(g) and 13(h) show the average per-hop latency among the packets successfully delivered to the destination node. The results show that *MobiRPL* delivers twice as many packets as RPL with slightly increased latency; it saves many packets by using more time for proper routing. It is important to note that low latency in RPL does not mean that it is effective in mobile LLNs but that it delivers packets only from the nodes *nearby* the root. Moreover, due to its effective mobile routing, the maximum latency in *MobiRPL* is much shorter than that in RPL, meaning that *MobiRPL* is not likely to make packets wander in the network. We note that this latency will vary according to the underlying link layer protocol.

*2) **Impact of Allowing Mobile Nodes to Participate in Routing:*** From now on, we evaluate *MobiRPL* in *Cooja-2* and *Cooja-3*. We set the number of mobile nodes to 18, and test the performance of *MobiRPL* and RPL. We designed *MobiRPL* to allow mobile nodes to participate in packet forwarding, assuming this will improve mobile nodes' packet delivery in mobile LLNs. To examine this, we simulate two cases: 1) Mobile nodes are not allowed to participate in packet forwarding, and 2) mobile nodes participate in packet forwarding. In case 1, a mobile node operates as a leaf node that does not generate multicast DIO messages and does not become the preferred parent of other nodes.

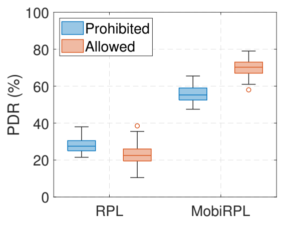
Fig. 14 plots the performance of RPL and *MobiRPL*. Figs. 14(a) and 14(b) show the PDR of mobile nodes in *Cooja-2* and *Cooja-3*. In both scenarios, regardless of whether packet forwarding of mobile nodes is prohibited or not, *MobiRPL* successfully increases the PDR by two to three times compared to RPL. *Cooja-3* has a wider shaded area than *Cooja-2*, and it shows lower PDRs. However, *MobiRPL* still shows higher PDR than RPL.

The PDR of RPL decreases if mobile nodes participate in packet forwarding due to its ineffective mobile routing. In *MobiRPL*, however, the participation of mobile nodes in packet forwarding improves the PDR. This is because a mobile node that moves into the shaded area can acquire connectivity to the root node with the help of other mobile nodes. The performance improvement is more significant in *Cooja-3* than *Cooja-2*; using mobile nodes for packet forwarding becomes more useful as the shaded area becomes broader.

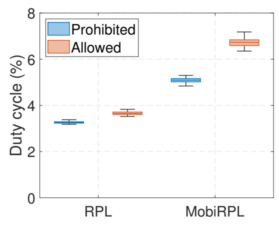
We now analyze energy consumption. Figs. 14(c), 14(d), 14(e), and 14(f) show the duty cycle of RPL and *MobiRPL* in the both scenarios. In the same scenarios, for both static and mobile nodes, *MobiRPL* shows a higher duty cycle than RPL



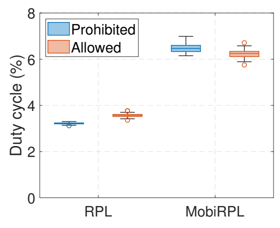
(a) PDR in *Cooja-2* (mobile node)



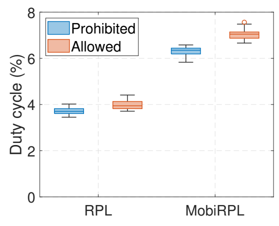
(b) PDR in *Cooja-3* (mobile node)



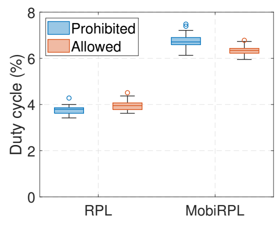
(c) Duty cycle in *Cooja-2* (mobile node)



(d) Duty cycle in *Cooja-3* (mobile node)



(e) Duty cycle in *Cooja-2* (static node)



(f) Duty cycle in *Cooja-3* (static node)

Fig. 14. Performance of *MobiRPL* compared to RPL in complicated scenarios (*Cooja-2* and *Cooja-3*).

since *MobiRPL* generates more control packets than RPL to maintain connectivity.

In *Cooja-2* (a narrow shaded area), if mobile nodes are allowed to forward packets, the duty cycles of static and mobile nodes increase in the both protocols due to the mobile nodes' routing and forwarding overheads. On the other hand, in *Cooja-3* (a broad shaded area), allowing packet forwarding of mobile nodes still increases the duty cycle of RPL, but decreases the duty cycle of *MobiRPL*. Despite mobile nodes' additional control overhead, *MobiRPL*'s timely management of mobile routes significantly reduces route repair overhead, resulting in lower duty cycle. In RPL, however, there is no benefit for mobile nodes to participate in packet forwarding.

Lastly, note that, as discussed in Section II, most of RPL-based mobile routing protocols prohibit mobile nodes from participating in packet forwarding [10]–[17]. Without careful design choices, allowing mobile nodes to forward packets can result in performance degradation as RPL. For example, some RPL-based mobile routing protocols allow mobile nodes to forward packets [20], [23]–[25], but exploits ETX-based MRHOF which is inappropriate for mobile LLNs as shown in Section VI-B. In contrast, the results show that our design choices for *MobiRPL* to allow mobile nodes' packet forwarding is effective.

F. Performance of MobiRPL in Real World

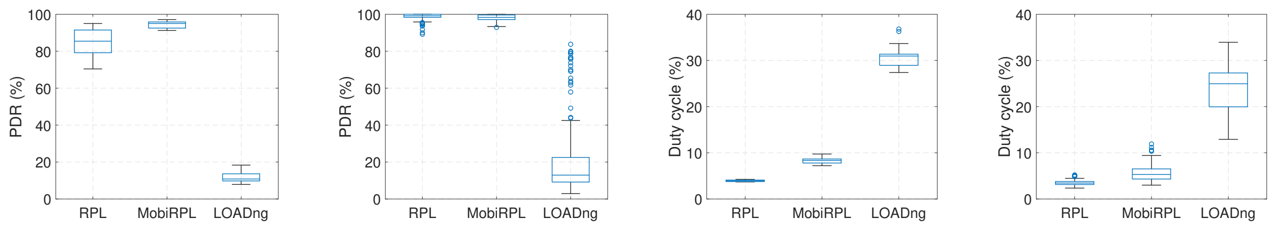
We evaluate *MobiRPL* on a real indoor testbed, the same as in Section III-A. There are 31 TelosB-clone *static* nodes, including one root node, as depicted in Fig. 1. Besides, three mobile nodes (nodes 32, 33, and 34) are deployed. Each node uses -10 dBm transmission power and an antenna of 5 dB gain. With various real-world obstacles, the testbed setting forms a 4-hop network where the communication range is 10–15 m, shorter than that in Cooja-based simulations (50 m). Considering that the mobile nodes move around at the speed of 0.4 m/s, we apply the same system parameters: $T_{l,\min} = 16$ s and $N = 2$. For the underlying link layer, ContikiMAC with a channel check rate of 32 Hz is used. We consider a bidirectional traffic scenario where all nodes generate one upward packet and one downward packet every 60 s (120 upward packets and 120 downward packets in total). We allow mobile nodes to participate in packet forwarding. We compare *MobiRPL* with RPL and LOADng in terms of PDR and duty cycle. Fig. 15 and Table VII show the results.

As can be seen in Figs. 15(a), and 15(b), although the mobile nodes' movement paths is simpler than those in the previous Cooja simulation scenarios, RPL provides significantly lower PDR for mobile nodes than static nodes. LOADng shows the lowest PDR because LOADng's flooding-based routing operation incurs severe congestion, preventing proper route discovery. On the other hand, *MobiRPL* stably provides high PDR both for mobile and static nodes.

For static nodes, *MobiRPL* provides slightly low PDR in the early stage since its *mobility detection mechanism* requires some time for each node to identify itself: Static or mobile node. Once static nodes identify themselves as static, however, they start to provide high PDR values, with a smaller deviation compared RPL. This is because *MobiRPL* utilizes the mobility type information, letting static nodes select other static nodes (robust paths) as preferred parents, instead of mobile nodes (fragile paths). Without mobility detection, RPL sometimes selects mobile nodes as preferred parents.

As presented in Figs. 15(c), and 15(d), compared to RPL, *MobiRPL* increases the duty cycle (energy consumption) of both static nodes and mobile nodes due to more control traffic (Table VII). This increase is larger than that observed in the simulations (*Cooja-1*, *Cooja-2*, and *Cooja-3*) because the topology in the testbed is much denser than that in the simulation environments, resulting in more control packets. However, the control traffic is needed to timely update mobile routes, resulting in more parent changes in *MobiRPL* than RPL as shown in Table VII. In addition, *MobiRPL*'s control traffic still low enough to deliver most data packets without congestion problems, which is verified by the high PDR in Figs. 15(a) and 15(b). Compared to LOADng that shows the worst duty cycle and PDR performance (see queue loss in Table VII) due to too much control traffic, *MobiRPL* provides a reasonable trade-off between control traffic and PDR performance.

We have observed that when a mobile node goes far away from the root node, it can be an attractive parent candidate even for static nodes since it has a smaller Rank compared to



(a) PDR (mobile node) (b) PDR (static node) (c) Duty cycle (mobile node) (d) Duty cycle (static node)

Fig. 15. Performance of *MobiRPL* compared to RPL and LOADng on a real world mobility-augmented testbed (Fig. 9).

TABLE VII
PERFORMANCE OF *MobiRPL* COMPARED TO RPL AND LOADNG.

Protocol	Node type	PDR (%)	Duty cycle (%)	Routing overhead (routing pkts /min /node)	Parent change (parent change /min /node)	Queue loss (queue loss /min /node)
RPL	Static node	98.63	3.47	0.64	0.03	0.00
	Mobile node	84.75	3.94	1.52	0.17	0.00
MobiRPL	Static node	98.06	5.58	13.07	0.51	0.01
	Mobile node	94.36	8.28	29.83	2.96	1.41
LOADng	Static node	22.01	23.88	146.71	-	21.17
	Mobile node	11.92	31.07	180.96	-	96.60

the nodes it will meet. If parent selection relies only on Rank, even static nodes will handover to mobile nodes. However, our RHOF chooses preferred parents by considering the priority derived from RSSI and mobility with Rank. Therefore, *MobiRPL* successfully prevents mobile nodes from becoming preferred parents of static nodes.

VII. DISCUSSION

There were three considerations for the design of *MobiRPL*: (1) It should operate in non-hybrid mobile LLNs, (2) it should operate with minimal assumptions and external mechanisms, and (3) it should operate proactively. While satisfying these three considerations, *MobiRPL* shows improved performance over RPL and LOADng, even at a speed of 2 m/s (similar to human movement), even when duty cycling is applied. However, *MobiRPL* did not completely solve all the problems with mobile LLNs. *MobiRPL* can perform better if some assumptions or external mechanisms are applied. For example, if an external localization method can accurately detect mobility, *MobiRPL* would be able to make more correct routing decisions.

Considering the proactive nature of *MobiRPL*, a slight deterioration in the accuracy of routing decisions is inevitable. For example, *MobiRPL*'s connectivity management mechanism can misclassify connectable parents as disconnected due to the aggressive timeout. However, the proactive discovery mechanism in *MobiRPL* can find new parent nodes before all parent nodes are blacklisted. As such, *MobiRPL* overcomes the inaccuracy of proactive routing through the cooperation between its mechanisms. At the same time, parameter settings are important in *MobiRPL*. Although we provide some guidelines to choose appropriate parameters, additional parameter tuning considering operation environments will be required for better energy efficiency.

Despite these limitations, we have shown that *MobiRPL* improves the performance of mobile LLNs as a stand-alone proactive routing protocol. The results from Cooja-2 and Cooja-3 demonstrate that *MobiRPL* operates effectively even in complicated situations. Besides, the results support our intuition that allowing mobile nodes' routing participation is helpful for mobile nodes to deliver more packets. According to the results from the testbed, we confirmed the capability of *MobiRPL* to cope well with network dynamics. Therefore, we believe that *MobiRPL* will perform well, even in large-sized random mobile LLNs. In addition, for mobile LLN routing protocols that require some assumptions and mechanisms, *MobiRPL* may provide basic connectivity as long as the requirements are satisfied.

As future work, we plan to investigate how to find the best *MobiRPL* parameter values according to the environment. We also plan to apply the state-of-the-art link layer protocol instead of the currently used ContikiMAC. We are considering time-slotted channel hopping (TSCH) [57] to test how TDMA link layer protocol affects the performance of *MobiRPL*. While resource allocation methods [58], [59] for TSCH protocols considering mobility have been proposed, combining TSCH with mobile routing protocols requires further research. Maintaining synchronization would be a challenge, but the application of TDMA link layer protocol would help lower contention and improve the performance of *MobiRPL*.

VIII. CONCLUSIONS

In this paper, we investigated the routing issues of mobile LLNs. In particular, we designed a routing protocol that operates well in a general mobile LLN, which uses duty cycling and has static and mobile nodes. In this scenario, we examined the performance of two representative routing protocols: RPL and LOADng, through experiments using an indoor testbed and Cooja simulator. As a result, we found that LOADng suffers severe performance degradation as the number of transmitting nodes increases due to its reactive operation. On the other hand, we found that RPL does not experience such a performance deterioration because of its proactive characteristics.

Through extensive experiments, we showed the reasons why RPL cannot support node mobility. Aiming to support node mobility while maintaining RPL's proactive characteristics in mobile LLNs, we designed a more general routing protocol named *MobiRPL*. *MobiRPL* includes three new mechanisms: *Mobility detection*, *connectivity management*, and *RSSI and hop distance-based objective function*. We implemented *MobiRPL* on Contiki OS and evaluated its performance through simulation and testbed evaluation. According to the evaluation results, we confirm that *MobiRPL* outperforms RPL in reliability and LOADng in energy efficiency. Our *MobiRPL* can be applied for more general and various mobile LLNs.

REFERENCES

[1] T. Winter, "RPL: IPv6 routing protocol for low-power and lossy networks," *Internet Eng. Task Force, RFC 6550*, Mar. 2012.

[2] H.-S. Kim, J. Ko, D. E. Culler, and J. Paek, "Challenging the IPv6 routing protocol for low-power and lossy networks (RPL): A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2502–2525, 2017.

[3] H.-S. Kim, H. Kim, J. Paek, and S. Bahk, "Load balancing under heavy traffic in RPL routing protocol for low power and lossy networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 4, pp. 964–979, 2016.

[4] H.-S. Kim, H. Cho, M.-S. Lee, J. Paek, J. Ko, and S. Bahk, "MarketNet: An asymmetric transmission power-based wireless system for managing e-Price tags in markets," in *Proc. ACM SenSys*, 2015.

[5] M. Dohler, T. Watteyne, T. Winter, and D. Barthel, "Routing requirements for urban low-power and lossy networks," *Internet Eng. Task Force, RFC 5548*, May 2009.

[6] K. Pister, P. Thubert, S. Dwars, and T. Phinney, "Industrial routing requirements in low-power and lossy networks," *Internet Eng. Task Force, RFC 5673*, Oct. 2009.

[7] A. Brandt, J. Buron, and G. Porcu, "Home automation routing requirements in low-power and lossy networks," *Internet Eng. Task Force, RFC 5826*, Apr. 2010.

[8] J. Martocci, P. D. Mil, N. Riou, and W. Vermeylen, "Building automation routing requirements in low-power and lossy networks," *Internet Eng. Task Force, RFC 5867*, June 2010.

[9] J. Ko, C. Lu, M. B. Srivastava, J. A. Stankovic, A. Terzis, and M. Welsh, "Wireless sensor networks for healthcare," *Proc. IEEE*, vol. 98, no. 11, pp. 1947–1960, 2010.

[10] M. Barcelo, A. Correa, J. L. Vicario, A. Morell, and X. Vilajosana, "Addressing mobility in RPL with position assisted metrics," *IEEE Sensors J.*, vol. 16, no. 7, pp. 2151–2161, 2015.

[11] C. Cobârzan, J. Montavont, and T. Noel, "Integrating mobility in RPL," in *Proc. EWSN*, 2015.

[12] J. Park, K.-H. Kim, and K. Kim, "An algorithm for timely transmission of solicitation messages in RPL for energy-efficient node mobility," *Sensors*, vol. 17, no. 4, p. 899, 2017.

[13] S. Hoghooghi and R. N. Esfahani, "Mobility-aware parent selection for routing protocol in wireless sensor networks using RPL," in *Proc. IEEE ICWR*, 2019.

[14] M. Bouaziz, A. Rachedi, A. Belghith, M. Berbineau, and S. Al-Ahmadi, "EMA-RPL: Energy and mobility aware routing for the Internet of mobile things," *Future Gener. Comput. Syst.*, vol. 97, pp. 247–258, 2019.

[15] M. Bouaziz, A. Rachedi, and A. Belghith, "EKF-MRPL: Advanced mobility support routing protocol for Internet of mobile things: Movement prediction approach," *Future Gener. Comput. Syst.*, vol. 93, pp. 822–832, 2019.

[16] G. Violettas, S. Petridou, and L. Mamatas, "Evolutionary software defined networking-inspired routing control strategies for the Internet of things," *IEEE Access*, vol. 7, pp. 132 173–132 192, 2019.

[17] I. Rabet *et al.*, "Pushing IoT mobility management to the edge: Granting RPL accurate localization and routing," in *Proc. IEEE WF-IoT*, 2021.

[18] R. Elhabyan, W. Shi, and M. St-Hilaire, "Coverage protocols for wireless sensor networks: Review and future directions," *J. Commun. Netw.*, vol. 21, no. 1, pp. 45–60, 2019.

[19] C. Zhu, C. Zheng, L. Shu, and G. Han, "A survey on coverage and connectivity issues in wireless sensor networks," *J. Netw. Comput. Applicat.*, vol. 35, no. 2, pp. 619–632, 2012.

[20] I. El Korbi, M. B. Brahim, C. Adjih, and L. A. Saidane, "Mobility enhanced RPL for wireless sensor networks," in *Proc. IEEE NOF*, 2012.

[21] H. Fotouhi, D. Moreira, and M. Alves, "mRPL: Boosting mobility in the Internet of things," *Ad Hoc Networks*, vol. 26, pp. 17–35, 2015.

[22] H. Fotouhi, D. Moreira, M. Alves, and P. M. Yomsi, "mRPL+: A mobility management framework in RPL/6LoWPAN," vol. 104. Elsevier, 2017, pp. 34–54.

[23] J. Ko and M. Chang, "MoMoRo: Providing mobility support for low-power wireless applications," *IEEE Syst. J.*, vol. 9, no. 2, pp. 585–594, 2015.

[24] O. Gaddour, A. Koubâa, and M. Abid, "Quality-of-service aware routing for static and mobile IPv6-based low-power and lossy sensor networks using RPL," *Ad Hoc Networks*, vol. 33, pp. 233–256, 2015.

[25] J. Wang, G. Chalhoub, and M. Misson, "Mobility support enhancement for RPL," in *Proc. IEEE PEMWN*, 2017.

[26] H. Kharrufa, H. Al-Kashoash, and A. H. Kemp, "A game theoretic optimization of RPL for mobile Internet of things applications," *IEEE Sensors J.*, vol. 18, no. 6, pp. 2520–2530, 2018.

[27] A. Mohammadsalehi *et al.*, "ARMOR: A reliable and mobility-aware RPL for mobile Internet of things infrastructures," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1503–1516, 2021.

[28] T. Clausen *et al.*, "The lightweight on-demand Ad hoc distance-vector routing protocol - Next generation (LOADng)," *Internet Eng. Task Force, Draft*, 2016.

[29] M. Vučinić, B. Tourancheau, and A. Duda, "Performance comparison of the RPL and LOADng routing protocols in a home automation scenario," in *Proc. IEEE WCNC*, 2013.

[30] U. Herberg and T. Clausen, "A comparative performance study of the routing protocols LOAD and RPL with bi-directional traffic in low-power and lossy networks (LLN)," in *Proc. ACM MSWiM*, 2011.

[31] S. Elyengui, R. Bouhouchi, and T. Ezzedine, "A comparative performance study of the routing protocols RPL, LOADng and LOADng-CTP with bidirectional traffic for AMI scenario," in *Proc. IEEE ICSGCE*, 2015.

[32] J. Yi, T. Clausen, and Y. Igarashi, "Evaluation of routing protocol for low power and lossy networks: LOADng and RPL," in *Proc. IEEE ICWISE*, 2013.

[33] J. Tripathi and J. C. de Oliveira, "Proactive versus reactive revisited: IPv6 routing for low power lossy networks," in *Proc. IEEE CISS*, 2013.

[34] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proc. IEEE LCN*, 2006.

[35] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing,"*Internet Eng. Task Force, RFC 3561*, 2003.

[36] P. Levis and T. H. Clausen, "The trickle algorithm," *Internet Eng. Task Force, RFC 6206*, Mar. 2011.

[37] O. Gnawali, "The minimum rank with hysteresis objective function," *Internet Eng. Task Force, RFC 6719*, Sep. 2012.

[38] A. Oliveira and T. Vazão, "Low-power and lossy networks under mobility: A survey," *Comput. Netw.*, vol. 107, pp. 339–352, 2016.

[39] P. O. Kamgueu, E. Nataf, and T. D. Ndie, "Survey on RPL enhancements: A focus on topology, security and mobility," *Comput. Commun.*, vol. 120, pp. 10–21, 2018.

[40] Z. Shah, A. Levula, K. Khurshid, J. Ahmed, I. Ullah, and S. Singh, "Routing protocols for mobile Internet of things (IoT): A survey on challenges and solutions," *Electronics*, vol. 10, no. 19, p. 2320, 2021.

[41] K. Levis *et al.*, "RSSI is under appreciated," in *Proc. EmNets*, 2006.

[42] S. Lin *et al.*, "ATPC: Adaptive transmission power control for wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 12, no. 1, p. 6, 2016.

[43] E. M. Royer *et al.*, "A review of current routing protocols for ad hoc mobile wireless networks." *IEEE Personal Commun.*, vol. 6, no. 2, pp. 46–55, 1999.

[44] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," *ACM SIGCOMM Comput. Commun. Review*, vol. 24, no. 4, pp. 234–244, 1994.

[45] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)," Tech. Rep. 2003.

[46] D. Johnson, Y.-c. Hu, and D. Maltz, "The dynamic source routing protocol (DSR) for mobile Ad Hoc networks for IPv4," Tech. Rep. 2007.

[47] J. Tripathi, J. C. De Oliveira, and J.-P. Vasseur, "Proactive versus reactive routing in low power and lossy networks: Performance analysis and scalability improvements," *Ad Hoc Netw.*, vol. 23, pp. 121–144, 2014.

[48] T. Clausen, J. Yi, and A. C. De Verdiere, "LOADng: Towards AODV version 2," in *Proc. IEEE VTC Fall*, 2012.

[49] T. Clausen, J. Yi, and U. Herberg, “Lightweight on-demand Ad hoc distance-vector routing-next generation (LOADng): Protocol, extension, and applicability,” *Comput. Netw.*, vol. 126, pp. 125–140, 2017.
[50] J. Yi and T. Clausen, “Collection tree extension of reactive routing protocol for low-power and lossy networks,” *Int. J. Distributed Sensor Netw.*, vol. 10, no. 3, p. 352421, 2014.
[51] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *Proc. IEEE LCN*, 2004.
[52] A. Dunkels, “The Contikimac radio duty cycling protocol,” 2011.
[53] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis, “An empirical study of low-power wireless,” *ACM Trans. Sensor Netw.*, vol. 6, no. 2, pp. 1–49, 2010.
[54] H.-S. Kim, S. Kumar, and D. E. Culler, “Thread/OpenThread: A compromise in low-power wireless multihop network architecture for the Internet of things,” *IEEE Commun. Mag.*, vol. 57, no. 7, pp. 55–61, 2019.
[55] S. Jeong, E. Park, D. Woo, H.-S. Kim, J. Paek, and S. Bahk, “MAPLE: Mobility support using asymmetric transmit power in low-power and lossy networks,” *J. Commun. Netw.*, vol. 20, no. 4, pp. 414–424, 2018.
[56] D. B. Johnson and D. A. Maltz, “Dynamic source routing in ad hoc wireless networks,” *Mobile computing*, 1996.
[57] “IEEE standard for local and metropolitan area networks–Part 15.4: Low-rate wireless personal area networks (LR-WPANs) amendment 1: MAC sublayer,” *IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, pp. 1–225, 2012.
[58] A. Elsts, J. Pope, X. Fafoutis, R. J. Piechocki, and G. Oikonomou, “Instant: A TSCH schedule for data collection from mobile nodes,” in *EWSN*, 2019, pp. 35–46.
[59] O. Tavallaie, J. Taheri, and A. Y. Zomaya, “Design and optimization of traffic-aware TSCH scheduling for mobile 6TiSCH networks,” in *Proc. ACM/IEEE IoTDI*, 2021.

Hongchan Kim (Student Member, IEEE) received the B.S. degree in Electrical Engineering from Seoul National University, in 2015. He is currently pursuing the Ph.D. degree with the School of Electrical and Computer Engineering, Seoul National University, Seoul, South Korea. His research interests include designing routing protocols for low-power networks and constructing the mobile IoT systems. He received the National Research Foundation (NRF) Global Ph.D. Fellowship, in 2016.

Hyung-sin Kim (Member, IEEE) received the B.S. degree in Electrical Engineering and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Science (EECS) from Seoul National University (SNU), Seoul, South Korea, in 2009, 2011, and 2016, respectively, all with outstanding thesis awards. He was a Postdoctoral Scholar at Network Laboratory (NETLAB), SNU, until August 2016 and Real-time, Intelligent, Secure, Explainable systems (RISELab), University of California, Berkeley, until August 2019, and a Software Engineer at Google Nest until February 2020. He received the Qualcomm Fellowship in 2011 and the National Research Foundation (NRF) Global Ph.D. Fellowship and Postdoctoral Fellowship in 2011 and 2016, respectively. He is currently an Assistant Professor at Graduate School of Data Science, SNU. His research interest includes Internet of Things and ambient artificial intelligence.

Saewoong Bahk (Senior Member, IEEE) received the B.S. and M.S. degrees in Electrical Engineering from Seoul National University (SNU), in 1984 and 1986, respectively, and the Ph.D. degree from the University of Pennsylvania, in 1991. He was with AT&T Bell Laboratories as a Member of Technical Staff from 1991 to 1994, where he had worked on Network Management. From 2009 to 2011, he served as Director of the Institute of New Media and Communications. He is currently a Professor at SNU. He has been leading many industrial projects on 5G/6G and IoT connectivity supported by Korean industry. He has published more than 300 technical articles and holds more than 100 patents. He is a member of the National Academy of Engineering of Korea (NAEK). He was a Recipient of the KICS Haedong Scholar Award, in 2012. He was President of the Korean Institute of Communications and Information Sciences (KICS) in 2020. He served as Chief Information Officer (CIO) of SNU from 2016 through 2021. He was General Chair of the IEEE WCNC 2020 (Wireless Communication and Networking Conference), IEEE ICCE 2020 (International Conference on Communications and Electronics), and IEEE DySPAN 2018 (Dynamic Spectrum Access and Networks). He was Director of the Asia–Pacific Region of the IEEE ComSoc. He is an Editor of the IEEE Network Magazine and IEEE Transactions on Vehicular Technology. He was TPC Chair of the IEEE VTC-Spring 2014, and General Chair of JCCI 2015, Co-Editor-in-Chief of the Journal of Communications and Networks (JCN), and on the Editorial Board of Computer Networks Journal and the IEEE Tran. on Wireless Communications.