공학박사 학위논문

Reliability and Throughput Enhancement in Low-power and Lossy Networks

저전력 무선 네트워크의 신뢰도 및 수율 개선

2023년 8월

서울대학교 대학원

전기정보공학부

김 홍 찬

공학박사 학위논문

Reliability and Throughput Enhancement in Low-power and Lossy Networks

저전력 무선 네트워크의 신뢰도 및 수율 개선

2023년 8월

서울대학교 대학원

전기정보공학부

김 홍 찬

Reliability and Throughput Enhancement in Low-power and Lossy Networks

지도 교수 박 세 웅

이 논문을 공학박사 학위논문으로 제출함

2023년 8월

서울대학교 대학원

전기정보공학부

김 홍 찬

김홍찬의 공학박사 학위 논문을 인준함

2023년 7월

위 원 장: 심 병 효 (인)

부위원장: 박 세 웅 (인)

위 원: 이 경 한 (인)

위 원: 김 형 신 (인)

위 원: 주 창 희 (인)

Abstract

This dissertation addresses various challenges in low-power and lossy networks (LLNs) by proposing innovative solutions that improve performance and efficiency. The study explores three key aspects: mobile routing, enhancing the throughput of time-slotted communication, and accelerating network formation in LLNs.

We first focus on mobile routing in LLNs. The existing *IPv6 routing protocol for LLNs* (RPL) lacks explicit support for mobility. To address this limitation, we design and implement an adaptive and robust mobile routing protocol called *MobiRPL*. *MobiRPL* utilizes the received signal strength indicator (RSSI) to enable efficient routing in mobile LLNs. Extensive evaluations through simulations and testbed experiments demonstrate the effectiveness of *MobiRPL*, with improvements in the packet delivery ratio by 11.3% compared to RPL and a 73.3% reduction in energy consumption when compared to the *lightweight on-demand ad-hoc distance-vector routing protocol - next generation* (LOADng).

We then target the throughput enhancement of time-slotted communications in LLNs, such as the IEEE 802.15.4e time-slotted channel hopping (TSCH). In many time-slotted systems, the slot length is typically fixed to a value suited for maximum-sized packets. Consequently, time slots can be underutilized when the packet length is shorter than the maximum, leading to residue time and throughput degradation. To overcome this inefficiency, we propose a utility-based adaptation of slot size and packet aggregation called *ASAP*. *ASAP* dynamically adjusts the slot length based on the packet size distribution and employs packet aggregation to maximize time-efficiency. Experimental evaluations using real embedded devices and large-scale testbeds demonstrate a significant improvement in throughput by 2.21x and a reduction in latency by 78.7% compared to the default time-slotted communication of TSCH.

Finally, we address the network formation process in *IPv6 over IEEE 802.15.4e*

TSCH mode (6TiSCH) network, the standard LLN. The formation of the 6TiSCH network often encounters severe collisions and congestion, leading to significant delays. We identify the root cause as the synchronized transmission timing among nodes within the time slot, rendering collision avoidance ineffective. To enhance the network formation efficiency, we propose a novel offset-based collision avoidance and prioritization method named *TOP*. This approach assigns different transmission offsets to packets, introducing diversity in their starting points. Such diversification facilitates collision detection and enables the prioritization of formation-critical packets. Real testbed experiments validate the effectiveness of *TOP*, demonstrating up to a 50% reduction in network formation time.

In aggregate, this dissertation presents novel approaches to address various challenges in LLNs, such as mobile routing, throughput enhancement of time-slotted communication, and network formation acceleration. The proposed solutions demonstrate significant improvements in performance, highlighting their potential to make LLNs more reliable and efficient.

keywords: Low-power and lossy network, Internet of Things, mobile routing protocol, reliability, throughput, latency, network formation

student number: 2015-22782

Contents

Abstract **i**

Contents **iii**

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**

1.1 Motivation 1

1.2 Contributions and Outline 2

2 MobiRPL: Adaptive, Robust, and RSSI-based Mobile Routing in Low Power and Lossy Networks **4**

2.1 Introduction 4

2.2 Background and Related Work 8

2.2.1 RPL and Mobility 8

2.2.2 LOADng, a MANET Protocol for LLN 13

2.3 Preliminary Study 14

2.3.1 Static Scenario: RPL vs. LOADng 14

2.3.2 Mobile Scenario: RPL's Problems 17

2.4 Design Requirements 21

2.5 MobiRPL Design 22

2.5.1 Mobility Detection . . . 23
2.5.2 Connectivity Management . . . 25
2.5.3 RSSI and Hop Distance-based Objective Function . . . 29
2.6 Performance Evaluation . . . 33
2.6.1 Implementation and Evaluation Environments . . . 33
2.6.2 Impact of *MobiRPL* Mechanisms . . . 35
2.6.3 Impact of *MobiRPL* Parameters . . . 39
2.6.4 Impact of circumstance parameters . . . 43
2.6.5 Performance of *MobiRPL* in complicated scenarios . . . 44
2.6.6 Performance of *MobiRPL* in real world . . . 50
2.7 Discussion . . . 52
2.8 Summary . . . 53

3 Slot-size Adaptation and Utility-based Aggregation for Time-Slotted Communication 55

3.1 Introduction . . . 55
3.2 Background and Motivation . . . 58
3.2.1 Time-Slotted Channel Hopping (TSCH) . . . 58
3.2.2 TSCH scheduling . . . 59
3.2.3 Problem and Motivation . . . 61
3.3 Related Work . . . 62
3.4 *ASAP* Design . . . 63
3.4.1 *SLA* Design . . . 64
3.4.2 *UPA* Design . . . 68
3.5 Evaluation . . . 73
3.5.1 Implementation and experiment setup . . . 73
3.5.2 Performance of *SLA* . . . 75
3.5.3 Performance of *UPA* . . . 78
3.5.4 Performance of *ASAP*: an ablation study . . . 80

3.5.5 Performance of *ASAP*: a comparative study 82
3.5.6 Performance of *ASAP* in different environment 84
3.6 Summary 86

4 Offset-based Prioritization for Accelerating Formation of 6TiSCH Networks 87

4.1 Introduction 87
4.2 Background 89
4.2.1 Time-Slotted Channel Hopping (TSCH) 89
4.2.2 TSCH scheduling and common shared cell 90
4.2.3 6TiSCH network and formation 91
4.3 Motivation 93
4.4 Approach and Considerations 95
4.4.1 Approach: Transmission offset-based prioritization 95
4.4.2 Considerations 96
4.5 Proposed Scheme 99
4.5.1 Transmission offset assignment policy 99
4.5.2 Determination of packet urgency 100
4.5.3 Differentiation between broadcast and unicast packets 101
4.5.4 Multi-offset assignment to urgent unicast packets 102
4.5.5 Transmission offset escalation 102
4.5.6 Modification of the backoff mechanism 103
4.6 Evaluation 103
4.6.1 Implementation and experiment setup 103
4.6.2 Performance of *TOP* 104
4.7 Related Work 107
4.7.1 Toward fast 6TiSCH network formation 107
4.7.2 Offset-based differentiation in TSCH slot 108
4.8 Summary 108

5 Conclusion **110**

Abstract (In Korean) **127**

List of Tables

2.1 Simulation scenarios for evaluating the performance of RPL in mobile scenarios 18

2.2 RHOF with a controllable threshold and a fixed hysteresis (4 dB) . . . 31

2.3 RHOF priority calculation from the perspective of a static node and a mobile node 32

2.4 Evaluation settings and parameters 38

2.5 Combination cases of *MobiRPL* mechanisms for evaluating the impact of *MobiRPL* mechanisms 39

2.6 Parameter settings for evaluating the impact of *MobiRPL* parameters . 40

2.7 Performance of *MobiRPL* compared to RPL and LOADng 50

List of Figures

2.1 An indoor testbed with 31 static nodes and 3 mobile nodes. The mobile nodes move along the path shown in the figure. 15

2.2 Performance of RPL and LOADng on a testbed with 31 static nodes (Fig. 2.1) according to the number of end-to-end sessions. 16

2.3 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. 17

2.4 Performance of MRHOF-based RPL in the simulation scenarios described in Fig. 2.3 and Table 2.1. 19

2.5 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. 22

2.6 *MobiRPL*'s mobility detection based on the parent change interval. . . 24

2.7 *MobiRPL* 's connectivity management including adaptive timeout and adaptive probing. 28

2.8 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 [1]. 35

2.9 Mobile node configuration and the travel path of mobile nodes. The model rail is installed in the path shown in Fig. 2.1, and the mobile node travels back and forth along the model rail. 36
2.10 Performance of *MobiRPL* depending on the type of mechanisms applied. We test four cases described in Table 2.5 37
2.11 Performance of *MobiRPL* depending on various parameter settings described in Table 2.6. 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. 41
2.12 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*. . . 43
2.13 Performance of *MobiRPL* compared to RPL varying the number of mobile nodes in *Cooja-2*. 45
2.14 Performance of *MobiRPL* compared to RPL in complicated scenarios (*Cooja-2* and *Cooja-3*). 48
2.15 Performance of *MobiRPL* compared to RPL and LOADng on a real world mobility-augmented testbed (Fig. 2.9). 49

3.1 Time usage breakdown of a regular TSCH Tx and Rx slot (of 10 ms) according to packet size, including ACK in the opposite direction. There are a lot of *idle time* (in white color) within a slot. 56
3.2 Example of TSCH operation. 58
3.3 *SLA*'s slot length adaptation process 64
3.4 Illustration of *SLA*'s slot length decision for adjustment. 65
3.5 An example of *UPA*'s packet aggregation and batch transmission (bottom) compared to the default slotted operation of TSCH (top) 69
3.6 Example of impact of *UPA*'s packet aggregation on slot utility and bit sequence. 72

3.7 Node deployment topology at Grenoble and Lille testbeds. In both testbeds, the node located in the upper left corner and marked in yellow serves as the root. 74
3.8 Real-time operation of *SLA*: Evolution of slot length over time in accordance to payload (packet) size changes. 76
3.9 Performance of *SLA* with varying k-percentile values vs. ALICE, at the Grenoble testbed, for data collection scenario. 76
3.10 Performance of *UPA*: maximum slot utility and an example of aggregating seven packets to reduce residue time. 78
3.11 Ablation study on *ASAP*: Goodput and latency results for upward vs. downward scenarios and 14 vs. 63 byte payloads 79
3.12 Performance comparison of *ASAP*, ALICE, DBT, and A3 at Grenoble testbed with dual-linear topology. *ASAP* achieves better goodput, latency, and PDR with only a slight increase in duty-cycle ($<1\%$) . . 83
3.13 Performance comparison of *ASAP*, ALICE, DBT, and A3 at Lille testbed with grid-like topology. *ASAP* achieves better goodput, latency, and PDR with only a slight increase in duty-cycle ($<1\%$) 85

4.1 TSCH common shared cell with slotframe size of 101. 90
4.2 6TiSCH network stack. 91
4.3 An example of state transition during 6TiSCH network formation. . . 92
4.4 Time evolution of packet transmissions within the common shared cell and the time when the state transition of all nodes completed. 93
4.5 Collision is inevitable in common shared cell due to the closely aligned transmission start times. 94
4.6 Assigning differentiated offsets to the packet transmission start times enables the detection of collisions among packets. 95

4.7 Example of potential side effects of offset-based prioritization. Orange, blue, and green rectangles represent CCA, transmission of data packet, and transmission of ACK respectively. 98
4.8 Transmission offset assignment policy. 99
4.9 Node deployment topology at Lille testbed. he node located in the upper left corner and marked in yellow serves as the root. 104
4.10 Network formation time for different slotframe sizes 105
4.11 Time evolution of transmissions within the common shared cell and the time when the state transition of all nodes completed for *Random* and *TOP*. 106

Chapter 1

Introduction

1.1 Motivation

LLNs play a pivotal role in various domains, including Internet of Things (IoT) applications, smart cities, and industrial automation. However, LLNs face diverse challenges that impede their performance and efficiency. In this dissertation, our motivation is to tackle these obstacles head-on and present innovative solutions to enhance the overall performance of LLNs.

The first challenge arises from the increasing need for mobility support in LLNs. The standard routing protocol, RPL, lacks an explicit mechanism to handle mobile nodes. With the growing deployment of LLN applications involving mobile devices, it becomes crucial to design a mobile routing protocol that can adapt to network dynamics and ensure reliable communication. The absence of a robust and efficient mobile routing protocol motivates us to develop a reliable mobile routing protocol termed *MobiRPL*.

The second challenge stems from inherent inefficiencies in time-slotted communication systems, such as TSCH. Traditionally, time slots are designed to accommodate maximum-sized packets, leading to significant wastage of resources when smaller packets are prevalent. This inefficiency adversely affects both throughput and latency,

limiting the overall performance of time-slotted systems. Consequently, there is a pressing need to develop techniques that can minimize residue time within each slot. As a result, we propose a utility-based adaptation of slot size and packet aggregation called *ASAP*.

The third motivation arises from the challenges faced during the network formation process in 6TiSCH networks. Synchronized transmission timing among nodes within a time slot poses difficulties in collision avoidance and congestion management. These issues hinder the efficient establishment of network connectivity, leading to increased network formation time and degraded overall performance. Therefore, it is essential to develop techniques that can enhance the efficiency of network formation in 6TiSCH networks. This motivation drives our research in proposing the offset-based prioritization technique, named *TOP*, to optimize the network formation process.

Overall, the motivation behind this dissertation is to address the challenges in mobile routing, time-slotted communication, and network formation in LLNs. By tackling these key issues, we aim to enhance the reliability and efficiency of LLNs, unlocking their full potential in various application domains.

1.2 Contributions and Outline

This dissertation makes significant contributions in addressing key challenges in LLNs and proposes innovative solutions to enhance performance of LLNs. The contributions can be summarized as follows:

Mobile routing protocol: We propose *MobiRPL*, an adaptive and robust mobile routing protocol based on RPL. *MobiRPL* utilizes RSSI and focuses on maintaining reliable routing topology in mobile LLNs. Extensive evaluations demonstrate improved packet delivery ratio by 11.3% compared to RPL and a 73.3% reduction in energy consumption compared to LOADng.

Throughput enhancement of time-slotted communication: We present *ASAP*, a

utility-based adaptation of slot-size and packet aggregation technique for time-slotted systems. *ASAP* dynamically adjusts slot length based on actual packet size distribution and employs packet aggregation to maximize slot utility. Experimental evaluations show a 2.21x improvement in throughput and a reduction in latency by 78.7%.

Acceleration of 6TiSCH network formation: We propose *TOP*, an offset-based prioritization technique to enhance network formation efficiency in 6TiSCH networks. *TOP* assigns offsets to packets during transmission initiation to diversify starting points within slot, enable collision detection, and prioritize critical packets. Real testbed experiments demonstrate up to a 50% reduction in network formation time.

In summary, these contributions address critical issues in mobile routing, time-slotted communication, and network formation, thereby improving the overall performance of LLNs.

This dissertation is structured as follows. In Chapter 2, we introduce *MobiRPL*. In Chapter 3, we psesent *ASAP*. In Chapter 4, we propose *TOP*. Chapter 5 concludes this dissertation.

Chapter 2

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

2.1 Introduction

RPL, the IPv6 routing protocol for LLNs standardized in 2012, has been considered as a building block of IoT and received great attention from LLN researchers [2–5]. 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 [6–9], 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 [3].

At the same time, however, there are *non-zero* mobile devices in the above application scenarios: remote controllers and wearable devices at home [8], cranes, forklifts, and workers in an industrial environment [7], and occupants and movable assets in a building [9]. In addition, clinical applications include mobile medical staffs and patients [10]. To minimize network dynamics with these mobile nodes, there is a routing design guideline in [9]: "*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 [11–18]. 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 non-trivial 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 [19, 20].

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* [21–28]. 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 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 proto-*

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

col originally designed for mobile networks, such as (LOADng) [29], 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 [30–34]. 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

[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.

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 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 [35] 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 chapter is organized as follows. We first discuss the brief background and related work in §2.2. In §2.3, we present the preliminary study results. We summarize the requirements for our proposed scheme in §2.4. We introduce our proposed scheme, *MobiRPL*, and elaborate on its main functional blocks in §2.5. We discuss the implementation details and the evaluation results in §2.6. We provide discussion in §2.7 and conclude the chapter in §2.8.

2.2 Background and Related Work

There have been two types of research efforts to support mobile LLN: (1) extending RPL [2] to cover mobile nodes and (2) modifying routing protocols for MANETs to support LLNs. The latter results in LOADng [29], a lightweight version of ad-hoc on-demand distance vector (AODV) (a standard MANET routing protocol) [36]. This section reviews the two representative protocols, RPL and LOADng, and their related work.

2.2.1 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

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

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 [37], 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) [38], 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 [3, 39–41].

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 [11]. 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 [14] 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 [15] 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 [16] 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 [12] 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 [13] 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 [21] 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 [22] and mRPL+ [23] 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 [24] 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 [25] defines corona ID (minimum hop distance) to indicate how close a node is to the root node and uses it for parent selection. RRD [26] 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 [27] 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 [42, 43]. 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 [17] 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 [18] 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 [28] 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.

2.2.2 LOADng, a MANET Protocol for LLN

Many routing protocols have been developed for MANETs [44], such as destination sequenced distance vector (DSDV) [45], optimized link-stated routing (OLSR) [46], AODV [36], and dynamic source routing (DSR) [47]. In LLN, however, these protocols cause significant control overhead and/or slow recovery [48]. To alleviate the problems, LOADng [29, 49, 50], 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 [48] 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 [30–34]. Otherwise, LOADng underperforms RPL. As an improvement, LOADng-CTP [50, 51] 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.

2.3 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.

2.3.1 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. 2.1. Each node uses -10 dBm transmission power and an antenna of 5 dB gain. For the routing layer, RPL and LOADng implementations on Contiki OS [52] version 3.0 are used. The underlying link layer is ContikiMAC [53], the asynchronous duty-cycling MAC of Contiki OS. We set the sleep interval of Con-

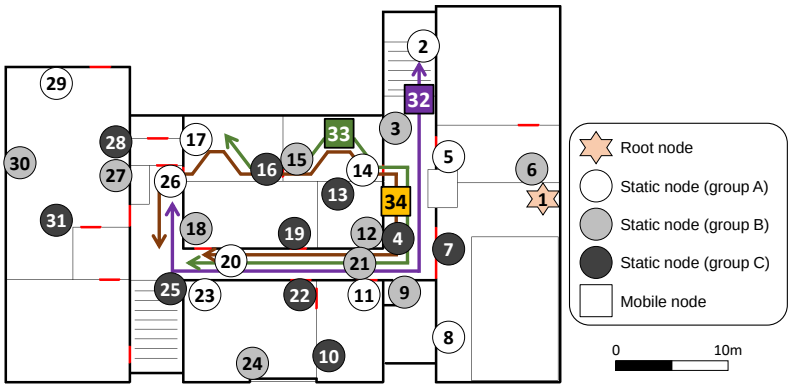
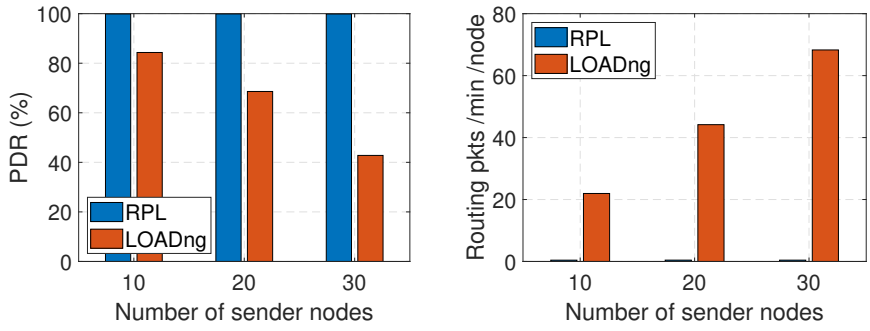


Figure 2.1: An indoor testbed with 31 static nodes and 3 mobile nodes. The mobile nodes move along the path shown in the figure.

tikiMAC as 31.25 msec (32 Hz channel check rate).

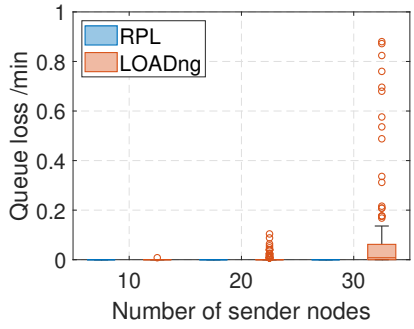
To evaluate the scalability of RPL and LOADng according to the number of end-to-end sessions, we divide 30 nodes (except the root) into three groups (group A, B, and C), each of which has ten nodes. Then, we observe the performance of RPL and LOADng with varying the number of data senders. We consider 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. 2.2 plots various performance metrics in the scenario (the average of five repetitive experiments).

As seen in Fig. 2.2a, 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 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

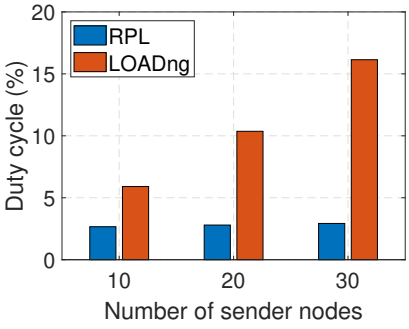


(a) PDR

(b) Routing overhead



(c) Queue loss



(d) Duty cycle

Figure 2.2: Performance of RPL and LOADng on a testbed with 31 static nodes (Fig. 2.1) according to the number of end-to-end sessions.

the number of data senders (i.e., the number of routes). This is verified in Fig. 2.2b, 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 $\sim$*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. 2.2c shows that in LOADng cases, several nodes suffer severe queue loss when the number of senders is

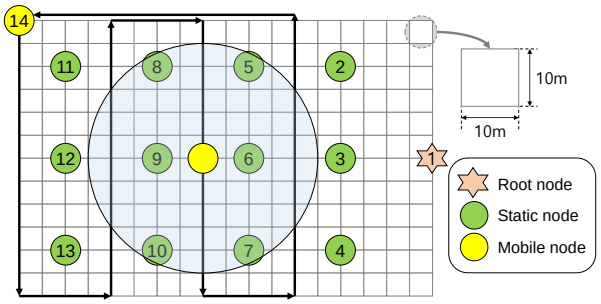


Figure 2.3: A mobile LLN scenario on the Cooja simulator with 12 static nodes, a mobile node, and a root. All the nodes have the same transmission range (50 m), and one example is indicated by a large light blue circle.

large. This confirms that the congestion level caused by LOADng is above the threshold which resource-constrained nodes can cope with. According to Fig. 2.2d, 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.

2.3.2 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

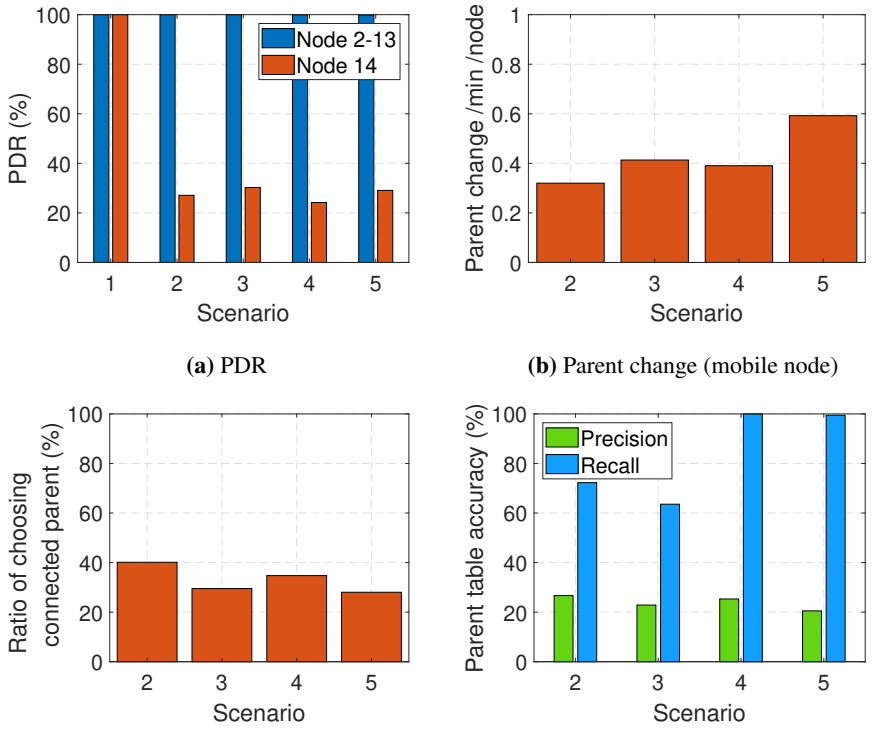
Table 2.1: Simulation scenarios for evaluating the performance of RPL in mobile scenarios

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

in Fig. 2.3. 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.

We perform simulations on five different scenarios, as shown in Table 2.1. 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. 2.3 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. 2.4 plots the simulation results (the average of five repetitive experiments with different random seeds). As Fig. 2.4a 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



(a) PDR

(b) Parent change (mobile node)

(c) Accuracy of routing decision (mobile node)

(d) Accuracy of parent table (mobile node)

Figure 2.4: Performance of MRHOF-based RPL in the simulation scenarios described in Fig. 2.3 and Table 2.1.

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. 2.4b, 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 mo-

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

bile node selects a new preferred parent that is actually connected when changing its preferred parent. Fig. 2.4c 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. 2.4d 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) [54], 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 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.

2.4 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.

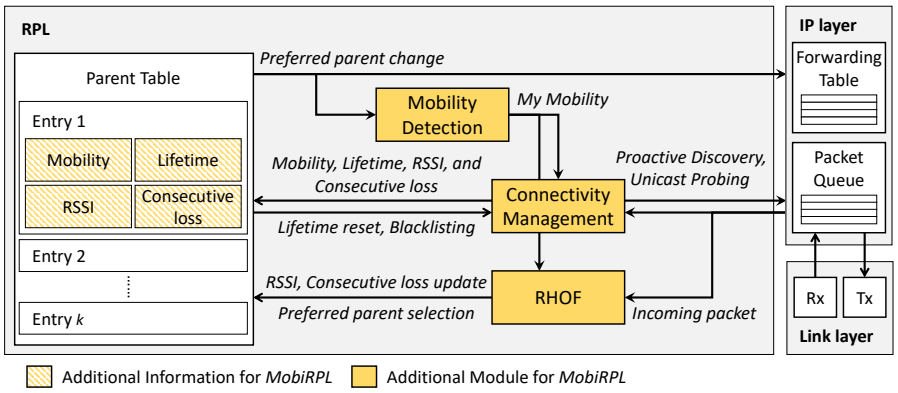


Figure 2.5: Overview of *MobiRPL* Design. We propose three new mechanisms (mobility detection, connectivity management, and RHOF) as a part of RPL. Our new mechanisms better cope with mobility by interacting with existing RPL operations and IP layer.

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.

2.5 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) *connec-*

tivity management, and (3) *RSSI and hop distance-based objective function*, as shown in Fig. 2.5.

2.5.1 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 ($t_{c,thr}$) 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

$$\overline{t_{c,i}} = \alpha \cdot \overline{t_{c,i-1}} + (1 - \alpha) \cdot t_{c,i}. \tag{2.1}$$

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 ($t_{c,thr}$), 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

[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.

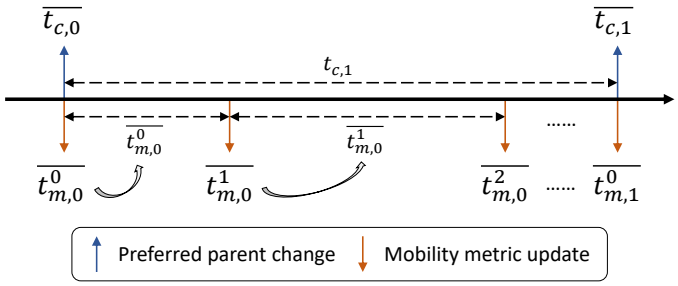


Figure 2.6: *MobiRPL*'s mobility detection based on the parent change interval.

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. 2.6, when the i-th parent change occurs, *MobiRPL* calculates $\overline{t^0_{m,i}}$ as (using the newly calculated $\overline{t_{c,i}}$)

$$\overline{t^0_{m,i}} = \overline{t_{c,i}}. \tag{2.2}$$

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

$$\overline{t^{j+1}_{m,i}} = \alpha \cdot \overline{t_{c,i}} + (1-\alpha) \cdot \sum_{k=0}^{j} \overline{t^k_{m,i}}. \tag{2.3}$$

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 $t_{c,thr}$ to determine mobility. In this way, static nodes can fast increase $\overline{t_m}$ even without their parent changes, avoiding misclassification.

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 ($t_{c,thr}$) 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.

2.5.2 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*.

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

$$t_{l,0} = T_{DIO,max} \cdot 2^{-m}, \quad 0 \leq m \leq M. \tag{2.4}$$

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 [12]. However, reducing the DIO interval violates the default Trickle-Timer 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.

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

$$t_p = t_{l,0}/(N+1). \tag{2.5}$$

If a data transmission has been recently performed, the next probing is skipped to reduce control overhead. Fig. 2.7 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 implementation, *MobiRPL* uses unicast DIS messages

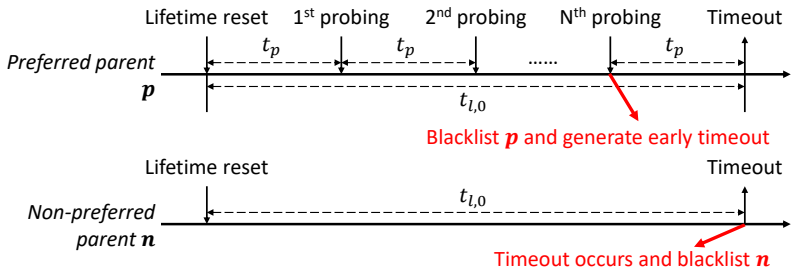


Figure 2.7: *MobiRPL* 's connectivity management including adaptive timeout and adaptive probing.

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.

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 §2.5.3). 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.

2.5.3 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.

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.

$$Rank(N) = Rank(P) + MinHopRankInc. \tag{2.6}$$

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 [26, 55, 56]. 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.

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 2.2. Inspired by Thread [55], a network protocol currently being actively used in the IoT domain, RHOF classifies nodes with RSSI above $RSSI_{thr}$ 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

Table 2.2: RHOF with a controllable threshold and a fixed hysteresis (4 dB)

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

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.

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 2.3 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

Table 2.3: 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

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.

2.6 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.

2.6.1 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 msec (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. 2.3 and Table 2.1. We will call this scenario *Cooja-1*. Fig. 2.8 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. 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).

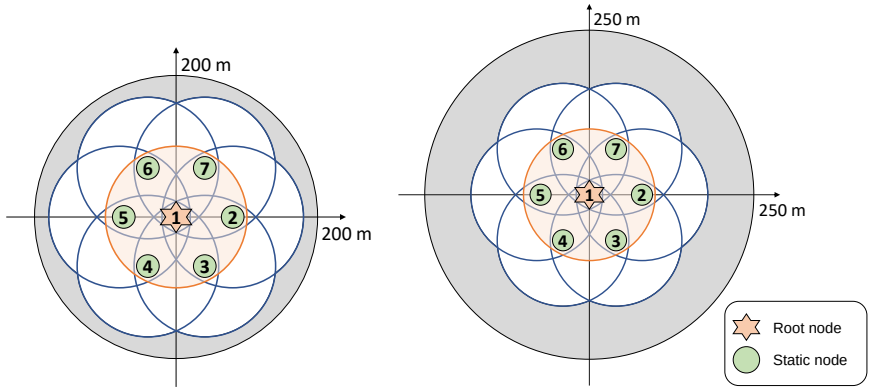
There are shaded areas where mobile nodes cannot be connected to static nodes, which is represented as a gray area in Fig. 2.8. Mobile nodes independently move following Random way-point model [1] with the minimum and maximum speeds of 0.5 m/s and 2.0 m/s, respectively. We will name the scenarios with 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. 2.1. 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. 2.9a. On the line drawn with arrows in Fig. 2.1, 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. §2.6.1 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 2.4 summarizes the default experimental parameters. Unless explicitly stated in each experiment, we apply the default parameters in Table 2.4.

[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.



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

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

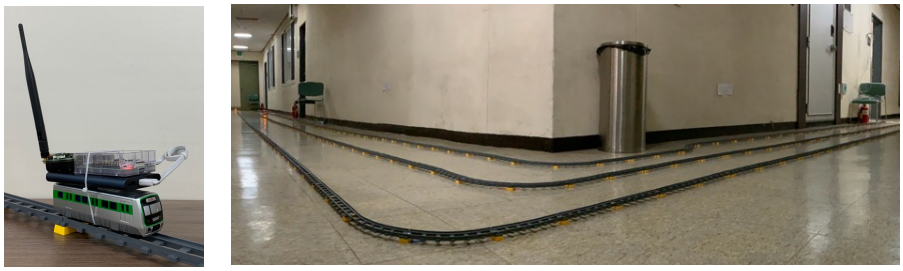
Figure 2.8: Mobile LLN scenarios on the Cooja simulator with one root, 6 static nodes, and up to 18 mobile nodes. A small circle indicates each node's transmission range. The mobile nodes move within the outer circle following Random way-point model [1].

2.6.2 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 along the line illustrated in Fig. 2.3 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 2.5, 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 ($T_{l,min}$) is 16 s, and the number of probing (N) for adaptive probing is 2 (twice). The threshold of parent change interval for mobility



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

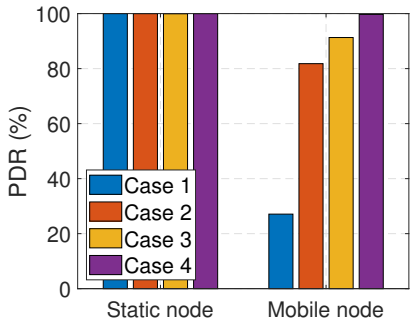
Figure 2.9: Mobile node configuration and the travel path of mobile nodes. The model rail is installed in the path shown in Fig. 2.1, and the mobile node travels back and forth along the model rail.

detection ($t_{c,thr}$) is 120 s. The RSSI threshold ($RSSI_{thr}$) of RHOF is -83 dBm.

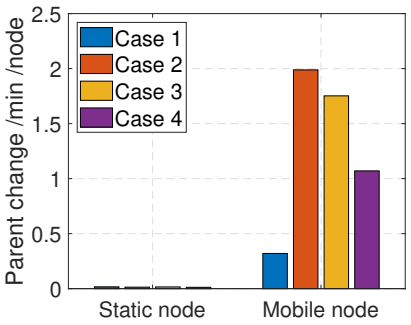
Fig. 2.10a shows the average upward PDR. While RPL shows PDR lower than 30%, the connectivity management mechanism raises the PDR of mobile nodes to 80%. Other mechanisms, such as RHOF and proactive discovery, further increase the PDR of mobile nodes. The performance improvement comes from accurate routing decisions achieved by our proposed mechanisms. As presented in Fig. 2.10c, *MobiRPL*'s mechanisms successfully improve the accuracy of routing decisions made by mobile nodes.

Fig. 2.10d, 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.

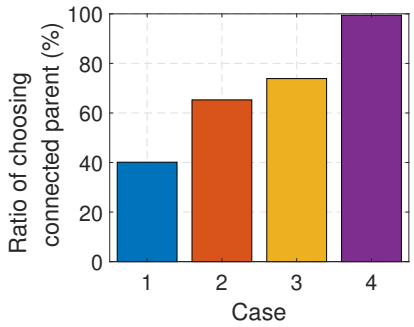
Since the connectivity management mechanism deletes the parent entries aggressively, we can observe that the recall also drops. However, our RHOF and proactive



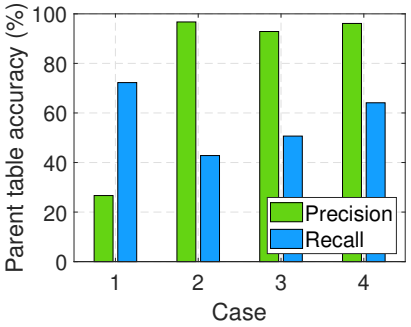
(a) PDR



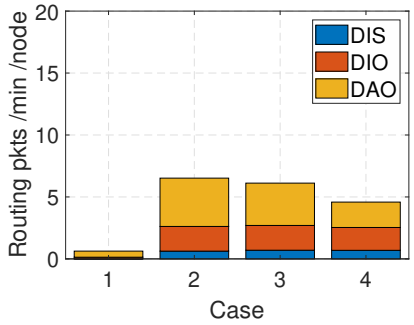
(b) Parent change (mobile node)



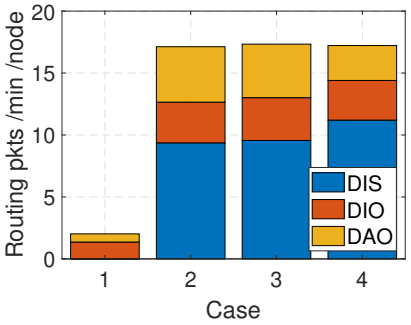
(c) Accuracy of routing decision (mobile node)



(d) Accuracy of parent table (mobile node)



(e) Routing overhead (static node)



(f) Routing overhead (mobile node)

Figure 2.10: Performance of *MobiRPL* depending on the type of mechanisms applied. We test four cases described in Table 2.5

Table 2.4: Evaluation settings and parameters

Parameters	Testing environments	Values
$t_{c,thr}$	All	120 s
$T_{l,min}$	All	16 s
N	All	2
$RSSI_{thr}$	All	-83 dBm
$T_{DIO,min}$	All	4.096 s
$T_{DIO,max}$	All	1048.576 s
ContikiMAC channel check rate	All	32 Hz
Traffic pattern	Cooja-1 Cooja-2 and 3 Testbed	120 upward packets (1 pkt / 30 secs) 100 up/downward packets (1 pkt / 60 secs) 120 up/downward packets (1 pkt / 60 secs)
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

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. 2.10b, *MobiRPL*'s

Table 2.5: 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

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. 2.10e and 2.10f 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.

2.6.3 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 §2.6.2. 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

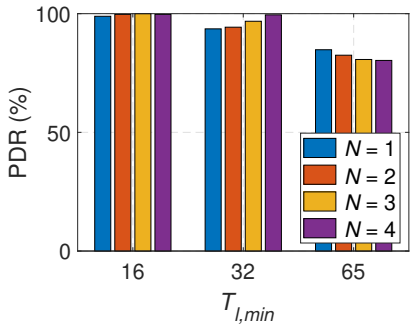
Table 2.6: Parameter settings for evaluating the impact of *MobiRPL* parameters

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

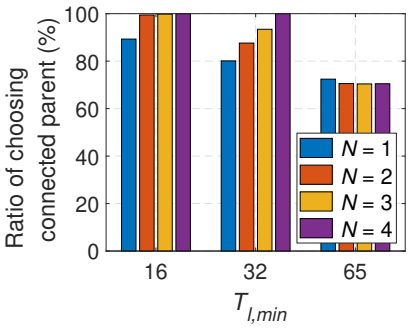
how *MobiRPL* aggressively blacklists neighbor nodes. *MobiRPL* determines the link with N-consecutive packet losses as disconnected. Therefore, increasing N improves the accuracy of connectivity examination, but a larger N causes a greater delay in connectivity judgment.

We evaluate *MobiRPL* with different parameter settings as described in Table 2.6 and plot the results in Fig. 2.11. Fig. 2.11a 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. 2.11b, 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.

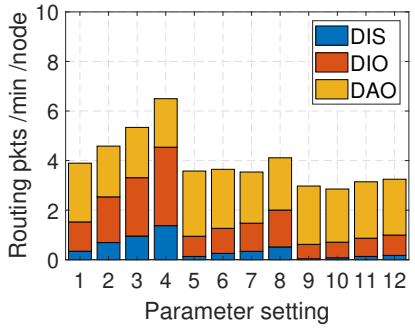
We then discuss why N affects routing accuracy and PDR differently according to $T_{l,min}$. If $T_{l,min}$ is set large, *MobiRPL* generates timeouts for non-preferred parents slowly. Considering that the probing interval for the preferred parent (t_p) is set proportionally to $T_{l,min}$, using large $T_{l,min}$ causes *MobiRPL* to take a longer time performing probing N times. Such a delay in timeout and probing can make the parent table of



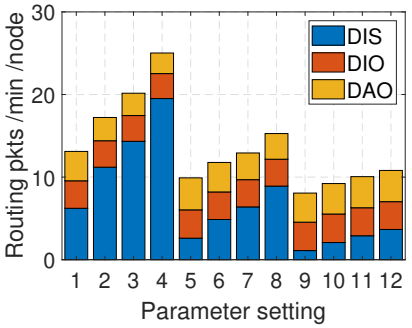
(a) PDR (mobile node)



(b) Accuracy of routing decision (mobile node)



(c) Routing overhead (static node)



(d) Routing overhead (mobile node)

Figure 2.11: Performance of *MobiRPL* depending on various parameter settings described in Table 2.6. 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.

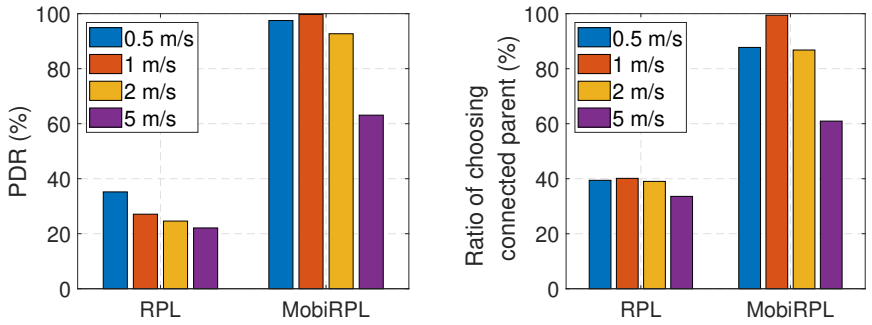
MobiRPL full of outdated entries. In this situation, increasing N causes probing operation to take more time, and *MobiRPL* cannot avoid wrong routing decisions. In short, if $T_{l,min}$ is not set short enough, increasing N does not have any advantage other than accurately determining connectivity to 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. 2.11c and 2.11d 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 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.



(a) PDR (mobile node)

(b) Accuracy of routing decision (mobile node)

Figure 2.12: 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*.

2.6.4 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 §2.6.2, 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. 2.12 plots the result. As shown in Fig. 2.12a, 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. 2.12b, 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 ($t_{c,thr}$) 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 cor-

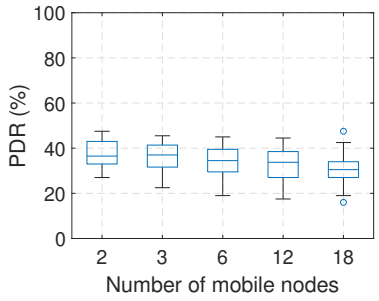
rectly detects its mobility most of the time. However, if the mobile node moves quickly (e.g., 2 m/s), the routing decision accuracy can degrade due to the proactive nature of *MobiRPL*. Nevertheless, as shown in Fig. 2.12a, *MobiRPL* overcomes deterioration in routing decision accuracy and achieve high 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.

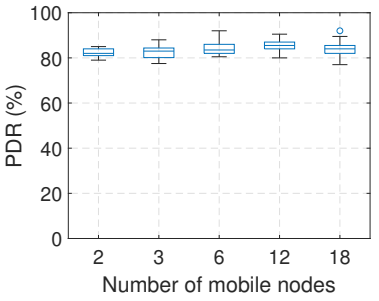
2.6.5 Performance of *MobiRPL* in complicated scenarios

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.

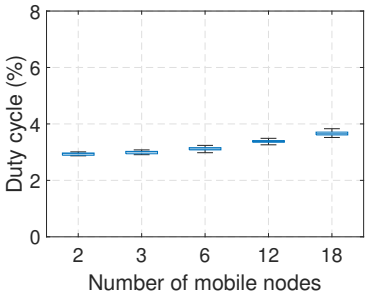
Impact of the number of mobile nodes: We now test the impact of the number of mobile nodes. To this end, in *Cooja-2*, we evaluate the performance of RPL and *MobiRPL*, changing the number of mobile nodes to 2, 3, 6, 12, and 18. There are six static nodes (excluding the root node) in *Cooja-2*; thus, 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



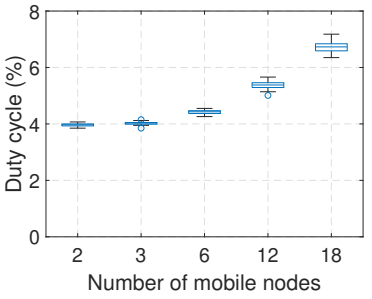
(a) PDR of RPL (mobile node)



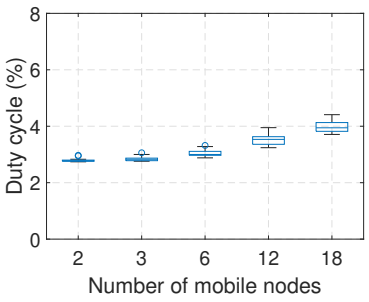
(b) PDR of *MobiRPL* (mobile node)



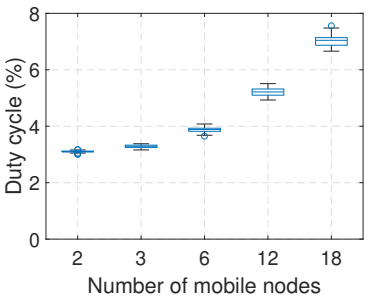
(c) Duty cycle of RPL (mobile node)



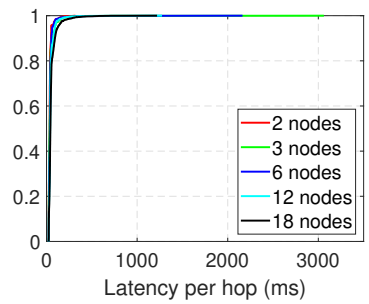
(d) Duty cycle of *MobiRPL* (mobile node)



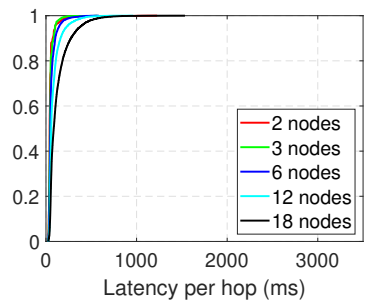
(e) Duty cycle of RPL (static node)



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



(g) Latency of RPL (mobile node)



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

Figure 2.13: Performance of *MobiRPL* compared to RPL varying the number of mobile nodes in *Cooja-2*.

to compare energy consumption. We also examine per-hop latency.

Fig. 2.13 plots the performance of RPL and *MobiRPL*. Figs. 2.13a and 2.13b 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. 2.13c to 2.13f 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. 2.13g and 2.13h 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.

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'

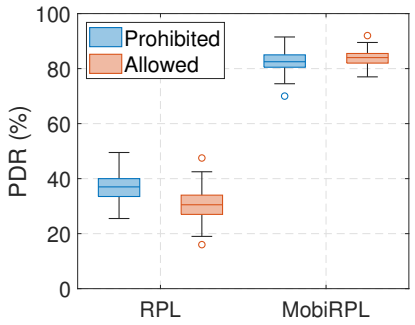
packet delivery in mobile LLNs. To examine this, we simulate two cases: 1) mobile nodes are not allowed to participate in packet forwarding, and 2) mobile nodes participate in packet forwarding. In case 1, a mobile node operates as a leaf node that does not generate multicast DIO messages and does not become the preferred parent of other nodes.

Fig. 2.14 plots the performance of RPL and *MobiRPL*. Figs. 2.14a and 2.14b 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.

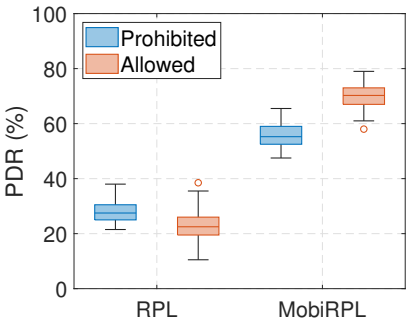
The PDR of RPL decreases if mobile nodes participate in packet forwarding due to its ineffective mobile routing. In *MobiRPL*, however, the participation of mobile nodes in packet forwarding improves the PDR. This is because a mobile node that moves into the shaded area can acquire connectivity to the 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. 2.14c to 2.14f 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 since *MobiRPL* generates more control packets than RPL to maintain connectivity.

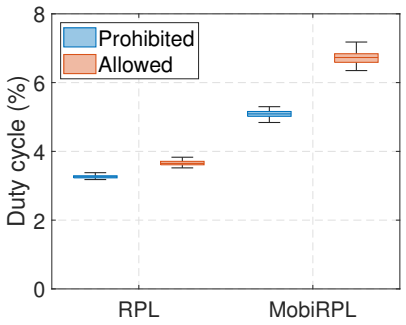
In *Cooja-2* (a narrow shaded area), if mobile nodes are allowed to forward packets, the duty cycles of static and mobile nodes increase in the both protocols due to the mobile nodes' routing and forwarding overheads. On the other hand, in *Cooja-3* (a broad shaded area), allowing packet forwarding of 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 signifi-



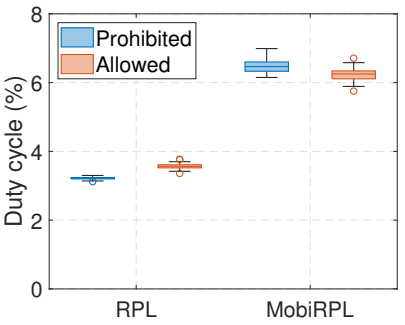
(a) PDR in *Cooja-2* (mobile node)



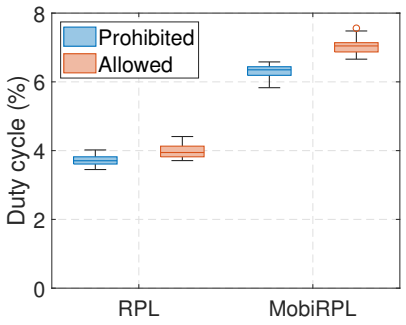
(b) PDR in *Cooja-3* (mobile node)



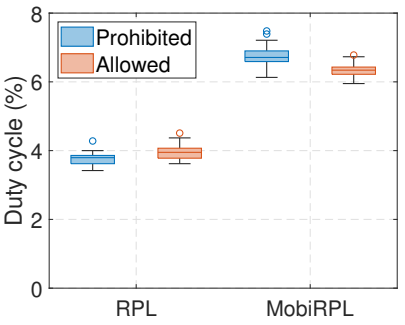
(c) Duty cycle in *Cooja-2* (mobile node)



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

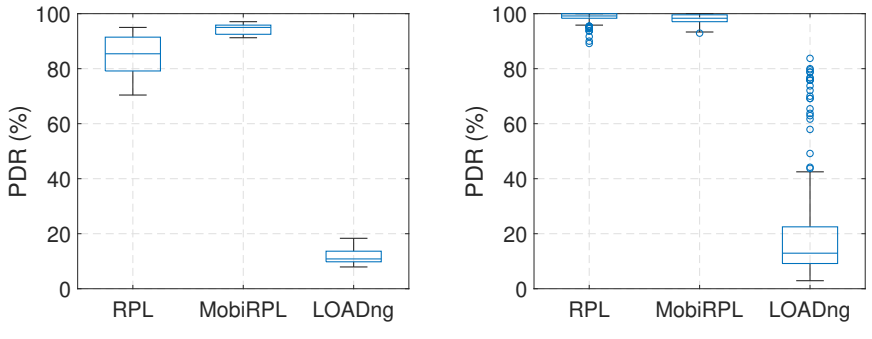


(e) Duty cycle in *Cooja-2* (static node)



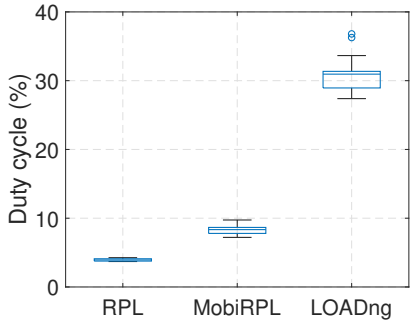
(f) Duty cycle in *Cooja-3* (static node)

Figure 2.14: Performance of *MobiRPL* compared to RPL in complicated scenarios (*Cooja-2* and *Cooja-3*).

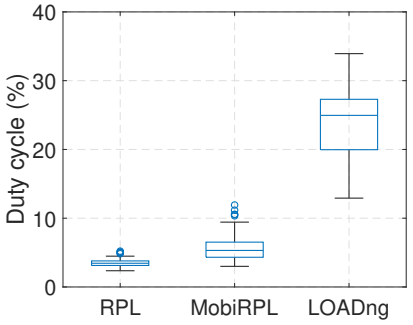


(a) PDR (mobile node)

(b) PDR (static node)



(c) Duty cycle (mobile node)



(d) Duty cycle (static node)

Figure 2.15: Performance of *MobiRPL* compared to RPL and LOADng on a real world mobility-augmented testbed (Fig. 2.9).

cantly 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 §2.2, most of RPL-based mobile routing protocols prohibit mobile nodes from participating in packet forwarding [11–18]. 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 [21, 24–26], but exploits ETX-based MRHOF which is inappropriate for mobile LLNs as shown in §2.6.2. In contrast, the results show that our design choices for *MobiRPL* to allow mobile nodes' packet forwarding is effective.

Table 2.7: 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	98.63	3.47	0.64	0.03	0.00
	Mobile	84.75	3.94	1.52	0.17	0.00
MobiRPL	Static	98.06	5.58	13.07	0.51	0.01
	Mobile	94.36	8.28	29.83	2.96	1.41
LOADng	Static	22.01	23.88	146.71	-	21.17
	Mobile	11.92	31.07	180.96	-	96.60

2.6.6 Performance of *MobiRPL* in real world

We evaluate *MobiRPL* on a real indoor testbed, the same as in §2.3.1. There are 31 TelosB-clone *static* nodes, including one root node, as depicted in Fig. 2.1. 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: $T_{l,min} = 16s$ 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. 2.15 and Table 2.7 show the results.

As can be seen in Figs. 2.15a and 2.15b, 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. 2.15c and 2.15d, compared to RPL, *MobiRPL* increases the duty cycle (energy consumption) of both static nodes and mobile nodes due to more control traffic (Table 2.7). 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 2.7. 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. 2.15a and 2.15b. Compared to LOADng that shows the worst duty cycle and PDR performance (see queue loss in Table 2.7) 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 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, *Mo-*

biRPL successfully prevents mobile nodes from becoming preferred parents of static nodes.

2.7 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*.

2.8 Summary

In this chapter, 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.

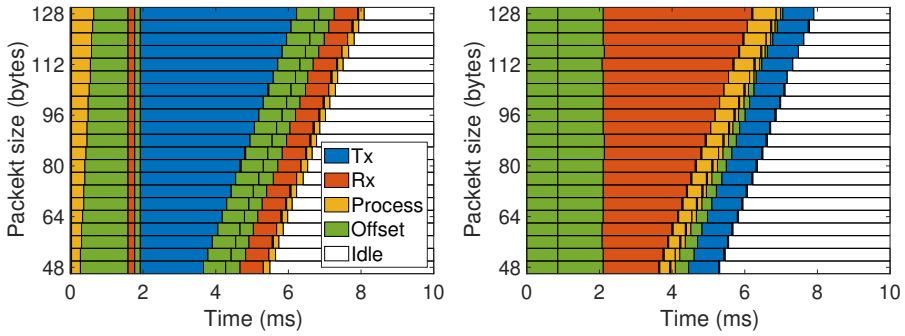
Chapter 3

Slot-size Adaptation and Utility-based Aggregation for Time-Slotted Communication

3.1 Introduction

Time-slotted communication has been long-loved by various communication protocols and systems. It synchronizes the network, divides time into slots, and a communication transaction occurs within each timeslot. In contrast to asynchronous random access approaches such as pure ALOHA and carrier sense multiple access (CSMA), time-slotted communication can easily coordinate communication or better allocate resources between devices given that there is a coordinator/master to manage the synchronization. Therefore, it can improve reliability and throughput by preventing collisions and interference due to uncoordinated transmissions, and also reduce energy waste attributed to redundant rendezvous attempts or idle listening [60].

IEEE 802.15.4 TSCH [57], a MAC protocol for LLN, is one of those examples. It has been designed to satisfy the growing demand for more reliable and energy-efficient LLNs in emerging IoT applications such as industrial IoT [61–66], in-vehicle IoT [67, 68], environmental monitoring [69–72], home IoT [73], and health IoT [74–76]. TSCH brings the benefits of time-slotted communication to LLN, and its channel hopping



(a) TSCH Tx slot

(b) TSCH Rx slot

Figure 3.1: Time usage breakdown of a regular TSCH Tx and Rx slot (of 10 ms) according to packet size, including ACK in the opposite direction. There are a lot of *idle time* (in white color) within a slot.

allows the network to become more robust to external interference or multi-path fading through frequency diversity. As such, TSCH has shown remarkable performance in the literature [77–84]

However, time-slotted systems have one fundamental drawback; *'a slot'* is predefined to be sufficiently long enough to accommodate one exchange of a maximum-sized packet and an acknowledgment (ACK) [1]. If most packets in the system are much shorter than the maximum, substantial amount of residue time within each slot are wasted, leading to corresponding amount of loss in effective data rate. The same is true for TSCH. Fig. 3.1 plots the time usage breakdown of a TSCH slot according to data packet length, for both transmission (Tx) side and reception (Rx) side including ACK, measured from an actual testbed experiment. *Idle time* ratio increases to almost 50% as the packet size decreases; i.e., nearly half of the time may be wasted (§3.2.3). This means, conceptually, a 250 kbps IEEE 802.15.4 PHY can only achieve up to ∼125 kbps effective data rate using TSCH.

A naive approach would be to shorten the time-slot length. But to what size? Ob-

[1] There are variant systems where multiple slots can be assigned for a large transaction (e.g. cellular), but the fundamental concept still holds.

viously, a size smaller than the packets would break the basic assumptions of slotted operation. What if the system has mix of packet sizes from small to big? Furthermore, what if the application running on the network (and thus the packet sizes) changes after configuring the slot size? or if multiple applications are running concurrently? These questions cannot be answered using a pre-configured fixed size slot.

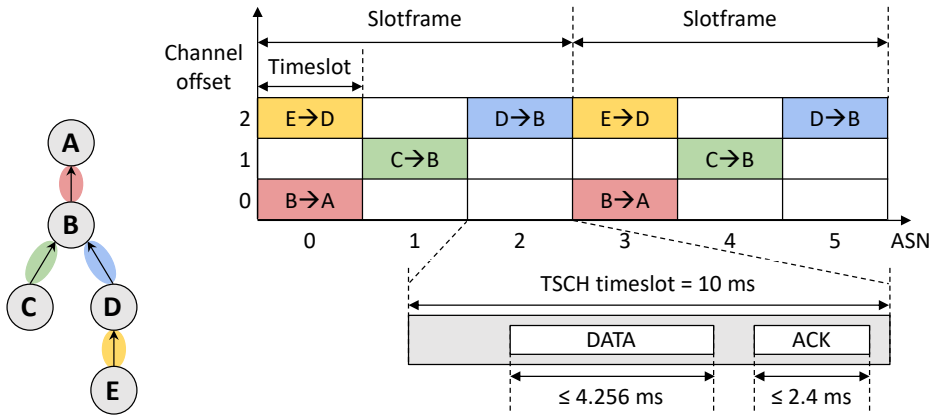
To address this fundamental challenge, we propose *"utility-based adaptation of slot-size and aggregation of packets"* (*ASAP*), a scheme that enables time-slotted systems to operate more *time-efficiently* by reducing the idle residue time and improving the time utility of the slots. *ASAP* consists of two orthogonal methods: (1) *slot-length adaptation* (*SLA*) adjusts timeslot length network-wide according to the distribution of packet and ACK sizes observed in the network. (2) *utility-based packet aggregation* (*UPA*) aggregates packets and transmits them in a batch over multiple consecutive slots when and only when beneficial in terms of *slot-utility*[2]. Both methods aim to minimize wasted time and maximize slot utility, thus achieving higher throughput and lower latency than the fixed-size time-slotted operation.

We case-study *ASAP* in the context of TSCH. We implement *ASAP* on real embedded IEEE 802.15.4 devices using Contiki-OS [85], and evaluate on multiple large-scale topologies in the FIT/IoT-LAB public LLN testbed [86] with various state-of-the-art TSCH schedulers. Results show that *ASAP* improves throughput and reduces latency of TSCH network by up to 2.21x and 78.7% respectively. Given that our approaches are not limited to TSCH but can be applied to other time-slotted communication protocols and systems, we believe *ASAP* can be a generic solution to the fundamental challenge of time-slotted communication.

Our contributions can be summarized as follows.

- We present an analysis of time usage breakdown in TSCH through real measurements to demonstrate the time wastage in slotted communication.

[2]We later (in §3.4) define *'slot utility'* as the ratio of the number of packets transmitted to the number of slots used for those transmissions.



(a) Topology **(b)** TSCH timeslot and channel hopping operations

Figure 3.2: Example of TSCH operation.

- We propose *ASAP*, consisting of *SLA* and *UPA*, to address the problem.
- We case-study *ASAP* in the context of TSCH. We implement *ASAP* on real embedded devices, and evaluate in multiple sizeable public testbeds to demonstrate significant performance improvement.

The remainder of this chapter is organized as follows. We present the background and motivation in §3.2, and discuss related work in §3.3. We present the design of *ASAP* in §3.4, and evaluate *ASAP* in §3.5. Finally, §3.6 concludes the chapter.

3.2 Background and Motivation

We first provides a brief introduction of TSCH and representative TSCH schedulers that we case-study on. We then describe the problem and motivation of this work.

3.2.1 Time-Slotted Channel Hopping (TSCH)

TSCH is a MAC protocol standardized in IEEE 802.15.4e [57] that combines time-slotted communication and channel hopping. TSCH synchronizes the network, and devices communicate in a time-slotted manner to improve reliability and energy effi-

ciency. Channel hopping enables TSCH to be robust to external interference and fading through channel diversity.

As Fig. 3.2 illustrates, TSCH divides time into *timeslots*. The length of a timeslot is typically set to 10 ms, sufficiently long enough for exchanging a maximum-sized (128 Bytes) frame and an ACK of up to 70 Bytes. Each timeslot has an *absolute slot number (ASN)*, which is initialized to zero at the beginning of the network and then sequentially incremented. A set of timeslots constructs a *slotframe*, which is repeated in time and functions as a unit of TSCH schedule. The number of timeslots in a slotframe is called slotframe length (L_{SF}). Then, *time offset* (t_o) is a relative position of a specific timeslot within a slotframe calculated as,

$$t_o = mod(ASN, L_{\mathrm{SF}}). \tag{3.1}$$

Each schedule has a *channel offset* (c_o) used for channel selection in TSCH's channel hopping. TSCH decides which channel to operate in each timeslot based on the following calculation,

$$Channel = List_{\mathrm{c}}[mod(ASN + c_{\mathrm{o}}, sizeof(List_{\mathrm{c}}))] \tag{3.2}$$

where $List_{\mathrm{c}}$ is a set of channels to be used and $sizeof(List_{\mathrm{c}})$ is the number of channels in $List_{\mathrm{c}}$. As ASN increases, each timeslot with a particular c_o hops over different channels. Even on a timeslot with the same ASN, different c_o leads to the usage of distinct channels.

3.2.2 TSCH scheduling

TSCH standard defines how to perform time-slotted communication and channel hopping. However, it leaves *resource scheduling* (i.e., determining when (t_o) and on which channel (c_o) for each device to communicate) as an open problem. Nevertheless, as with any other slotted communication protocols, TSCH requires a scheduling method for efficient and reliable packet exchange. To this end, various TSCH schedulers have

been proposed. Most TSCH schedulers can be categorized into centralized, distributed, and autonomous schedulers.

Centralized schedulers [87–92] use global network information (e.g., topology, link quality, etc.) to construct a schedule, and distribute it to the network for each node to use. Although they can potentially optimize the schedule based on a global view of the network, collecting network information and disseminating the schedule requires a huge communication overhead. With **distributed schedulers** [80–82, 93–95], every node has a scheduling function that determines its schedule based on local information or negotiation with neighboring nodes. Although distributed schedulers can lower control overhead compared to centralized schedulers, they still suffer from non-negligible overhead. Lastly, **autonomous schedulers** [77, 79, 83] determine a schedule according to predetermined rules (e.g., a hash function) and self-obtainable information (e.g., node ID) in each node. Therefore, autonomous schedulers have the advantage of requiring no additional control overhead. However, since each node schedules itself autonomously and independently, autonomous schedulers inherently cannot consider the schedules or situations of neighboring nodes.

It is important to note that the problem we are trying to solve is orthogonal to how the schedulers operate. Regardless of the type of the scheduler, residue time may always exist under fixed-size time-slotted operation. Therefore, without loss of generality, we select ALICE [79], a state-of-the-art autonomous scheduler among many, to case-study *ASAP*. In ALICE, nodes autonomously determine their own unicast schedule by utilizing routing information (i.e., the node IDs of neighboring nodes) and a hash function that is shared between all nodes in the network. For instance, the time offset of a unicast schedule for a directional link from node *A* to node *B* can be calculated as;

$$t_o(A, B) = mod(\mathit{Hash}(\alpha \cdot \mathit{ID}(A) + \mathit{ID}(B) + \mathit{ASFN}), L_{\mathrm{SF}}). \tag{3.3}$$

The coefficient α is used to differentiate traffic direction, while $\mathit{ID}(x)$ denotes the node ID of x. *ASFN* stands for *absolute slotframe number*, which is initialized to zero

at the beginning of the network and then incremented as the slotframe progresses. *ASFN* makes the outcome of the hash function distinctive every slotframe, leading to time-varying resource assignment. This time-varying resource assignment prevents repetitive overlaps between resources or repeated disruption from interference that can occur with fixed location of resources. Since each node can obtain the node ID of its neighboring nodes through the routing layer, ALICE no longer requires the exchange of control messages for scheduling.

3.2.3 Problem and Motivation

Here we further analyze the time usage breakdown of a TSCH slot in Fig. 3.1. To obtain the result, we measured the execution time of various TSCH operations within a slot while exchanging packets between two IEEE 802.15.4 M3 devices with varying packet sizes from 48 to 128 bytes. ACK length is set to 20 bytes, the typically used size in Contiki-OS. We enabled the CCA feature before packet transmission on the Tx side, and classified the operation times into five categories as follows:

- **Transmission** ($T_{\mathbf{Tx}}$): Time to transmit packet or ACK.
- **Reception** ($T_{\mathbf{Rx}}$): Time to receive packet or ACK
- **Process** ($T_{\mathbf{proc}}$): Time to pre-/post-process transmission/ reception, including the time to turn the radio on and off.
- **Offset** ($T_{\mathbf{offset}}$): Required wait time to meet the predefined operation timing, including the time to listen on wireless channel before reception and the time to perform CCA. Cannot be regarded as idle time.
- **Idle** ($T_{\mathbf{idle}}$): Idle time without any Tx/Rx-related action.

Intuitively, smaller packet size leads to decrease in Tx/Rx times, and thus an increase in idle time within a slot (Fig. 3.1). Idle time ratio reaches almost 50% when the packet size is at its minimum. Even with the largest packet size, $\sim$20% of the timeslot is still wasted. This is because the TSCH timeslot length is set to accommodate a maximum-

sized packet and also a maximum-sized ACK. Since the typical ACK size is far smaller than the maximum, significant idle time exists even when a max-sized packet is sent. In summary, 20−50% of bandwidth is wasted in TSCH due to the fixed slot length, and therefore, reducing the idle/residue time is crucial to increase network throughput. This is the problem that we aim to address in this work. Finally, although we case-study in the context of TSCH, the problem we solve is not limited to TSCH but is a common problem of time-slotted systems. This observation motivated us to the design of *ASAP*, which we believe will apply as a general solution to many time-slotted systems.

3.3 Related Work

The goal of this work is to improve the throughput of time-slotted communication by reducing the residue time within each slot. In TSCH where we conduct case study, various attempts have been made to improve network throughput.

One approach is to utilize temporary resources in addition to the ones allocated by the scheduler. TSCH standard [57] defines the *default burst transmission* (DBT) method that allows using an additional slot temporarily allocated between a sender and a receiver if the sender has non-zero packets in its queue toward the receiver and there is no schedule in the next slot for both nodes. On-demand provisioning in OST [81] is similar to DBT, but it checks the schedule of a predetermined number of subsequent slots and uses the earliest available slot among them to enable additional transmission. Another approach is to adjust the amount of resources adaptively based on traffic load. In OST [81], each node measures the traffic load on its links, and allocates non-overlapping exclusive resources for each link accordingly through negotiation between nodes. A3 [83] divides a slotframe into multiple zones and assigns resources to each zone, with the number of active zones adjusted adaptively according to the measured traffic load on the links. However, none of the aforementioned approaches address the residue time problem within a timeslot.

There are other approaches that utilize residue time rather than reducing it. In [96, 97], methods are proposed to mitigate collisions in shared slots and improve throughput by performing additional collision avoidance during the residue time. However, these approaches are only effective in shared slots and the benefits may be limited when traffic load is not heavy enough to cause collisions.

Attempts have been made to aggregate application layer payloads from multiple nodes into a single frame in order to increase throughput, taking advantage of the fact that packet sizes in TSCH networks are typically short [67, 89, 98–100]. However, aggregation size is limited to a single frame and is applicable only for the same application-layer destination whereas *ASAP* is a link-layer solution that allows aggregating larger number of frames and slots. The idea of frame aggregation for throughput enhancement has been used in other domains as well. For example in Wi-Fi [101], Aggregated MAC Protocol Data Unit (A-MPDU) is a well-known approach that combines multiple data frames into a larger frame, improving transmission efficiency. Similar attempts have also been made in Bluetooth [102]. However, their focus is on reducing the header/control overhead for transmission efficiency rather than reducing residue time within a timeslot.

3.4 *ASAP* Design

ASAP consists of two orthogonal methods: *slot length adaptation* (*SLA*) and *utility-based packet aggregation* (*UPA*). *SLA* is responsible for reducing residue time by dynamically adjusting the slot length based on the current packet size distribution. *UPA* further reduces residue time by aggregating and transmitting multiple packets over consecutive slots if beneficial in terms of slot utility. *SLA* and *UPA* complement each other, achieving better time efficiency for slotted communication. The ideas in *ASAP* can generalize to many, if not all, slotted communication systems, and we case-study on the TSCH protocol in this chapter.

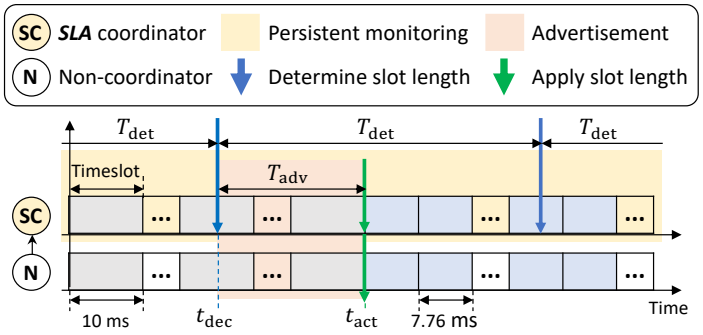


Figure 3.3: *SLA*'s slot length adaptation process

3.4.1 *SLA* Design

SLA operates in a centralized manner where a designated coordinator is responsible for managing the slot length adaptation. *SLA* operates in three phases:

1. *SLA* coordinator *persistently monitors* the packet size distribution of the network.
2. Then, it periodically *determines an appropriate slot length*.
3. If the *SLA* coordinator decides to change the slot length, the new slot length and its activation time is *advertised throughout the network*. Then, once the activation time is reached, all nodes *apply the new slot length simultaneously*.

By repeating this procedure, *SLA* adapts the slot length to the packet size distribution in the network in near real-time to reduce wasted residue time. It leverages the fact that any time-slotted system would require some form of coordinator or master that synchronizes the network. We describe the details of *SLA* using an illustration in Fig. 3.3.

Persistent monitoring: To determine an appropriate slot length for the entire network, the *SLA* coordinator needs to know the distribution of packet sizes generated throughout the network. For this purpose, we take advantage of the fact that on a multi-hop TSCH network, RPL (*IPv6 Routing Protocol for LLN*) [2, 3] is the de-facto standard

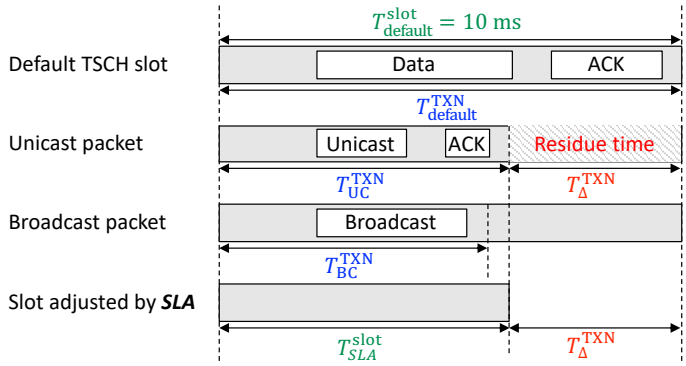


Figure 3.4: Illustration of *SLA*'s slot length decision for adjustment.

routing protocol, and the TSCH coordinator operates as an RPL root[3]. Almost all types of packets flow in and out through the RPL root under most scenarios (e.g. data collection, command dissemination, DIO/DAO, etc.), and thus the TSCH coordinator can acquire the packet size distribution of the entire network by observing the packets it sends and receives. Therefore, we assign the role of *SLA* coordinator to the TSCH coordinator and have it monitor the packet size distribution necessary for determining the slot length.

In our *SLA* design, the *SLA* coordinator quantizes the observed packet sizes into 8-byte interval bins, and uses the longest length within each bin as a representative value. This is to manage the packet size distribution in a memory-efficient manner and to prevent excessively fine-grained adjustment of the slot length. To determine for how long (T_{adv}) to advertise the new slot length and when to apply it (t_{act} in Fig. 3.3), the *SLA* coordinator also needs to know the depth of the multi-hop network (details explained shortly). For this purpose, our *SLA* design utilizes the time-to-live (TTL) field in the IPv6 header. The *SLA* coordinator monitors the number of hops each packet traverses to derive the maximum hop distance, and use it as the network depth. However, it is also possible to derive it directly from RPL's routing table.

Determination of slot length: Based on the packet size distribution collected from

[3]DODAG root in RPL's terminology

persistent monitoring, *SLA* coordinator determines an appropriate slot length periodically every T_{det}. Fig. 3.4 illustrates how this is done. An appropriate slot length is determined based on the estimated *transaction time* required for sending and receiving a packet. The default transaction time ($T^{\text{TXN}}_{\text{default}}$) in TSCH is 10 ms, long enough for exchanging a pair of maximum-sized packet and ACK. However, the actual required transaction time varies depending on the type (i.e., unicast or broadcast) and length of the packet. For instance, a unicast packet requires a transaction time of $T^{\text{TXN}}_{\text{UC}}$ for both packet and ACK, whereas a broadcast packet requires $T^{\text{TXN}}_{\text{BC}}$ without an ACK, calculated as,

$$T^{\text{TXN}}_{\text{UC}} = T_{\text{proc}} + T_{\text{offset}} + (B_{\text{UC}} + B_{\text{ACK}})/R \tag{3.4}$$

$$T^{\text{TXN}}_{\text{BC}} = T_{\text{proc}} + T_{\text{offset}} + B_{\text{BC}}/R \tag{3.5}$$

where B_{UC}, B_{BC}, and B_{ACK} represent the total bytes in each corresponding packet type, including both the packet body and control fields such as the preamble. R is the data rate of the PHY layer, which is 250 kbps for IEEE 802.15.4 PHY. T_{proc} and T_{offset} denote the total process and the total offset time within a slot, respectively, as defined in §3.2.3. We note that the sum of T_{proc} and T_{offset} is nearly constant when *SLA* is not applied (Fig. 3.1). As *SLA* does not affect these values, their sum remains constant even when *SLA* is used. Therefore, the transaction time can be determined by adjusting B_{UC} and B_{ACK}, or B_{BC} according to the desired packet size.

Then, what packet size should *SLA* adjust the slot length to? We propose a simple yet effective '*k*-th percentile policy.' From the packet size distribution, the *SLA* coordinator determines the *k*-th percentile size separately for unicast and broadcast packets. Based on their sizes, data rate, and pre-defined offset times, the coordinator calculates the transaction times of each type, denoted as $T^{\text{TXN}}_{\text{UC,k}}$ and $T^{\text{TXN}}_{\text{BC,k}}$, respectively. Finally, the coordinator selects the larger value between the two as the *target transaction time* to which to match the new slot length (T^{slot}_{SLA}). As a result, *SLA* can reduce residue time by T^{TXN}_{Δ}, the difference between the default and target transaction times.

Advertising and applying the new slot length: When the *SLA* coordinator decides to change the slot length, the new slot length must be advertised and applied throughout the network. *SLA* updates the slot length of all nodes *at once*. This design choice considers that synchronization in a TSCH network occurs across multiple hops. *Gradually changing the slot length may break the synchronization, as nodes may have to maintain synchronization with nodes with different slot lengths at the same time.* For this simultaneous activation, the *SLA* coordinator must determine when the new slot length should be applied. As illustrated in Fig. 3.3, if the *SLA* coordinator decides to change the slot length at t_{dec} and determines the advertisement duration T_{adv}, then the endpoint of this advertising period becomes the activation time as, $t_{\text{act}} = t_{\text{dec}} + T_{\text{adv}}$. Then, the new slot length and its activation time are advertised throughout the entire network during T_{adv}, and all nodes apply the new slot length simultaneously when t_{act} is reached.

Then, what should T_{adv} be? *SLA* coordinator must determine an appropriate T_{adv} that ensures all nodes in the network are aware of the new slot length and the activation time. To achieve this, we consider how our *SLA* design propagates such information. In a TSCH network, network synchronization information is propagated through a control message called *Enhanced Beacon* (EB). Starting from the TSCH coordinator, all TSCH nodes periodically transmit EBs, and each node maintains synchronization by listening to EBs. Our *SLA* design adds the new slot length and the activation time in this EB. When the new slot length and the activation time are determined, the *SLA* coordinator begins to transmit EBs containing this information. Then, all nodes that receive the new slot length and the activation time via EBs also transmit EBs including this information during T_{adv}.

Considering this advertisement procedure, *SLA* coordinator determines T_{adv} to be long enough to ensure that all nodes in the network can acquire the information. Specifically, the *SLA* coordinator uses the network depth obtained during the persistent monitoring. Assume that the network depth is h hops and every TSCH node has resources

to send EBs at a period of L_{EB} slots. Then, considering that all nodes transmit EB at every resource for EB transmission during the advertisement period, all nodes can receive at least one EB after $L_{\text{EB}} \cdot h$ slots. Based on these observations, the *SLA* coordinator calculates the required advertisement duration T_{adv} as,

$$T_{\text{adv}} = (\alpha \cdot L_{\text{EB}} \cdot h + \beta) \times (\textit{current slot length}) \text{ [ms]} \tag{3.6}$$

where α and β are coefficients considering re-transmissions[4].

Discussion: We note that the responsiveness of *SLA*'s adaptation varies depending on the size of T_{det}. Smaller T_{det} allows *SLA* to adjust the slot length more quickly in response to changes in packet size distribution. However, setting T_{det} too small can lead to excessive overhead due to frequent slot length change and advertisement, as well as the possibility of making inaccurate judgment based on too few packet size samples. Additionally, T_{det} must be set longer than T_{adv} for *SLA* to function properly. In our implementation, considering all these factors, we set T_{det} to 5 minutes, which is sufficiently longer than T_{adv} for deep networks, provides enough samples for slot length adjustment, and allows for slot length adaptation to occur multiple times (at least 10) during our experiments. However, the optimal value of T_{det} may vary depending on network characteristics or user requirements.

3.4.2 *UPA* Design

UPA presents another approach to minimize residue time by aggregating multiple packets and transmitting them over multiple (but fewer) consecutive slots to fully utilize the time within the slots. Our intuition is that if *UPA* can transmit the same number of packets with fewer slots, it would be possible to effectively reuse the previously wasted residue time in each slot and increase effective data rate.

An example: Fig. 3.5 exemplifies how *UPA* operates compared to the default TSCH. We first define *'slot utility'* as the ratio of number of packets transmitted to the number

[4] α and β are both set to 1 in our experiments.

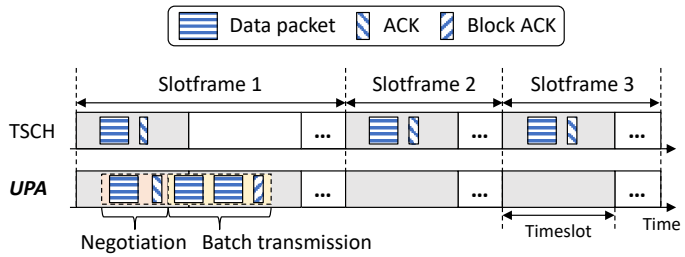


Figure 3.5: An example of *UPA*'s packet aggregation and batch transmission (bottom) compared to the default slotted operation of TSCH (top)

of slots used for those transmissions. Assume that a Tx node has three packets to send to a Rx node, and one slot is scheduled for the link in each slotframe as in Fig. 3.5. The default TSCH behaviour would be to send three packets using three slots (slot utility of one) over three slotframes. Instead in *UPA*, the Tx and Rx nodes first exchange one packet and an ACK through which they negotiate whether additional batch transmission is doable and can be advantageous in terms of slot utility.

If batch transmission is determined to be beneficial, then the Tx node sends the remaining two packets immediately in a batch, and the Rx node acknowledges the result with a *block ACK*. In this way, *UPA* finishes transmissions of three packets in two slots within a slotframe, thus increasing the slot utility from 1 to 1.5 (3 packets over 2 slots) and reducing latency to less than half (within one slotframe, see Fig. 3.5). By reducing the number of slots required per packet, this increase in slot utility allows acquiring additional resources for other transmissions and delivering multiple packets faster in terms of both datarate (less residue time) and latency (less slotframes). To realize this, *UPA* requires two phases as shown in Fig. 3.5: (1) utility-based negotiation and (2) batch transmission.

Slot utility-based negotiation: *UPA* aggregates packets only when there is a gain in terms of slot utility. To achieve this, the Tx and Rx nodes must determine whether batch transmission is doable and beneficial. However, there is an information asymmetry between the Tx and Rx. Only the Tx node knows the number of packets it wishes to

transmit and the size of each packet, and thus only the Tx can estimate the slot utility depending on the number of packets to aggregate. On the other hand, queue status of the Rx node may limit how many packets can be received in a batch, and the Tx node does not know this. Therefore, the Tx and Rx nodes first perform a negotiation whenever there is more than one packet to be sent in the Tx queue. Tx node estimates the slot utilities for varying number of aggregated packets, and informs them to the Rx node. Then, the Rx node selects the number of packets that maximizes the slot utility within its buffer availability.

Specifically, the Tx node constructs a *slot utility information* (SUI) to represent the slot utility according to the number of packets to be aggregated as exemplified in Fig. 3.6[5]. SUI consists of two bit sequences (the two 8-bit sequences shown on the right table in Fig. 3.6). The first contains the total number of packets pending in the Tx node, and the second represents an increment in the number of required slots according to the number of aggregated packets. The latter is calculated as follows: Assume total of n packets in the Tx node's queue. The time it takes for the negotiation to complete (T_{N}), i.e., the time duration until the Tx node receives an ACK from the Rx node for the first packet, is calculated as,

$$T_{\mathrm{N}} = T^{\mathrm{N}}_{\mathrm{proc}} + T^{\mathrm{N}}_{\mathrm{offset}} + (B_{\mathrm{UC}}(1) + B_{\mathrm{ACK}})/R \tag{3.7}$$

where $T^{\mathrm{N}}_{\mathrm{proc}}$ and $T^{\mathrm{N}}_{\mathrm{offset}}$ denote the total process and offset times within the negotiation phase. $B_{\mathrm{UC}}(i)$ represent the total bytes of i-th packet, including both the packet body and the control fields such as preamble. Again, R is the data rate of the PHY layer (e.g., 250 kbps for IEEE 802.15.4 PHY.)

When the Tx node sends i-th packet during a batch transmission, the additional time required until the end of transmission ($T^{\mathrm{Tx}}_{\mathrm{B}}(i)$) is calculated as,

$$T^{\mathrm{Tx}}_{\mathrm{B}}(i) = T^{\mathrm{B,Tx}}_{\mathrm{proc}} + T^{\mathrm{B,Tx}}_{\mathrm{offset}} + B_{\mathrm{UC}}(i)/R \tag{3.8}$$

[5]The example in Fig. 3.6 is simplified to a uniform packet size, but the actual implementation reflects the individual (possibly distinct) packet sizes during slot utility calculation

where $T_{\text{proc}}^{\text{B,Tx}}$ and $T_{\text{offset}}^{\text{B,Tx}}$ denote the total process and offset times while sending the i-th packet.

Once all n packets have been sent, the Rx node sends a block ACK to the Tx node. The time taken for exchanging a block ACK at the end of the batch transmission (T_{B}^{BA}) can be modeled as,

$$T_{\text{B}}^{\text{BA}} = T_{\text{proc}}^{\text{B,BA}} + T_{\text{offset}}^{\text{B,BA}} + B_{\text{BA}}/R \tag{3.9}$$

where $T_{\text{proc}}^{\text{B,BA}}$ and $T_{\text{offset}}^{\text{B,BA}}$ denote the total process and offset times while exchanging the block ACK at the end of the batch transmission. Then the total time required for transmitting the entire n packets (i.e., $T_{UPA}(n)$) via *UPA* can be derived as,

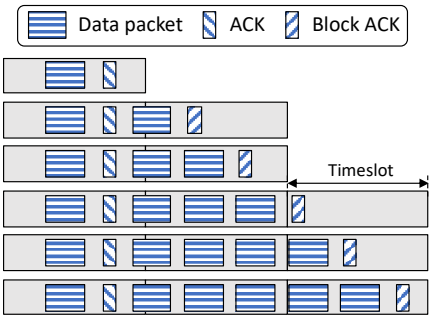
$$T_{UPA}(n) = T_{\text{N}} + \sum_{i=2}^{n} T_{\text{B}}^{\text{Tx}}(i) + T_{\text{B}}^{\text{BA}} \tag{3.10}$$

and the number of slots according to the number of aggregated packets (i.e., $S_{UPA}(n)$) is derived as,

$$S_{UPA}(n) = \lceil T_{UPA}(n)/(\textit{current slot length}) \rceil. \tag{3.11}$$

We note that B_{ACK}, B_{BA}, and all the process and offset times in Eqs. (3.7) to (3.11) are predefined constants. Therefore, the Tx node can estimate the slot utility according to the number of aggregated packets only by considering $B_{\text{UC}}(i)$. When aggregating packets, if the number of required slots (i.e., $S_{UPA}(n)$) increases, the bit of the second bit sequence corresponding to the packet count is set to one; otherwise, it is set to zero. Then, SUI is sent by piggybacking on the triggering (first) unicast packet. In the example of Fig. 3.6, five packets are pending in addition to the triggering packet, and the number of slots required increases by one when sending one additional or three more packets. Accordingly, Tx node can construct a bit sequence representing the increase in the number of slots as indicated by the red bits. Then the number of aggregated packets is represented in the first bit sequence as shown by the blue numbers.

Upon receiving the SUI, the Rx node can calculate the slot utility for each number of aggregated packets. Rx node restores $S_{UPA}(n)$ for each value of n from the received



Number of packets	Bit sequence	Slot utility
1	00000001 00000000	1
2	00000010 01000000	1
3	00000011 01000000	1.5
4	00000100 01010000	1.33
5	00000101 01010000	1.67
6	00000110 01010000	2

Figure 3.6: Example of impact of *UPA*'s packet aggregation on slot utility and bit sequence.

SUI, and calculates the slot utility as $n/S_{UPA}(n)$. Then, the Rx node selects the number of packets that maximizes the slot utility within its buffer limit (l), by solving the following problem:

$$\underset{2\leq n\leq l}{\operatorname{argmax}} \frac{n}{S_{UPA}(n)} \tag{3.12}$$

It chooses zero if there is no way to improve the slot utility. Then the selected number of packets to aggregate is conveyed to the Tx node through ACK. Since the negotiation information are piggybacked on unicast and ACK packets, no additional control packet is required.

Batch transmission: Upon agreement from the negotiation, the Tx and Rx nodes start batch transmission and reception. For each batch, the Tx node assigns a separate batch sequence number to the packets. Then, since the Rx node knows in advance how many packets will be received in the current batch, it can detect packet loss(es) using this sequence number. Reception status of the packets within a batch is then converted into a bit sequence, and sent back as a *block ACK*. Upon receiving the block ACK, Tx node finds the transmission result of each packet from the bit sequence, and re-enqueues the lost packets for retransmission.

Discussions for *UPA*: Since the batch transmission of *UPA* is performed beyond the slot originally scheduled for the Tx and Rx nodes of *UPA*, it may overlap with the schedules of other links. If the overlapped schedules are executed simultaneously,

a collision will occur. The collision can reduce the number of packets successfully delivered by *UPA*, decreasing the slot utility. Therefore, we enable non-participants to avoid collision with *UPA*'s batch transmission through clear channel assessment (CCA). Specifically, we modify and apply the CCA operation proposed in [53] to fit the operation procedure and timing of *UPA*.

3.5 Evaluation

In this section, we evaluate the effectiveness of *ASAP* and its key techniques. First, we investigate whether each individual method, *SLA* and *UPA*, operates properly according to its design purpose. Then, we conduct an ablation study to verify the efficacy of *ASAP* by examining the synergistic collaboration between *SLA* and *UPA*. Finally, we compare *ASAP* against two state-of-the-art TSCH schedulers, ALICE and ALICE with A3 (referred to as A3 in the rest of this chapter), in addition to ALICE with the IEEE 802.15.4 *default burst transmission* (DBT).

3.5.1 Implementation and experiment setup

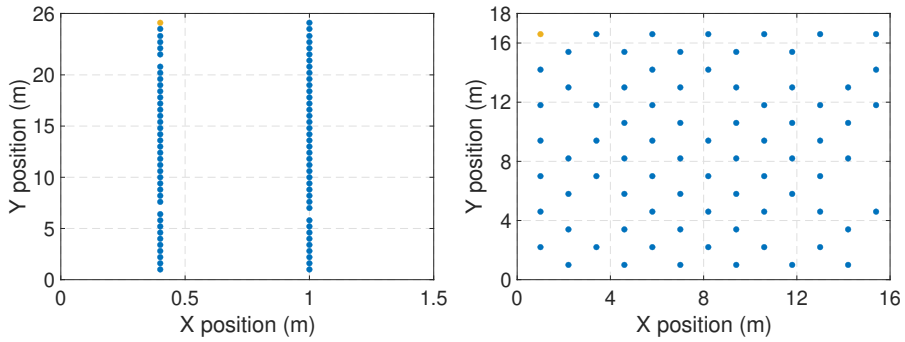
We implement *ASAP* on M3 board using Contiki-NG[6] [85]. For the comparison schemes, we use the publicly available implementations of ALICE[7], DBT[8], and A3[9], which are also implemented in Contiki-NG. The slot length is set to 10 ms by default, and we set the slotframe lengths of EB, broadcast, and unicast slotframes to 397, 17, and 20, respectively, referring to the configuration in A3 paper [83]. For channel hopping, we utilize the four IEEE 802.15.4 channels 15, 20, 25, 26. For the routing layer, we use RPL storing mode [103] and MRHOF [38] with ETX [104] for the objective function in RPL. We enable the DAO-ACK option in RPL to promote reliable downward rout-

[6]https://github.com/iot-lab/iot-lab-contiki-ng

[7]https://github.com/skimskimskim/ALICE

[8]It is included in the implementation of Contiki-NG.

[9]https://github.com/skimskimskim/A3



(a) Grenoble testbed **(b)** Lille testbed

Figure 3.7: Node deployment topology at Grenoble and Lille testbeds. In both testbeds, the node located in the upper left corner and marked in yellow serves as the root.

ing and resource scheduling. We set the transmission power of each node to -17 dBm.

We conduct experiments on the FIT/IoT-LAB testbed [86], a large-scale public testbed, at three sites: Lyon, Grenoble, and Lille. Lyon testbed was used for the preliminary study in Fig. 3.1 and the evaluation of *UPA* in §3.5.3. Grenoble and Lille testbeds were used to assess the performance of *ASAP* in a multi-hop topology utilizing 79 M3 nodes at each site. Fig. 3.7 depicts the physical deployment topology of the 79 nodes at Grenoble and Lille. At Grenoble, the nodes are arranged in two narrow and long rows, while at Lille, the nodes are distributed almost evenly in a rectangular grid shape. Approximately 8-hop routing topology is formed at Grenoble, while a 3-4 hop topology is formed at Lille. Through experiments in these two sites with very distinct characteristics, we comprehensively verify the performance of *ASAP*.

With this setup, we focus on two application scenarios: data collection (upward traffic) and data dissemination (downward traffic) with varying traffic loads. In the upward scenario, each node periodically transmits data packets to the root node. In the downward scenario, the root transmits data packets to each non-root node periodically in a round-robin fashion. Unless specified explicitly, data transmission lasts for 30 minutes in each experiment run, and sufficient time is provided for initialization and bootstrap before starting data transmission. For each experimental case, we repeat the

experiment three times.

Here we list the key performance metrics for evaluation:

- **App. layer goodput** is the end-to-end per-minute packet goodput for each node excluding losses and retransmissions.
- **Per-hop latency** is obtained by dividing the end-to-end latency of each data packet by the path length it traverses.
- **End-to-end packet delivery ratio (PDR)** is the PDR of data packets from/to each node to/from the root node.
- **Slot length** is calculated as the average length of the slots in which data packet transmission occurs.
- **Slot utility** is calculated as the number of packets transmitted per slot with *UPA*.
- **Duty cycle** is calculated as the ratio of the time the radio is on to the total operating time of the network.

3.5.2 Performance of *SLA*

We first verify the proper functioning of *SLA*'s slot length adaptation across the entire network, and evaluate the impact of k in k-th percentile policy in a data collection scenario at the Grenoble site.

Network-wide slot length adaptation: To verify whether *SLA* operates correctly at run-time across the network, we conduct a four-hour experiment in which the packet size varies over time. During four hours, each node periodically transmits a total of 960 data packets to the root node while changing the payload length to 63, 14, 46, and 30 bytes every 240 packets. We set the relevant *SLA* parameters as follows: T_{det} is set to 5 minutes, and α and β are both set to 1, which are used to determine T_{adv} based on the depth of the network. We apply the same parameters in the subsequent experiments. The k for the k-th percentile policy is set to 90. TSCH slot length is initially set to 10 ms.

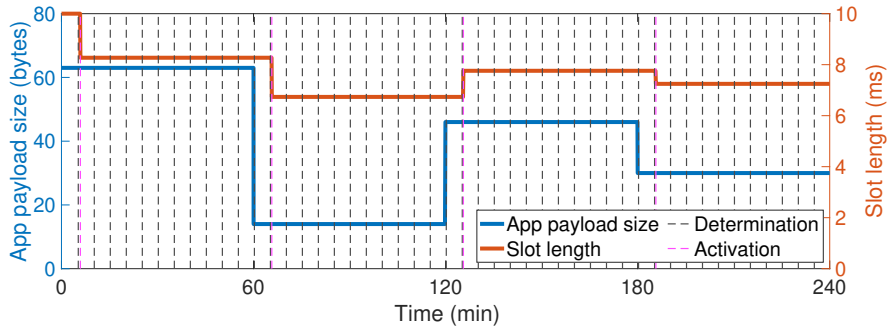
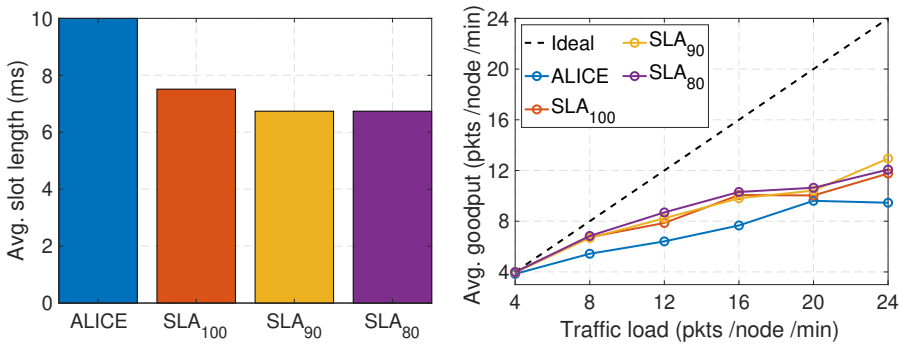


Figure 3.8: Real-time operation of *SLA*: Evolution of slot length over time in accordance to payload (packet) size changes.



(a) TSCH slot length

(b) App layer goodput

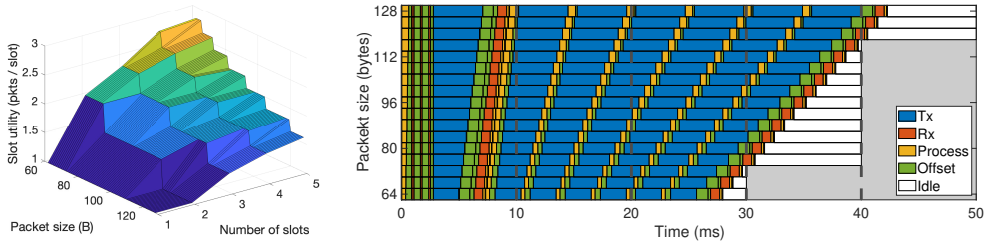
Figure 3.9: Performance of *SLA* with varying k-percentile values vs. ALICE, at the Grenoble testbed, for data collection scenario.

Fig. 3.8 plots the time-evolution of slot length in accordance to payload size changes over time, together with the timing of *SLA*'s slot length determination and activation actions. *SLA* coordinator continuously monitors the packet size distribution and periodically determines the appropriate slot length, as indicated by the black dashed line. Whenever there is a change in the distribution, the *SLA* coordinator detects it and adjusts the slot length accordingly. The *SLA* coordinator also determines the activation time, represented by the pink dashed line. After all nodes in the network advertise the new slot length and activation time, the new slot length is applied simultaneously at the designated time. The overall result demonstrates that *SLA* successfully adjusts the

slot length whenever the packet size decreases or increases. We note that at the first determination point of *SLA*, the slot length decreases even though the packet length remains the same. This happens because the original slot length of 10 ms is adjusted to fit the data packets of 63 bytes payload.

Choice of an appropriate *k* for the *k*-th percentile policy: The *k*-th percentile policy serves as a reference for adjusting the slot length based on the packet size distribution, and impacts the performance of *SLA*. Therefore, choosing an appropriate *k* value is crucial. In most of our experiments, a single-size data packet is used. While data packets constitute the majority of total packets, there are also RPL control packets (DIS, DIO, DAO, and DAO-ACK) and TSCH control messages (e.g., EB), which may be larger or smaller than the data packet size. Thus, we need to find an appropriate *k* value that results in a good slot length for improved performance.

Fig. 3.9a plots the average adjusted slot length according to the value of *k* in the data collection (upward) scenario at the Grenoble testbed when the traffic load is 4 packets per minute per node. The payload size is 14 bytes, resulting in a maximum data packet size of 67 bytes, with an ACK of 20 bytes. If these sizes are used as a reference, *SLA* adjusts the slot length to 6.736 ms (§3.4.1). We found that a value of *k*=90 is appropriate for *SLA* to adjust the slot length to the data packet size. *k* larger than 90 (e.g., 100) will end up setting the slot length to the infrequent but large control packets, while smaller *k* has no effect due to large number of data packets. Therefore, we set *k* to 90 in the subsequent experiments. For example, Fig. 3.9b plots the goodput of *SLA* with *k*=90 and ALICE as a function of traffic load (same experiment as Fig. 3.9a). By dynamically adjusting the slot length, *SLA* improves goodput compared to ALICE which uses a fixed 10 ms slot. In a more complex scenario where data packets of several different sizes coexist, selecting an optimal value for *k* can be a more challenging issue. We leave this as a future work.



(a) Slot utility vs. packet size and num. of slots

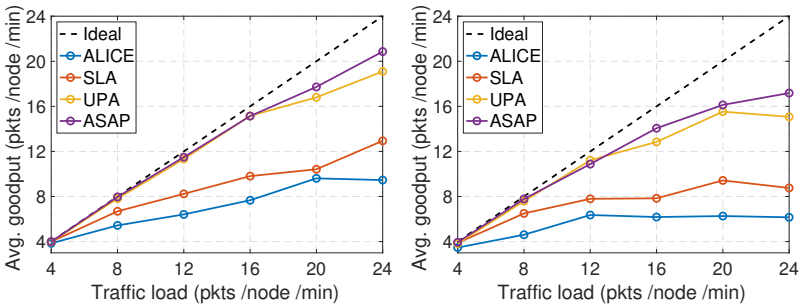
(b) Aggregating seven packets in *UPA*. Seven packets can be sent in three∼five slots.

Figure 3.10: Performance of *UPA*: maximum slot utility and an example of aggregating seven packets to reduce residue time.

3.5.3 Performance of *UPA*

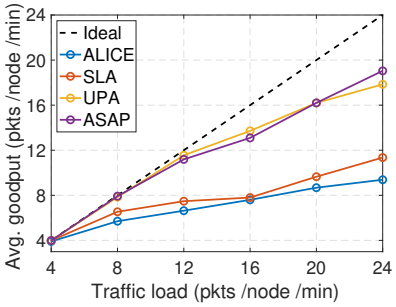
Next, we measure the maximum achievable slot utility of *UPA* depending on the packet size and number of slots, and also investigate the time usage breakdown of *UPA* to attain insights into how the slot utility gain is achieved. For these experiments, we use two nodes at the Lyon site and fix the TSCH slot length to 10 ms.

Fig. 3.10a plots the maximum achievable slot utility of *UPA* as a function of packet size and number of slots used for aggregation. For example, if *UPA* utilizes five consecutive slots, slot utility can go up to 2.8 by aggregating 14 minimum-sized packets. This figure can guide *UPA* in terms of when and how many packets to aggregate. It shows that *UPA* can achieve slot utility of 1.5∼2.8 in most cases, and smaller packets provide more opportunity for higher improvement. This analysis is corroborated by the time usage breakdown of aggregating seven packets as illustrated in Fig. 3.10b. Following the triggering packet and ACK, six packets are transmitted in a batch with very short intervals, reducing the number of slots required from seven to three∼five depending on the packet length. For example, for the seven minimum-sized packets, *UPA* can improve the slot utility to 2.33 by utilizing 3 consecutive slots.

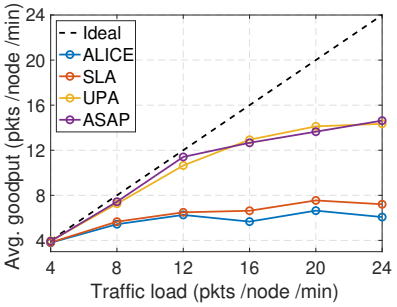


(a) Goodput, 14 bytes, upward

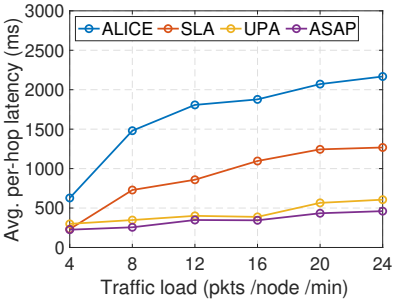
(b) Goodput, 14 bytes, downward



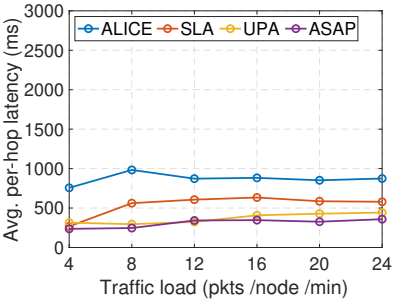
(c) Goodput, 63 bytes, upward



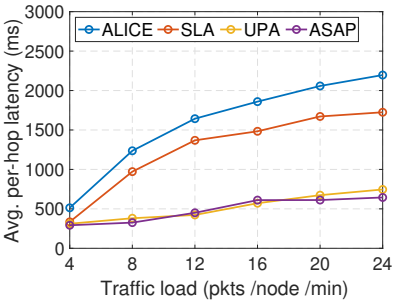
(d) Goodput, 63 bytes, downward



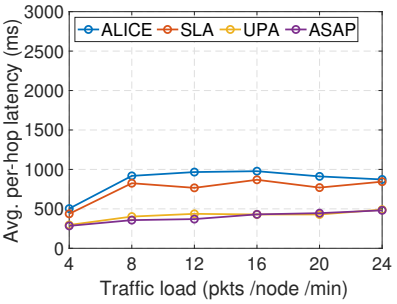
(e) Latency, 14 bytes, upward



(f) Latency, 14 bytes, downward



(g) Latency, 63 bytes, upward



(h) Latency, 63 bytes, downward

Figure 3.11: Ablation study on *ASAP*: Goodput and latency results for upward vs. downward scenarios and 14 vs. 63 byte payloads

3.5.4 Performance of *ASAP*: an ablation study

Next, we evaluate the performance of *ASAP* through an ablation study with varying traffic load (from 4 to 24 packets per minute per node) and payload length (14 bytes and 63 bytes) in upward and downward traffic scenarios. We use ALICE as the baseline scheduler and compare the performance of 'ALICE with *SLA*', 'ALICE with *UPA*', and 'ALICE with *ASAP*' (referred to as *SLA*, *UPA*, and *ASAP*, respectively, hereafter) with ALICE. The experiments are conducted at Grenoble.

Application layer goodput: Fig. 3.11a plots the goodput in the data collection (upward) scenario with the minimum payload length (14 bytes). ALICE performs the worst beyond traffic load of 4 pkts/min/node, and fails to deliver a significant number of packets as the traffic load increases. This is because, the traffic load was higher than ALICE's effective data rate at the bottleneck nodes in most cases. *SLA* improves over ALICE, demonstrating that *SLA* has successfully increased the effective data rate by adjusting the slot length. *UPA* exhibits even more improvement, showing that *UPA* has successfully increased the effective data rate through packet aggregation. *UPA* achieves higher improvement than *SLA* because *UPA* can flush the bottleneck node's queue backlog in a burst within a slotframe whereas *SLA* still requires same number of slotframes as the number of packets. The collaboration of *SLA* and *UPA* within *ASAP* leads to further improvement outperforming *UPA* as the traffic load increases.

Per-hop latency: Fig. 3.11e plots the latency result in the collection (upward) scenario with the smallest payload length (14 bytes). *SLA* achieves lower latency than ALICE by adapting the slot length to be shorter. *UPA* further reduces latency since its packet aggregation allows each packet to be transmitted significantly earlier than the baseline schedule. By harmonizing the effects of these two techniques, *ASAP* achieves the lowest latency.

Synergy of *SLA* and *UPA* in *ASAP*: Here we discuss the combined benefits of *SLA* and *UPA*. As the slot length decreases (by *SLA*), the utility of packet aggregation (by

UPA) diminishes due to fewer packets fitting in smaller slots. Therefore, *UPA* becomes slightly less effective when used together with *SLA* compared to *UPA* alone. However, *SLA* increases the frequency of resource repetitions (i.e., next slotframe comes earlier), creating more opportunities for transmissions and aggregations in the time domain. This compensates for the reduced slot utility. Furthermore, *SLA*'s shorter slot length and more frequent opportunities improve latency by enabling quicker transmission for all nodes. Overall, *UPA* and *SLA* together create a synergy to improve both throughput and latency beyond what can be done by each technique alone.

Impact of traffic direction: In the downward traffic scenario, the root node generates and disseminates downward traffic throughout the network. Since packets are distributed to multiple nodes, aggregation opportunity in each link may decrease, reducing the slot utility. Furthermore, the root is the main bottleneck of throughput, resulting in an overall decrease in goodput across all schemes as shown in Fig. 3.11b. Nevertheless, *SLA*, *UPA*, and *ASAP* still outperforms ALICE for the same reasons discussed earlier, with *ASAP* having the highest performance. Latency improves for the downward scenario as well, as shown in Fig. 3.11f. However, in contrast to the upward scenario, downward traffic has relatively constant per-hop latency. This is because the root node is the bottleneck, and the nodes below the root forward fewer packets than the root and rarely experience congestion. Thus, the per-hop latency does not change significantly per traffic load.

Impact of packet length: We lastly explore the impact of packet length on the performance of *ASAP* using experiments with a payload size of 63 bytes. The results are plotted in Figs. 3.11c, 3.11d, 3.11g and 3.11h. We observe that the trends for goodput and latency remain unchanged from the experiments with 14-byte payload, in both upward and downward traffic scenarios. In the case of ALICE, the goodput and latency remain almost the same because the packet length difference does not affect ALICE's operation. However, for *SLA*, *UPA*, and *ASAP*, the goodput and latency degrade slightly as the packet size increases. This is because, with longer packets, *SLA* cannot shorten

the slot length as much as with shorter packets, and *UPA* cannot aggregate as many packets. Nevertheless, *SLA* and *UPA* still achieve significantly better performance than ALICE, and so does *ASAP*.

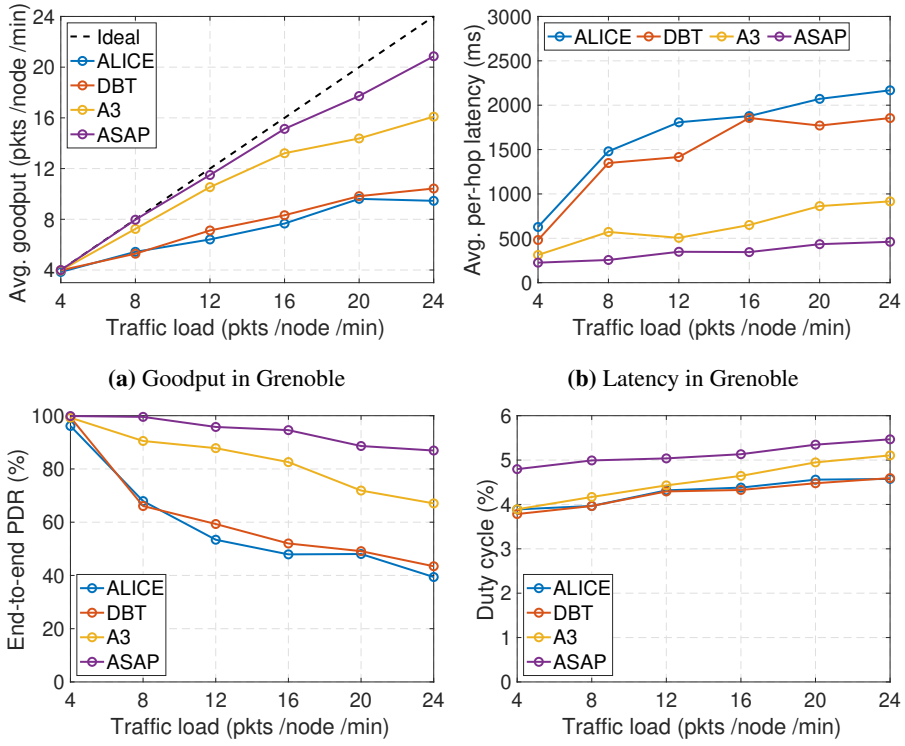
Summary: Overall, the results validate the superiority of *ASAP*. Across all scenarios, irrespective of traffic direction and packet size, the proposed schemes improve effective data rate by addressing the inefficiencies of ALICE resulting from wasted residue time, leading to enhanced goodput and latency. Notably, by leveraging the synergetic strengths of both *SLA* and *UPA*, *ASAP* achieves the best performance. Specifically, at a traffic load of 24 pkts/min/node, *ASAP* improves throughput and reduces latency of the TSCH network by 2.21x and 78.1%, respectively, compared to those of ALICE.

3.5.5 Performance of *ASAP*: a comparative study

We now compare the performance of *ASAP* against other state-of-the-art schedulers, namely ALICE, DBT, and A3. For this purpose, we run experiments for the data collection scenario (i.e., upward traffic) at both Grenoble and Lille sites. While varying the traffic load, we measure the application layer goodput, per-hop latency, end-to-end PDR, and duty cycle.

Overall performance: As shown in Fig. 3.12, *ASAP* significantly outperforms all three baseline schemes in terms of goodput, latency, and PDR. DBT performs similarly to ALICE, with only minor improvements in some cases. A3, on the other hand, exhibits notably better performance than ALICE and DBT. Nevertheless, *ASAP* has significantly better performance especially at higher traffic load. However, *ASAP* exhibit a slightly higher duty cycle than other schemes. This is because *ASAP* delivers more packets successfully; That is, the radio on time is used for useful work, significantly improving throughput.

Comparison with DBT: It is worth noting that DBT attempts to send additional packets by recursively allocating additional resource when a queue backlog occurs and there is no schedule in the subsequent slot. This approach does provide DBT with more



(a) Goodput in Grenoble

(b) Latency in Grenoble

(c) PDR in Grenoble

(d) Duty cycle in Grenoble

Figure 3.12: Performance comparison of *ASAP*, ALICE, DBT, and A3 at Grenoble testbed with dual-linear topology. *ASAP* achieves better goodput, latency, and PDR with only a slight increase in duty-cycle ($<1\%$)

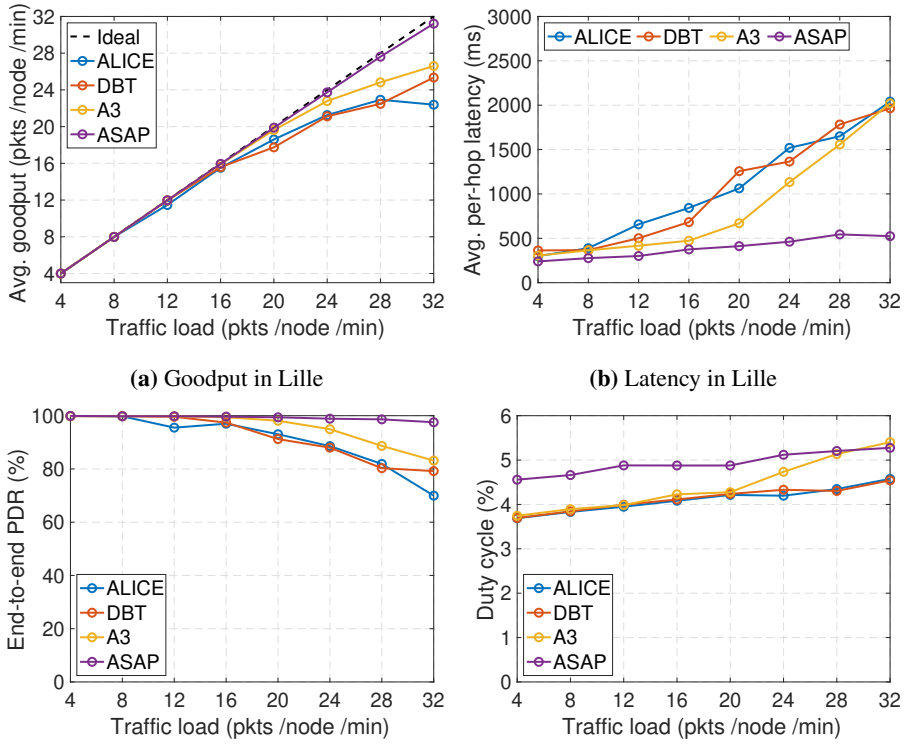
opportunities to send packets than ALICE. However, it does not cope with residue time within each timeslot. Moreover, when an existing schedule is present in the subsequent slot, DBT prioritizes it instead of allocating additional resource, resulting in limited improvements in reducing queue backog or alleviating congestion at the bottleneck compared to ALICE. In contrast, *ASAP* exhibits superior performance compared to DBT (Figs. 3.12a to 3.12c), primarily due to its *UPA* mechanism; i.e., it enables more packets to be transmitted in fewer slots than DBT. Moreover, *ASAP* prioritizes transmissions with high slot utility over the existing schedule, which significantly enhances network time-efficiency.

Comparison with A3: A3 assigns additional periodic resources based on traffic load, allowing for transmission of more packets compared to ALICE. In our experiments, A3 can allocate up to four times more resources than ALICE when needed. For this reason, A3 significantly improves goodput, latency, and PDR performance compared to both ALICE and DBT. Nevertheless, *ASAP* achieves even better performance than A3. This is because, as the traffic load increases, A3's additional resource allocation will eventually reach its limit where it becomes impossible for A3 to send more packets. It is important to highlight that A3's approach of allocating additional resources is orthogonal to *ASAP*'s approach of reducing residue time. There may still be a significant waste of residue time within each slot, and *ASAP* can reduce residue time and allow A3 to send more packets in such a situation. Another limitation of A3 is the occurrence of conflicts due to overlapping resources from different links when allocating more resources to support higher traffic load. In such situations, efficiently utilizing residue time within each slot may be a better solution than allocating more resources.

3.5.6 Performance of *ASAP* in different environment

Finally, we validate whether *ASAP* maintains good performance in a different environment with drastically different physical topology. For this purpose, we evaluate ALICE, DBT, A3, and *ASAP* at the Lille testbed which has a compact grid deployment (Fig. 3.7b) unlike the long dual-linear topology of the Grenoble site (Fig. 3.7a). The network topology characteristics of the two locations are markedly different. At Lille, the hop distance is reduced by about half compared to Grenoble, and the bottleneck problem becomes less severe, but more nodes are connected to the root directly. Considering this topological difference, we increase the traffic load up to 32 pkts/min/node at Lille from that of 24 at Grenoble. By comparing Fig. 3.12 and Fig. 3.13, we can analyze the impact of topology on *ASAP* and other compared schemes.

As shown in Fig. 3.13, the performance trends among ALICE, DBT, A3, and *ASAP* at the Lille site are similar to those observed at the Grenoble site; *ASAP* outperforms



(a) Goodput in Lille

(b) Latency in Lille

(c) PDR in Lille

(d) Duty cycle in Lille

Figure 3.13: Performance comparison of *ASAP*, ALICE, DBT, and A3 at Lille testbed with grid-like topology. *ASAP* achieves better goodput, latency, and PDR with only a slight increase in duty-cycle (<1%)

all other schemes. However, at the Lille site, ALICE, DBT, and A3 perform well even at much higher traffic loads than those at the Grenoble site. This is due to the shallower topology where many nodes can directly deliver packets to the root node, thus have less resource shortage issues. Nonetheless, both ALICE and DBT experience a decrease in goodput and an increase in latency at a traffic load of 20, while A3 experiences these issues at traffic load of 24. In contrast, *ASAP* maintains goodput close to ideal and minimizes latency increase even at traffic load of 32. Regarding the duty cycle, similar trend but slightly lower energy consumption is observed at the Lille testbed for the same reason of shallower network depth. To sum up, *ASAP* has shown

successful performances in both linear and grid shape topologies. Based on the results, we believe that *ASAP* can generalize to other topologies and perform well in various environments.

3.6 Summary

Time-slotted communication systems set the timeslot length to be sufficient for transmitting a maximum-sized packet and an ACK, resulting in a significant residue time within each timeslot and decrease in effective data rate. To address this problem, this chapter proposed *ASAP* with two orthogonal approaches: *SLA* and *UPA*. *SLA* reduces residue time in timeslots by adjusting the length based on the packet size distribution, while *UPA* reduces residue time by aggregating multiple packets and transmitting them in a burst over consecutive slots. Through a case study on TSCH, we verified that *ASAP* improves performance significantly compared to the state-of-the-art TSCH schemes such as ALICE, A3 and DBT. Specifically, *ASAP* improves throughput by up to 2.21x and reduces latency by up to 78.7% through a synergistic collaboration of *SLA* and *UPA*. As a future work, we plan to investigate the performance of *ASAP* when applied in combination with other orthogonal approaches (e.g., A3) and explore ways to improve the performance of *ASAP*.

Chapter 4

Offset-based Prioritization for Accelerating Formation of 6TiSCH Networks

4.1 Introduction

6TiSCH [105], which is IPv6 over the TSCH mode of IEEE 802.15.4e [57] networks have attracted considerable attention for their applicability in diverse industrial and IoT applications. However, the process of network formation in 6TiSCH networks poses challenges, particularly in terms of efficiency and collision management [106].

Efficient network formation is crucial as it directly impacts the overall performance and reliability of the network. Formation in 6TiSCH networks encompasses the process of joining at the TSCH layer, joining at the RPL layer, and the allocation of resources (i.e., cells) for exchanging data packets. During the 6TiSCH network formation process, all nodes communicate through the common shared cell, which is a shared resource accessible to all nodes, as resource allocation has not yet been established. However, this can lead to collisions and congestion in the common shared cell, resulting in packet loss, retransmissions, and severely delayed network formation time.

In relation to the congestion issue in the common shared cell during network formation, we have observed that the highly synchronized transmission start times among

nodes within the cell significantly impede collision detection and congestion mitigation. To address this challenge, we propose *TOP*, a transmission offset-based prioritization technique to improve the efficiency of 6TiSCH network formation. The core concept is to allocate offsets to packets during their transmission initiation, providing distinct starting points for packets and minimizing the likelihood of collisions.

TOP comprises several components. Firstly, we introduce an offset assignment policy that allocates offsets based on packet urgency, distinguishing between unicast and broadcast packets. Secondly, we propose an adaptive offset adjustment mechanism that dynamically modifies offsets based on network conditions, aiming for even faster packet delivery. Lastly, we modify the backoff mechanism of TSCH to prevent collisions and enable the offset-based prioritization to fulfill its intended performance objectives. These three components collaborate in unison to hasten the process of network formation in 6TiSCH networks.

We highlight that the congestion issue in the common shared cell can be a significant concern not only during the network formation phase but also in the operational phase after network formation is completed. For example, in the event of a specific node experiencing an anomaly (e.g., a security attack or battery depletion) that necessitates other nodes to find new routing paths or synchronization targets, these nodes will rely on the common shared cell for communication, thereby encountering the aforementioned issues. Even in such scenarios, *TOP* can serve as an effective countermeasure.

We implement *TOP* on real embedded IEEE 802.15.4 devices using Contiki-NG [85], and evaluate it on a large-scale topology in the FIT/IoT-LAB public LLN testbed [86]. The results demonstrate substantial improvements in network formation time, with reductions of up to 50%. *TOP* effectively mitigates collisions, congestion, and packet losses, all of which contribute to the accelerated network formation process.

Our contributions can be summarized as follows.

• We identify collisions in the common shared cell as a significant factor causing

network formation delays.

- We highlight that the specific characteristic of closely aligned transmission start times in TSCH exacerbates the challenges in collision management.
- We propose *TOP*, an transmssion offset-based prioritization technique for efficient network formation in 6TiSCH networks.
- We implement *TOP* on real embedded devices, and evaluate in a sizeable public testbed to demonstrate significant performance improvement.

The remainder of this chapter is organized as follows. We present the background and motivation in §4.2, and §4.3. We present our approach in §4.4, and the design of the proposed scheme in §4.5. We evaluate the proposed scheme in §4.6. We discuss related work in §4.7. Finally, §4.8 concludes the chapter.

4.2 Background

4.2.1 Time-Slotted Channel Hopping (TSCH)

TSCH is a MAC protocol standardized in IEEE 802.15.4e that integrates time-slotted communication and channel hopping. By synchronizing the network and enabling devices to communicate in a time-slotted manner, TSCH enhances reliability and energy efficiency. The utilization of channel hopping supports resilience to external interference and fading by leveraging channel diversity.

As shown in Fig. 4.1, TSCH divides time into *timeslots*. The duration of a timeslot is typically set to 10 ms, allowing for the exchange of a maximum-sized (128 Bytes) frame and an ACK of up to 70 Bytes. Each timeslot is assigned an *absolute slot number (ASN)*, which starts at zero when the network begins and increments sequentially. A collection of timeslots forms a *slotframe*, which repeats over time and serves as a scheduling unit in TSCH. The number of timeslots within a slotframe is referred to as the slotframe length (L_{SF}). Then, *time offset* (t_o) represents the relative position of a

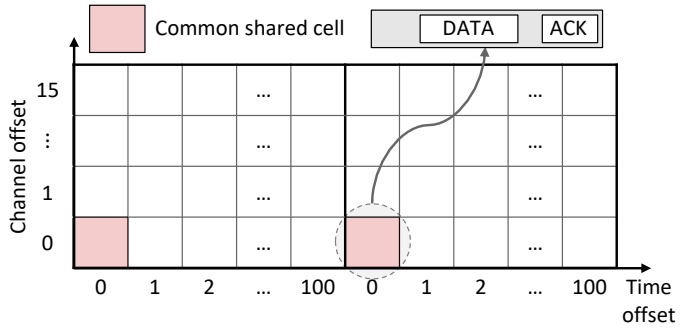


Figure 4.1: TSCH common shared cell with slotframe size of 101.

particular timeslot within a slotframe and is calculated as,

$$t_o = mod(ASN, L_{\mathrm{SF}}). \tag{4.1}$$

To determine the channel to be used for channel hopping, TSCH utilizes a *channel offset* (c_o) within its schedule. The specific channel for each timeslot is determined through a calculation based on the aforementioned channel offset as,

$$Channel = List_{\mathrm{c}}[mod(ASN + c_{\mathrm{o}}, sizeof(List_{\mathrm{c}}))] \tag{4.2}$$

where $List_{\mathrm{c}}$ is a set of channels to be used and $sizeof(List_{\mathrm{c}})$ is the number of channels in $List_{\mathrm{c}}$. As ASN increases, each timeslot with a specific c_o hops over different channels. Even with the same ASN timeslot, different c_o results in the selection of different channels.

4.2.2 TSCH scheduling and common shared cell

The TSCH standard specifies the principles of time-slotted communication and channel hopping, but it does not provide a specific solution for resource scheduling, which involves determining the timing and channel selection for each device's communication. To fill this gap, several TSCH schedulers have been proposed [77, 79, 81–83, 94]. In this study, we mainly consider ALICE [79], a state of the art TSCH scheduler.

However, in order to enable communication both prior to scheduling and during the scheduling process, a minimal communication path is required. Therefore, in 6TiSCH protocol, a common shared cell is established as the essential resource, allowing all nodes to communicate on the same channel at the same time. Fig. 4.1 presents an example scenario where a common shared cell is positioned at time offset 0 and channel offset 0 within a common shared slotframe of size 101.

While the specific usage of the common shared cell may vary depending on the functioning of the scheduler, it remains consistent that when communicating with nodes that have not been allocated dedicated schedules or when attempting resource scheduling, communication must pass through this common shared cell.

4.2.3 6TiSCH network and formation

A 6TiSCH network is an IPv6 communication network implemented on top of the IEEE 802.15.4e TSCH mode. It involves the integration of various standard protocols across different layers, as illustrated in Fig. 4.2. Specifically, the TSCH protocol is utilized as the MAC protocol at the data link layer, while the RPL protocol is adopted for the routing layer. For the 6TiSCH network to operate fully, network formation, which includes join processes over multiple layers and resource scheduling, must be carried out.

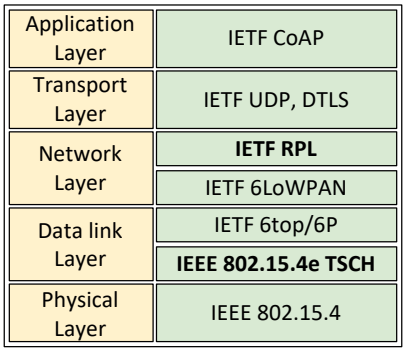


Figure 4.2: 6TiSCH network stack.

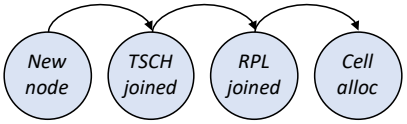


Figure 4.3: An example of state transition during 6TiSCH network formation.

The network formation process in 6TiSCH involves the sequential transition of nodes through different states. Typically, as shown in Fig. 4.3, the states include the new node, *TSCH joined*, *RPL joined*, and *cell allocated* states. Initially, in the new node state, a node waits for TSCH Enhanced Beacons (EBs) broadcasted by the nodes in the *TSCH joined* state, such as the TSCH coordinator. When an EB is received, the node extracts TSCH network information, including synchronization timing, and joins the TSCH network (*TSCH joined* state), enabling participation in time-slotted communication.

The node then listens for RPL DODAG Information Objects (DIOs) broadcasted by the nodes in the *RPL joined* state, such as the RPL root. This allows the node to acquire routing information and join the RPL network (*RPL joined* state). In *RPL joined* state, the node can send Destination Advertisement Object (DAO) packets to its RPL parent node to establish downward routes. In our considered network, where the ALICE scheduler is applied, successful DAO transmission allocates dedicated cells for links between the node and its RPL parent. That is, by successfully transmitting DAO packets, the node enters the *cell allocated* state, and it can utilize these dedicated cells for efficient and coordinated communication within the 6TiSCH network.

As aforementioned, the transition between these states relies on the exchange of specific packets, including EBs, DIOs, and DAOs. Hence, to ensure the efficient progression of 6TiSCH network formation, it is crucial that the formation-critical (i.e., urgent) packets are exchanged in a timely manner, allowing each node to successfully complete its state transitions.

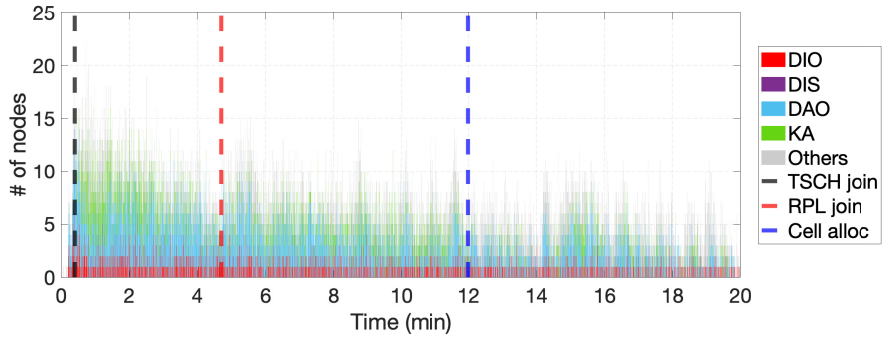


Figure 4.4: Time evolution of packet transmissions within the common shared cell and the time when the state transition of all nodes completed.

4.3 Motivation

The 6TiSCH network offers several advantages, such as deterministic latency, low energy consumption, and high reliability. However, to fully utilize these benefits, network formation is a prerequisite. Efficient network formation plays a crucial role in the timely deployment and operation of 6TiSCH networks. Nevertheless, we have identified challenges in network formation within the 6TiSCH framework, primarily due to the congestion issues in the common shared cell.

To elaborate, the common shared cell serves as the primary communication path during network formation, which leads to significant collisions within this shared resource. Fig. 4.4 illustrates the number of nodes attempting packet transmissions in the common shared cell over time. The different colors represent the various types of transmitted packets. Additionally, it shows the time taken for all nodes to reach the *TSCH joined* state, *RPL joined* state, and *cell allocated* state. This analysis was conducted using a real testbed comprising 79 nodes, where network formation was performed with a common shared slotframe size of 31.

The result reveals that immediately after network initiation, a substantial number of nodes engage in packet transmissions within the common shared cell. In some cells, more than 20 nodes attempt packet transmissions. However, the experimental results

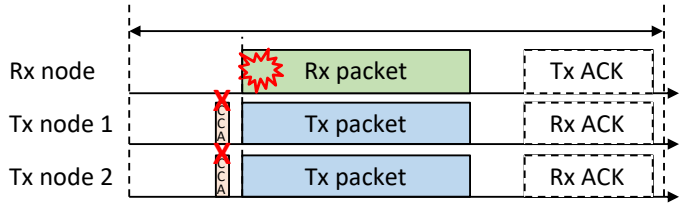


Figure 4.5: Collision is inevitable in common shared cell due to the closely aligned transmission start times.

indicate a notably low success rate for these transmitted packets. As a result, it takes more than 12 minutes for all nodes to reach the *cell allocated* state.

What exacerbates the problem further is that the transmission mechanism of TSCH makes it challenging to effectively mitigate the collision issue in the common shared cell. As depicted in Fig. 4.5, in TSCH, packets transmitted within the same slot have closely aligned transmission start times. Despite the inclusion of clear channel assessment (CCA) to determine channel availability prior to packet transmission, nodes in the network are unable to detect collisions between packets. As a consequence, TSCH nodes repeatedly attempt transmissions without detecting collisions, aggravating the congestion problem.

Nevertheless, addressing the challenges of common shared cell and optimizing the network formation process are crucial for achieving reliable and efficient communication in 6TiSCH networks. This motivates us to proposes *TOP*, a transsmision offset-based prioritization technique to mitigate collisions and prioritize urgent packets, aiming to expedite network formation and improve overall performance of 6TiSCH networks.

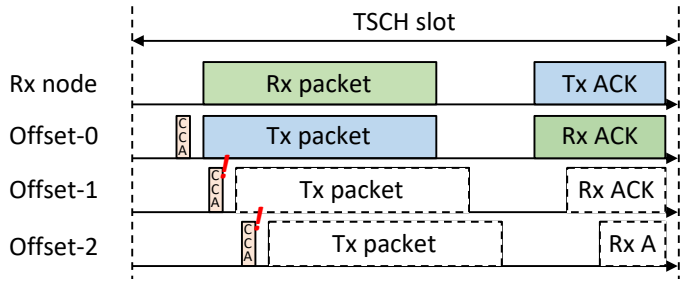


Figure 4.6: Assigning differentiated offsets to the packet transmission start times enables the detection of collisions among packets.

4.4 Approach and Considerations

4.4.1 Approach: Transmission offset-based prioritization

To address the issue of unavoidable collisions in the common shared cell, we employ a transmission offset-based prioritization technique. Fig. 4.6 provides an example of assigning differentiated transmission offsets to multiple packets within the common shared cell. By dispersing the transmission start times of the packets, it becomes possible for each packet to detect the presence of preceding packets through CCA. This enables the postponement of subsequent packet transmissions, effectively mitigating the collision problem that was previously inevitable and promoting the packet delivery within the common shared cell.

Furthermore, taking into consideration that packets with smaller offsets initiate their transmissions earlier, the utilization of offsets allows for packet prioritization. This prioritization can ensure that packets requiring more timely delivery, such as those related to the state transitions (i.e., urgent packets), are transmitted more quickly by being delivered ahead of other packets with longer offsets.

4.4.2 Considerations

To successfully apply offset-based prioritization in 6TiSCH networks and accelerate network formation, the following considerations need to be taken into account:

- The number of available offsets
- State transitions in network formation
- Differences between broadcast and unicast packets
- Impact of offset-based prioritization
- Characteristics of TSCH common shared cell

These considerations play a crucial role in optimizing the use of offsets and ensuring efficient communication in the network formation process.

The number of available offsets: Taking into account the timeslot template in the standard and the performance of commodity off-the-shelf devices, up to six offsets can be conditionally supported. Among these offsets, the first five can accommodate both unicast and broadcast transmissions, while the sixth offset is reserved exclusively for broadcast packets. Out of these offsets, we utilize the first five for our research purposes.

State transitions in network formation: The second consideration is the state transition in the 6TiSCH network formation. We especially focus on the transition from the *TSCH joined* state to the subsequent states, namely the *RPL joined* and the *cell allocated* states. As discussed earlier, in order to move from the *TSCH joined* state to the *RPL joined* state, it is crucial to receive DIO packet. Similarly, to proceed from the *RPL joined* state to the *cell allocated* state, the successful transmission of DAO packet is necessary. Considering these factors, a careful understanding of the state transitions and associated message exchanges is necessary for effective offset-based prioritization in the 6TiSCH network formation.

Differences between broadcast and unicast packets: During the network formation process, both broadcast packets and unicast packets are generated, and they are

transmitted through a common shared cell. However, these two types of packets have distinct characteristics that need to be considered in offset assignment. Specifically, unicast packets are accompanied by ACKs, enabling the determination of successful or failed transmissions. On the other hand, broadcast packets do not have ACKs, making it impossible to ascertain the success or failure of their transmissions. Therefore, it is desirable to avoid situations where broadcast packets and unicast packets collide by being transmitted on the same offset.

Impact of transmission offset-based prioritization: Next, we discuss the impact of offset-based prioritization on the network and highlight the considerations that need to be taken into account. Firstly, it is important to note that packets with smaller offsets will block the transmission of packets with larger offsets. While this is a design choice to prioritize the delivery of urgent packets (assigned smaller offsets), it is desirable to ensure that lower priority packets are not excessively delayed and are transmitted as well. To achieve this, the smaller offsets should be emptied as quickly as possible. Additionally, for packets that repeatedly get larger offsets and experience transmission postponement, appropriate adjustments to the offsets can promote faster delivery.

Another consideration is to assign multiple offsets to packets of a specific type, reducing the collision probability and promoting packet delivery. For instance, let's consider a scenario where multiple nodes need to transmit DAO packets. If these DAO packets are assigned a single offset, collisions between them within the same cell would be inevitable. However, by allowing DAO packets to have multiple offsets and randomly determining the offset for each node, it becomes probabilistically possible to have DAO packets transmitted without collisions. Therefore, it is worth consideration to allocate multiple offsets to packets of a specific type to accelerate the packet delivery.

We now discuss the potential side effects of offset-based prioritization that need to be addressed to achieve the desired performance. Fig. 4.7 illustrates the behavior of five nodes across four common shared cells, omitting non-common shared cells for

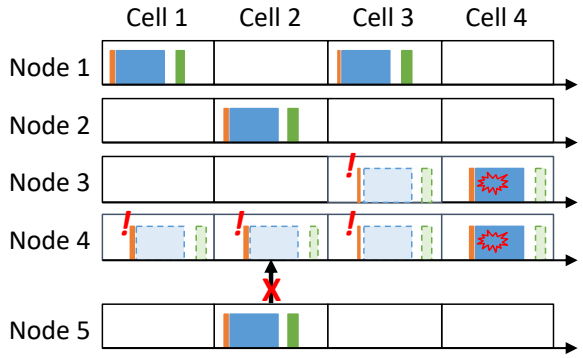


Figure 4.7: Example of potential side effects of offset-based prioritization. Orange, blue, and green rectangles represent CCA, transmission of data packet, and transmission of ACK respectively.

convenience. Assume that nodes 1 and 2 transmit packets at offset 0, while nodes 3 and 4 transmit packets at offset 2. In this scenario, until the fourth cell where nodes 1 and 2 do not transmit packets, nodes 3 and 4 postpone their packet transmissions. In this situation, if nodes 3 and 4 both attempt to transmit their packets in the fourth cell, immediately after deferring packet transmissions in the third cell, a collision between nodes 3 and 4 becomes unavoidable. Preventing such collisions is needed for *TOP* to accomplish its intended purpose.

Furthermore, an issue may arise when a node defers packet transmission based on offset-based prioritization and attempts retransmissions repeatedly. For instance, as shown in Fig. 4.7, node 4 retries packet transmission from cell 1 to cell 3 due to the presence of preceding packets from nodes 1 and 2. The problem lies in the fact that the common shared cell serves as a resource not only for packet transmission but also for reception. As a result, repetitive transmission behaviors within consecutive common shared cells can limit opportunities for receiving operations. In the mentioned scenario, if there is another node, node 5, that needs to send a packet to node 4, communication between node 5 and node 4 will continue to fail until node 4 no longer requires transmission operations. This can lead to further delays in packet delivery.

For *TOP* to effectively expedite network formation, successful transmission should be accompanied by successful reception.

4.5 Proposed Scheme

Taking into account the aforementioned factors, we propose transmission offset-based prioritization (*TOP*) as a means to expedite 6TiSCH network formation. *TOP* comprises an offset assignment policy that allocates suitable offsets to packets based on node state, packet type, and urgency. It incorporates offset escalation mechanisms to promote fast packet transmission and a modified backoff mechanism to prevent collisions and further improve transmission efficiency. Together, these components synergistically work towards accelerating network formation in 6TiSCH networks.

4.5.1 Transmission offset assignment policy

We first propose an offset assignment policy that is tailored to the characteristics of 6TiSCH networks. Fig. 4.8 visualizes our proposed offset assignment policy by indicating the assigned offsets for different packet types through color coding. As discussed earlier, the formation of a 6TiSCH network involves transitions between specific states, which necessitate the transmission or reception of certain packets. Depending on the network implementation, there may also be packets that need to be transmitted or received to fulfill specific conditions, such as network stabilization. *TOP*

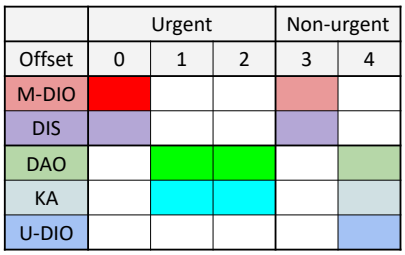


Figure 4.8: Transmission offset assignment policy.

classifies these formation-critical packets as “urgent packets,” requiring higher priority. To prioritize them, we assign smaller offsets, ensuring their prompt delivery. In Fig. 4.8, offsets from 0 to 2 are allocated to urgent packets, while offsets 3 and 4 are assigned to non-urgent packets.

4.5.2 Determination of packet urgency

To determine which packets should be classified as urgent, we consider specific packets that play a vital role in state transitions of network formation. We first focus on the conditions for transition from the *TSCH joined* state to the *RPL joined* state. For this transition, the reception of RPL DIO packets from neighboring *RPL joined* nodes is prerequisite. It is important to note that the challenge lies not in transmitting but in receiving these DIO packets. Therefore, it is important for nodes in the *RPL joined* state or in the subsequent states to anticipate the neighboring nodes' need for reception of DIO and prioritize it accordingly.

To address this, we consider the behavior of DIO Trickle timer algorithm [37], which governs the transmission interval of DIO packets. The Trickle timer dynamically adjusts the interval by manipulating the DIO interval exponent. At the beginning of the network, the Trickle timer sets the interval to a minimum value to facilitate the rapid dissemination of RPL network information. Subsequently, it gradually increases the interval exponentially unless there are changes in the routing topology. Meanwhile, whenever a node in the *RPL joined* state receives a RPL DIS packet, it resets the Trickle timer and reverts the interval to the minimum value, enabling more frequent DIO packet transmissions.

As a result, a smaller Trickle timer interval signifies the necessity for frequent DIO transmissions to neighboring nodes. Hence, we classify DIO packets as urgent when the Trickle timer exponent falls below a certain threshold. This classification remains consistent even when the DIO Trickle timer period is reset due to changes in the routing topology, irrespective of the state transition for network formation, as the

swift dissemination of network information through DIO packets remains paramount.

We next consider DAO packets. DAO packets are generated periodically and sent to RPL parents. However, they are especially crucial when transitioning from the *RPL joined* state to the *cell allocated* state. In other situations, their transmission does not require immediate attention. Taking this into account, we classify DAO packets transmitted during the RPL joined state as urgent packets. For the remaining cases, DAO packets are categorized as non-urgent packets.

In Contiki-NG, where we implement *TOP*, after a node enters the *TSCH joined* state by receiving an EB, if it fails to transmit the KA packet before clock drift correction occurs at least once, it is considered unable to achieve synchronization and reverts back to the new node state. Therefore, it is important to successfully transmit KA packet after transitioning to the *TSCH joined* state to maintain a stable state. Consequently, under this condition, we classify KA packet as an urgent packet. However, in other cases, particularly when clock drift has been corrected at least once, we classify the KA packet as a non-urgent packet.

The remaining packets, as they do not directly participate in the state transition and state stabilization during the network formation process, are classified as non-urgent packets.

4.5.3 Differentiation between broadcast and unicast packets

As previously discussed, broadcast packets are more susceptible to collisions compared to unicast packets. Considering this, we allocate offsets for both urgent packets and non-urgent packets in a way that prioritizes broadcast packets over unicast packets (i.e., assigning smaller offsets to the broadcast packets). As shown in Fig. 4.8, offset 0 is assigned for broadcast packets among the offsets designated for urgent packets, while offset 3 is assigned for broadcast packets among the offsets designated for non-urgent packets.

4.5.4 Multi-offset assignment to urgent unicast packets

In the current approach, offset 0 is assigned to urgent broadcast packets, offset 3 is assigned to non-urgent broadcast packets, and offset 4 is assigned to non-urgent unicast packets, leaving offsets 1 and 2 available. During the network formation process, broadcast packets are transmitted in a one-to-many fashion, while unicast packets are transmitted in a one-to-one manner. Consequently, the number of packets required for state transitions of all nodes is generally greater for unicast packets than for broadcast packets. Taking this into consideration, we randomly allocate the remaining two offsets to the urgent unicast packets. This multi-offset allocation strategy reduces the collision probability of urgent unicast packets, thereby increasing transmission success rates and facilitating efficient network formation.

4.5.5 Transmission offset escalation

To provide transmission opportunities for packets that are repeatedly delayed due to being assigned a large offset, *TOP* incorporates offset escalation. We note that, it is important to handle offset adjustments with care, as they have the potential to increase congestion levels that were reduced through offset-based prioritization. Therefore, we specifically target non-urgent broadcast packets for offset escalation. While non-urgent broadcast packets may not be urgent in terms of network formation, they often contain vital network information, making excessive transmission delays undesirable. Conversely, non-urgent unicast packets generally do not require immediate delivery or can be offloaded to dedicated cells once transitioning to the *cell allocated* state. Considering these factors, we implement offset escalation exclusively for broadcast packets.

We leverage the parameter *macMaxRetries* defined in the TSCH standard, which represents the maximum number of transmission attempts for a packet at the MAC layer. If a packet experiences a greater transmission delay than this value, it has been delayed longer than its intended transmission time. In such cases, we assign offset 0 to non-urgent broadcast packets, allowing them to be transmitted with higher priority.

4.5.6 Modification of the backoff mechanism

As mentioned earlier, when applying offset-based prioritization, allowing packets that have been delayed due to the detection of preceding packets to immediately attempt retransmission can lead to collisions between nodes. Besides, this can result in prolonged transmission without reception operations. This can exacerbate congestion and delay packet delivery, ultimately slowing down network formation. To address this, we propose a slight modification to the default backoff mechanism in TSCH to align it with the goal of achieving fast network formation through offset-based prioritization.

Firstly, we introduce random backoff mechanism for broadcast packets, which was previously only applied to unicast packets. We extend the random backoff behavior defined for unicast packets to also encompass broadcast packets that have experienced delays due to the detection of preceding packets. Furthermore, in the TSCH retransmission mechanism, if the number of failed transmission reaches the maximum defined by *macMaxRetries*, the transmission is considered failed, and the packet is dropped. However, when a node detects a preceding packet and defers its transmission in the common shared cell, it is an intended postponement thus we exclude such deferred transmissions from the count of retransmission attempts. By incorporating these modifications, we ensure that the backoff mechanism aligns with the principles of offset-based prioritization, mitigating congestion and facilitating faster network formation.

4.6 Evaluation

4.6.1 Implementation and experiment setup

We implement *TOP* on the M3 board using Contiki-NG. As for the baseline scheme, we utilized publicly available implementations of ALICE. For the comparison schemes, we first employed random offset assignment (referred to as *Random* hereafter). The slotframe lengths for EB and unicast are set to 397 and 20, respectively. We then investigate the network formation time while varying the common shared slotframe size

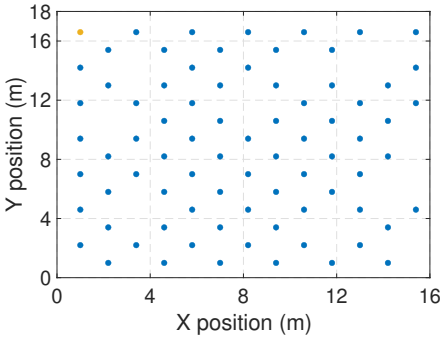


Figure 4.9: Node deployment topology at Lille testbed. he node located in the upper left corner and marked in yellow serves as the root.

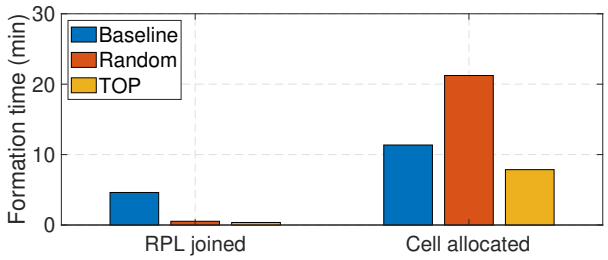
from 31 to 41.

We conduct experiments on the FIT/IoT-LAB testbed, which is a large-scale public testbed located in Lille. The physical deployment topology of the 79 nodes in Lille is illustrated in Fig. 4.9. The nodes are almost evenly distributed in a rectangular grid shape, forming a 3-4 hop topology. With this setup, we measured the network formation time for each scheme. We repeat the experiment three times for each experimental case.

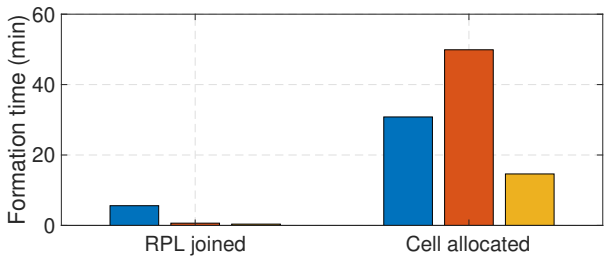
4.6.2 Performance of *TOP*

Fig. 4.10 shows the time taken for all nodes to reach the *RPL joined* state and the *cell allocated* state for two different common shared slotframe lengths, 31 and 41. In both slotframe sizes, *TOP* shows reduced transition times in two states compared to the baseline. This improvement can be attributed to *TOP* effectively mitigating collisions in the common shared cell and delivering the necessary packets promptly.

On the other hand, in the *Random* case, the transition to the *RPL joined* state is faster compared to the baseline, while reaching the *cell allocated* state takes longer than the baseline. The faster transition to the *RPL joined* state in *Random* can be attributed to the fact that transitioning to the *RPL joined* state requires receiving DIO packets. We note that broadcast packets like DIO can propagate throughout the net-



(a) Network formation time with common shared slotframe size of 31



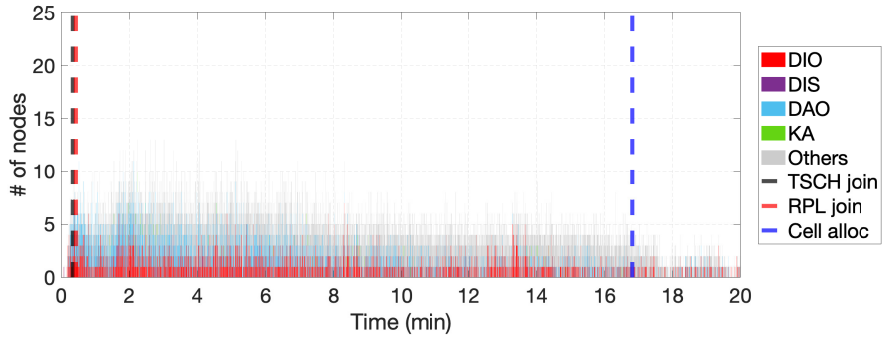
(b) Network formation time with common shared slotframe size of 41

Figure 4.10: Network formation time for different slotframe sizes

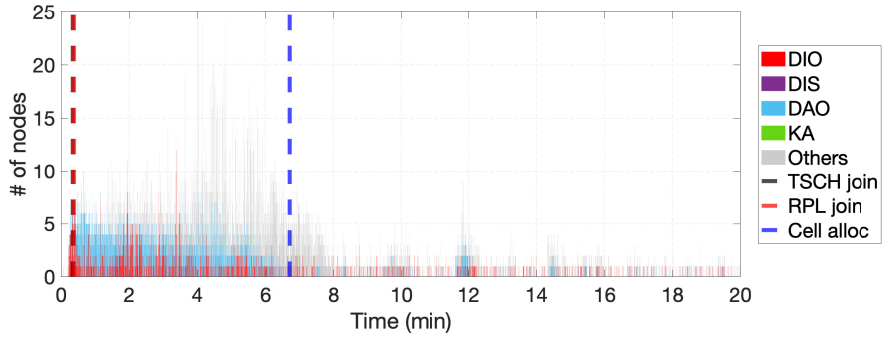
work even with a few packets since they are delivered in a one-to-many manner. Therefore, simply assigning offsets randomly can reduce the collision probability of DIOs enough to expedite the transition to the *RPL joined* state.

However, proceeding to the *cell allocated* state requires successful one-to-one unicast transmission of DAO packets from all nodes to their parent nodes. In Random offset assignment, DAO packets are also assigned offsets randomly. Therefore, the transmissions of DAO packets with larger offsets can be repeatedly delayed by packets with smaller offsets, resulting in delayed state transition. That is, in the *Random* policy, although collisions among packets can be detected and avoided through random offset assignment, it alone was not sufficient to ensure the timely delivery of unicast packets.

On the contrary, in *TOP*, after successful propagation of the initial few DIO packets, DAO packets are prioritized and promptly delivered. As a result, *TOP* demonstrates the shortest formation time not only in the *RPL joined* state but also in the *cell allocated* state.



(a) Time evolution of *Random*



(b) Time evolution of *TOP*

Figure 4.11: Time evolution of transmissions within the common shared cell and the time when the state transition of all nodes completed for *Random* and *TOP*.

Moreover, as the length of the common shared slotframe increases, the severity of the collision problem intensifies due to the reduced frequency of the common shared cell repetition. Consequently, it is evident that the overall formation time for both states increases when the slotframe length is 41 compared to 31. However, *TOP* continues to exhibit superior performance by efficiently delivering the necessary packets for formation through effective prioritization. Notably, the enhancement in *TOP*'s performance is more pronounced when the slotframe length is 41, underscoring its successful mitigation of collision and congestion issues in the common shared cell.

Fig. 4.11, similar to Fig. 4.4, the number of nodes attempting packet transmissions in the common shared cell over time, as well as the time at which all nodes transition to

the *TSCH joined*, *RPL joined*, and *cell allocated* states, for both the *Random* policy and *TOP*. Under the *Random* policy, where offsets are assigned randomly without explicit priorities, a notable number of packets are repeatedly transmitted at the same common shared cell, leading to packet delivery failures and subsequent delays in transitioning to the *cell allocated* state. In contrast, *TOP* demonstrates a different behavior. The packets necessary for state transitions, such as DIO and DAO, are transmitted in a prioritized manner. As a result, a prompt transition to the *cell allocated* state is achieved.

In summary, *TOP* effectively alleviates collisions in the TSCH common shared cell by assigning different offsets to the start time of packet transmission. Moreover, it prioritizes packet delivery based on the assigned offsets, enabling the swift transmission of essential packets required for network formation and facilitating the timely completion of the formation process in the 6TiSCH network.

4.7 Related Work

4.7.1 Toward fast 6TiSCH network formation

Numerous research studies have focused on enhancing the speed of 6TiSCH network formation. For instance, Vallati et al. [107] proposed dynamic allocation of common shared cells based on control packet load to mitigate collisions. However, this approach converts dedicated cells into common shared cells, resulting in decreased network efficiency and increased energy consumption. Vucinic et al. [108] found that EB transmissions can cause congestion in the common shared cell and suggested a fixed and low EB rate to alleviate this issue, but it can potentially delay state transitions. C2DBI [109] dynamically adjusts the EB generation interval to address congestion, while Kalita et al. [110] dynamically adjust the priority of control packets to prevent delays caused by the highest priority given to EBs. TRGB [111] introduces a novel approach by dividing the common shared cell into three types and differentiating packets transmitted within each type of common shared cell. The goal is to reduce collision probability, but this

approach can potentially lead to resource scarcity issues. As such, there have been various research efforts aimed at improving 6TiSCH network formation. However, none of these studies investigated a fundamental causes of delay in network formation: the unavoidable collision problem within the common shared cell due to closely aligned transmission start times.

4.7.2 Offset-based differentiation in TSCH slot

The concept of assigning different offsets to the transmission start times within TSCH slots has been explored in previous studies [96, 97]. However, these studies primarily focused on implementing this approach during the data transmission phase rather than the network formation phase. In [96], transmission offset differentiation is utilized in dedicated cells, allowing non-owner nodes to utilize the cell when the owner node is not transmitting packets, with the goal of reducing communication latency. DualBlock [97] introduces multiple offsets in shared cells for unicast transmission, aiming to alleviate packet collisions and enhance packet delivery. However, DualBlock does not consider offset-based prioritization, which limits the effectiveness of its collision mitigation. In contrast, *TOP* incorporates packet differentiation and prioritization based on offsets to address collisions in the common shared cell during the network formation process, facilitating the timely delivery of essential packets.

4.8 Summary

In conclusion, this chapter addresses the challenges of network formation in 6TiSCH networks by proposing *TOP*, an offset-based prioritization technique. The proposed scheme enhances the efficiency of network formation by assigning offsets and prioritizing packets, leveraging a comprehensive understanding of 6TiSCH networks. Through implementation on real devices and extensive evaluations conducted on a sizable testbed, we have successfully validated the effectiveness of *TOP* in signifi-

cantly reducing network formation time. These findings contribute to the optimization of 6TiSCH networks, enabling faster and more reliable network deployment.

Chapter 5

Conclusion

In this dissertation, we have addressed significant challenges in LLNs and proposed innovative solutions to enhance performance of LLNs. Through extensive research and empirical evaluations, we have made notable contributions in three key areas: mobile routing, time-slotted communication, and 6TiSCH network formation.

In the domain of mobile routing, we introduced *MobiRPL*, an adaptive and robust mobile routing protocol designed to overcome the limitations of the existing IPv6 RPL protocol. By leveraging RSSI, *MobiRPL* enables reliable and efficient routing in mobile LLNs. The evaluations have demonstrated its superiority, achieving substantial improvements in packet delivery ratio and energy consumption compared to conventional approaches.

To enhance the time-efficiency of time-slotted communication, we presented *ASAP*, a utility-based adaptation of slot-size and packet aggregation technique. By dynamically adjusting slot sizes based on packet size distribution and employing packet aggregation, *ASAP* maximizes resource utilization in time-slotted systems. Experimental evaluations have validated its effectiveness, showcasing significant improvements in throughput and latency compared to traditional approaches.

Moreover, we addressed the challenges in network formation in 6TiSCH networks and proposed *TOP*, an offset-based prioritization technique. Within the common shared

cell, *TOP* assigns offsets to packets during transmission initiation, diversifies their starting points, and facilitates collision detection and prioritization of critical packets. Testbed experiments have demonstrated the remarkable reductions in network formation time achieved by *TOP*, further enhancing the efficiency of 6TiSCH networks.

Overall, the contributions made in this dissertation significantly advance the field of LLNs by providing novel solutions to critical challenges. The proposed solutions have demonstrated their effectiveness through extensive evaluations and have shown the potential to improve the overall performance of LLNs in various application domains.

Bibliography

[1] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile computing*. Springer, 1996, pp. 153–181.

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

[3] 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 Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2502–2525, Sep. 2017.

[4] 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 Transactions on Mobile Computing*, vol. 16, no. 4, pp. 964–979, 2016.

[5] 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 *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, 2015, pp. 281–294.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] J. Ko, C. Lu, M. B. Srivastava, J. A. Stankovic, A. Terzis, and M. Welsh, "Wireless Sensor Networks for Healthcare," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1947–1960, 2010.

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

[12] C. Cobârzan, J. Montavont, and T. Noel, "Integrating Mobility in RPL," in *European conference on wireless sensor networks*. Springer, 2015, pp. 135–150.

[13] 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.

[14] S. Hoghooghi and R. N. Esfahani, "Mobility-Aware Parent Selection for Routing Protocol in Wireless Sensor Networks using RPL," in *2019 5th International Conference on Web Research (ICWR)*. IEEE, 2019, pp. 79–84.

[15] 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 Generation Computer Systems*, vol. 97, pp. 247–258, 2019.

[16] M. Bouaziz, A. Rachedi, and A. Belghith, "EKF-MRPL: Advanced mobility support routing protocol for internet of mobile things: Movement prediction approach," *Future Generation Computer Systems*, vol. 93, pp. 822–832, 2019.

[17] 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.

[18] I. Rabet, S. P. Selvaraju, M. H. Adeli, H. Fotouhi, A. Balador, M. Vahabi, M. Alves, and M. Björkman, "Pushing IoT Mobility Management to the Edge: Granting RPL Accurate Localization and Routing," in *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. IEEE, 2021, pp. 338–343.

[19] R. Elhabyan, W. Shi, and M. St-Hilaire, "Coverage protocols for wireless sensor networks: Review and future directions," *Journal of Communications and Networks*, vol. 21, no. 1, pp. 45–60, 2019.

[20] C. Zhu, C. Zheng, L. Shu, and G. Han, "A survey on coverage and connectivity issues in wireless sensor networks," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 619–632, 2012.

[21] I. El Korbi, M. B. Brahim, C. Adjih, and L. A. Saidane, "Mobility Enhanced RPL for Wireless Sensor Networks," in *Network of the Future (NOF), 2012 Third International Conference on the*. IEEE, 2012, pp. 1–8.

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

[23] 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.

[24] J. Ko and M. Chang, "MoMoRo: Providing Mobility Support for Low-Power Wireless Applications," *IEEE Systems Journal*, vol. 9, no. 2, pp. 585–594, 2015.

[25] 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.

[26] J. Wang, G. Chalhoub, and M. Misson, "Mobility support enhancement for RPL," in *Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN), 2017 International Conference on*. IEEE, 2017, pp. 1–6.

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

[28] A. Mohammadsalehi, B. Safaei, A. M. H. Monazzah, L. Bauer, J. Henkel, and A. Ejlali, "ARMOR: A Reliable and Mobility-aware RPL for Mobile Internet of Things Infrastructures," *IEEE Internet of Things Journal*, 2021.

[29] T. Clausen, A. C. de Verdiere, J. Yi, A. Niktash, Y. Igrashi, H. Satoh, U. Herberg, C. Lavenu, T. Lys, and J. Dean, "The Lightweight On-demand Ad hoc Distance-vector Routing Protocol - Next Generation (LOADng)," *Internet Eng. Task Force, Draft*, July 2016.

[30] M. Vučinić, B. Tourancheau, and A. Duda, "Performance comparison of the RPL and LOADng routing protocols in a Home Automation scenario," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*. IEEE, 2013, pp. 1974–1979.

[31] 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 *Proceedings of the 8th ACM Symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. ACM, 2011, pp. 73–80.

[32] 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 *2015 International Conference on Smart Grid and Clean Energy Technologies (ICSGCE)*. IEEE, 2015, pp. 43–49.

[33] J. Yi, T. Clausen, and Y. Igarashi, "Evaluation of routing protocol for low power and Lossy Networks: LOADng and RPL," in *2013 IEEE Conference on Wireless Sensor (ICWISE)*. IEEE, 2013, pp. 19–24.

[34] J. Tripathi and J. C. de Oliveira, "Proactive versus reactive revisited: IPv6 routing for Low Power Lossy Networks," in *2013 47th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2013, pp. 1–6.

[35] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Local computer networks, proceedings 2006 31st IEEE conference on*. IEEE, 2006, pp. 641–648.

[36] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," Tech. Rep., July 2003.

[37] P. Levis and T. H. Clausen, "The Trickle Algorithm," *Internet Eng. Task Force, RFC 6206*, Mar. 2011.

[38] O. Gnawali and P. Levis, "The Minimum Rank with Hysteresis Objective Function," *RFC 6719*, Sep 2012.

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

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

[41] 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.

[42] K. Levis *et al.*, "RSSI is under appreciated," in *Proceedings of the Third Workshop on Embedded Networked Sensors, Cambridge, MA, USA*, vol. 3031, 2006, p. 239242.

[43] S. Lin, F. Miao, J. Zhang, G. Zhou, L. Gu, T. He, J. A. Stankovic, S. Son, and G. J. Pappas, "ATPC: Adaptive Transmission Power Control for Wireless Sensor Networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 12, no. 1, p. 6, 2016.

[44] E. M. Royer, C.-K. Toh *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.

[45] C. E. Perkins and P. Bhagwat, "Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers," in *ACM SIGCOMM computer communication review*, vol. 24, no. 4. ACM, 1994, pp. 234–244.

[46] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," Tech. Rep., 2003.

[47] D. Johnson, Y.-c. Hu, and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," Tech. Rep., 2007.

[48] 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 Networks*, vol. 23, pp. 121–144, 2014.

[49] T. Clausen, J. Yi, and A. C. De Verdiere, "LOADng: Towards AODV Version 2," in *2012 IEEE Vehicular Technology Conference (VTC Fall)*. IEEE, 2012, pp. 1–5.

[50] T. Clausen, J. Yi, and U. Herberg, "Lightweight On-demand Ad hoc Distance-vector Routing-Next Generation (LOADng): Protocol, extension, and applicability," *Computer Networks*, vol. 126, pp. 125–140, 2017.

[51] J. Yi and T. Clausen, "Collection Tree Extension of Reactive Routing Protocol for Low-Power and Lossy Networks," *International Journal of Distributed Sensor Networks*, vol. 10, no. 3, p. 352421, 2014.

[52] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*. IEEE, 2004, pp. 455–462.

[53] A. Dunkels, "The contikimac radio duty cycling protocol," Swedish Institute of Computer Science (SICS), Tech. Rep. T2011:13, 2011.

[54] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis, "An empirical study of low-power wireless," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 2, pp. 1–49, 2010.

[55] 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 Communications Magazine*, vol. 57, no. 7, pp. 55–61, 2019.

[56] 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," *Journal of Communications and Networks*, vol. 20, no. 4, pp. 414–424, 2018.

[57] R. Heile, R. Alfvin, P. Kinney, J. Gilb, and C. Chaplin, "IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer," IEEE, Tech. Rep., 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 *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, 2021, pp. 234–246.

[60] L. G. Roberts, "ALOHA packet system with and without slots and capture," *ACM SIGCOMM Computer Communication Review*, vol. 5, no. 2, pp. 28–42, 1975.

[61] T. Watteyne, J. Weiss, L. Doherty, and J. Simon, "Industrial IEEE802.15.4e networks: Performance and trade-offs," in *IEEE International Conference on Communications (ICC)*, 2015, pp. 604–609.

[62] V. Scilimati, A. Petitti, P. Boccadoro, R. Colella, D. Di Paola, A. Milella, and L. Alfredo Grieco, "Industrial Internet of things at work: a case study analysis in robotic-aided environmental monitoring," *IET wireless sensor systems*, vol. 7, no. 5, pp. 155–162, 2017.

[63] X. Vilajosana, T. Watteyne, M. Vučinić, T. Chang, and K. S. Pister, "6TiSCH: Industrial performance for IPv6 Internet-of-Things networks," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1153–1165, 2019.

[64] S. Raza, M. Faheem, and M. Guenes, "Industrial wireless sensor and actuator networks in industry 4.0: Exploring requirements, protocols, and challenges—A MAC survey," *International Journal of Communication Systems*, vol. 32, no. 15, p. e4074, 2019.

[65] C. Orfanidis, P. Pop, and X. Fafoutis, "Active Connectivity Fundamentals for TSCH Networks of Mobile Robots," in *18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2022, pp. 191–198.

[66] H. Kim, H.-S. Kim, and S. Bahk, "MobiRPL: Adaptive, robust, and RSSI-based mobile routing in low power and lossy networks," *Journal of Communications and Networks*, vol. 24, no. 3, pp. 365–383, 2022.

[67] R. Tavakoli, M. Nabi, T. Basten, and K. Goossens, "Topology Management and TSCH Scheduling for Low-Latency Convergecast in In-Vehicle WSNs," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1082–1093, 2018.

[68] I. Khoufi, P. Minet, and B. Rmili, "Beacon advertising in an IEEE 802.15.4e TSCH network for space launch vehicles," *Acta Astronautica*, vol. 158, pp. 76–88, 2019.

[69] T. Watteyne, A. L. Diedrichs, K. Brun Laguna, J. E. Chaar, D. Dujovne, J. C. Taffernaberry, and G. A. Mercado, "Peach: Predicting frost events in peach orchards using iot technology," *EAI Endorsed Trans. Internet Things*, vol. 2, no. 5, Dec. 2016.

[70] S. A. Malek, F. Avanzi, K. Brun-Laguna, T. Maurer, C. A. Oroza, P. C. Hartsough, T. Watteyne, and S. D. Glaser, "Real-Time Alpine Measurement System Using Wireless Sensor Networks," *Sensors*, vol. 17, no. 11, p. 2583, 2017.

[71] K. Brun-Laguna, A. L. Diedrichs, D. Dujovne, C. Taffernaberry, R. Leone, X. Vilajosana, and T. Watteyne, "Using SmartMesh IP in Smart Agriculture and Smart Building Applications," *Computer Communications*, vol. 121, pp. 83–90, 2018.

[72] Z. Zhang, S. Glaser, T. Watteyne, and S. Malek, "Long-Term Monitoring of the Sierra Nevada Nnowpack Using Wireless Sensor Networks," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 185–17 193, 2020.

[73] G. Yang, A. R. Urke, and K. Øvsthus, "Mobility Support of IoT Solution in Home Care Wireless Sensor Network," in *Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS)*. IEEE, 2018, pp. 475–480.

[74] P. Woznowski, A. Burrows, T. Diethe, X. Fafoutis, J. Hall, S. Hannuna, M. Camplani, N. Twomey, M. Kozlowski, B. Tan *et al.*, "SPHERE: A Sensor Platform for Healthcare in a Residential Environment," *Designing, Developing, and Facilitating Smart Cities: Urban Design to IoT Solutions*, pp. 315–333, 2017.

[75] A. Elsts, X. Fafoutis, P. Woznowski, E. Tonkin, G. Oikonomou, R. Piechocki, and I. Craddock, "Enabling Healthcare in Smart Homes: The SPHERE IoT Network Infrastructure," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 164–170, 2018.

[76] A. Elsts, X. Fafoutis, G. Oikonomou, R. Piechocki, and I. Craddock, "TSCH Networks for Health IoT: Design, Evaluation, and Trials in the Wild," *ACM Transactions on Internet of Things*, vol. 1, no. 2, pp. 1–27, 2020.

[77] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH," in *Proceedings of the 13th ACM conference on embedded networked sensor systems*, 2015, pp. 337–350.

[78] A. Elsts, X. Fafoutis, J. Pope, G. Oikonomou, R. Piechocki, and I. Craddock, "Scheduling High-Rate Unpredictable Traffic in IEEE 802.15.4 TSCH Networks," in *13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2017, pp. 3–10.

[79] S. Kim, H.-S. Kim, and C. Kim, "ALICE: Autonomous Link-based Cell Scheduling for TSCH," in *IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2019, pp. 121–132.

[80] S. Jeong, J. Paek, H.-S. Kim, and S. Bahk, "TESLA: Traffic-Aware Elastic Slotframe Adjustment in TSCH Networks," *IEEE Access*, vol. 7, pp. 130 468–130 483, 2019.

[81] S. Jeong, H.-S. Kim, J. Paek, and S. Bahk, "OST: On-Demand TSCH Scheduling with Traffic-awareness," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 69–78.

[82] J. Jung, D. Kim, T. Lee, J. Kang, N. Ahn, and Y. Yi, "Distributed Slot Scheduling for QoS Guarantee over TSCH-based IoT Networks via Adaptive Parameterization," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2020, pp. 97–108.

[83] S. Kim, H.-S. Kim, and C.-k. Kim, "A3: Adaptive Autonomous Allocation of TSCH Slots," in *International Conference on Information Processing in Sensor Networks (IPSN)*, 2021, pp. 299–314.

[84] Z. Yu, X. Na, C. A. Boano, Y. He, X. Guo, P. Li, and M. Jin, "SmarTiSCH: An Interference-Aware Engine for IEEE 802.15.4e-based Networks," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2022, pp. 350–362.

[85] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The Contiki-NG open source operating system for next generation IoT devices," *SoftwareX*, vol. 18, p. 101089, 2022.

[86] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele *et al.*, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 459–464.

[87] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-Time Scheduling for WirelessHART Networks," in *IEEE Real-Time Systems Symposium*, 2010, pp. 150–159.

[88] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic Aware Scheduling Algorithm for Reliable Low-Power Multi-Hop IEEE 802.15.4e networks," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2012, pp. 327–332.

[89] Y. Jin, P. Kulkarni, J. Wilcox, and M. Sooriyabandara, "A centralized scheduling algorithm for IEEE 802.15.4e TSCH based industrial low power wireless networks," in *IEEE Wireless Communications and Networking Conference*, 2016, pp. 1–6.

[90] D. Gunatilaka and C. Lu, "Conservative Channel Reuse in Real-Time Industrial Wireless Sensor-Actuator Networks," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 344–353.

[91] R. Brummet, D. Gunatilaka, D. Vyas, O. Chipara, and C. Lu, "A Flexible Retransmission Policy for Industrial Wireless Sensor Actuator Networks," in *IEEE International Conference on Industrial Internet (ICII)*, 2018, pp. 79–88.

[92] O. Harms and O. Landsiedel, "MASTER: Long-Term Stable Routing and Scheduling in Low-Power Wireless Networks," in *International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2020, pp. 86–94.

[93] T. Chang, T. Watteyne, Q. Wang, and X. Vilajosana, "LLSF: Low Latency Scheduling Function for 6TiSCH Networks," in *International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2016, pp. 93–95.

[94] T. Chang, M. Vucinic, X. Vilajosana, S. Duquennoy, and D. Dujovne, "6TiSCH Minimal Scheduling Function (MSF)," 2019.

[95] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "LDSF: Low-Latency Distributed Scheduling Function for Industrial Internet of Things," *IEEE Internet of Things journal*, vol. 7, no. 9, pp. 8688–8699, 2020.

[96] R. Tavakoli, M. Nabi, T. Basten, and K. Goossens, "Hybrid Timeslot Design for IEEE 802.15.4 TSCH to Support Heterogeneous WSNs," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2018, pp. 1–7.

[97] J. Park, H. Kim, H.-S. Kim, and S. Bahk, "DualBlock: Adaptive Intra-Slot CSMA/CA for TSCH," *IEEE Access*, vol. 10, pp. 68 819–68 833, 2022.

[98] D. Vasiljević and G. Gardašević, "Packet Aggregation-Based Scheduling in 6TiSCH Networks," in *IEEE EUROCON 18th International Conference on Smart Technologies*, 2019, pp. 1–5.

[99] H. Hajian, M. Nabi, M. Fakouri, and F. Veisi, "LaDiS: a Low-Latency Distributed Scheduler for Time-Slotted Channel Hopping Networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–7.

[100] C. Michaelides, T. Adame, and B. Bellalta, "ECTS: Enhanced Centralized TSCH Scheduling with Packet Aggregation for Industrial IoT," in *IEEE Con-*

ference on Standards for Communications and Networking (CSCN), 2021, pp. 40–45.

[101] IEEE 802.11-2012, *Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*, IEEE Std., Mar. 2012.

[102] Bluetooth SIG, "Bluetooth Core Specification: 5.0," 2021.

[103] J. Ko, J. Jeong, J. Park, J. A. Jun, O. Gnawali, and J. Paek, "DualMOP-RPL: Supporting Multiple Modes of Downward Routing in a Single RPL Network," *ACM Transactions on Sensor Networks (TOSN)*, vol. 11, no. 2, pp. 39:1–39:20, Mar 2015.

[104] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," in *ACM International Conference on Mobile Computing and Networking (MobiCom)*, Sep. 2003.

[105] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "Ietf 6tisch: A tutorial," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 595–615, 2019.

[106] A. Kalita and M. Khatua, "6tisch–ipv6 enabled open stack iot network formation: A review," *ACM Transactions on Internet of Things*, vol. 3, no. 3, pp. 1–36, 2022.

[107] C. Vallati, S. Brienza, G. Anastasi, and S. K. Das, "Improving network formation in 6tisch networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 98–110, 2018.

[108] M. Vučinić, T. Watteyne, and X. Vilajosana, "Broadcasting strategies in 6tisch networks," *Internet Technology Letters*, vol. 1, no. 1, p. e15, 2018.

[109] A. Kalita and M. Khatua, "Channel condition based dynamic beacon interval for faster formation of 6tisch network," *IEEE Transactions on Mobile Computing*, vol. 20, no. 7, pp. 2326–2337, 2020.

[110] A. Kalita and M. Khatua, "Opportunistic transmission of control packets for faster formation of 6tisch network," *ACM Transactions on Internet of Things*, vol. 2, no. 1, pp. 1–29, 2021.

[111] A. Kalita and M. Khatua, "Time-variant rgb model for minimal cell allocation and scheduling in 6tisch networks," *IEEE Transactions on Mobile Computing*, 2023.

초 록

본 논문은 저전력 및 손실 네트워크에서 모바일 라우팅, 시간 슬롯 기반 통신, 6TiSCH 네트워크의 형성이라는 세 가지 영역에 대한 문제를 해결한다. 첫째로 모바일 라우팅 기법, *MobiRPL* 은 수신 신호 세기와 주변 디바이스와의 연결성 관리 동작을 활용하여 모바일 노드가 포함된 저전력 및 손실 네트워크에서 효율적인 라우팅이 가능하도록 한다. 시뮬레이션 및 테스트베드 실험 결과, *MobiRPL* 은 비교 기법에 비해 패킷 전달률을 11.3% 향상시키고 에너지 소비를 73.3% 감소시킨다. 또한 시간 슬롯 기반 통신 시스템에서의 슬롯 내 유휴 시간으로 인한 네트워크의 통신 효율 저하를 해결하기 위해 실제 패킷 길이 분포에 따라 슬롯 크기를 동적으로 조절하고 슬롯 사용 효율에 따라 패킷 집단 전송을 수행하는 *ASAP* 를 제안한다. 실제 테스트베드에서의 TSCH 기반의 연구 결과, *ASAP* 은 기본 TSCH에 비해 throughput을 2.21배 향상시키고 지연 시간을 78.7% 줄인다. 세 번째로, 6TiSCH 네트워크 형성 과정의 비효율성을 개선하기 위해 TSCH 슬롯 내 전송 시작 시점에 offset을 부여함으로써 충돌 감지 및 offset 기반 우선 순위에 따른 중요 패킷의 우선 전송을 가능케 하는 *TOP* 을 제안한다. 실제 테스트베드 실험 결과 *TOP* 는 네트워크 형성 시간을 최대 50% 감소시킨다. 이처럼 실제 구현 및 실험을 통해 입증된 제안 기법들의 성능 향상은 본 논문이 저전력 및 손실 네트워크의 전반적인 효율성 향상에 기여할 수 있음을 보여준다.

주요어: 저전력 손실 네트워크, 사물인터넷, 모바일 라우팅 프로토콜, 신뢰도, 통신 수율, 지연 시간, 네트워크 형성

학번: 2015-22782