Ph.D. DISSERTATION

Design of Adaptive Duty-cycled Protocols in Low-power and Lossy Networks

저전력 네트워크를 위한 적응적 듀티사이클 프로토콜 설계

BY

JEONG SEUNG-BEOM

AUGUST 2019

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Design of Adaptive Duty-cycled Protocols in Low-power and Lossy Networks

저전력 네트워크를 위한 적응적 듀티사이클 프로토콜 설계

BY

JEONG SEUNG-BEOM

AUGUST 2019

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Design of Adaptive Duty-cycled Protocols in Low-power and Lossy Networks

저전력 네트워크를 위한 적응적 듀티사이클 프로토콜 설계

지도교수 박 세 웅
이 논문을 공학박사 학위논문으로 제출함

2019년 8월

서울대학교 대학원

전기 컴퓨터 공학부

정 승 범

정승범의 공학박사 학위 논문을 인준함

2019년 8월

위 원 장:	김 종 권	(인)
부위원장:	박 세 웅	(인)
위 원:	최 성 현	(인)
위 원:	심 병 효	(인)
위 원:	백 정 엽	(인)

Abstract

Internet of Things (IoT) has opened a new era with low-power embedded devices. In industrial IoT networks, numerous sensors and actuators are deployed for system monitoring and remote control. From smart homes to smart cities, new applications and network services are emerging such as electricity management, home security, health care, and smart grid. As IoT applications become diverse, the need for reliable, energy-efficient, and flexible (*i.e.,* adaptable to diverse and dynamic applications) network protocols is growing up steadily. In this dissertation, to this end, we design three different duty-cycled protocols for dynamic low-power and lossy networks (LLNs).

Firstly, we focus on mobile LLNs. With the proliferation of emerging Internet of Things devices and applications, mobility is becoming an integral part of low-power and lossy networks (LLNs). However, most LLN protocols have not yet focused on the support for mobility with an excuse of resource constraints. Some work that do provide mobility support fail to consider radio duty-cycling, control overhead, or memory usage, which are critical on resource-limited low-power devices. To tackle theses problems, we introduce *MAPLE*, an asymmetric transmit power-based routing architecture that leverages a single resource-rich LLN border router. It supports mobility in duty-cycled LLNs using received signal strength indicator (RSSI) gradient field-based routing. We implement *MAPLE* on a low-power embedded platform, and evaluate through experimental measurements on a real multihop LLN testbed consisting of 31 low-power ZigBee nodes and 1 high-power gateway. We show that *MAPLE* improves the performance of mobile devices in LLN by 27.2%/55.7% and 17.9% in terms of both uplink/downlink reliability and energy efficiency, respectively.

Next, we move our attention to Time Slotted Channel Hopping (TSCH), which is a promising TDMA-like link layer protocol standardized by the IEEE 802.15.4-2015. Compared with conventional asynchronous duty-cycled MAC protocols, it provides

both higher reliability and lower energy operation. For this reason, a number of TSCH scheduling schemes have been proposed recently. However, they lack one thing: flexibility to support a wide variety of applications and services with unpredictable traffic load and routing topology due to "fixed" slotframe sizes. To this end, we propose *TESLA*, a traffic-aware elastic slotframe adjustment scheme for TSCH networks which enables each node to dynamically self-adjust its slotframe size at *run time*. *TESLA* aims to minimize its energy consumption without sacrificing reliable packet delivery by utilizing incoming traffic load to estimate channel contention level experienced by each neighbor. We extensively evaluate the effectiveness of *TESLA* on large-scale 110-node and 79-node testbeds, demonstrating that it achieves up to 70.2% energy saving compared to Orchestra (the *de facto* TSCH scheduling mechanism) while maintaining 99% reliability.

Lastly, we point out the limitations of *TESLA*. In *TESLA*, a reception (Rx) slot is shared for multiple transmitters. To prevent their transmissions from being collided, *TESLA* inevitably allocates more Rx slots than actually needed. To tackle this inefficient resources usage, we propose *OST*, a on-demand TSCH scheduling with traffic-awareness, which improves *TESLA* further. *OST* basically schedules timeslots based on estimation of average traffic load. Moreover, if there are queued packets due to instantaneous traffic burst, it additionally allocates timeslots exactly as needed. We implement *OST* on ContikiOS, and evaluate *OST* with state-of-the-arts on large-scale multi-hop testbed showing the superiority of *OST* over others.

keywords: Low-power and lossy network (LLN), IEEE 802.15.4, Mobility, Asymmetric transmit power (ATP), Time-slotted channel hopping (TSCH), Dynamic scheduling, Wireless network protocol

student number: 2013-20877

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Motivation

Low-power and lossy network (LLN), multihop wireless network composed of resource-constrained embedded devices, has been used for a variety of applications including smart grid automated metering infrastructure (AMI), environmental monitoring, and wireless sensor network (WSN). In industrial IoT networks, numerous sensors and actuators are deployed for system monitoring and remote control. From smart homes to smart cities [1, 2], new applications and network services are emerging such as electricity management, home security, and health care [3].

As IoT applications become diverse, the need for network protocols achieving following three requirements is growing up steadily. Firstly, network protocols should be reliable. In other words, it guarantees high end-to-end packet delivery ratio (e.g., more 99%). Next, they need to be energy-efficient, since most of IoT devices are battery-powered. Lastly, they are required to be flexible (*i.e.,* adaptable to diverse and dynamic applications).

With the emergence of Internet of Things (IoT) and cyber-physical systems (CPS), LLN is now going into a new phase for smart and daily life applications which include medical care services [4, 5, 6], smart market maintenance [7, 2], networked

robots [8, 9], and more. A key challenge in many of these emerging applications is that they incorporate not only stationary but also *mobile nodes*. As an example, a hospital network can be connected with sensing and actuating devices on mobile patients and patient beds, which enables remote monitoring of medical signals. In smart market applications, mobile shopping carts are connected to an LLN, which is used for real-time advertisement of hot deals, cart location tracking, and virtual fencing. Market staffs can also carry low-power portable terminals for reporting status of inventory/stock and market condition. For over a decade, LLN research community has elaborated network layer protocols [10, 11, 7, 12] for energy efficiency and high reliability on resource-constrained devices. Although these protocols have been making progress gradually under the assumption of stationary network, they cannot be apparently adopted in LLNs with mobility, due to lack of providing any specific operation for mobility and identifying mobile nodes [13, 14, 15]. As the first work in this dissertation, we design *MAPLE*, an asymmetric transmit power-based routing architecture , which provides flexible connectivity between the LLN border router (LBR, also referred to as 'gateway') and each *mobile node* in LLN, both reliably and energy efficiently.

Meanwhile, the IEEE 802.15.4-2015 [16] standardized the time-slotted channel hopping (TSCH) protocol for low-power and lossy networks (LLNs), a promising TDMA-like link layer protocol providing both high reliability and low energy operation. TSCH's slotframe structure is the basis of TSCH operation, but its size is set offline as a fixed global constant. On top of significant burden for empirical optimization, even if the slotframe size is optimized, it is still problematic since all nodes share a single slotframe size, disregarding routing topology and traffic intensity for each node: (1) When the slotframe is too small for the node experiencing low traffic load, it will waste energy due to idle listening. (2) When the slotframe is too large for the node under heavy traffic load, it cannot receive/forward many packets due to channel contention or queue overflow. To address this issue, we introduce *TESLA*, a novel

traffic-aware elastic slot-frame adjustment scheme as a solution. Furthermore, we also point out the limitation of *TESLA*, and propose *OST*, a on-demand TSCH scheduling with traffic-awareness.

1.2 Related Work

1.2.1 Representative Standard LLN Protocols

This section presents two representative LLN routing protocols as our benchmark, RPL (IETF standard) [10] and LOADng (IETF draft) [17], and their related work.

RPL and Mobility Support. RPL [10] is a routing protocol for low-power IPv6 networks which enables IPv6 Internet connectivity to embedded devices by providing reliable routes over lossy wireless links. Several studies [13, 14, 15], however, found that RPL suffers significant performance degradation when operating with mobile endpoints since it was not designed with mobility in mind. To alleviate this problem, ME-RPL [18] gives lower priority to mobile parent candidates than static candidates when choosing a preferred parent. In addition, when parent changes occur frequently, DIS transmission interval is reduced for prompt neighbor discovery. Ko *et al.* introduced MoMoRo [13], which detects route disconnections based on uplink packet losses and quickly gathers neighbor information by requesting a unicast reply. It can find out neighbors with a good link based on a fuzzy estimator. In mRPL [19], a mobile node broadcasts a batch of DIS messages when the RSSI from its parent drops, triggering replies from its neighbors. The mobile node measures RSSI from the neighbors and selects the neighbor with a good RSSI as a preferred parent. Gaddour *et al.* used position information for mobile routing in [20] where corona ID is defined as the minimum of reachable hop distances from the DAG root.

However, most of these protocols are designed assuming *no radio duty-cycling* for more responsiveness, sacrificing low-power operation. If a duty-cycled MAC is adopted under these protocols, it necessarily delays the update of routing costs, result-

ing in more packet losses and control overhead [19]. Moreover, proposals in [18, 13, 19, 20] require maintaining *up-to-date topology information*. This incurs a significant control overhead to keep track of topology changes caused by mobile nodes. Lastly, evaluations of [18, 21, 20] are performed only on simulators, which cannot show their feasibility in the unpredictable real world.

LOADng LOADng [17] is a simplified version of well known AODV (Ad hoc On-demand Distance Vector Routing) [22] to support mobile LLNs. Like AODV, LOADng discovers a route between a source and a destination based on flooding of routing packets from the source. However, Clausen *et al.* showed that LOADng suffers from flooding overhead, particularly in applications with collection traffic [23]. To resolve this problem, the authors designed LOADng-CTP, where only the sink (not a source but a destination) floods routing messages to enable each data source to obtain a path towards the sink. It outperforms LOADng with respect to delivery ratio, overhead, and delay. However, both LOADng and LOADng-CTP are evaluated without a duty-cycled MAC and only with simulations.

1.2.2 TSCH Scheduling

Numerous duty-cycling MACs have been proposed for LLNs. Among those, asynchronous approaches [24, 25, 26, 27, 28, 11] have the advantage of neither requiring strict time-synchronization on resource-constrained devices, nor relying critically on certain parameter configurations. With technical progress, however, synchronized communication became a viable option [29], such as TSCH, which opens the scheduling problem. Below we summarize prior work on TSCH scheduling , which is categorized into centralized and decentralized approaches.

Centralized TSCH Scheduling. Centralized scheduling is often employed in industrial scenarios [30, 31, 32] where algorithms are built in and maintained by a central controller. TASA [33] is a *centralized* traffic aware scheduling algorithm using graph theory methods of matching and coloring. TASA builds the schedule based on traffic

load offered by each source node, and allocates timeslots and channel offsets based on network topology in order to maximize parallel transmissions. In [34, 35], the authors have also derived fundamental bounds on the minimum number of slots achievable with TASA for a given topology. Overlooking practical challenges in LLNs, however, it showed high loss rate or high duty-cycle on a real multihop testbed evaluation [36].

AMUS [37] is a centralized adaptive scheduling scheme which gives more Tx slots to nodes that are closer to the sink assuming that those nodes have more traffic to forward. However, it does not consider traffic load, routing topology (other than hop distance), nor link quality, resulting in inefficient allocation.

In [38], retransmission slots are added and shared according to reliability and delay constraints. However, it does not handle the side effect: collisions and excessive idle listening.

For high data rate scenarios, Elsts *et al.* [39] proposed a hybrid approach where dedicated and shared slots coexist in the same schedule. However, they assume that the number of channels used in the network is greater than the number of forwarding nodes, which is unrealistic in a channel-resource limited network. In addition, all nodes are forced to wake up at every timeslot, disregarding low-power operation.

In centralized link scheduling (CLS) [40], the sink reserves slots for a newly joining node at every node along the path to that node. When a node changes its preferred parent, it sends a removal request to the sink, de-allocating the slots in each intermediate hop. However, it requires an end-to-end multihop signaling phase, resulting in massive communication overhead.

Distributed TSCH Scheduling. The goal of distributed schemes is to adapt to dynamic topology and traffic load changes efficiently without the signaling overhead to a central controller. For example, in local lock-based algorithms [41, 42, 43, 44], each node selects and reserves a timeslot not used by nearby interfering nodes. By announcing this reservation locally, the timeslot is *locked* for the node solely. However, notifying the reservation to the interfering nodes selectively is a complicated task in

practice. Besides, none of these works addressed the reservation overlapping problem.

Vallati *et al.* [45] improves the 6TiSCH minimal schedule by allocating shared slots dynamically. However, it still uses shared slots only, like the minimal configuration, thus suffering severe packet collisions and redundant overhearing.

6TiSCH defines SF0 [46], a minimal scheduling function using 6top protocol (6P) [47]. It estimates the number of slots required between two neighbors, and lets them know when to add or delete slots. However, it does not define which timeslots they should reallocate. Based on SF0, LLFS [48] daisy-chains the timeslots in a multi-hop path to reduce end-to-end latency. However, slot reallocation by SF0 occurs between every neighbor, incurring significant overhead for 6P negotiation such as 6P Request/Response messages.

DeTAS [49] is a decentralized version of TASA. In DeTAS, a node collects bandwidth requests from its children, adds them with its own bandwidth, and then forwards it to its parent recursively. Then, slot allocation starts from the sink. To reduce end-to-end delay and queue overflows, DeTAS schedules alternatively Rx/Tx slots along the path to the sink. However, if a packet is lost due to poor link quality, all the subsequent slots scheduled are wasted.

Some scheduling proposals [50, 51, 52, 53] allocate timeslots randomly. Then, using local information, nodes detect schedule collision between two interfering radio links and re-allocate the colliding slots. These works may handle traffic dynamics but have only been evaluated on small-scale low-density deployments. Higher density limits the available slots due to a large number of interfering nodes, and as a result, communication overhead increases substantially since more negotiation procedures are needed [32].

1.3 Outline and Contributions

This dissertation is organized as follows. In Chapter 2, we propose ATP-based *MAPLE* system for reliable and low-power bidirectional communication in mobile LLNs. It provides single-hop downlink based on the high-power gateway and multihop uplink based on an RSSI gradient field based opportunistic routing. We design several mechanisms to support this system architecture: (1) NACK-based local downlink retransmission improves downlink reliability without topology information. (2) High-resolution multi-sampling makes RSSI measurement stable enough to be used as a routing metric. (3) RSSI adaptation addresses the local maximum problem of the RSSI-based gradient field. We implement *MAPLE* on real embedded devices and experimentally evaluate its performance against the standard RPL, ORPL, and LOADng on a real 32-node testbed. Our evaluation shows that *MAPLE* achieves significantly better packet delivery performance and route adaptation according to topology change than RPL.

In Chapter 3, we analyze the impact of slotframe size, showing the limitation of setting it as a fixed global constant, offline. We design *TESLA* which includes four elements: (1) traffic information exchange by piggybacking on each frame, (2) contention level estimation, (3) periodic slotframe adjustment and sharing, and (4) multi-channel scheduling. We also implement prototype of *TESLA* and evaluate its performance on two distinct testbeds with 110 nodes and 79 nodes, showing that *TESLA* outperforms the state-of-the-art in terms of reliability and energy-efficiency using distributed dynamic scheduling.

In Chapter 4, we introduce *OST*, which improves *TESLA* by dedicating timeslot to each pair of nodes. In addition, *OST* allocates timeslots in on-demand manner when the burst of traffic occurs. We implement and evaluate *OST* on a large-scale multi-hop testbed, showing its outstanding performances.

Lastly, we conclude the dissertation in Chapter 5.

Chapter 2

MAPLE: Mobility Support using Asymmetric Transmit Power in Low-power and Lossy Networks

2.1 Introduction

Low-power and lossy network (LLN), multihop wireless network composed of resource-constrained embedded devices, has been used for a variety of applications including smart grid automated metering infrastructure (AMI), environmental monitoring, and wireless sensor network (WSN). Furthermore, with the emergence of Internet of Things (IoT) and cyber-physical systems (CPS), LLN is now going into a new phase for smart and daily life applications which include medical care services [4, 5, 6], smart market maintenance [7, 2], networked robots [8, 9], and more. A key challenge in many of these emerging applications is that they incorporate not only stationary but also *mobile nodes*. As an example, a hospital network can be connected with sensing and actuating devices on mobile patients and patient beds, which enables remote monitoring of medical signals. In smart market applications, mobile shopping carts are connected to an LLN, which is used for real-time advertisement of hot deals, cart location tracking, and virtual fencing. Market staffs can also carry low-power portable terminals for reporting status of inventory/stock and market condition.

For over a decade, LLN research community has elaborated network layer protocols for energy efficiency and high reliability on resource-constrained devices. As a cornerstone work, 6LoWPAN [54] and IPv6 routing protocol for LLNs (RPL) [10] were designed and standardized, enabling multihop IoT networks. Numerous network protocols based on RPL have been devised [11, 7, 12], such as ORPL [11], surpassing performance of RPL in terms of delay, energy-consumption, and reliability. However, although these protocols have been making progress gradually under the assumption of stationary network, they cannot be apparently adopted in LLNs with mobility, due to lack of providing any specific operation for mobility and identifying mobile nodes [13, 14, 15]. Several studies [13, 14, 15] confirm that protocols designed for stationary network, such as RPL, experience significant performance degradation when operating with mobile devices.

There have been several work [19, 13, 17, 23, 20, 18], such as LOADng (LLN On-demand Ad-hoc Distance-vector Routing Protocol - Next Generation) [17], designed to provide seamless connectivity for mobile nodes in LLN. However, each of these work has at least one of the limitations among the followings. (1) They disregard the protocol operation with radio duty-cycling, which is one of the most critical characteristics of LLN with battery-powered nodes, (2) up-to-date neighbor or routing information is required according to topology dynamics caused by mobility, adding a significant network overhead, and (3) even if they showed performance improvement in idealistic simulator-based evaluation, the same has not been shown in practice via real experiments.

This work investigates how to provide bi-directional connectivity between the LLN border router (LBR, also referred to as 'gateway') and each *mobile node* in LLN, both reliably and energy efficiently. In contrast to previous approaches, we exploit *asymmetric transmit power (ATP) architecture* [55] for LLN with mobile endpoints. It has been more common place that LLN routers are plugged in, especially in indoor environments which have abundant outlets [7, 5], and only the endpoints are battery-operated.

It is also possible for smart grid applications such as automated metering infrastructure (AMI) where power is a given [56]. In this context, recent LLN protocols, such as Thread [57] and BLEmesh [58], even 'force' LLN routers to be plugged in. Without an energy constraint, these plugged-in routers can utilize much higher transmit power than battery-powered endpoint devices (e.g., 30 dBm vs. 0 dBm), which allows asymmetric transmit power to be a viable design choice in this regime.

As an example of ATP-based applications, a smart market [7, 2] uses a high-power gateway for disseminating price information and advertisements. At the same time, it collects stock status from portable terminals carried by mobile staffs, or locations of shopping carts. In a hospital network, low-power sensing and actuating devices report patient's condition such as vital signals. The information gathers in a single network gateway, and it can maintain and process the information. In addition, when an emergency occurs, it can control remote medical devices (attached to patients) immediately using high-power transmissions.

ATP-LLN provides a *single hop downlink* (from gateway to endpoints) and *multihop uplink* (from endpoints to gateway) architecture. A number of studies have explored its potential [55, 7, 2], showing improvement in downlink reliability and energy consumption when all nodes are static. Building on these previous work, *we argue that with a careful design, the ATP architecture is also useful for supporting mobile nodes*. To this end, we design *MAPLE*, an ATP-based LLN protocol that supports mobile endpoints by providing reliable and energy-efficient bi-directional communication under dynamic topology variation.

In our *MAPLE* design, each low-power node expects to receive a downward packet *directly* from the high-power gateway (*i.e.,* single hop). To improve reliability, each node sends a negative acknowledgement packet (NACK) when detecting a missed downward packet, which triggers *local retransmissions* from its neighbor nodes. Unlike the previous approach in [55], *MAPLE*'s neighbor forwarding works *without topology information*. On the other hand, low-power uplink transmissions require multihop

routing. To obtain path diversity in dynamic mobile environments, *MAPLE* uses *opportunistic routing* [11] where a sender simply broadcasts packets and each receiver decides whether to relay the packets by considering its and the sender's uplink routing metric. For the routing metric, *MAPLE* uses *RSSI of the gateway's high power transmissions* given that RSSI (Received Signal Strength Indicator) generally decreases with distance (*i.e.,* providing indirect geographical information). A single high-power transmission can update all nodes' routing metric at once, creating an *RSSI gradient field* in the network, and low-power nodes are completely free from a control packet overhead and routing table size limitation.

In doing so, we address well-known concerns about the RSSI measurements; it is unstable and time varying [59, 60, 61]. To this end, *MAPLE* obtains a *stable and interference-free RSSI value* from each packet of the gateway by using a high-resolution multi-sampling technique [62, 63], and updates this value with periodic high-power beacon transmissions [7]. Furthermore, we also tackle the *local maximum problem* [64], a representative problem in geographical routing, through dynamic and distributed adaptation of the RSSI metric. We implement *MAPLE* on ContikiOS and extensively evaluate the performance of *MAPLE* on a real LLN testbed in both stationary and mobile scenarios. Our results show that *MAPLE* significantly outperforms representative LLN protocols, *i.e.,* RPL [10], ORPL [11], and LOADng [17], in terms of reliability and energy consumption.

The contributions of this work are threefold:

- We propose ATP-based *MAPLE* system for reliable and low-power bidirectional communication in mobile LLNs. It provides single-hop downlink based on the high-power gateway and multihop uplink based on an RSSI gradient field based opportunistic routing.

- We design several mechanisms to support this system architecture: (1) NACK-based local downlink retransmission improves downlink reliability without topology information. (2) High-resolution multi-sampling makes RSSI measurement sta-

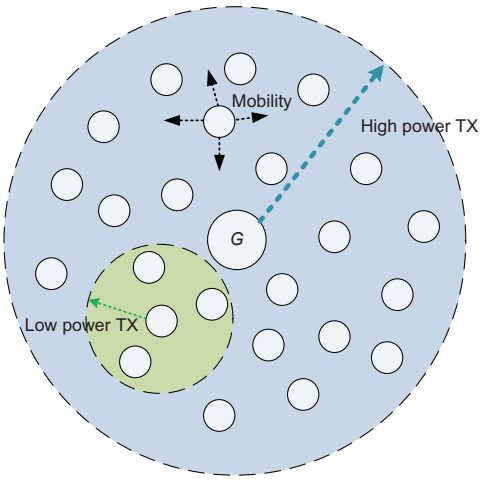


Figure 2.1: Network model of *MAPLE*.

ble enough to be used as a routing metric. (3) RSSI adaptation addresses the local maximum problem of the RSSI-based gradient field.

- We implement *MAPLE* on real embedded devices and experimentally evaluate its performance against the standard RPL, ORPL, and LOADng on a real 32-node testbed. Our evaluation shows that *MAPLE* achieves significantly better packet delivery performance and route adaptation according to topology change than RPL.

The remainder of this chapter is organized as follows: In Section 2.2, we present the design of our proposed scheme, *MAPLE*, and elaborate on its main functional blocks. We discuss the implementation details and present the evaluation results in Section 2.3. We conclude the chapter in Section 2.4.

2.2 MAPLE Design

In this section, we describe *MAPLE*, Asymmetric Transmit Power-based Mobile LLN architecture that provides mobility support with low radio duty-cycle. Fig. 2.1 de-

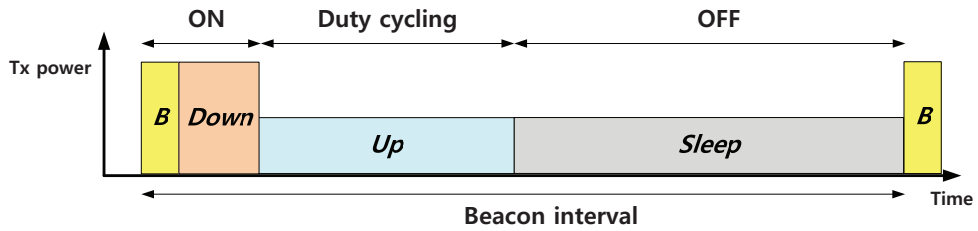


Figure 2.2: Superframe structure of *MAPLE*.

picts the system model we consider. There are a large number of low-power nodes and a single high-power gateway node (G). We assume that each node could be mobile, but is always located within high-power transmission coverage of the gateway. Thus transmission of gateway reaches all the nodes in a single hop. However, uplink communication works in a multihop manner due to low transmit power of low-power nodes.

As shown in Fig. 2.2, *MAPLE* repeats a superframe every beacon interval, which is divided into four periods: (1) beacon period (B), (2) downlink period (*Down*), (3) uplink period (*Up*), and (4) sleep period (*Sleep*). First, the gateway transmits a beacon in the B period and downward packets in the *Down* period, both with high transmit power. During these two periods, the other low-power nodes are not allowed to send any packet to ensure that the gateway's transmissions are free from contention and collision [7]. Instead, they continuously listen to the medium to receive packets from the gateway. In the subsequent *Up* period, the low-power nodes send and receive upward packets on top of a duty-cycling MAC protocol for low-power operation[1]. At the end of *Up* period, a low-power node stops duty-cycling and turns off its radio for energy saving. During this *Sleep* period, there is no packet communication. After the *Sleep*

[1]Any of synchronous [65, 66] or asynchronous duty-cycle MAC protocols [67, 68, 69] could be used in *MAPLE*. Without loss of generality, we use the ContikiMAC [67], a representative asynchronous MAC protocol for LLN.

period, every low-power node turns its radio on again to receive next beacon at the right time.

Except for the *B* period, lengths of the other periods could be controlled by the gateway according to traffic generation rate and network size. For instance, *Sleep* duration might be set to zero in order to minimize channel contention in the *Up* period when intensity of uplink traffic is high. On the contrary, *Sleep* period can be made longer for ultra low-power operation when uplink traffic load is light. The remainder of this section provides detailed descriptions of *B*, *Down*, and *Up* periods.

2.2.1 Beacon and Beacon Period

In the *B* period, the gateway transmits beacons using high transmit power so that all nodes can receive them. The beacon has three major roles. Firstly, it is used for *network-wide time synchronization* of all the nodes. Every beacon includes durations of *B*, *Down*, *Up*, and *Sleep* periods, and a node willing to join the *MAPLE* network must wait and listen for the first beacon reception. Once a node receives a beacon correctly, it can be *synchronized* and share the superframe structure illustrated in Fig. 2.2. Secondly, the beacon includes the destinations of the downward packets which will be sent in the subsequent *Down* period. The destination information is used for NACK-based local retransmission of downward packets (explained in Section 2.2.2). In our experiments, we have used 5 seconds as the beacon interval.

Lastly, the beacon is used to generate an RSSI-based gradient field throughout the network. Whenever each low-power node receives a high-power packet from the gateway, it records the RSSI ($RSSI_G$). Ideally, the closer a node is to the gateway, the larger $RSSI_G$ it obtains. *MAPLE* exploits this $RSSI_G$-gradient field for multihop opportunistic routing in the *Up* period, as described in Section 2.2.3.

2.2.2 Downlink Transmission: Local NACK and Retransmission

In the *Down* period, the gateway transmits downward packets with a high transmit power. Low-power nodes are not allowed to transmit any packet during this period to avoid packet collision [7]. While the single-hop high-power downlink transmission based on the ATP architecture removes the need for downlink routing, a subtle issue still remains: how to acknowledge a downward packet for reliable packet delivery. This is because a low-power destination node is not likely to deliver an acknowledgement (ACK) to the gateway in a single hop due to its limited transmit power.

An ACK may be forwarded towards the gateway through a *multihop* route [70, 71], which creates a significant communication overhead. Another approach is for the destination node to send ACK packets *locally* to its neighbors [55, 7]. When the destination's neighbors overhear a downward packet but do not receive a local ACK from the destination, they locally retransmit the downward packet on behalf of the gateway. However, this requires up-to-date neighbor information which is hard and expensive to get in mobile LLNs since topology changes continuously. Furthermore, this approach creates large number of (potentially redundant) local ACKs, which may be an overkill under high-power gateway transmissions where downlink loss rates are typically low for most endpoints.

For these reasons, our solution is to send a *local negative-ACK (NACK)* for a downward packet loss. As described in Section 2.2.1, a low-power node knows the destinations of all downward packets that are sent in a *Down* period by receiving a beacon in the previous *B* period. When a low-power node expects to receive a downward packet for itself in a *Down* period but misses it, the node sends a local NACK. Given that the *Down* period is only for the gateway's transmission, local NACK is transmitted in the following *Up* period with a low transmit power.

At the same time, each low-power node overhears all downward packets in a *Down* period and holds them until the end of the subsequent *Up* period. The gateway also holds these downward packets. Upon receiving a NACK in the *Up* period, a low-power

node (or the gateway) searches the stored downward packets to check if any of them is destined to the NACK sender. If it has one, it forwards the downward packet to the NACK sender with a low transmit power. The node can try the local retransmission several times until receiving an ACK from the destination. This approach improves downlink reliability without topology information nor redundant ACK transmissions.

When a node fails to receive a beacon in a *B* period, it can still use the superframe structure given that the time synchronization is valid for a while. But it does not find out the destination information of downlink transmissions in the following *Down* period. In this case, the node assumes that a downward packet towards itself is lost and transmits a NACK in the following *Up* period. This triggers local retransmissions from the neighbors if the downward packet loss really happens, providing reliable downward packet delivery regardless of a beacon loss.

MAPLE also supports network-wide broadcast service from the gateway. To this end, the gateway can inform all low-power nodes of the existence of a broadcast packet by including IPv6 link-local broadcast address in the beacon, instead of a downward unicast destination. The rest of operation with local NACK and retransmission is the same as the unicast case.

Lastly, like high-power beacon transmissions, each high-power downlink transmission is also used to update $RSSI_G$ for uplink routing, regardless of its destination. This enables a low-power node to update its RSSI-based routing metric frequently even if it fails to receive a recent beacon.

2.2.3 Uplink Transmission: RSSI Gradient-based Routing

In the *Up* period, the gateway mainly listens to the medium, but can transmit ACKs for uplink traffic or perform local retransmissions for NACKs, both with a low level of transmit power. Meanwhile, low-power nodes send/receive packets with duty-cycle for energy saving.

For multihop uplink transmissions, *MAPLE* borrows opportunistic routing concept

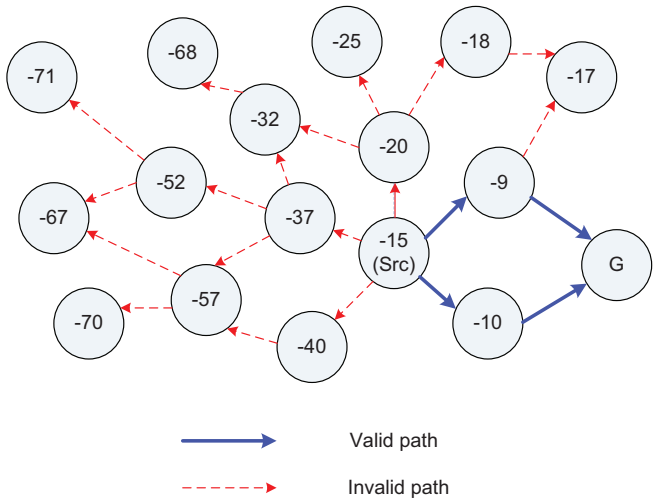


Figure 2.3: Opportunistic uplink routing using RSSI-gradient field.

in ORW [11] but uses a gradient-field of $RSSI_G$ rather than the EDC metric. Specifically, each packet sender piggybacks its $RSSI_G$ value in an upward packet and simply broadcasts it. Given that large $RSSI_G$ indicates high proximity to the gateway, when a node receives an upward packet and it has higher $RSSI_G$ than the packet sender, it sends an ACK and forwards the packet. Fig. 2.3 shows an example of the $RSSI_G$-gradient based opportunistic forwarding. The number in each node indicates $RSSI_G$ value obtained through a previous beacon or downward packet reception (The unit is dBm). Although there are five neighbors of data source (expressed as Src), only two nodes among them have higher $RSSI_G$ values (-9 dBm and -10 dBm) than $RSSI_G$ of the source (-15 dBm), and thus are valid candidates for data forwarding.

Compared to the state-of-the-art LLN routing protocols such as RPL [10] and LOADng [17], $RSSI_G$-based opportunistic routing of *MAPLE* has three primary strengths. Above all, *MAPLE*'s opportunistic routing requires each node to maintain only its $RSSI_G$ value without any neighbor information, letting a resource-constrained device keep low and constant memory footprint regardless of network size or density. In addition, $RSSI_G$ is updated solely based on the gateway's high power transmissions,

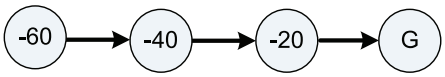
which enables a low-power node to maintain a valid routing metric without any control packet overhead. Lastly, given that one high-power transmission can update $RSSI_G$ of all nodes, the gateway can freely adjust periodicity of $RSSI_G$ update depending on mobility scenarios.

However, an $RSSI_G$-gradient has the local maximum problem which is common in geographical routing. It is even more so when applying the $RSSI_G$-gradient in a real wireless environment since RSSI is highly variable even if all nodes are stationary due to multipath fading, shadowing, and various interference. When a route with a non-monotonic $RSSI_G$-gradient is encountered, uplink packets may not be relayed anymore in the middle of the path, leading to packet losses. *MAPLE* addresses this phenomenon by adjusting $RSSI_G$ intentionally, but carefully. This is explained in Section 2.2.4.

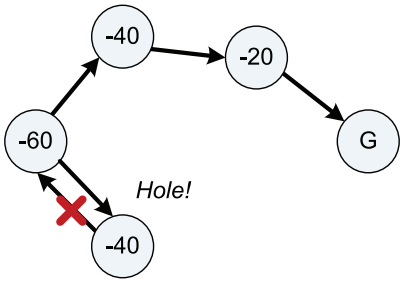
2.2.4 Local Maximum Problem and RSSI Adaptation

In a free space where every node has a line-of-sight (LOS) link with the gateway, $RSSI_G$ can be used to approximately indicate the straight-line distance to the gateway using free-space RF propagation models. Meanwhile, *MAPLE* interprets this information as the distance along the routing path. If the network density is high enough to provide sufficient number of forwarding nodes such that linear shaped routing path can be obtained for any node, as depicted in Fig. 2.4(a), this interpretation is valid.

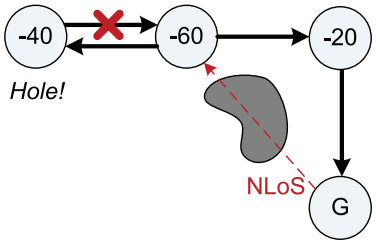
However, this interpretation might not hold when the path is curved like Fig. 2.4(b), which is highly probable when the node density is low. Furthermore, it is also invalid if there are obstacles which block off LOS with the gateway as shown in Fig. 2.4(c). In an ATP network where LOS and Non-LOS (NLOS) nodes with the gateway coexist, $RSSI_G$ field is not generated with a monotonic gradient along the desired path. For example, in Figs. 2.4(b) and 2.4(c), a packet which is generated by or sent to a node with *local maximum RSSI* (denoted as a '*Hole!*' in the figures) cannot be forwarded any further towards the gateway as if it gets stuck at a dead-end, or in what we call a



(a) Ideal RSSI gradient



(b) Local maximum problem 1



(c) Local maximum problem 2

Figure 2.4: Ideal case and local maximum problems in RSSI gradient based routing.

hole. This problem makes the hole node suffer from consecutive packet losses either at the link since there is no forwarder with better $RSSI_G$, or at the queue due to memory overflow as resource-constrained nodes have very small size queues.

To resolve this *hole problem*, we design a light-weight but effective algorithm for $RSSI_G$ adaptation at the hole node. Each low-power node maintains the most recent transmission history list ($list_{tx}$). As a first-in first-out (FIFO) list, $list_{tx}$ is updated whenever an anycast transmission is completed. If the transmission is acknowledged, which means there is at least one neighbor who offers routing progress as the next hop,

transmission success (S) is recorded in $list_{tx}$. On the other hand, when there is no incoming ACK packet before a timeout (which is, for example, a whole sleep interval in asynchronous sender-based duty-cycle MAC protocols), transmission failure (F) is added to the list. If the number of F (N_F) exceeds a pre-specified threshold, N_F^{th}, the node recognizes itself as a *hole*. Then it tries to escape from the hole by lowering its $RSSI_G$ deliberately. With N_F above N_F^{th}, $RSSI_G$ is reduced by Δ_{hole} every packet loss in the link. The updated RSSI in the hole, $RSSI_{hole}$, is expressed as,

$$RSSI_{hole} = \max\left(RSSI_G - (N_F - N_F^{th}) \cdot \Delta_{hole},\ RSSI_{\min}\right),$$

where $RSSI_{\min}$ is the minimum RSSI value available in a radio. In our implementation, N_F^{th} and Δ_{hole} are 2 and 20 dBm, respectively.

2.2.5 Implementing Reliable $RSSI$ Capture

Even though *MAPLE* handles the hole problem, instantaneous RSSI is well known to be unpredictable in wireless links due to multipath fading, external/internal interference and various environmental factors. On the other hand, the primary principle for uplink routing of *MAPLE* is to adapt an $RSSI_G$-gradient field to physical topology changes like node's mobility, while minimizing the effect of wireless unpredictability. As one of possible approaches, more $RSSI_G$ samples could be collected by increasing the number of high-power transmissions. Then, the average $RSSI_G$ can be used for an uplink routing metric. However, this inevitably brings about more energy consumption.

We consider an IEEE 802.15.4 compliant radio, *i.e.,* CC2420 [72], as an implementation example. During a packet reception, a 2-byte frame check sequence (FCS) follows the last MAC payload byte. FCS is automatically generated and verified by the hardware when the `MODECTRL0.AUTOCRC` control bit is set[2]. Then the first FCS byte is replaced with the 8-bit RSSI value, which can be read by the upper layer. In

[2]It is recommended to always have this control bit enabled, except possibly for debug purposes [72].

CC2420, this RSSI value is measured over the first 8 symbols following the start of frame delimiter (SFD), and can be obtained from the `RSSI.RSSI_VAL` register.

In our system, instead of reading only the first byte of FCS for RSSI, we obtain *multiple* RSSI values from a *single* packet in a similar way to [62]. The RSSI value in `RSSI.RSSI_VAL` is always averaged over 8 symbol periods (128 microseconds) and continuously updated for each symbol after RSSI has become valid. We let a low-power node detect an SFD interrupt for the *B* and *Down* periods and then immediately read and store `RSSI.RSSI_VAL` register value every 8 symbols. Given that, following SFD, a frame length byte and IEEE 802.15.4 MPDU (maximum size of 127 bytes) come, 32 RSSI samples can be acquired at most from a single packet. In our implementation, we use the beacon size of 51 bytes and let 10 RSSI samples obtained. We average these RSSI samples and use the averaged value as $RSSI_G$.

2.3 Performance Evaluation

In this section, we evaluate *MAPLE* experimentally through a prototype implementation, and compare it with RPL, ORPL, and LOADng in terms of reliability and energy efficiency. We evaluate on a network with and without mobility using three scenarios; 1) static network (no mobility) 2) a single mobile device, and 3) three mobile devices. In addition, we also run Cooja simulations to evaluate and compare the performance under high level of network mobility.

2.3.1 Methodology and Experiment Setup

Fig. 2.5 presents the topology of our testbed where a total of 32 TelosB clone devices are deployed with one node acting as the gateway (or root) of the network. For *MAPLE*, we use the MTM-CM3300MSP device as the high-power gateway, which is similar to a TelosB with a 10 dB power amplifier. The other low-power nodes use a transmit power of -20 dBm while the high-power root uses 10 dBm. For other compared schemes,

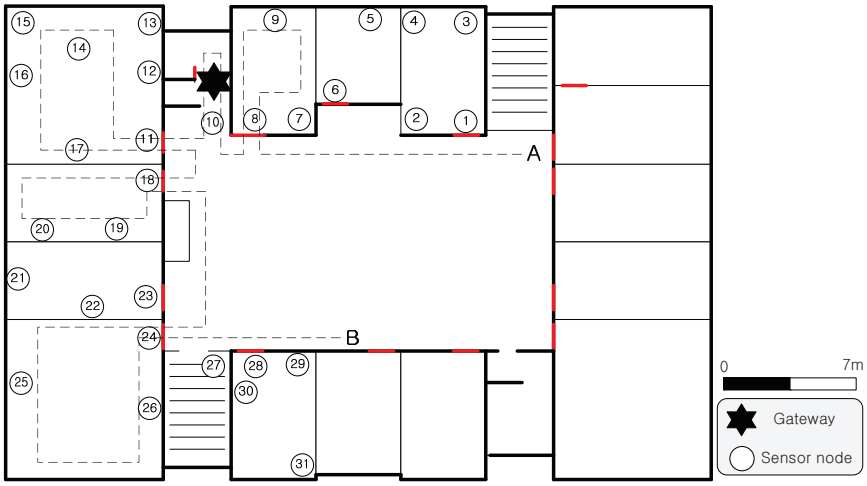


Figure 2.5: Topology map of indoor 32-node LLN testbed with a moving path for node mobility.

the gateway uses the same transmit power as other nodes. This leads to a maximum diameter of 5 hops in case of RPL.

We consider a bidirectional traffic scenario. Each node generates an uplink packet every 75 seconds, while the gateway generates equal rate of downlink packet per node, resulting in average inter-packet interval of 2.5 seconds in both directions. In our experiments with *MAPLE*, within the beacon interval of 5 seconds, the duration of *Down* period is 20 ms to accommodate two downlink packets. Remaining time is used for *Up* period (with no *Sleep* period). In all experiments, the application payload is 24 bytes, which is carried in UDP datagrams over 6LoWPAN. All our experiments were done on Zigbee/IEEE 802.15.4 channel 26 (*i.e.,* no WiFi interference) and in a stable channel environment with minimal external interfering factors, such as uncontrolled human movement and environment changes. Unless specified, all our results are an average of three runs of 1-hour experiments from different times of the day.

For comparison with state-of-the-arts, we use RPL [10], ORPL [11], and LOADng [17]. All these protocols including *MAPLE* are implemented on top of ContikiMAC [67] in ContikiOS [73]. We use Contiki's default values for the number of transmission at-

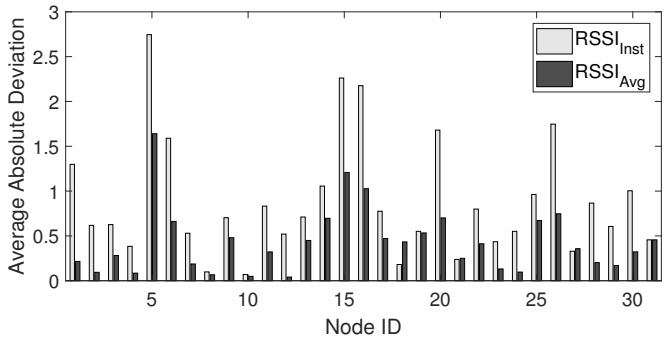


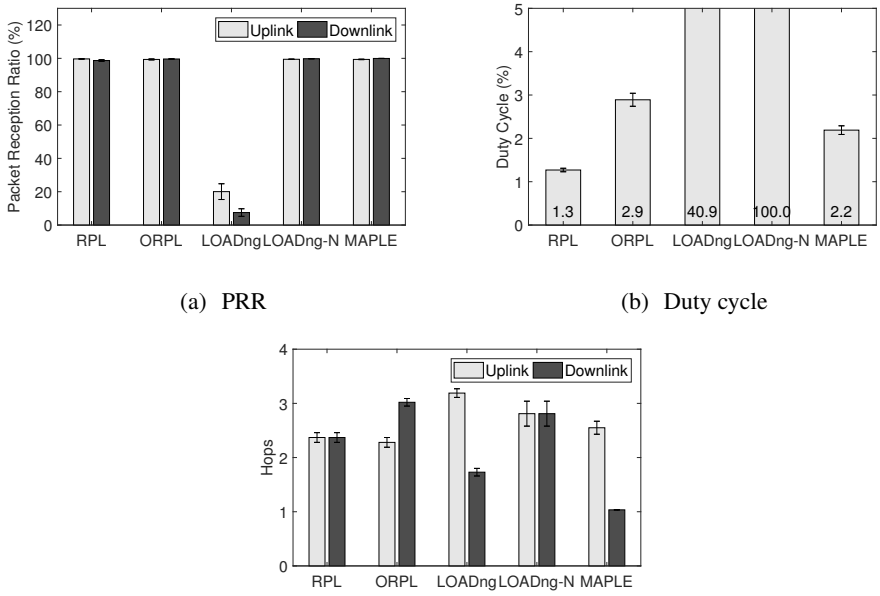
Figure 2.6: Average absolute deviation for instantaneous RSSI and average RSSI.

tempts and duty cycle rate, 5 and 8 Hz, respectively. Note that the gateway does not duty-cycle in order to handle all network traffic. ContikiMAC has a phase-lock mechanism, where a sender records wake-up phase of its neighbors, and uses it for timely transmission in an energy-efficient manner. Phase-lock can be used for unicast of RPL or LOADng, neither broadcast nor anycast of ORPL and *MAPLE*.

With this configuration, we first check the total size of volatile memory for each protocol. RPL and ORPL consume 8.6 and 8.4 kBytes, respectively, and LOADng uses 9.5 kBytes of RAM. On the other hand, *MAPLE* spends only 7.3 kBytes of memory as it does not need to maintain neighbor or routing information. This result verifies *MAPLE* outperforms state-of-the-arts with regard to memory footprint.

2.3.2 Static Network

We first evaluate the effect of multiple RSSI sampling in a single packet, which is used in *MAPLE* for reliable RSSI capture. To this end, we run *MAPLE* with beacon interval of 5 seconds in the testbed shown in Fig. 2.5. Whenever a low-power node receives a beacon, it records two kinds of RSSI values, $RSSI_{Inst}$ and $RSSI_{Avg}$. $RSSI_{Inst}$ indicates the instantaneous RSSI value read from the first FCS byte of the received beacon. On the other hand, $RSSI_{Avg}$ is the average of 10 RSSI samples stored after an SFD interrupt based on our approach described in Section 2.2.5. The experiment



(a) PRR

(b) Duty cycle

(c) Hops

Figure 2.7: PRR, duty-cycle, and number of hops results from a static network.

ran for 4-hours, leading to about 2,900 beacon receptions. We observed that the difference between the mean values of $RSSI_{Inst}$ and $RSSI_{Avg}$ is marginal less than 1%. However, their deviations over time are quite different. Fig. 2.6 presents the average absolute deviations over time for each node. For the $RSSI_{Inst}$ measurements, in the worst case (*i.e.,* node 5), the deviation is close to 3. We stabilized this unpredictable RSSI by obtaining and averaging multiple RSSI samples, resulting in 40% improvement.

Using our stabilized $RSSI_{Avg}$, now we evaluate the performance of *MAPLE* against RPL, ORPL, and LOADng in a static network. Fig. 2.7(a) shows the end-to-end packet reception ratio (PRR) for uplink and downlink traffic. We observe that LOADng is not suited for a duty-cycled LLN, showing severe PRR degradation. This result comes from the way of its reactive route search based on network flooding. A broadcast message for route request occupies the medium within a whole sleep interval (*i.e.,* 125 ms in 8 Hz duty-cycle rate). What is worse, it is propagated throughout the network

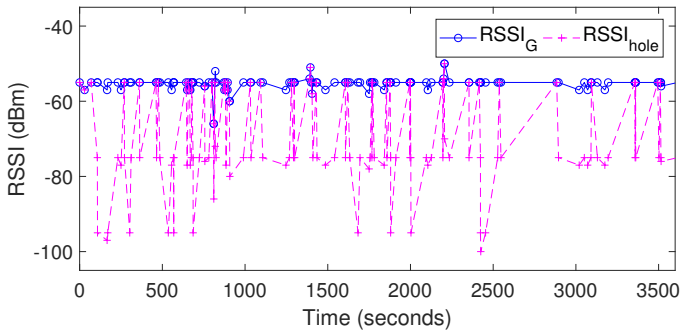


Figure 2.8: RSSI adaptation of a hole node (node 27).

hop-by-hop. This significant overhead incurs network congestion extremely, leading to excessive energy consumption, as illustrated in Fig. 2.7(b) as a metric of duty-cycle, the portion of radio on-time.

Apart from duty-cycling MAC, in order to see routing performance of LOADng solely, we also examine LOADng without duty-cycling. Instead of ContikiMAC, we build LOADng on NullRDC (a simple MAC implementation without duty-cycling, provided by Contiki), denoted as *LOADng-N* in Fig. 2.7. Figs. 2.7(a) and 2.7(b) show that LOADng-N has comparable PRR with RPL and ORPL with the cost of 100% duty-cycle. RPL shows the lowest duty-cycle with nearly perfect PRR, owing to the phase-lock mechanism used in ContikiMAC for unicast transmissions. ORPL also achieves about 100% reliability, but spends more energy than RPL due to anycast-based transmissions without the phase-lock operation. Fig. 2.7(c) presents the number of hops for uplink and downlink traffic. In ORPL, uplink hop is shorter than that of downlink. As the root always listens to the medium without duty-cycling, it is likely to receive and acknowledge an uplink packet from a neighbor earlier than any other duty-cycling nodes. In case of downward traffic, on the other hand, it is common that the packets are relayed by other early wake-up nodes.

MAPLE has over 99% uplink PRR with reasonable energy consumption in the static network even though *MAPLE* was devised for mobile network. During the ex-

periment, we observed local maximum RSSI problem in node 27 (see Fig. 2.5). It was deployed in a relatively open space. As a result, it receives beacons or downlink packets from the gateway with higher RSSI than its neighbors, which are located inside the rooms. Fig. 2.8 depicts how the node stuck in a hole adapts its RSSI. Whenever it detects itself in a local-maximum point, experiencing transmission failures, it escapes from the hole by adjusting its routing metric from $RSSI_G$ into $RSSI_{hole}$. With this approach, *MAPLE* tackles the local-maximum problem, guaranteeing high reliability. However, the hole node shows the highest energy consumption (*i.e.,* 4.15% of duty cycle) due to transmission failures it experiences during the RSSI adaptation. Nevertheless, *MAPLE* shows lower energy consumption than ORPL since it benefits from its ATP architecture which enables single-hop downlink transmission.

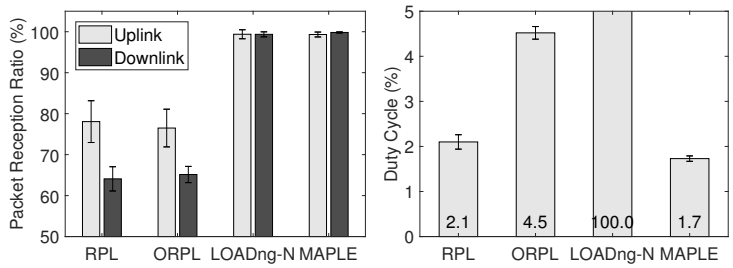
Thanks to NACK-based local retransmission, *MAPLE*'s downlink PRR is also nearly perfect. We discovered that more than 3% of high-power downlink packets were lost during the experiments. In particular, the nodes which have low SNR from the gateway, such as nodes 26 and 27, went through about 20% of downlink packet loss. Nevertheless, by broadcasting NACK locally, they could receive the downlink packets successfully from neighbors, reaching 99.94% downlink PRR with average hop count of 1.035, as shown in the Fig. 2.7.

2.3.3 Network with Mobility

Having the experimental results from a static LLN as a basis, we now move on to our main evaluation with mobility.

One Mobile Node

In this experiment, we first introduce a single mobile node into the network to examine its performance under mobility. To keep the number of nodes consistent, we use node 15, which is in a corner of testbed topology, as a mobile node. While carrying the node, we walked back and forth along the path between points *A* and *B* shown in Fig. 2.5. In



(a) PRR (b) Duty cycle

Figure 2.9: PRR and duty-cycle results for the mobile node.

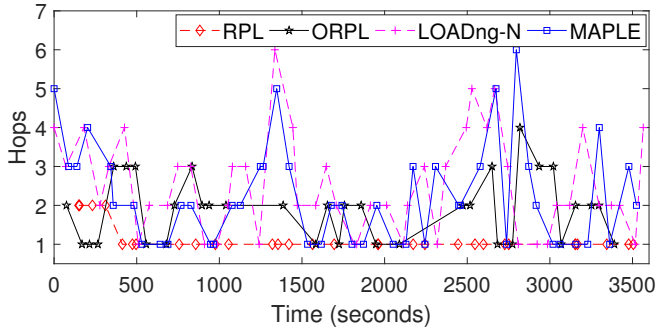
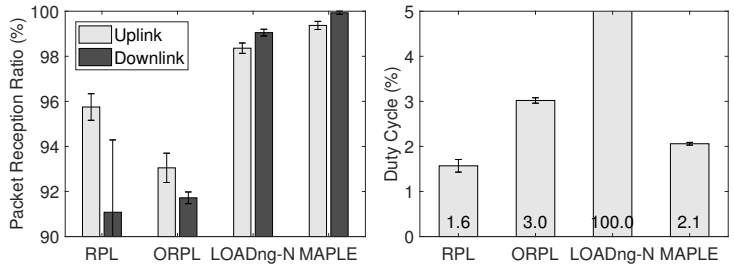


Figure 2.10: Hops for uplink traffic from the moving node.

each room we enter, we stay 90 seconds while still walking, and then exit. With this movement, one-way trip time for the whole path is about 5 minutes.

Fig. 2.9(a) presents the PRR achieved for the uplink and downlink traffic from/to the mobile node. The performance of RPL and ORPL degrades severely. RPL did not react properly to link disconnections. Fig. 2.10 shows the number of hops that uplink traffic from the mobile node goes through. In RPL, the mobile node has the root as its preferred parent most of time. Even if it sometimes detects link failure with the root, the link becomes valid again by node's mobility. It makes the link quality remain good enough for the mobile node to keep the current preferred parent. In addition, *the effort after detecting packet losses* could not be a solution to provide seamless connectivity. The main problem in ORPL is that, to update link quality with neighbors, it relies on



(a) PRR (b) Duty cycle

Figure 2.11: PRR and duty-cycle results from the network with three mobile nodes.

Trickle algorithm [74], thus not reacting topology dynamics promptly. In both RPL and ORPL, uplink PRR is better than downlink since the root is always-on. From the view of the always-on root, a train of transmission strobes, which were intended for duty-cycling receivers, have an effect of multiple retransmissions.

Fig. 2.10 also shows the mobile node with *MAPLE* or LOADng-N[3] adapts its uplink hops according to change of its location, achieving PRR above 99% in all cases as shown Fig. 2.9(a). Fig. 2.9(b) illustrates duty-cycle of the mobile node. *MAPLE* shows the highest energy efficiency whereas RPL and ORPL suffer from frequent packet losses and retransmission, incurring more energy consumption.

Three Mobile Nodes

Now we consider three mobile nodes with 29 static nodes. Among three mobile nodes, two nodes continuously moved back and forth at typical walking speed along the path shown in Fig. 2.5, but in opposite direction to each other. Their one-way trip time is 3 minutes. The other mobile node moved in the same manner as the previous experiment with a single mobile device. Overall, the performance of each mobile node was consistent with our previous results. Thus, in this subsection, we focus on how much the mobile nodes affect the performance of the whole network.

[3]We exclude LOADng with duty-cycling from mobile experiments since we already identified its chaotic performance through the previous static experiment.

Fig. 2.11(a) plots the average PRR for all low-power nodes. For RPL and ORPL, not only the mobile nodes but also their descendants sequentially are affected by wrong routing decisions with outdated information. The PRR of LOADng-N remains still good under mobility. However, it sometimes fails to transmit a packet with more than 1% of loss rate for both downlink and uplink. As an intrinsic drawback of unicast transmission, it cannot benefit from multi-path diversity.

Meanwhile, there are three reasons why *MAPLE* shows the highest PRR for uplink traffic. Firstly, *MAPLE* is more robust to link failure with spatial diversity using opportunistic transmissions. Next, routing information throughout the network (*i.e.,* an $RSSI_G$ gradient) is newly updated every beacon interval of 5 seconds. Given the network mobility of human walking speed, with this interval, it is enough for the $RSSI_G$ gradient to be tuned to topology dynamics. Lastly, even though the routing information is not perfect, (*i.e.,* with a non-monotonic $RSSI_G$ gradient along a path) incurring some transmission failures, the node tries more and more paths by lowering its $RSSI_G$ gradually before the packet is dropped. Downlink PRR of *MAPLE* is more reliable, by using high-power transmission and introducing local NACK-based retransmission mechanism.

As presented in Fig. 2.11(b), *MAPLE*'s energy-efficiency is also good, which is mainly attributed to the effect of eliminating multihop downlink relay between low-power nodes. Additionally, with regard to routing overhead, while the compared protocols use broadcast packets which occupy the medium during a whole sleep interval and need to be forwarded hop-by-hop, the cost in *MAPLE* is negligible, a single timely transmission of the beacon.

2.3.4 Simulation Study under More Mobility

In our testbed experiments, we were unable to increase the number of mobile nodes to more than 3 due to limitations in human resources. For this reason, we instead used Contiki-based Cooja simulator to add more mobile nodes and expand our evaluation

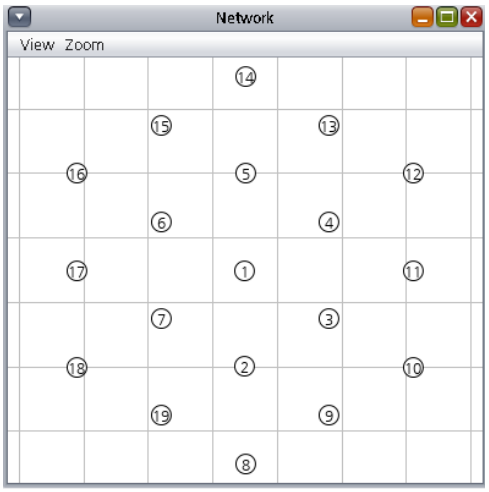
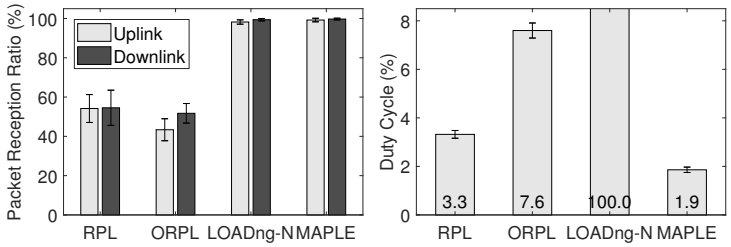


Figure 2.12: Network topology for static nodes in Cooja simulation.



(a) PRR (b) Duty cycle

Figure 2.13: PRR and duty-cycle results from 50-node simulation.

to showcase the performance of *MAPLE* under higher level of network mobility. As shown in Fig. 2.12, we firstly deployed 19 static nodes to guarantee connectivity between mobile nodes and the gateway (*i.e.,* node 1). The distance between two adjacent nodes are 15 m. Then, for network mobility, we added another 31 nodes (total of 50 nodes) which independently move within the range of 45 m from the gateway. Each node follows Random way-point model [**?**] with the minimum and maximum speeds of 0.5 m/s and 2.0 m/s, respectively. For *MAPLE*, the gateway uses a transmit power of 0 dBm. The other low-power nodes use -10 dBm, having transmission range of about 17 m. All the other experimental settings are identical to those of previous testbed

experiments.

Fig. 2.13(a) and Fig. 2.13(b) plots the average PRR and duty-cycle of all low-power nodes, respectively. Compared to the previous experiment results, we found that RPL and ORPL are impacted severely by increased network mobility, showing less than 60% PRR with larger duty-cycle. Meanwhile, LOADng-N achieves PRR near 100% with the expense of 100% of duty-cycle. Most importantly, *MAPLE* outperforms the others in terms of both PRR and duty-cycle. Surprisingly, it shows lower energy consumption than RPL despite RPL works over ContikiMAC which includes the phase-lock operation to minimize energy consumption for unicasts. Overall, simulation results are in-line with the experiment results, and shows that *MAPLE* achieves significantly better reliability as well as energy efficiency under high mobility.

2.4 Conclusion

We presented *MAPLE*, an asymmetric transmit power-based routing architecture that supports mobility of resource-constrained devices in LLNs. Using high transmit power of the gateway, LLN nodes are synchronized for low duty-cycle operation, and RSSI gradient field based opportunistic routing is designed which eliminates the need for any neighbor or routing table. This enables scalability, low and constant memory footprint, and provides responsive routing metric without control overhead. We obtain reliable RSSI measurements via multi-sampling approach, and resolve the local maximum problem through adaptive and local adjustment of the routing metric. We implemented *MAPLE* on a low-power embedded platform, and evaluated through experiments on a real multihop LLN testbed consisting of 31 low-power ZigBee nodes and 1 high-power gateway. We showed that *MAPLE* improves the performance of mobile devices in a multi-hop LLN testbed by 27.2%/55.7% and 17.9% in terms of both uplink/downlink reliability and energy efficiency, respectively. As future work, we plan to improve *MAPLE* in terms of latency and energy-consumption, and evaluate *MAPLE*

on large-scale testbed such as Indriya and IoT-LAB. We envision that our approach can be used in many practical indoor IoT applications where mobility is becoming an integral part of LLNs.

Chapter 3

TESLA: Traffic-aware Elastic Slotframe Adjustment in TSCH Networks

3.1 Introduction

Internet of Things (IoT) has opened a new era with low-power embedded devices. In industrial IoT networks, numerous sensors and actuators are deployed for system monitoring and remote control. From smart homes to smart cities [1, 2], new applications and network services are emerging such as electricity management, home security, health care [3], and smart grid. As IoT applications become diverse, the need for reliable, energy-efficient, and flexible (*i.e.,* adaptable to diverse and dynamic applications) network protocols is growing up steadily.

The IEEE 802.15.4-2015 [16] standardized the time-slotted channel hopping (TSCH) protocol for low-power and lossy networks (LLNs), a promising TDMA-like link layer protocol providing both high reliability and low energy operation. Compared with asynchronous duty-cycled MAC protocols [24, 25, 28], time-slot operation of TSCH saves redundant transmissions or listening for rendezvous time of data exchange. Additionally, channel hopping enables low-power communication to be resilient from narrow-band interference and multipath fading [29]. For the implementation of TSCH network, timeslot scheduling is required, but how to build and maintain the schedule is

out of scope of the IEEE 802.15.4-2015 standard. For this reason, a number of TSCH scheduling schemes have been proposed recently, such as the *minimal configuration schedule* [75] of 6TiSCH [76] and Orchestra [12] (Section 3.2.2 and 3.2.3).

Challenge. Any well designed protocol can end up with miserable performance if its parameters are not set appropriately [77, 78, 79]. Setting proper network parameters has been one of the most painful tasks in LLNs as well. Since it is hard to predict the impact of a parameter change on performance, it is exhaustive, empirical, and environment specific. In addition, a network parameter is usually set as a global constant (*i.e.,* all nodes have the same value), which cannot satisfy all nodes having different environments and roles. This may cause significant inefficiency since each node's situation is different and may change at run time, not only due to its physical surroundings but also routing topology [80], forwarding traffic intensity [81, 82], and application behaviors [5, 2].

Parameter selection for TSCH is not an exception. TSCH's slotframe structure is the basis of TSCH operation, but its size is set offline as a fixed global constant. On top of significant burden for empirical optimization, even if the slotframe size is optimized, it is still problematic since all nodes share a single slotframe size, disregarding routing topology and traffic intensity for each node: (1) When the slotframe is too small for the node experiencing low traffic load, it will waste energy due to idle listening. (2) When the slotframe is too large for the node under heavy traffic load, it cannot receive/forward many packets due to channel contention or queue overflow (Section **??**). To address this issue, each node should use a *different* slotframe size and *adjust* it with traffic-awareness at run time. To align with the basic design paradigm of LLN (simple and low overhead), this adjustment procedure should be light-weight and operate in a distributed manner based on local information.

Approach. How can each node *self-adjust* TSCH slotframe size at *run time*? We introduce *TESLA*, a novel traffic-aware elastic slotframe adjustment scheme as a solution (Section **??**). *TESLA* inherits and extends the Orchestra's receiver-based scheduler [12]

where each node has a single reception (Rx) slot per slotframe and sends a packet to a neighbor in the neighbor's Rx slot. Beyond Orchestra, in *TESLA*, each node obtains the amount of incoming traffic using locally piggybacked information from neighbors. It periodically self-estimates the contention level of the neighbors based on the traffic load, and adjusts its slotframe size: (1) When the contention level is high, it decreases slotframe size to receive more traffic from neighbors. (2) When the contention level is low, it increases slotframe size to save energy. (3) Otherwise it maintains slotframe size. Upon slotframe size change, the node informs its one-hop routing neighbors of the new slotframe size for seamless communication. Furthermore, *TESLA* also supports multi-channel operation to fully utilize available channel resources. Although our implementation is based on Orchestra, the state-of-the-art TSCH scheduling mechanism, the core idea of *TESLA* is general, applicable to any TSCH scheduling mechanism.

Contributions. The contributions of this work are threefold.

- Analysis on the impact of slotframe size, showing the limitation of setting it as a fixed global constant, offline.
- Design of *TESLA* which includes four elements: (1) traffic information exchange by piggybacking on each frame, (2) contention level estimation, (3) periodic slotframe adjustment and sharing, and (4) multi-channel scheduling.
- Prototype implementation (the code will be open after acceptance) and extensive evaluation on two distinct testbeds with 110 nodes and 79 nodes (Section **??**), showing that *TESLA* outperforms the state-of-the-art in terms of reliability and energy-efficiency using distributed dynamic scheduling.

3.2 Background

In this section, we provide a brief overview of TSCH, and two instances of TSCH scheduling implementation: 6TiSCH minimal configuration and Orchestra.

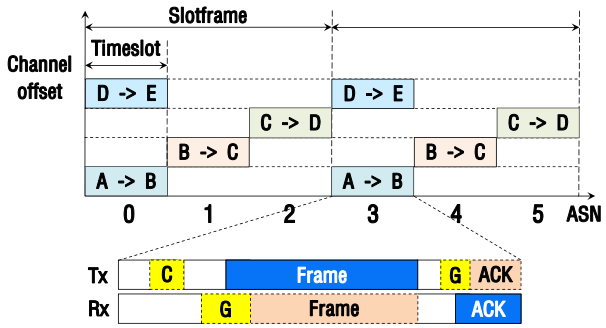


Figure 3.1: An example of TSCH slotframe schedule and timeslot with slotframe size of 3.

3.2.1 IEEE 802.15.4 TSCH

TSCH is a time-synchronous MAC specified in the IEEE 802.15.4-2015 standard [16]. Its synchronous operation saves energy by reducing redundant transmissions or idle listening compared to asynchronous MACs [27, 25, 28], and its channel hopping enables resilient operation over narrow-band interference and multipath fading [29].

TSCH network is globally time-synchronized, and time is divided into *timeslots* as in Fig. 3.1. Typical length of a timeslot is 10 ms, long enough for a single frame and an acknowledgement (ACK) to be exchanged. A *slotframe* is a collection of timeslots, continuously repeated in time. The number of timeslots in a slotframe, *i.e., slotframe size*, determines the period of each slotframe. Within a slotframe, *time offset* is defined as when the timeslot occurs, and *channel offset* denotes an offset value for channel selection. The total number of timeslots that has elapsed since the start of a TSCH network is defined as the absolute slot number (*ASN*). It increases globally every timeslot. In Fig. 3.1, when ASN is 2, node *C* can transmit a frame, node *D* can receive it, and the others sleep.

For channel hopping, the channel on which a timeslot operates is determined by the timeslot's ASN, as

$$Channel = List_c[(ASN + offset_{\text{channel}}) \ \% \ N_{List_c}] \tag{3.1}$$

where $List_c$ is a set of channels to be hopped over, $offset_{\text{channel}}$ is the channel offset, and N_{List_c} is the number of elements in $List_c$. By introducing ASN in channel determination, each timeslot with a fixed $offset_{\text{channel}}$ can exploit different frequencies per timeslot. The $offset_{\text{channel}}$ enables different channels to be used in the same timeslot.

Then for each timeslot, a TSCH *schedule* specifies (1) the activity (*i.e.,* whether to transmit, receive, or sleep), (2) the channel to be used for the corresponding activity, and (3) whether the slot is shared or dedicated. However, how to build and maintain the schedule is out of the scope of the IEEE 802.15.4-2015 standard, and is left as an open research problem. For this reason, a number of TSCH scheduling schemes have been proposed recently. We will describe the two representative state-of-the-arts below. A common characteristic of widely used TSCH scheduling mechanisms is their *simple* operation; each node *self-allocates* its timeslot without any additional control packet exchange. This is for robust and energy-efficient operation on time-varying routing topology in wireless environments.

3.2.2 6TiSCH and its Minimal Configuration

In 2013, IETF Working Group 6TiSCH [76] was established for the purpose of designing IPv6 support on top of TSCH. 6TiSCH defines a TSCH *minimal configuration* [75], which is a simple fixed scheduling scheme designed to enable basic and necessary functions for TSCH network. It simply consists of a single shared timeslot per slotframe to run IPv6 traffic on top of low-power TSCH networks with basic interoperability. This timeslot is used for both transmission and reception of *all nodes* in a TSCH network.

3.2.3 Orchestra

Orchestra [12] provides autonomous TSCH scheduling together with the RPL routing layer[1]. For the construction of TSCH and RPL network, Orchestra employs two types

[1]RPL is the standard IPv6 routing protocol for LLNs. The detailed description and related work for RPL are in [10, 83], which is out of the scope of this paper.

of slotframes. The first is the *EB (Enhanced Beacon) slotframe* which has two active timeslots in each node, one dedicated for EB transmission and the other for EB reception from the time source. Reliable EB communication is possible since a channel offset is dedicated for this slotframe and a node's reception (Rx) slot is synchronized with the transmission (Tx) slot of its TSCH time source. The second is the *RPL shared slotframe* for RPL control packets (DIO, DAO, and DIS), which has another dedicated channel offset. This slotframe has one active slot, which is used for both Tx and Rx of all nodes' RPL control packets.

In addition, Orchestra proposes two approaches for unicast data communication slotframe, sender-based or receiver-based, where either of them can be selected. Thus, a total of three slotframes are employed in each Orchestra implementation. A different channel offset from EB and RPL shared slotframes is used for the unicast slotframe. In a sender/receiver-based schedule, a node self-allocates a single Tx/Rx slot per slotframe based on its MAC address, respectively. The time offset is computed as,

$$\mathit{offset}_{\text{time}} = h(MAC) \ \% \ S_{\text{sf}} \tag{3.2}$$

where h is a hash function shared in the network, MAC is the hardware address of the node, and S_{sf} is the size of the unicast slotframe. As all nodes use the same hash function, a neighbor's schedule can be computed directly based on the neighbor's MAC address, without any exchange of additional control packets. In conjunction with the standard RPL network layer, Orchestra updates schedules autonomously as network topology changes.

Fig. 3.2 depicts an example of receiver-based scheduling in Orchestra. *R* denotes a timeslot allocated for unicast packet reception. In this example, each node computes $\mathit{offset}_{\text{time}}$ of *R* using its ID as output of $h(MAC)$ where S_{sf} is 5. For example, node 1 or 7 has the $\mathit{offset}_{\text{time}}$ of 1 or 2, respectively. When any node has a packet to transmit towards node 1, it transmits on the first timeslot within a slotframe. In receiver-based scheduling, while a node's Rx slot is single and fixed within a slotframe, its Tx slots can be multiple; each Tx slot corresponds to Rx slot of each neighbor node. On the

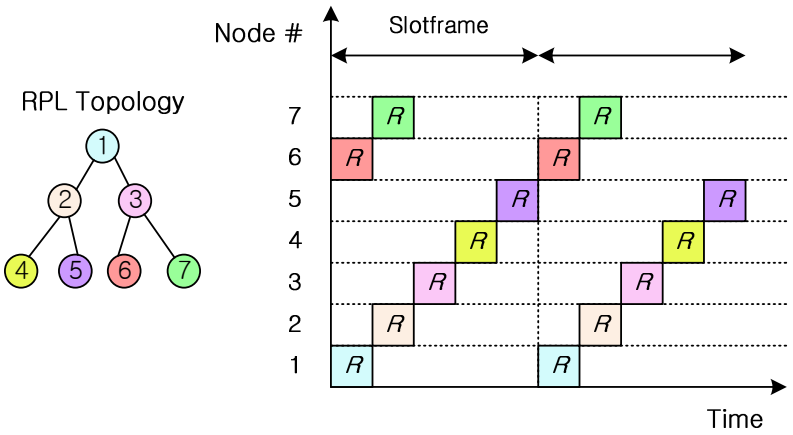


Figure 3.2: An example of TSCH scheduling in receiver-based Orchestra.

other hand, in sender-based scheduling, a node has a fixed Tx slot and multiple Rx slots.

3.3 Preliminary and Motivation

While the contributions of the state-of-the-art techniques are substantial in enabling TSCH to operate on embedded devices in real wireless environments, they have *one possible drawback*: static scheduling with globally identical slotframe size, which is pre-defined at compile time. Nodes in a network usually neither transmit nor receive the same amount of traffic. Depending on routing topology and traffic generation pattern, each node observes a different volume of traffic. Consequently, a uniform and constant schedule may bring about three kinds of undesired situations: (1) A node responsible for forwarding packets more often than its Tx or Rx timeslots suffers from severe packet losses. (2) A node who experiences little traffic wastes energy due to idle listening in timeslots allocated unnecessarily. (3) When routing topology or traffic pattern changes, there is no mechanism to adjust its slotframe size according to network dynamics.

To confirm this hypothesis and motivate our *TESLA*, we present a preliminary study on the performance of three representative state-of-the-arts: 6TiSCH minimal

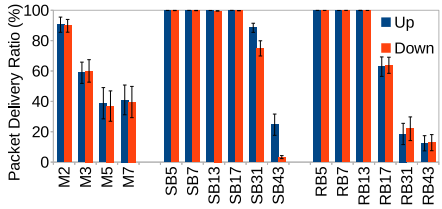
configuration [75], sender-based and receiver-based Orchestra [12], implemented on ContikiOS [84].

3.3.1 Methodology

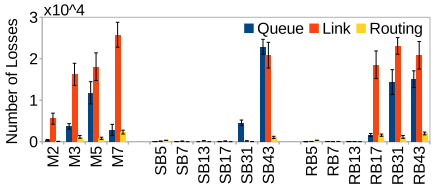
We evaluate the three schemes on FIT/IoT-LAB testbed [85] with 110 *M3* nodes having bidirectional traffic. The root node generates a downward packet every 0.5 second while altering destinations in a round-robin fashion. Each of 109 non-root nodes generates an upward packet with the period of 54.5 (=0.5$\times$109) seconds to equal the bidirectional traffic load. Detailed explanations of the experimental settings will be provided in Section 3.5.1.

For Orchestra, to focus on the impact of unicast slotframe size, we first optimize the size of RPL shared slotframe on this testbed. Small RPL shared slotframe size allows successful and stable RPL network formation at the cost of high energy consumption. On the other hand, large RPL shared slotframe size is unable to accommodate RPL control messages during network bootstrap and when preferred-parent changes occur, resulting in excessive collisions. In the worst case, this causes a TSCH node to fail to exchange packets with its time source (*i.e.,* RPL preferred parent in Orchestra) before a certain keep-alive timeout, and eventually lose time-synchronization.

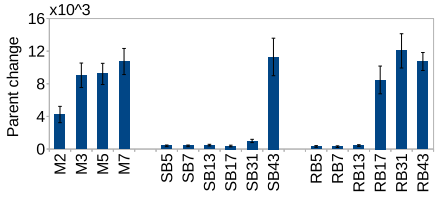
Interestingly, we found that the optimal size is different in two types of Orchestra because they deliver DAOs in different ways when a node changes its preferred parent. In receiver-based Orchestra, the node is able to self-calculate the new parent's Rx slot based on the parent's ID, and send a DAO to the parent. In sender-based Orchestra, however, if the child node sends a DAO to the new parent through the child's Tx slot, the DAO is likely to be lost. This is because the parent is yet unaware of the new child, thus not listening to the new child's Tx slot. To this end, the child node utilizes the RPL shared slotframe for DAO delivery until its new parent knows its Tx slot schedule. Consequently, sender-based Orchestra requires more resources for the RPL shared slotframe than receiver-based Orchestra. After a series of experiments on this



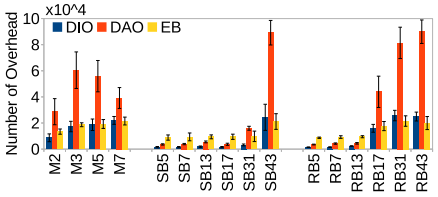
(a) End-to-end packet delivery ratio (PDR)



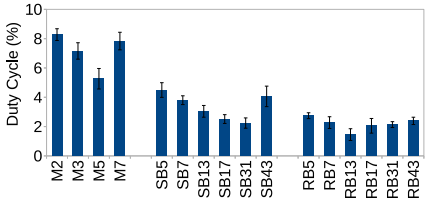
(b) Number of packet losses during 1-hour experiment



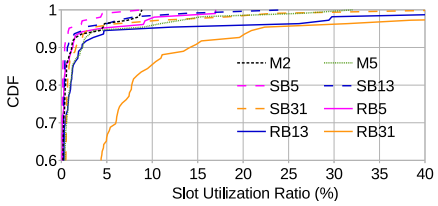
(c) Number of parent changes during 1-hour experiment



(d) Number of control packets during 1-hour experiment



(e) Radio duty-cycle (on time)



(f) CDF of slot utilization ratio (SUR)

Figure 3.3: Various performance metrics in Orchestra and 6TiSCH minimal configuration with different slotframe sizes.

testbed (figures are omitted for brevity), we set the sizes of RPL shared slotframes for receiver-based and sender-based Orchestra to 23 and 11, respectively.

3.3.2 Experimental Results

Fig. 3.3 summarizes our results where *M*, *SB*, and *RB* denote the 6TiSCH minimal configuration, sender-based Orchestra, and receiver-based Orchestra, respectively. The number shown after each label indicates the (unicast) slotframe size. Fig. 3.3(a) plots end-to-end packet delivery ratio (PDR) for both upward and downward traffic. The

minimal configuration never achieves perfect PDR even with the shortest slotframe (*i.e.,* 2) since every slot is shared by all nodes in the entire network resulting in frequent collisions. Its performance becomes even worse as the slotframe size increases. On the other hand, Orchestra provides significantly better PDR by dispersing active slots in time using a hash function in Eq. (3.2). SB and RB achieve PDR of more than 99% when they employ slotframe size less than 17 and 13, respectively. As the slotframe size increases, however, Orchestra also suffers from lack of communication opportunities.

To analyze the causes of PDR degradation more closely, Fig. 3.3(b) plots the number of three types of packet losses: *queue loss*, *link loss*, and *routing loss*. Fig. 3.3(b) shows that most of the packet losses are due to queue overflow and link failure, and we observed that most of these losses occur at a few bottleneck nodes due to the load imbalance problem in RPL [80]. For example, when RB employed a slotframe size of 31, 85% of lost packets disappeared at just two bottleneck nodes.

However, Fig. 3.3(b) also shows that detailed loss patterns at these bottleneck nodes are different depending on TSCH scheduling. Note that each node in RB has one Rx slot and multiple Tx slots within a unicast slotframe while each node in SB has one Tx slot and multiple Rx slots. This means that RB and SB provide fewer Rx and Tx opportunities, respectively. Accordingly in RB, neighbors of a bottleneck node contend for a single Rx slot of the bottleneck, which first leads to many link losses and then queue losses when the contention becomes more severe (due to redundant CSMA backoff). On the other hand, SB mainly suffers from queue losses due to lack of Tx opportunities. As an exception, SB43 also experiences significant link losses, but most of these losses (*i.e.,* 98.9%) occur in not the unicast slotframe but the RPL shared slotframe due to a large number of RPL control packets attempting to fix unstable routing topology. The minimal configuration shows numerous link losses since all nodes contend in one same slot to send packets regardless of receiver identity.

Figures 3.3(c) and 3.3(d) plot network stability and control overhead in terms of

the number of parent changes and the numbers of DIOs, DAOs, and EBs, respectively. As PDR is degraded in all the scheduling schemes, the RPL-TSCH network becomes unstable and generates more control packets. Despite its effort, however, their performance is not restored since the problem is attributed to how TSCH slots are scheduled. Figures 3.3(b) and 3.3(c) show that network stability is closely correlated to link loss. When packets are lost at links due to collision, RPL misunderstands it as bad link quality and triggers meaningless parent changes [78].

Fig. 3.3(e) represents average radio duty-cycle of each scheme. As the slotframe size increases, duty-cycle typically decreases due to low resource allocation. However, when the slotframe size becomes too long, duty-cycle rises again due to more Tx/Rx overhead coming from low PDR. The minimal configuration provides the lowest PDR among the three schemes, resulting in the highest energy consumption. SB consumes more energy than RB due to the two reasons. Given that, within a slotframe, RB allocates one Rx slot but SB allocates Rx slots as many as the number of RPL neighbors, *i.e.,* the preferred parent and children, SB uses more energy for listening. In addition, SB employs a smaller size of RPL shared slotframe than RB, as discussed in Section 3.3.1, consuming more energy.

Next, we define the slot utilization ratio (SUR) as the ratio of Rx slots used for successful packet reception over total Rx slots, and plot its CDF among nodes in Fig. 3.3(f). A higher SUR indicates more efficient use of resources and less redundant energy consumption. In the cases where PDR is nearly perfect, such as SB5, SB13, RB5, and RB13, they utilize slots very inefficiently. For example, more than 80% of nodes experience $<$1% SUR. This is because, compared to the given slotframe size, only a few bottleneck nodes receive a reasonable amount of traffic but most nodes experience too sparse traffic. If a larger slotframe is used as in RB31, SUR becomes better but PDR becomes miserable ($\sim$20%) as shown in Fig. 3.3(a). The minimal configuration cases exhibit low PDR with low SUR, an undesirable performance characteristic.

3.3.3 Summary

Overall, experimental results strongly support our hypothesis: under static globally-uniform scheduling methods, while bottleneck nodes suffer from packet losses due to insufficient opportunities for Tx/Rx, most of other nodes waste energy due to over-allocated timeslots. The conventional techniques will suffer even more when each node generates data with a different rate and/or a node changes its traffic pattern at run time. For example, in a smart building application, a temperature or humidity sensor generates light periodic traffic but an anemometer generates heavy continuous traffic [81]. A node's application traffic can change at run time due to emergency detection [5, 86], device control [87, 88], and firmware update. Even with a fixed application traffic pattern, "network" traffic can still vary at run time according to a reporting strategy, e.g., sending each data immediately or aggregating data for a while and sending as a batch [82]. This motivates *TESLA*, a technique for dynamic and local adjustment of slotframe size according to traffic load.

3.4 TESLA Design

In this section, we present our *TESLA* design. *TESLA* operates in conjunction with RPL and receiver-based Orchestra (*i.e.,* node ID-based Rx slot allocation). Each *TESLA* node monitors its incoming traffic load without any additional control overhead. Based on the traffic load information, each node periodically adapts its Rx slot schedules. Specifically, when a node detects overwhelming packets coming through its current Rx slots, it reduces its slotframe size to alleviate contention between neighboring nodes for reliable packet delivery. For energy efficiency, on the other hand, when a node notices many idle Rx slots, it increases its slotframe size in order to save energy by avoiding idle listening. In addition, *TESLA* attempts to allocate different channel offsets to nodes, if possible, leading to network capacity increase.

There is a price to pay for this dynamic slot scheduling. As each node's Rx sched-

ule varies, its change should be timely propagated to the RPL neighbors (*i.e.,* preferred parent and one-hop children) for their Tx schedules. This local exchange of Rx schedules slightly increases control overhead. Nevertheless, our intuition is that the gain from slotframe adjustment is more than enough to compensate the modest increase in control overhead.

3.4.1 Slotframe Structure

In *TESLA*, each node has four types of slotframes:

- **EB slotframe** is for TSCH enhanced beacons (EBs) with a constant periodicity and a dedicated channel offset.
- **RPL shared slotframe** is for RPL control packets, also with a constant periodicity and a dedicated channel offset.
- **Rx slotframe (RSF)** is for unicast reception with an elastic periodicity.
- **Tx slotframe (TSF)** is for unicast transmission, per neighbor, with an elastic periodicity.

The first two slotframes are for control messages similar to Orchestra [12]. In addition, each *TESLA* node maintains a single slotframe only for unicast packet reception, called *Rx slotframe (RSF)*. There is one Rx slot in each RSF. In contrast to Orchestra, *TESLA enables each node to adjust its own RSF size dynamically according to incoming traffic load*. Fig. 3.4 shows an example of *TESLA* scheduling for the same routing topology as Fig. 3.2. When a large amount of traffic converges to node 1, it reduces the RSF size for more Rx opportunities. On the contrary, if node 6 receives few packets, it enlarges its RSF size to reduce energy consumption. In this way, each node may end up using a different RSF size.

In addition, a node maintains a *Tx slotframe (TSF)* for each of its routing neighbors (*i.e.,* preferred parent and one-hop children). Each TSF has one Tx slot, which

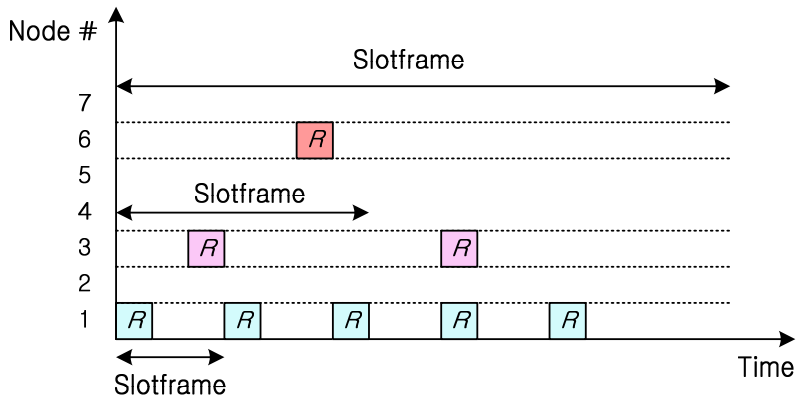


Figure 3.4: An example of *TESLA* scheduling.

matches the Rx slot in the RSF of the corresponding routing neighbor. A node can have a different TSF size for each neighbor node since each neighbor adjusts its RSF size independently. Overall, each *TESLA* node has an EB slotframe, a RPL shared slotframe, an RSF, and multiple independent TSFs as many as the number of its routing neighbors.

Inheriting Eq. (3.2) from Orchestra, the time offset for a *TESLA* node's Rx slot is computed as

$$\mathit{offset}_{\text{time}}(t) = h_t(MAC) \ \% \ S_{\text{rsf}}(t). \tag{3.3}$$

Differently from Orchestra, the RSF size (S_{rsf}) changes over time (t) and so does $\mathit{offset}_{\text{time}}$. Note that a node should know not only a neighbor's ID but also its *up-to-date* RSF size to calculate the neighbor's Rx schedule and maintain a correct TSF for the neighbor.

3.4.2 Rx Slotframe Size Adaptation

This section presents how a *TESLA* node monitors its incoming traffic load and self-adapts its RSF size accordingly.

Traffic Load Reporting

TESLA lets each node (*i*) inform each of its one-hop routing neighbors (*A*) of traffic load from node *i* to node *A*, namely $L(i, A)$. Specifically, when node *i* sends a unicast packet to the one-hop neighbor *A*, it piggybacks the traffic load information $L(i, A)$ in the packet by using Information Element (IE) field in the IEEE 802.15.4 frame; the traffic reporting process happens locally and requires no additional control overhead.

Given that the current TSCH scheduling techniques suffer both link loss and queue loss as discussed in Section 3.3, node i calculates the traffic load $L(i, A)$ by adding two elements, as

$$L(i, A) = M(i, A) + Q(i, A). \tag{3.4}$$

Specifically, assuming $t_{\text{update}}(A)$ as the time elapsed from the last RSF size update of node A, $M(i, A)$ indicates the number of node *i*'s Tx attempts towards node A during $t_{\text{update}}(A)$. Node *i* initializes $M(i, A)$ to 0 upon detecting neighbor node A changing its RSF size, and increases $M(i, A)$ in every MAC layer transmission destined for node A regardless of whether it is acknowledged or not. On the other hand, $Q(i, A)$ is simply the number of currently queued packets for node A waiting to be transmitted, which signifies the current congestion level experienced by node i towards node A.

Two Traffic Load Metrics

Based on the traffic load information reported from all routing neighbors, each node (*A*) calculates two complementary metrics for its periodic RSF adaptation (every T_{adapt}): (1) normalized total incoming traffic load, and (2) contention level.

To this end, we define $L_{\text{last}}(i, A)$ as the $L(i, A)$ at the last RSF adaptation of node A (*i.e.,* before T_{adapt}). $L(i, A)$ has been accumulated since node *A*'s last RSF size change (*i.e.,* during last $t_{\text{update}}(A)$, longer than or equal to T_{adapt} because the adaptation procedure may not always change the RSF size). Thus, the traffic load from node i to

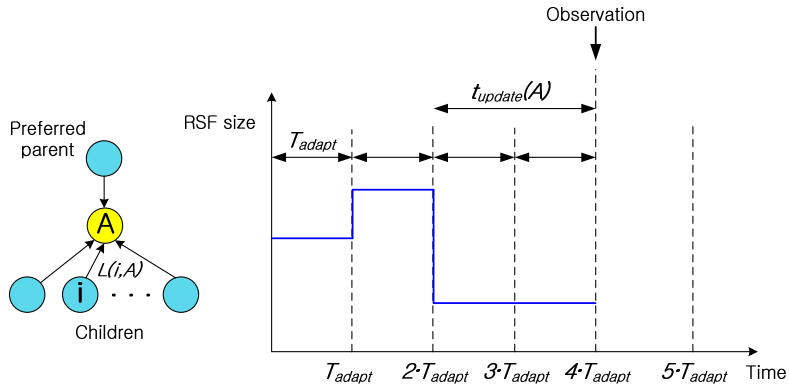


Figure 3.5: Topology of node A and its RPL routing neighbors, and an example of RSF size adaptation.

node A during recent T_{adapt}, namely $L_{\Delta}(i, A)$, is

$$L_{\Delta}(i, A) = L(i, A) - L_{\text{last}}(i, A). \tag{3.5}$$

Defining $\mathbb{N}(A)$ as the routing neighbor set of node A and W as the number of node A's Rx slots in last T_{adapt}, the normalized total incoming traffic load at node A, $L_{\Delta,\text{n}}(A)$, is computed as

$$L_{\Delta,\text{n}}(A) = \sum_{i \in \mathbb{N}(A)} \frac{L_{\Delta}(i, A)}{W}. \tag{3.6}$$

Finally, node A uses the metric $L_{\Delta,\text{n}}(A)$ for its RSF size adaptation.

Fig. 3.5 exemplifies RSF size adaptation, where node A executes RSF adaptation at time $4 \cdot T_{\text{adapt}}$ to decide the RSF size for the next period $[4 \cdot T_{\text{adapt}}, 5 \cdot T_{\text{adapt}}]$. Note that $2 \cdot T_{\text{adapt}}$ is when node A's most recent RSF size change happened. Then, W is the number of Rx slots in $[3 \cdot T_{\text{adapt}}, 4 \cdot T_{\text{adapt}}]$, $L(i, A)$ is the traffic load from node i during $t_{\text{update}}(A)$ (*i.e.,* $[2 \cdot T_{\text{adapt}}, 4 \cdot T_{\text{adapt}}]$), $L_{\text{last}}(i, A)$ is the traffic load in $[2 \cdot T_{\text{adapt}}, 3 \cdot T_{\text{adapt}}]$, and $L_{\Delta}(i, A)$ indicates the traffic load during recent T_{adapt}, *i.e.,* $[3 \cdot T_{\text{adapt}}, 4 \cdot T_{\text{adapt}}]$.

Next, node A estimates the contention level on its Rx slots. Specifically, node A interprets $L_{\Delta}(i, A)$ as the number of its Rx slots required to receive node i's traffic for

last T_{adapt}. Given that node A has W Rx slots for last T_{adapt}, the probability of node i to access an Rx slot of node A is $\frac{L_{\Delta}(i,A)}{W}$. Node A estimates packet reception ratio (PRR) from node i without any collision with the other routing neighbors, as

$$PRR_c(i, A) = \prod_{k(\neq i) \in \mathbb{N}(A)} (1 - \frac{L_{\Delta}(k, A)}{W}). \tag{3.7}$$

Then, node A uses $PRR_{c,\text{min}}(A)$, the minimum of $PRR_c(i, A)$ among all i in $\mathbb{N}(A)$ (*i.e.,* $PRR_c(i, A)$ for the worst-case node), for its RSF adaption, as the indicator of its contention level.

Note that the two metrics, $L_{\Delta,\text{n}}(A)$ and $PRR_{c,\text{min}}(A)$, are mutually complementary. For example, when traffic is heavy but comes from the only one neighbor node, $PRR_{c,\text{min}}(A)$ is always good (*i.e.,* 1) since there is no contention but node A may not receive all traffic successfully due to lack of Rx slots. In this case, $L_{\Delta,\text{n}}(A)$ helps node A to detect the problem. On the other hand, when traffic comes equally from many neighbor nodes, node A may lose many packets due to collision even though $L_{\Delta,\text{n}}(A)$ is relatively low. In this case, $PRR_{c,\text{min}}(A)$ helps to detect the problem. Overall, by combining the two metrics, each node detects not only the total incoming traffic load but also how it is distributed to the routing neighbors.

Prime Numbers for Rx Slotframe Size

When a *TESLA* node adapts its RSF size, it selects one from *prime numbers* excluding the pre-installed sizes for EB and RPL shared slotframes. There are two reasons for using prime numbers.

First, according to Eq. (3.1), Rx slots in consecutive RSFs (e.g., with $ASN = k$ and $ASN = k + S_{\text{rsf}}$) can use different channels when the RSF size (S_{rsf}) is a prime number, which increases frequency diversity. As an example, in Fig. 3.1, there are four available channels (e.g., IEEE 802.15.4 channels 15, 20, 25, and 26) and the slotframe size is 3. Based on Eq. (3.1), the timeslot (A->B) with channel offset 0 will select channel numbers 15, 26, 25, and 20 when ASNs are 0, 3, 6, and 9, respectively. Next,

as explained in Section 3.4.1, a *TESLA* network has many slotframes with different sizes: an EB slotframe, an RPL shared slotframe, and many RSFs (and corresponding TSFs). If all slotframes have lengths of different prime numbers, they are mutually prime, ensuring that the active slots overlap each other rarely and evenly. It prevents unintended synchronization effect. To this end, each node has an ordered list of prime numbers, $\mathbb{P}$, where the elements are in ascending order from 2, 3, 5, and so on.

Traffic-Aware Rx Slotframe Size Adaptation

Based on the aforementioned design, each node executes the following adaptation procedures every T_{adapt}:

- Attempt to decrease RSF size, if required (Algorithm 1).
- Otherwise, attempt to increase the size (Algorithm 2).

In Algorithm 1, node *A* first initializes $index$ as that of the element in the prime number list $\mathbb{P}$, which is equal to the current RSF size (S_{rsf}). For example, $\textit{index} = 1$ when $S_{\text{rsf}} = 2$ and $\textit{index} = 3$ when $S_{\text{rsf}} = 5$. Next, W_{new} is defined as the expected number of Rx slots during next T_{adapt} with a new RSF size, and initialized to W, the number of Rx slots with the current RSF size in last T_{adapt}. Then in the loop on line 3, the two traffic load metrics, $PRR_{c,\text{min}}(A)$ and $L_{\Delta,\text{n}}(A)$, are calculated using W_{new} instead of W.

At the beginning of the loop, node *A* checks if it is suffering high incoming traffic by using the two traffic load metrics as follows:

1. $PRR_{c,\text{min}}(A)$ is worse than a lower bound ($PRR_{\text{th,low}}$).
2. $L_{\Delta,\text{n}}(A)$ exceeds a threshold (L_{th}).

Condition (1) is satisfied if any neighbor in $\mathbb{N}(A)$ is expected to suffer from channel contention, and condition (2) is used as a precaution for sudden traffic increment due to change of network topology or traffic generation pattern. If at least one of these two

conditions is satisfied, the RSF size needs to be reduced to give more transmission opportunities for the neighbors. Therefore, a new RSF size $S_{\text{rsf,new}}$ becomes a one-step smaller prime number than S_{rsf} (line 4). In the following T_{adapt} with the new RSF size, node A is expected to wake up $\frac{S_{\text{rsf}}}{S_{\text{rsf,new}}}$ times more. Thus, W_{new} goes up accordingly (line 5), which increases $PRR_{c,\text{min}}(A)$ and decreases $L_{\Delta,\text{n}}(A)$. It iterates until neither of the two conditions is satisfied, which means none of $\mathbb{N}(A)$ is expected to suffer from low PRR due to contention, and the number of Rx slots is enough to cope with traffic variation.

Algorithm 1: How to decrease Rx slotframe (RSF) size

1 $index \leftarrow$ FindIndex(S_{rsf}, $\mathbb{P}$);
2 $W_{\text{new}} \leftarrow W$;
3 **while** $(PRR_{c,\text{min}}(A) < PRR_{\text{th,low}})$ —— $(L_{\Delta,\text{n}}(A) > L_{\text{th}})$ **do**
4 $S_{\text{rsf,new}} \leftarrow \mathbb{P}[--index]$;
5 $W_{\text{new}} \leftarrow \frac{W \cdot S_{\text{rsf}}}{S_{\text{rsf,new}}}$;
6 $S_{\text{rsf}} \leftarrow S_{\text{rsf,new}}$;

Algorithm 2: How to increase Rx slotframe (RSF) size

1 $index \leftarrow$ FindIndex(S_{rsf}, $\mathbb{P}$);
2 $W_{\text{new}} \leftarrow W$;
3 **while** $(PRR_{c,\text{min}}(A) > PRR_{\text{th,up}})$ && $(L_{\Delta,\text{n}}(A) < L_{\text{th}})$ **do**
4 $S_{\text{rsf,new}} \leftarrow \mathbb{P}[++index]$;
5 $W_{\text{new}} \leftarrow \frac{W \cdot S_{\text{rsf}}}{S_{\text{rsf,new}}}$;
6 **if** $\frac{S_{\text{rsf,new}}}{S_{\text{rsf}}} > \varepsilon$ **then**
7 **break**;
8 $S_{\text{rsf}} \leftarrow S_{\text{rsf,new}}$;

Algorithm 2 is executed when the RSF size is not decreased by Algorithm 1. The two algorithms are similar in structure, but their conditions for the RSF size update are different. In Algorithm 2, an RSF size increases if both of the following conditions are

satisfied.

1. $PRR_{c,\min}(A)$ is better than an upper bound ($PRR_{\text{th,up}}$).

2. $L_{\Delta,\text{n}}(A)$ is less than the threshold (L_{th}).

In other words, if $PRR_c(i, A)$ for all *i* in $\mathbb{N}(A)$ are high enough and the number of Rx slots is sufficient to accommodate all traffic from the neighbors, node *A* increases the RSF size to reduce idle Rx slots. Note that Algorithm 2 uses another threshold $PRR_{\text{th,up}}$, higher than $PRR_{\text{th,low}}$ used in Algorithm 1. Having double thresholds improves stability by preventing the ping-pong effect (repetition of increasing/decreasing the RSF size too frequently).

To prioritize high reliability over energy saving, we design *TESLA* to increase RSF size conservatively by introducing a bounding factor ε. Specifically, if the ratio of $S_{\text{rsf,new}}$ to S_{rsf} exceeds ε, it breaks the loop and stops increasing the RSF size (lines 6 and 7).

3.4.3 Tx Slotframe Size Adaptation

If a node ends up changing its RSF size through the periodic RSF adaptation, it announces the new RSF size to its routing neighbors, then each of which modifies its TSF size for the node.

Local Update of the Rx Slotframe Size

Reliable delivery of an updated RSF size to routing neighbors is critical to *TESLA*'s robust operation; if a neighbor is unaware of the RSF size change, it may continuously fail to deliver packets to the node due to schedule mismatch. To this end, node A delivers the new RSF size and its version number[2] through DAO, DIO, EB, and Enhanced ACK (EACK)[3] packets using reserved fields of DAO and DIO, and IEEE 802.15.4 header IE for EB and EACK. Neighbor nodes are informed of node A's up-to-date

[2]The version number increases by one whenever the RSF size changes.

[3]In TSCH, IEEE 802.15.4 EACK is used normally with timing information embedded.

RSF size whenever receiving these packets. DAO updates the preferred parent, DIO and EB update all neighbors of node A simultaneously, and EACK updates any node which transmits a unicast packet to node A.

Announcing the RSF size relying solely on existing traffic incurs no additional control overhead but may not provide timely update. For immediate RSF size update, node A transmits an additional DAO (for the preferred parent) and an EB (for the one-hop children) right after changing its RSF size, if they are not already scheduled. This greatly improves robustness with slightly more control overhead.

However, both of these messages may be lost, especially the EB which is broadcasted without ARQ. To address this problem, *TESLA* has two backup mechanisms. (1) After node A changes its RSF size, it does not eliminate the previous RSF but temporarily maintains it together with the new RSF. When an outdated neighbor successfully sends a packet in node A's temporary RSF, it is updated by receiving an EACK from node A. (2) If a neighbor node fails to receive the new RSF size even until the temporary double RSF schedule ends, it will continuously fail to transmit unicast packets to node A. In this case, the neighbor suspects schedule mismatch, sends packets destined for node A through the RPL shared slotframe, and is updated by receiving an EACK from node A.

Tx Slotframe Update

When a node notices an update of a neighbor's RSF size by comparing the versions, it changes the periodicity and time offset of corresponding TSF. Sometimes, Tx slots of two TSFs, each of which is allocated for a different neighbor, may overlap unfortunately. In this case, *TESLA* compares the lengths of Tx queues for the two neighbors, and prioritizes the transmission for the neighbor with more queued packets.

3.4.4 Multi-channel Operation

Although *TESLA* is designed to avoid schedule overlap between neighbors by using prime numbers for RSF size, overlaps may occur inevitably, especially under extremely heavy traffic. This is because bottleneck nodes end up using very small RSF sizes, increasing probability of timeslot overlap. For example, assume that a bottleneck node (*B*) reduces its RSF size to the minimum prime number (*i.e.,* 2). Then, any transmission in the vicinity of node B, not destined to node B, can collide with a packet towards *B* with a probability of up to 50%.

To enlarge network capacity, *TESLA* utilizes multiple channels for unicast slotframes. Specifically, the channel offset of a node's RSF is computed as,

$$\mathit{offset}_{\text{channel}} = h_c(MAC) \ \% \ (N_{\mathit{List}_c} - 2). \tag{3.8}$$

Note that $\mathit{offset}_{\text{channel}}$ uses a modulo operator with $N_{\mathit{List}_c} - 2$, instead of N_{List_c}, since we dedicate two channel offsets for the EB and RPL shared slotframes not to hinder the basic TSCH and RPL operations. For example, when there are 16 channels available (max. number of channels in IEEE 802.15.4), *TESLA* can allocate 14 channels for unicast communications. Since each node is likely to use a different Rx channel, collision between packets for different receivers occurs rarely and temporarily in *TESLA*, only when traffic load is very high and both timeslot and channel offset schedules are overlapped.

3.4.5 Collaboration with RPL

A practical embedded network is typically designed as a vertical silo where multiple layers intimately collaborate [83, 12]. To this end, we discuss how *TESLA* jointly operates with RPL when routing topology changes. *TESLA* maintains the RSF size information of only current routing neighbors. When a node switches its preferred parent, it does not have the new parent's RSF information and neither does the parent. The new child and new parent should know each other's RSF size for unicast communication.

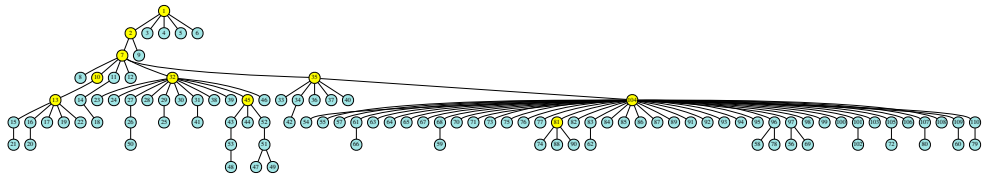


Figure 3.6: A snapshot of RPL topology for 110 nodes with -17 dBm of Tx power on FIT/IoT-LAB testbed in Lille.

In this case, the node's RPL layer schedules a DAO for the new parent, and its *TESLA* layer sends the DAO (including its RSF size) on the RPL shared slotframe and finds out the parent's RSF information by receiving an EACK. Meanwhile, the new parent detects the addition of a new child by receiving the DAO. Its RPL layer establishes a new downward route for the child and its *TESLA* layer installs a TSF for the child using the RSF information included in the DAO. On the other hand, the old parent removes both the route and TSF for the previous child after receiving a No-path DAO from the child (scheduled by RPL) or the expiration of the route.

When RPL's routing topology is unstable or being repaired, e.g., when the network bootstraps or wireless environments change significantly, many nodes change their parents simultaneously and it is difficult to exchange RSF sizes through DAO and EACK. In this case, however, since each node's RPL layer generates many DIO packets due to Trickle re-initialization [89], most nodes are able to know routing neighbors' RSF sizes quickly by receiving their DIOs. When a node receives a DIO from its new parent quickly, it sends a DAO on the parent's RSF instead of the RPL shared slotframe. This synergistic joint operation enables *TESLA* to maintain modest contention on the RPL shared slotframe.

3.5 Performance Evaluation

In this section, we evaluate *TESLA* on real world testbeds with various topologies. We compare *TESLA* against state-of-the-art TSCH schedules. We also examine the adaptability of *TESLA* to dynamics of network traffic. Lastly, the impact of parameter

setting is discussed.

3.5.1 Methodology and Experiment Setup

We implement *TESLA* on ContikiOS and compare it against 6TiSCH minimal scheduling (*M*), sender-based Orchestra (*SB*), and receiver-based Orchestra (*RB*), using 110 and 79 nodes on the FIT/IoT-LAB testbeds [85] in Lille and Grenoble, France, respectively. Each node features a 32-bit ARM Cortex-M3 microcontroller (STM32F103REY) and an AT86RF231 IEEE 802.15.4 radio chip. This node is representative of today's state-of-the art IoT devices [85]. We use Contiki-RPL implementation on top of the TSCH scheduling schemes. For Orchestra and *TESLA*, the length of EB slotframe is 397. As discussed in Section 3.3, the size of RPL shared slotframe for sender-based Orchestra is 11, and that for receiver-based Orchestra is 23. The size of RPL shared slotframe for *TESLA* is also 23, since *TESLA* exchanges most of DAOs in unicast slotframe as explained in Section 3.4.5.

All protocols use a maximum of 8 retransmissions per hop, and queue size of 16 packets. TSCH hops over four best channels: 15, 20, 25, and 26. Each instance of an experiment lasts for 1 hour, and results are averaged over 3$\sim$5 runs of experiments for each case. An error bar represents 95% confidence interval. In all experiments, the application payload is 59 bytes carried in UDP/IPv6 datagrams over 6LoWPAN, reaching 109 bytes of the data frame size.

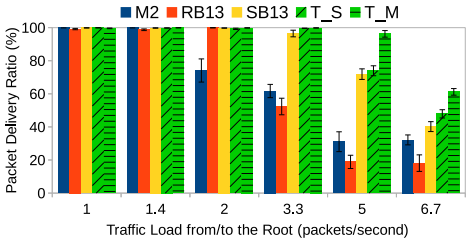
For *TESLA*, we use RSF adaptation period T_{adapt} of 15 seconds, and limitation factor ε is set to 1.5 to increase RSF size conservatively. In other words, RSF size cannot increase by more than 50% every T_{adapt}. The load threshold L_{th} is 50%, and $PRR_{\text{th,low}}$ and $PRR_{\text{th,up}}$ are 80% and 90%, respectively. The prime numbers for RSF size adaptation range from 2 to 97, allowing a node to wake up at least once in 1 second. To distinguish the effect of multi-channel operation in *TESLA*, we create two versions of *TESLA*: T_S uses a single channel offset for RSF/TSF, and T_M uses two channel offsets (excluding two offsets dedicated for EB and RPL shared slotframes).

3.5.2 Impact of Traffic Load

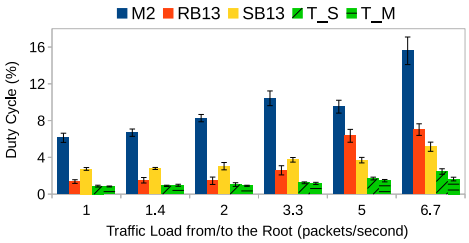
We first investigate the performance of the state-of-the-art TSCH schedules and *TESLA* with various traffic intensities on Lille testbed consisting of 110 nodes. We use slotframe size of 2 for the minimal configuration which had the highest PDR in Section 3.3. Both receiver/sender-based Orchestra use a unicast slotframe size of 13. Each node uses transmission power of -17 dBm. Fig. 3.6 shows a snapshot of the RPL routing topology during experiments where average depth of the network is 4.7 hops, and its maximum reaches 7 hops. In each experiment, the root node (*i.e.,* node 1) generates downward packets with a fixed rate while altering destinations. For upward packets, an equal aggregate rate is used for 109 sensor nodes. For instance, when the root generates 2 downward packets per second, each of 109 non-root nodes generates upward packets with 0.018 (=2/109) packets per second.

Reliability. Fig. 3.7(a) shows the average bidirectional end-to-end PDR under different traffic load. Under light traffic, all the protocols perform well with PDR over 99%. However, when the traffic load from/to the root is 2 or 3.3 packets/second, M, RB, and SB start to show performance degradation. On the other hand, *TESLA* is still capable of accommodating traffic by reducing slotframe sizes adaptively. For example, Fig. 3.9(b) presents a snapshot of RSF size of each node when traffic rate is 2 packets/second. It shows that bottleneck nodes marked with yellow color in Fig. 3.6 use much smaller RSF sizes than the other nodes to resolve contention. Fig. 3.7(a) also shows that T_M improves PDR of T_S by up to 30.1%

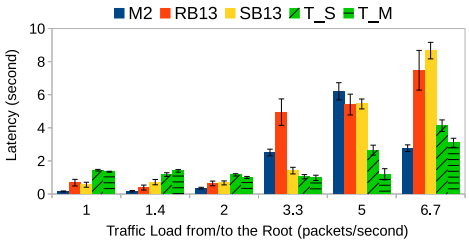
As presented in Fig. 3.8(a), M and RB experience considerable link losses since multiple nodes contend for an active slot, which is aggravated more in the vicinity of bottlenecks. Meanwhile, bottlenecks in SB suffer from frequent queue overflows due to its fewer Tx slots than Rx slots. In *TESLA*, however, bottlenecks can reduce its Rx slotframe size to the minimum (*i.e.,* 2), resulting in much less packet losses. When the traffic increases more, *TESLA* also encounters channel contention and packet losses, which is still less than those in other schemes. The results confirm that *TESLA* suc-



(a) End-to-end packet delivery ratio



(b) Radio duty-cycle (on time)

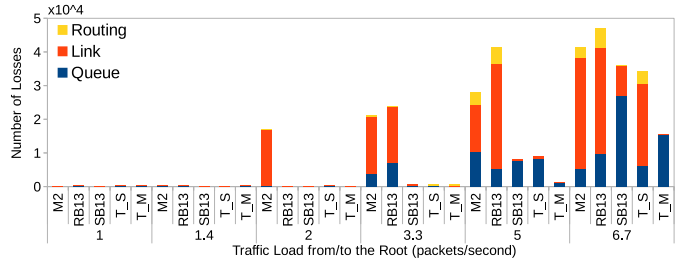


(c) End-to-end latency

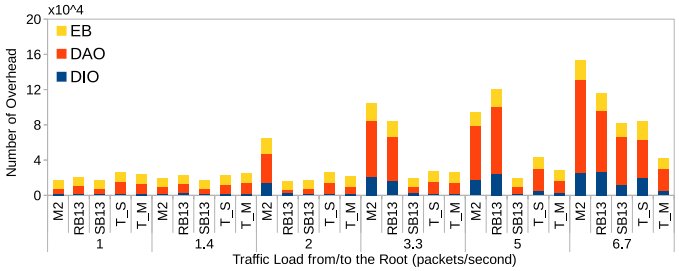
Figure 3.7: PDR, duty-cycle and latency according to different traffic load.

cessfully does its role at the link layer: rescuing bottleneck nodes from the contention hell, as much as possible.

Energy consumption. Fig. 3.7(b) shows the duty-cycle for each protocol. Obviously, when the traffic load increases, more energy is used to transmit and receive packets. M2 consumes the largest energy due to its short periodicity of slotframe and severe contention. Orchestra has duty-cycles from 1.3% to 7.0%. Under low traffic load, SB spends more energy than RB by allocating multiple Rx slots within a slotframe. However, RB with one Rx slot within a slotframe experiences more contention than SB,



(a) Number of packet losses during 1-hour experiment

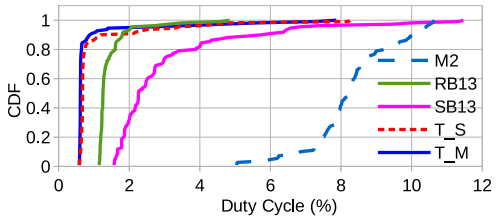


(b) Number of overhead during 1-hour experiment

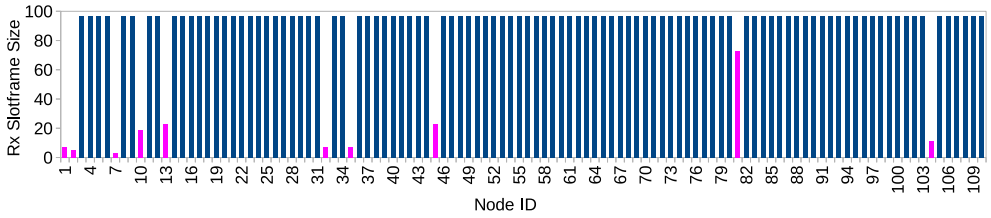
Figure 3.8: Number of Losses and overhead according to different traffic load.

bringing about higher duty-cycles than SB as traffic intensifies. *TESLA* significantly improves upon M, RB, and SB, maintaining duty-cycle from 0.8% to 1.6%. T_M has slightly lower duty-cycle than T_S since it reduces channel contention thanks through channel diversity. Compared to SB which showed the best PDR except *TESLA* in Fig. 3.7(a), T_M reduces duty-cycle by 67.1% on average.

Fig. 3.9(a) plots the distribution of duty-cycles among all 110 nodes for traffic load of 2 packets/second. At this traffic rate, RB13 outperforms M2 and SB13 as illustrated in Fig. 3.7(b), but *TESLA* performs even better, enabling more than 90% of nodes to save their energy by 50% compared to RB13. It is important to note that *TESLA* achieves energy saving without loss of reliability by increasing the RSF sizes only for the nodes with over-allocated slots. Fig. 3.9(b) confirms this: while most nodes utilize the maximum RSF size (*i.e.,* 97) for energy saving, a few bottleneck nodes use very small RSF sizes. In Fig. 3.9(a), *TESLA* does increase the duty-cycles of ¡5% bottleneck



(a) CDF of duty-cycle for traffic load of 2 packets/second



(b) Rx slotframe size of *TESLA* for traffic load of 2 packets/second

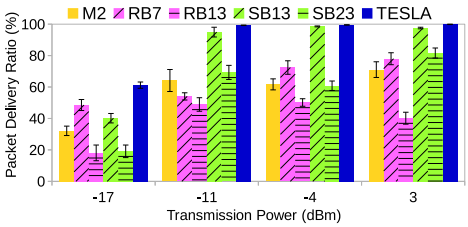
Figure 3.9: CDF of duty-cycle and Rx slotframe size of *TESLA* for traffic load of 2 packets/second

nodes compared to RB13, because they select RSF sizes shorter than 13 as shown in Fig. 3.9(b).

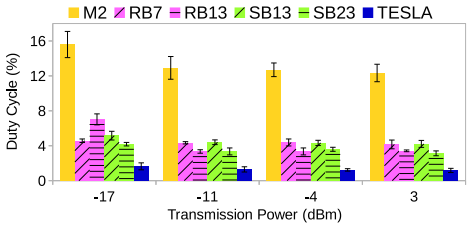
Interestingly, RB13 also achieved perfect reliability under the same traffic load, which means that RSF size of 13 is sufficient to handle the traffic even for bottlenecks. The reason why *TESLA* causes the bottleneck nodes to use RSF sizes less than 13 is its sensitive reaction to the contention: decreasing RSF size promptly when a burst of traffic is temporarily observed. Even if each node generates traffic with a fixed rate, the incoming traffic load of each node, especially bottleneck nodes, fluctuates due to randomness in the network such as channel quality and topology changes. As *TESLA* prioritizes reliability over energy efficiency, a few bottleneck nodes sacrifice their energy by using RSF sizes slightly shorter than actually needed, aiming for reliable packet delivery under network dynamics. We argue that this design choice is reasonable because sacrificing reliability can ruin the whole network.

End-to-end latency. Fig. 3.7(c) presents the average end-to-end latency for upward and downward traffic. When the traffic load is low, *TESLA* exhibits the longest delay, since most nodes use the maximum RSF size. Note that *TESLA* is designed for reliability and energy efficiency, rather than short latency. Interestingly, however, as the traffic load increases to 3.3 packets/second, the latency of *TESLA* decreases because RSF size is reduced throughout the network, while those of the other schemes increase due to channel contention. Eventually, beyond traffic load of 3.3 packets/second, *TESLA* provides the shortest delay with the best reliability and energy-efficiency. As an exceptional case, under the highest traffic load, it seems that the minimal configuration shows shorter delay than *TESLA*, but this is because only nodes with 1 or 2 hops away from the root can successfully deliver packets (*i.e.,* $\sim$30% PDR). Overall, although *TESLA* is not explicitly designed for latency improvement, its contention alleviation ends up with better latency.

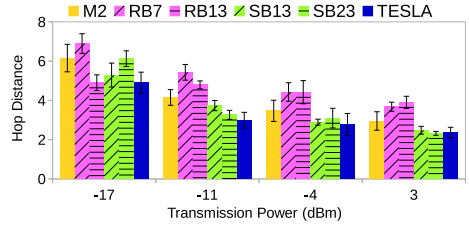
Network overhead. When a *TESLA* node changes its RSF size, it can generate additional DAO and EB packets in order to fast notify the new RSF size to the preferred parent and 1-hop children, as described in Section 3.4.3. Fig. 3.8(b) presents DIO, DAO, and EB overhead. In the lowest traffic load case, T_M makes 27.2% and 16.1% increments of DAO and EB packets, compared to RB13. Nevertheless, it does not impede *TESLA*'s reliability as shown in Fig. 3.7(a). In addition, this control overhead increase is more than compensated by *TESLA*'s substantial energy saving via reducing idle listening, which leads to significant duty-cycle improvement shown in Fig. 3.7(b). Fig. 3.8(b) also reveals that as the traffic increases, Orchestra and the minimal schedule incur more network overhead than *TESLA* to restore the network that has become unstable due to lack of reliable packet delivery. This also confirms why *TESLA*'s design choice, prioritizing reliability over energy efficiency, makes sense.



(a) End-to-end packet delivery ratio



(b) Radio duty-cycle (on time)



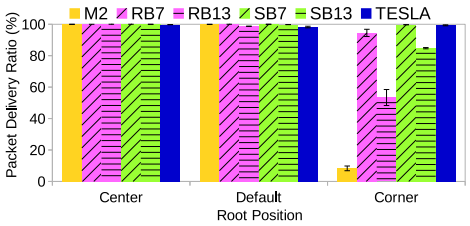
(c) Hop distance

Figure 3.10: Impact of transmission power.

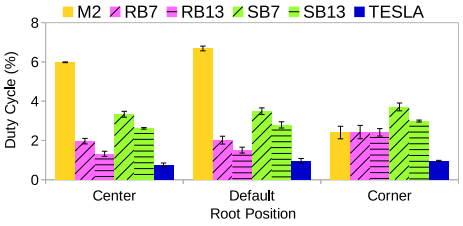
3.5.3 Impact of Network Topology

Now, we run experiments extensively with various network topologies. We first change Tx power of each node to investigate the impact of node density. Then, we change the location of the root to give drastic variation in the topology. Lastly, we run experiments in an entirely different environment, 79 nodes on Grenoble testbed.

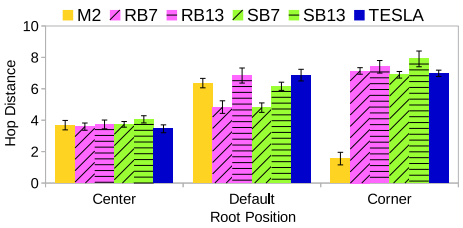
Different Tx power. In this experiment, we vary Tx power from the minimum (-17 dBm) to the maximum (3 dBm) value. We set traffic load to 6.7 packets/second, the highest load used in Section 3.5.2 anticipating that higher Tx power will result in



(a) End-to-end packet delivery ratio



(b) Radio duty-cycle (on time)



(c) Hop distance

Figure 3.11: Impact of root location.

better performance, and compare *TESLA* with M2, RB7, RB13, SB13, and SB23.

Fig. 3.10 plots end-to-end PDR, duty-cycle of radio, and average hop distance of RPL topology. As Tx power increases, Fig. 3.10(a) shows that all the schemes except RB13 provide better PDR and Fig. 3.10(b) shows that all the schemes provide lower radio duty-cycle. There are two reasons for this result. Firstly, a higher Tx power decreases average hop distance as shown in Fig. 3.10(c), which reduces network traffic since a packet can be delivered to its destination with fewer transmissions. This alleviates the level of contention. Secondly, a higher Tx power increases node density

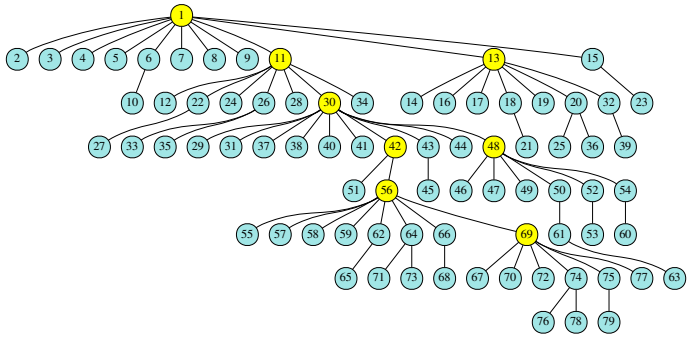
and provides more parent candidates for each node. Thus, each node can have a better chance to avoid choosing a bottleneck node as the preferred parent.

On the other hand, RB13's PDR performance shows a trade-off regarding Tx power increase: although lower network traffic can reduce contention, higher node density can cause more contention due to more nearby contenders. In RB13, the latter effect becomes stronger than the former when Tx power is higher than -4 dBm, resulting in PDR degradation. Note that RB7 escapes from the negative effect by using a shorter slotframe size, showing monotonic PDR increase with Tx power but more duty-cycle than RB13. SB usually experiences less contention than RB, providing better PDR than RB. Meanwhile, SB13 always provides better PDR but worse duty-cycle than SB23 due to its shorter slotframe size. Lastly, regardless of Tx power, *TESLA* outperforms the others considerably in terms of both reliability and energy efficiency.

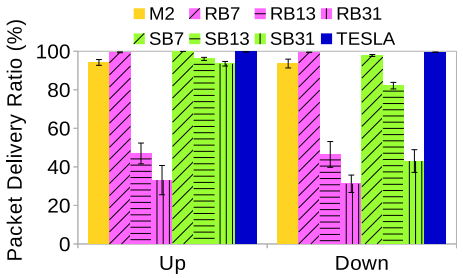
Different positions of the root. Here, we change the root location to create a totally different topology. In addition to the default root location of all the previous experiments, we also used a node at corner/center of the testbed as the root. We set Tx power to -17 dBm and aggregate traffic rate from/to the root to 1.4 packets/second, a rate at which M2 maintained more than 99% PDR in Section 3.5.2. In this experiment, *TESLA* is compared with M2, RB7, RB13, SB7, and SB13, and the results are shown in Figures 3.11(a) through 3.11(c).

When we use the center and default as the root positions, all the schemes achieve more than 99% PDR, but *TESLA* improves energy efficiency remarkably. For the case of corner root, as hop distance becomes longer and network traffic increases, PDR and duty-cycle performance drops in the minimal schedule and Orchestra. However, *TESLA* still maintains perfect reliability with the lowest energy consumption through slotframe size adaptation.

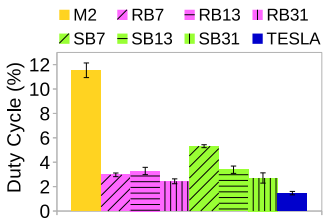
Different testbeds. We now compare *TESLA* with M2, RB7, RB13, RB31, SB7, SB13, and SB31 in a different environment: 79 nodes on the Grenoble testbed, which are deployed uniformly in a long linear topology with two lines. We use total traffic load



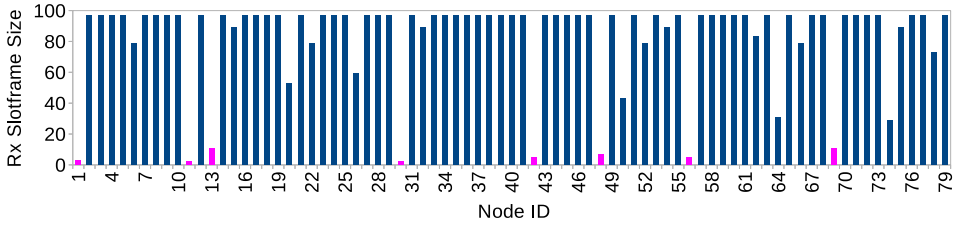
(a) A snapshot of RPL topology for 79 nodes on Grenoble testbed



(b) End-to-end packet delivery ratio



(c) Radio duty-cycle (on time)



(d) Rx slotframe size of *TESLA*

Figure 3.12: Results on the 79-node IoT-LAB Grenoble testbed.

of 2 packets/second for each of bidirectional traffic, and Tx power of -17 dBm.

The experiment results are summarized in Fig. 3.12. Compared to Fig. 3.6, the routing topology of Grenoble testbed illustrated in Fig. 3.12(a) is evenly spread out due to its linear deployment. However, there are still bottlenecks depicted as yellow-colored nodes. As two main performance metrics, Fig. 3.12(b) plots upward and downward PDR, and Fig. 3.12(c) plots radio duty-cycle. By adaptively controlling RSF sizes as illustrated in Fig. 3.12(d), *TESLA* shows the best performance in both aspects. Specifically, Fig. 3.12(d) and Fig. 3.9(b) show that *TESLA* uses more diverse RSF sizes on the Grenoble testbed than the Lille testbed. This confirms that *TESLA* does reflect the different routing topology on the Grenoble testbed, more balanced than that on the Lille testbed.

In Sections 3.5.2 and 3.5.3 so far, we have extensively evaluated the performance of *TESLA* against the state-of-the-arts with various slotframe sizes. We found that the optimal slotframe size for each of compared schemes differs according to the traffic load and network topology. It is notable, however, that we have never adjusted any of *TESLA*'s default parameters (explained in Section 3.5.1), but *TESLA* has always presented the best performances nevertheless.

3.5.4 Impact of Run-time Traffic Dynamics

To closely understand *TESLA*'s adaptability to traffic dynamics of the network at the link level, we run another experiment with a single-hop topology having five senders and a common receiver. In this experiment, each sender generates packets with two different traffic loads, 0.2 and 1 packet/second. The experiment comprises four 20-minute periods, and each sender uses the two traffic rates alternately within each period. The intervals for traffic load alternation in the four periods are 5 minutes, 1 minute, 15 seconds, and 5 seconds. This load change is shown explicitly in Fig. 3.13.

Fig. 3.13 also plots how *TESLA* adjusts the receiver's RSF size under the traffic load dynamics. Recall that we used the RSF adaptation period, T_{adapt}, of 15 seconds.

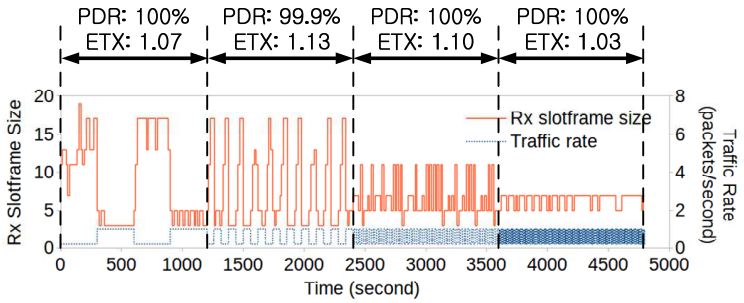


Figure 3.13: Time vs. Rx slotframe size.

The first two periods have larger intervals (*i.e.,* 5 min. and 1 min.) of traffic alternation than T_{adapt}, sufficient to adapt RSF size according to the changed traffic load. While the RSF size gradually reaches 17 when the traffic load is low, it is reduced down to 3 immediately when the traffic load becomes high. During the third and fourth periods, however, the traffic load changes faster than the RSF adaptation rate. Therefore, the estimated traffic load is always averaged through the two different traffic loads, and thus the range of RSF size variation declines. For example, it fluctuates only between 5 and 7 in the fourth period.

Nevertheless, PDR of each period is maintained above 99.9% with reasonable expected transmission counts (ETX) as indicated in Fig. 3.13. This proves that even when traffic load varies fast, *TESLA* does its best to adjust RSF size according to the average traffic load, maintaining reliability. Prioritizing reliability over energy efficiency is important at this point again.

3.5.5 Impact of *TESLA* Parameters

Lastly, we evaluate slot utilization ratio (SUR) of *TESLA* with the traffic load of 2 packets/second from/to the root, while changing the *TESLA* parameters. We compare that with Orchestra and the minimal configuration. From Fig. 3.3(a) which used the same traffic load, we chose a common slotframe size '13' for sender-based and receiver-based Orchestra, which achieves more than 99% reliability for both, and a slotframe size '2' for the minimal schedule, which provides the highest PDR. We eval-

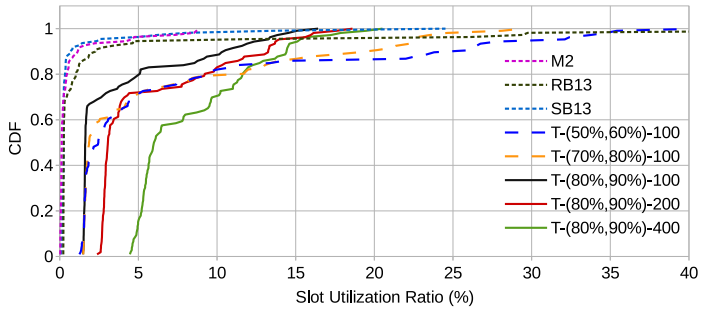


Figure 3.14: Slot utilization ratio with different PRR thresholds and upper bound of slotframe lengths.

uate *TESLA* with different pairs of ($PRR_{\text{th,low}}$, $PRR_{\text{th,up}}$) and different RSF size upper bounds. For instance, T-(80%,90%)-200 indicates *TESLA* with 80% of $PRR_{\text{th,low}}$, 90% of $PRR_{\text{th,up}}$, and the upper bound of 200.

Fig. 3.14 plots the SUR distribution of M2, RB13, SB13. In Orchestra and the minimal schedule, 80% of nodes show ¡1% SUR, most of which are leaf nodes wasting energy excessively in unnecessary Rx slots. On the contrary, *TESLA* improves their SURs, more when a larger RSF size upper bound is used. When the upper bound is 400, *TESLA* provides SUR from 5% to 12% for the 80% nodes. This result reveals that using a large maximum RSF size improves the group of nodes with low SUR (*i.e.,* leaf nodes). However, nodes with an excessively large RSF size cannot react to network dynamics promptly due to few wake-ups, degrading reliability. For example, upward and downward PDRs of T-(80%,90%)-400 are 93.0% and 95.9%, respectively, while T-(80%,90%)-100 achieves more than 99% for both. In addition, energy saving by using a large maximum RSF size is marginal since the RPL shared slotframe accounts for most of energy consumption in nodes with large RSF sizes.

On the other hand, $PRR_{\text{th,low}}$ and $PRR_{\text{th,up}}$ affect the bottleneck nodes with high SUR. With lower PRR thresholds, *TESLA* is reluctant to reduce RSF size under heavy traffic, resulting in higher SUR. However, we found *TESLA* with low $PRR_{\text{th,low}}$ and $PRR_{\text{th,up}}$ underperforms in terms of reliability, since it does not resolve poor link-layer PRR. Based on the results, we have used T-(80%,90%)-100 as our default configura-

tion for all the previous experiments.

3.6 Conclusion

We introduced *TESLA*, a dynamic scheduling solution for TSCH. In *TESLA*, each node adapts its Rx schedule with traffic awareness to improve energy efficiency while guaranteeing reliability. *TESLA* also aims to increase network capacity by using multiple channels. We implemented *TESLA* on a low-power embedded platform using ContikiOS, and evaluated it through extensive experiments on two large-scale multihop testbeds consisting of 110 and 79 low-power IEEE 802.15.4 devices. Consequently, we have shown that *TESLA* improves the state-of-the-arts with respect to both reliability and energy efficiency in any experimental environment and topology. We also demonstrated *TESLA*'s adaptability to traffic dynamics. As future work, we plan to design a dynamic TSCH scheduling for broadcast packets as well, which, collaborated with *TESLA*, can complete fully adaptive scheduling for all types of traffic.

Chapter 4

OST: On-demand TSCH Scheduling with Traffic-awareness

4.1 Introduction

As the Internet of Things (IoT) is growing up consistently, a number of applications are used and emerging. It is common that wearable devices are attached to people. In smart homes, IoT devices are connected each other for various purposes such as security and device control. The network gets bigger in smart buildings and smart cities. Industrial IoT (IIoT) networks are composed of numerous sensors and actuators, and they monitor and manage the automation of system.

To support such applications, Time-slotted channgel hopping (TSCH) was standardized by IEEE 802.15.4-2015 [16] for low-power and lossy networks (LLNs). A TSCH network is synchronized tightly and operated with in a timeslot manner. In a single timeslot, a data frame and an acknowledgement (ACK) can be exchanged. In each timeslot, a node choose a single its operation (transmit (Tx), receive (Rx), or sleep). For TSCH implementation, it is necessary to determine who (which node), when (which timeslot), and where (which channel) to act. It is called TSCH scheduling, which is not specified in the standard, and thus a open problem. The detailed explanation of TSCH is described in Section 3.2.1.

In Chapter 3, we designed a new TSCH scheduling, *TESLA*, which adapts Rx slot-

frame size according to incoming traffic load. However, there are a few of limitations of *TESLA*. Firstly, a Rx slot in *TESLA* is shared by multiple transmitters. As a result, collision happens between them. *TESLA* addresses this problem in two ways, by 1) exploiting exponential backoff for link-layer retransmissions and 2) allocating more Rx slots than actually required. This makes a lot of Rx slots unused, wasting energy with idle listening. Secondly, a *TESLA* node changes its Rx slotframe size by estimating incoming traffic from neighbors. However, receiver-side traffic estimation is not accurate, since it cannot include the packets lost due to collisions or bad link quality. Lastly, *TESLA* generates additional packet overhead to inform neighbors of a new schedule whenever an Rx slotframe size is updated.

By tackling these problems in *TESLA*, we introduce *OST*, a novel on-demand TSCH scheduling with traffic-awareness. *OST* inherits principle of traffic-aware schedule adaptation from *TESLA*. However, it dedicates each of Rx slots to a single sender, instead of using shared Rx slots. To adapt schedules to traffic load, sender-side traffic estimation is executed. After the estimation, a new dedicated schedule is exchanged between a pair of nodes (*i.e.,* a sender and receiver) without incurring any additional overhead. Furthermore, *OST* can handle a burst of traffic, by allocating additional dedicated slots in a on-demand way. We implement *OST* with ContikiOS on low-power embedded devices and evaluate *OST* with state-of-the-arts (including *TESLA*) in real multi-hop testbed with 72 nodes, showing that *OST* outperforms the others in terms of reliability and energy-efficiency.

The remainder of this chapter is organized as follows: In Section 4.2, we propose the design of *OST*, and elaborate on its main functional blocks. We discuss the implementation details and present the evaluation results in Section 4.3. We conclude the chapter in Section 4.4.

4.2 OST Design

In this section, we present *OST* design. Collaborated with RPL, *OST* adapts the schedules to updated routing neighbors (*i.e.,* the preferred parent and 1-hop children). Each *OST* node monitors traffic load from itself to each routing neighbor. Using average traffic load for each routing neighbor, the node updates its Tx periodicity, *i.e.,* Tx slotframe size, and negotiates with the corresponding neighbor for a available time offset (t_{offset}) to both. Although *OST* dedicates such periodic schedule for the pair of nodes based on average traffic load, more traffic than expected may come. This makes the packets queued, and in the worst case, queue is overflowed and packets are lost. To address this problem, *OST* additionally allocates on-demand timeslots promptly when packets are queued.

4.2.1 Slotframes

In *OST*, each node has five types of slotframes:

- **EB slotframe** is for TSCH enhanced beacons (EBs) with a constant periodicity and a dedicated channel offset.
- **Autonomous unicast slotframe (AUS)** is for autonomous unicast with a constant periodicity a dedicated channel offset.
- **Autonomous broadcast slotframe (ABS)** is for autonomous broadcast, also with a constant periodicity.
- **Periodic-provision Tx slotframe (PTS)** is for unicast transmission, per neighbor, with an elastic periodicity.
- **Periodic-provision Rx slotframe (PRS)** is for unicast reception, per neighbor, with an elastic periodicity.

The first three slotframes are same the ones with receiver-based Orchestra. They are used for TSCH/RPL control packets. PTS/PRS are made after negotiation between

a pair of nodes, but AUS/ABS are "*autonomous*". In other words, a node can always exploit AUS/ABS for its transmission without negotiation, since the node knows which t_{offset} the receiver(s) listens on. We distinguish AUS from ABS to alleviate collision of unicast packets. Thus, when a node changes its preferred parent, it transmits DAO to a new preferred-parent using AUS.

In addition, a *OST* node maintains PTS for a routing neighbor, while the neighbor has PRS for the node. Thus, a node ends up with having multiple PTSs and PRSs as many as the number of its routing neighbors. PTS and PRS have a single Tx and Rx slot, respectively. *OST* enables each node to adjust its own PTS size dynamically according to its Tx rate. This update is informed the neighbor who has corresponding PRS, and the neighbor changes its PRS size as such.

4.2.2 Periodic Provision

This section present how a pair of two neighboring nodes schedule PTS and PRS.

Selection of N

A *OST* node measures average traffic rate towards each of routing neighbors to select the size of PTS and PRS. Whenever a node (A) enqueues a unicast packet for Tx to a routing neighbor (i), A increases $L(A, i)$ by one, which indicates traffic load from A to i. For all i, $L(A, i)$ is initialized to 0 with the period of t_{update}. Before initializing $L(A, i)$, A calculates average traffic rate during last t_{update}, $R(A, i)$ for all i, as

$$R(A, i) = L(A, i)/(t_{\text{update}} \cdot n_{\text{slot}}). \tag{4.1}$$

n_{slot} is a constant indicating the number of TSCH timeslots during time unit. Thus, $t_{\text{update}} \cdot n_{\text{slot}}$ is the number of timeslots in t_{update}, and $R(A, i)$ means the average number of Tx towards i per a timeslot. Then, node A determines the size of its PTS for i as $2^{N(A,i)}$, where $N(A, i)$ is chosen as k satisfying $1/2^{k+1} < R(A, i) \leq 1/2^k$.

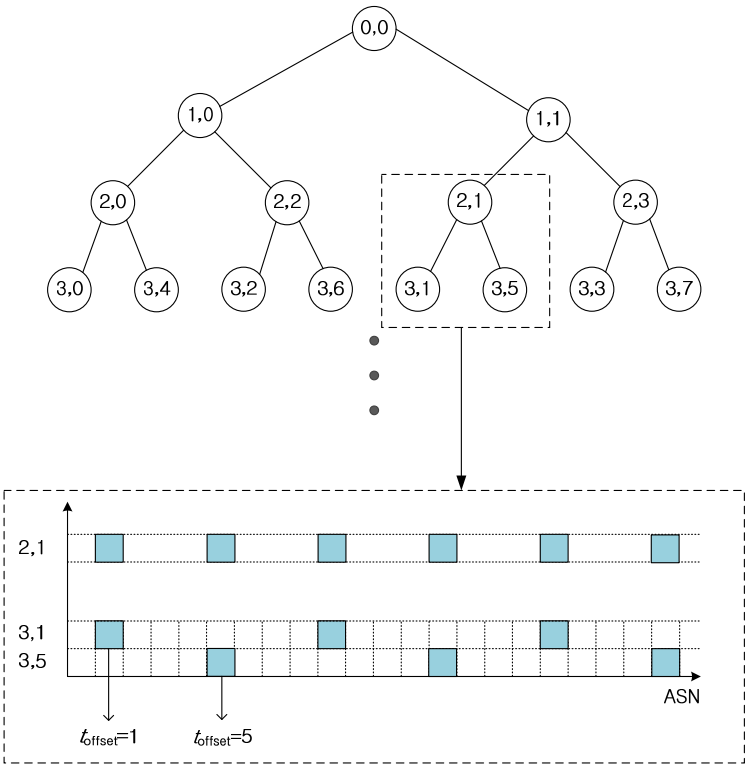


Figure 4.1: Timeslot tree.

$N(A, i)$ is updated every t_{update} before initializing $L(A, i)$. Node A piggybacks $N(A, i)$ on unicast packets destined to i, in order for i to update its PRS size for A.

Selection of t_{offset}

Each *OST* node maintains a resources tree as shown Fig. 4.1, where a circle with (n, t) represents periodic timeslot resources with N=n and t_{offset}=t. The resources (n, t) can be divided into (n+1, t) and (n+1, t+2^n). For instance, (2,1) indicates the resources with the slotframe size of 2^2 and t_{offset} of 1. This resources can be divided into (3,1) and (3,5) whose slotframe size is 2^3 and t_{offset} are 1 and 5, respectively, as exemplified in Fig. 4.1.

When node i detects a new $N(A,i)$ in a unicast packet from A, it selects $t_{\text{offset}}(N,i)$ newly for its PRS update. However, a new PRS schedule should not overlap with other PTS or PRS schedules. Thus, node i searches for the resource tree to find the resource with n=$N(A,i)$, which is not used for other PTSs and PRSs. If there are multiple resources available, node i select one randomly. Then, t in the selected resource becomes $t_{\text{offset}}(N,i)$, and node i allocates a PRS which has the periodicity of $2^{N(A,i)}$ a Rx slot with $t_{\text{offset}}(N,i)$.

Meanwhile, node i piggybacks $t_{\text{offset}}(N,i)$ on the acknowledgement (ACK) packet to node A. Then, A also schedules a PTS with $N(A,i)$ and $t_{\text{offset}}(N,i)$, which corresponds to the new PRS of node i.

4.2.3 On-demand Provision

Although a *OST* node allocates PRS/PTS based on average traffic rate, traffic pattern is usually irregular according to network topology and randomness of wireless links. If a node occasionally observes a burst of traffic, it may not be handled only with PRS/PTS, leading to queue overflow. To tackle this problem, *OST* enables a node to schedule more timeslots when there are queued packets.

Specifically, when a *OST* node A sends a unicast packet p_1 on a timeslot (ASN=t_1) towards a neighbor i, it checks whether there is another packet p_2 in Tx queue for i. If so, A makes a subsequent timeslot schedule (STS) by looking into all slotframe schedules. STS consists of $size_{STS}$ bits, where k-th bit indicates whether A has the schedule in the timeslot with ASN=t_1+k. If the timeslot is schedules, k-th bit is set to 1. Otherwise, it is set to 0.

Then, STS is included in the p_1 with the frame pending bit set. When a node i receives the p_1 in ASN=t_1, it finds out the frame pending bit is set, and compares STS of A (piggybacked on p_1) with its own STS. If m-th bits are 0 in both STSs, m is a *matching bit*, and it means neither A nor i has a schedule in ASN=t_1+m. Then, i replies the matching bit to A on the ACK for p_1, and schedules a single Rx slot

in ASN=t_1+m. When there are multiple matching bits, the earliest bit is selected. On receiving the ACK in ASN=t_1, node A allocates a Tx slot in ASN=t_1+m. Accordingly, node A is allowed to transmit p_2 in ASN=t_1+m. Note that this on-demand provision occurs recursively. In other words, when node A has more packets other than p_2, it updates its STS, and delivers it on p_2 again using the matching slot with ASN=t_1+m.

4.3 Evaluation

4.3.1 Methodology and Experiment Setup

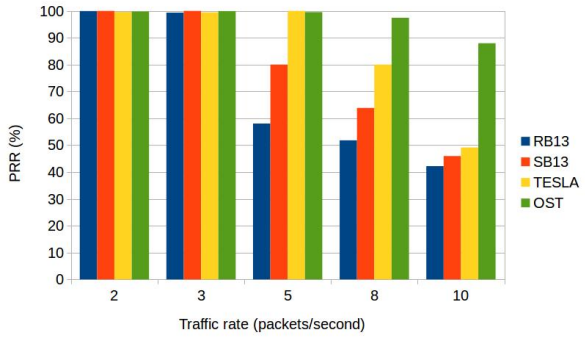
We implement *OST* on ContikiOS and compare it with receiver-based Orchestra (RB), sender-based Orchestra(SB) and *TESLA*. To this end, we evaluate them using 72 low-power nodes on FIT/IoT-lab in Grenoble. Each node features a 32-bit ARM Cortex-M3 microcontroller (STM32F103REY) and an AT86RF231 IEEE 802.15.4 radio chip. This node is representative of today's state-of-the art IoT devices [85]. We use Contiki-RPL implementation on top of the TSCH scheduling schemes.

In this experiment, we use the size of EB slotframe as 397 in all protocols. In RB and SB, the sizes of RPL shared slotframes are 57 and 41, respectively, and the sizes of unicast slotframes are 13 in common. Meanwhile, *TESLA* uses 41 for the length of RPL shared slotframe. *OST* uses AUS and ABS with the sizes of 47 and 57, respectively.

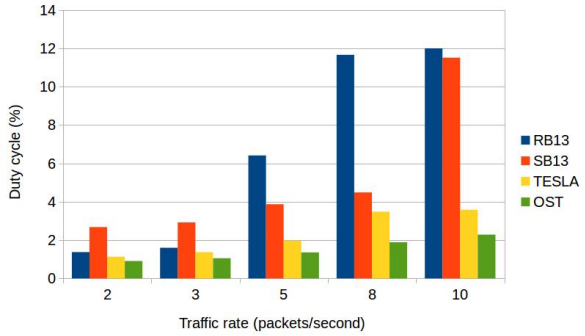
All schemes uses Tx queue size of 16 for each of routing neighbor, and link-layer maximum retransmissions is 8. In all experiments, the application payload is 59 bytes carried in UDP/IPv6 datagrams over 6LoWPAN, reaching 109 bytes of the data frame size. TSCH hops over four best channels: 15, 20, 25, and 26.

4.3.2 Experimental Results

We evaluate the protocols according to various traffic rate. There are bidirectional traffic (*i.e.,* upwards and downwards), aggregated traffic load is same. Aggregate traffic rate for each direction is from 2 to 10 packets/second. The results are shown in Fig. 4.2.



(a) End-to-end packet delivery ratio



(b) Radio duty-cycle (on time)

Figure 4.2: PDR, duty-cycle according to different traffic load.

Fig. 4.2(a) presents average end-to-end packet reception ratio (PRR). Generally, as traffic rate increases, PRR decreases due to insufficient resources. SB shows better PDR with the cost of high energy consumption, which is shown in Fig. 4.2(b) as radio duty-cycle. *TESLA* improves Orchestra in terms of both reliability and energy-efficiency as discussed in Section 3.5. However, *OST* outperforms even *TESLA* in both aspects, since it dedicates the resources and allocates more in real time whenever required.

4.4 Conclusion

In this chapter, we introduce *OST*, a novel on-demand TSCH scheduling with traffic-awareness. It adapts the slotframe sizes for periodic provision according average traffic rate. Moreover, it allocates more timeslots promptly when there are packets queued. It addresses the problem of unpredictable traffic. We implemented *OST* on a low-power embedded platform using ContikiOS, and evaluated it through extensive experiments on large-scale testbed consisting of 72 low-power IEEE 802.15.4 devices. We compared*OST*'s performance with state-of-the-arts, showing outstanding improvement in both reliability and energy-efficiency in various traffic load.

Chapter 5

CONCLUSION

In this dissertation, we focused on the adaptability in low-power and lossy networks. As a adaptable protocol to mobile devices, we first proposed *MAPLE*, an asymmetric transmit power-based routing architecture that supports mobility of resource-constrained devices in LLNs. By using high transmit power of the gateway in *MAPLE*, LLN nodes are synchronized for low duty-cycle operation, and RSSI gradient field based opportunistic routing is designed which eliminates the need for any neighbor or routing table. This enables scalability, low and constant memory footprint, and provides responsive routing metric without control overhead. Next, we designed two TSCH scheduling methods, which are adaptable to traffic load. In *TESLA*, we introduced a dynamic scheduling solution for TSCH with traffic-awareness. Each *TESLA* node adapts its Rx schedule with traffic awareness to improve energy efficiency while guaranteeing reliability. Then, we proposed *OST*, a novel on-demand TSCH scheduling with traffic-awareness. *OST* adapts the slotframe sizes for periodic provision according average traffic rate. Moreover, it allocates more timeslots promptly when there are packets queued. It addresses the problem of unpredictable traffic. For performance evaluation of *MAPLE*, *TESLA*, and *OST*, we implemented them on resource-constrained low-power embedded devices, and conducted extensive experiments on large-scale multi-hop testbeds. Compared with state-of-the-arts, we showed their superiority in terms of

packet delivery ratio and duty-cycle.

Bibliography

[1] J. Adkins, B. Ghena, N. Jackson, P. Pannuto, S. Rohrer, B. Campbell, and P. Dutta, "The signpost platform for city-scale sensing," in *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IEEE Press, 2018, pp. 188–199.

[2] H.-S. Kim, J. Ko, and S. Bahk, "Smarter markets for smarter life: applications, challenges, and deployment experiences," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 34–41, 2017.

[3] 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.

[4] J. Park, W. Nam, T. Kim, J. Choi, S. Lee, D. Yoon, J. Paek, and J. Ko, "Glasses for the third eye: Improving clinical data analysis with motion sensor-based filtering," in *The 15th ACM International Conference on Embedded Networked Sensor Systems (SenSys'17)*, Nov. 2017, pp. 99–112.

[5] J. Ko, J. H. Lim, Y. Chen, R. Musvaloiu-E, A. Terzis, G. M. Masson, T. Gao, W. Destler, L. Selavo, and R. P. Dutton, "Medisn: Medical emergency detection in sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, no. 1, p. 11, 2010.

[6] M. S. Shahamabadi, B. B. M. Ali, P. Varahram, and A. J. Jara, "A network mobility solution based on 6LoWPAN hospital wireless sensor network (NEMO-HWSN)," in *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2013, pp. 433–438.

[7] 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 *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2015, pp. 281–294.

[8] S. Berman, V. Kumar, and R. Nagpal, "Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 378–385.

[9] P. Tokekar, D. Bhadauria, A. Studenski, and V. Isler, "A robotic system for monitoring carp in Minnesota lakes," *Journal of Field Robotics*, vol. 27, no. 6, pp. 779–789, 2010.

[10] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.-P. Vasseur, and R. Alexander, "RPL: IPv6 routing protocol for low-power and lossy networks," *RFC 6550*, 2012.

[11] S. Duquennoy, O. Landsiedel, and T. Voigt, "Let the tree bloom: Scalable opportunistic routing with ORPL," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2013.

[12] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *ACM conference on embedded networked sensor systems (SenSys)*, 2015, pp. 337–350.

[13] 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.

[14] 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 & Tutorials*, vol. 19, no. 4, pp. 2502–2525, 2017.

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

[16] "IEEE standard for low-rate wireless networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, April 2016.

[17] T. Clausen, A. C. de Verdiere, J. Yi, A. Niktash, Y. Igarashi, H. Satoh, U. Herberg, C. Lavenu, T. Lys, C. Perkins *et al.*, "The lightweight on-demand ad hoc distance-vector routing protocol-next generation (LOADng)," *draft-clausen-lln-loadng-09*, 2013.

[18] I. El Korbi, M. B. Brahim, C. Adjih, and L. A. Saidane, "Mobility enhanced RPL for wireless sensor networks," in *IEEE International Conference on the Network of the Future (NOF)*, 2012, pp. 1–8.

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

[20] O. Gaddour, A. Koubâa, R. Rangarajan, O. Cheikhrouhou, E. Tovar, and M. Abid, "Co-RPL: RPL routing for mobile low power wireless sensor networks using Corona mechanism," in *IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2014, pp. 200–209.

[21] C. Cobarzan, J. Montavont, and T. Noel, "Analysis and performance evaluation of RPL under mobility," in *IEEE Symposium on Computers and Communication (ISCC)*, 2014, pp. 1–6.

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

[23] 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.

[24] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[25] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks," in *ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.

[26] D. Moss and P. Levis, "BoX-MACs: Exploiting physical and link layer boundaries in low-power networking," *Computer Systems Laboratory Stanford University*, vol. 64, no. 66, p. 120, 2008.

[27] A. Dunkels, "The contikimac radio duty cycling protocol," 2011.

[28] Y. Sun, O. Gurewitz, and D. B. Johnson, "RI-MAC: a receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks," in *ACM conference on Embedded network sensor systems (SenSys)*, 2008.

[29] T. Watteyne, A. Mehta, and K. Pister, "Reliability through frequency diversity: why channel hopping makes sense," in *Proceedings of the 6th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. ACM, 2009, pp. 116–123.

[30] J. Lee, T. Kwon, and J. Song, "Group connectivity model for industrial wireless sensor networks," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 5, pp. 1835–1844, 2010.

[31] M. Nobre, I. Silva, and L. A. Guedes, "Routing and scheduling algorithms for WirelessHART Networks: a survey," *Sensors*, vol. 15, no. 5, pp. 9703–9740, 2015.

[32] R. T. Hermeto, A. Gallais, and F. Theoleyre, "Scheduling for IEEE802.15.4-TSCH and slow channel hopping MAC in low power industrial wireless networks: A survey," *Computer Communications*, 2017.

[33] 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.

[34] ——, "Traffic-aware time-critical scheduling in heavily duty-cycled IEEE 802.15.4e for an industrial IoT," *Proc. of IEEE Sensors 2012*, pp. 1–4, 2012.

[35] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel, "On optimal scheduling in duty-cycled industrial IoT applications using IEEE 802.15.4e TSCH," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3655–3666, 2013.

[36] V. Sempere-Payá, J. Silvestre-Blanes, D. Todolí, M. Valls, and S. Santonja, "Evaluation of TSCH scheduling implementations for real WSN applications," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–4.

[37] 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 (WCNC)*, 2016, pp. 1–6.

[38] M. Hashimoto, N. Wakamiya, M. Murata, Y. Kawamoto, and K. Fukui, "End-to-end reliability-and delay-aware scheduling with slot sharing for wireless sen-

sor networks," in *International Conference on Communication Systems and Networks (COMSNETS)*, 2016.

[39] 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 *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2017.

[40] K.-H. Choi and S.-H. Chung, "A new centralized link scheduling for 6TiSCH wireless industrial networks," in *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Springer, 2016, pp. 360–371.

[41] I. Rhee, A. Warrier, J. Min, and L. Xu, "DRAND: Distributed randomized TDMA scheduling for wireless ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 8, no. 10, pp. 1384–1396, 2009.

[42] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu, "Z-MAC: a hybrid MAC for wireless sensor networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 16, no. 3, pp. 511–524, 2008.

[43] R. Soua, P. Minet, and E. Livolant, "DiSCA: A distributed scheduling for convergecast in multichannel wireless sensor networks," in *14th IFIP/IEEE Symposium on Integrated Network and Service Management (IEEE IM)*, 2015.

[44] A. Aijaz and U. Raza, "Deamon: a decentralized adaptive multi-hop scheduling protocol for 6TiSCH wireless networks," *IEEE Sensors Journal*, vol. 17, no. 20, pp. 6825–6836, 2017.

[45] C. Vallati, S. Brienza, G. Anastasi, and S. K. Das, "Improving network formation in 6TiSCH networks," *IEEE Transactions on Mobile Computing*, 2018.

[46] D. Dujovne, L. A. Grieco, M. R. Palattella, and N. Accettura, "6TiSCH 6top scheduling function zero (SF0)," *Internet Engineering Task Force, Tech. Rep. draft-ietf-6tisch-6top-sf0-01 [work in progress]*, vol. 8, 2016.

[47] Q. Wang, X. Vilajosana, and T. Watteyne, "6top protocol (6p)," *Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-6top-protocol-02*, 2016.

[48] T. Chang, T. Watteyne, Q. Wang, and X. Vilajosana, "LLSF: Low latency scheduling function for 6TiSCH networks," in *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2016.

[49] N. Accettura, E. Vogli, M. R. Palattella, L. A. Grieco, G. Boggia, and M. Dohler, "Decentralized traffic aware scheduling in 6TiSCH networks: Design and experimental evaluation," *IEEE Internet of Things Journal*, 2015.

[50] K.-H. Phung, B. Lemmens, M. Goossens, A. Nowe, L. Tran, and K. Steenhaut, "Schedule-based multi-channel communication in wireless sensor networks: A complete design and performance evaluation," *Ad Hoc Networks*, vol. 26, pp. 88–102, 2015.

[51] M. R. Palattella, T. Watteyne, Q. Wang, K. Muraoka, N. Accettura, D. Dujovne, L. A. Grieco, and T. Engel, "On-the-fly bandwidth reservation for 6TiSCH wireless industrial networks," *IEEE Sensors Journal*, vol. 16, no. 2, pp. 550–560, 2016.

[52] T. P. Duy, T. Dinh, and Y. Kim, "Distributed cell selection for scheduling function in 6TiSCH networks," *Computer Standards & Interfaces*, vol. 53, pp. 80–88, 2017.

[53] F. Theoleyre and G. Z. Papadopoulos, "Experimental validation of a distributed self-configured 6TiSCH with traffic isolation in low power lossy networks," in *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM, 2016, pp. 102–110.

[54] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," *RFC 4944*, Sep. 2007.

[55] H.-S. Kim, M.-S. Lee, Y.-J. Choi, J. Ko, and S. Bahk, "Reliable and energy-efficient downward packet delivery in asymmetric transmission power-based networks," *ACM Transactions on Sensor Networks*, vol. 12, no. 4, pp. 34–1, 2016.

[56] Cisco, "Connected grid networks for smart grid - Field area network," http://www.cisco.com/web/strategy/energy/field_area_network.html. [Online]. Available: http://www.cisco.com/web/strategy/energy/field_area_network.html

[57] T. Group, "Thread Stack Fundamentals," http://threadgroup.org/, Jul. 2015.

[58] M. W. Group, "Mesh profile," *Bluetooth Specification*, Jul. 2017.

[59] D. Lymberopoulos, Q. Lindsey, and A. Savvides, "An empirical characterization of radio signal strength variability in 3-D IEEE 802.15. 4 networks using monopole antennas," in *EWSN*. Springer.

[60] M. Lindhé, K. H. Johansson, and A. Bicchi, "An experimental study of exploiting multipath fading for robot communications," in *International Conference on Robotics Science and Systems (RSS)*, June 2007, pp. 289–296.

[61] M. Malajner, K. Benkic, P. Planinsic, and Z. Cucej, "The accuracy of propagation models for distance measurement between WSN nodes," in *IEEE International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2009, pp. 1–4.

[62] F. Hermans, O. Rensfelt, T. Voigt, E. Ngai, L.-Å. Norden, and P. Gunningberg, "SoNIC: classifying interference in 802.15. 4 sensor networks," in *ACM/IEEE IPSN*, 2013.

[63] W. Dong, J. Yu, and X. Liu, "CARE: Corruption-aware retransmission with adaptive coding for the low-power wireless," in *IEEE International Conference on Network Protocols (ICNP)*, 2015, pp. 235–244.

[64] F. Cadger, K. Curran, J. Santos, and S. Moffett, "A survey of geographical routing in wireless ad-hoc networks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 621–653, 2013.

[65] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *IEEE IPSN*, 2011, pp. 73–84.

[66] S. Duquennoy, A. Elsts, B. Nahas, and G. Oikonomou, "TSCH and 6TiSCH for Contiki: Challenges, design and evaluation," *IEEE DCOSS*, 2017.

[67] A. Dunkels, "The contikimac radio duty cycling protocol," *SICS Technical Report*, 2011.

[68] X. Ji, Y. He, J. Wang, W. Dong, X. Wu, and Y. Liu, "Walking down the stairs: Efficient collision resolution for wireless sensor networks," in *IEEE INFOCOM*, 2014, pp. 961–969.

[69] S. Jeong, H.-S. Kim, S.-G. Yoon, and S. Bahk, "Q-BT: Queue-based burst transmission over an asynchronous duty-cycle MAC protocol," *IEEE Communications Letters*, vol. 20, no. 4, pp. 812–815, 2016.

[70] V. Shah and S. Krishnamurthy, "Handling asymmetry in power heterogeneous ad hoc networks: A cross layer approach," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2005.

[71] X. Du, D. Wu, W. Liu, and Y. Fang, "Multiclass routing and medium access control for heterogeneous mobile ad hoc networks," *IEEE Transactions on Vehicular Technology*, vol. 55, no. 1, pp. 270–277, 2006.

[72] T. Instruments, "CC2420: 2.4 GHz IEEE 802.15. 4/ZigBee-ready RF transceiver," *Available at http://www.ti.com/lit/gpn/cc2420*, vol. 53, 2006.

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

[74] P. Levis and T. H. Clausen, "The trickle algorithm," *RFC 6206*, 2011.

[75] X. Vilajosana, K. Pister, and T. Watteyne, "Minimal 6TiSCH configuration - draft-ietf-6tisch-minimal-21," *IETF Draft*, 2017.

[76] P. Thubert, T. Watteyne, R. Struik, and M. Richardson, "An architecture for IPv6 over the TSCH mode of IEEE 802.15. 4. draft-ietf-6tisch-architecture-10," *IETF Draft, June*, 2016.

[77] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTunes: Runtime parameter adaptation for low-power MAC protocols," in *Proceedings of the 11th international conference on Information Processing in Sensor Networks*. ACM, 2012, pp. 173–184.

[78] H.-S. Kim, J. Paek, D. E. Culler, and S. Bahk, "Do not lose bandwidth: Adaptive transmission power and multihop topology control," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2017, pp. 99–108.

[79] T. Lee, J. Han, M.-S. Lee, H.-S. Kim, and S. Bahk, "CABLE: connection interval adaptation for BLE in dynamic wireless environments," in *2017 14th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2017, pp. 1–9.

[80] 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, 2017.

[81] S. Kumar, M. P. Andersen, H.-S. Kim, and D. E. Culler, "TCPlp: System design and analysis of full-scale TCP in low-power networks," *arXiv preprint arXiv:1811.02721*, 2018.

[82] S. Boubiche, D. E. Boubiche, A. Bilami, and H. Toral-Cruz, "Big data challenges and data aggregation strategies in wireless sensor networks," *IEEE Access*, vol. 6, pp. 20 558–20 571, 2018.

[83] 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 & Tutorials*, vol. 19, no. 4, pp. 2502–2525, 2017.

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

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

[86] D. Reina, M. Askalani, S. Toral, F. Barrero, E. Asimakopoulou, and N. Bessis, "A survey on multihop ad hoc networks for disaster response scenarios," *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, p. 647037, 2015.

[87] M. Erdelj, M. Król, and E. Natalizio, "Wireless sensor networks and multi-UAV systems for natural disaster management," *Computer Networks*, vol. 124, pp. 72–86, 2017.

[88] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-time wireless sensor-actuator networks for industrial cyber-physical systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1013–1024, 2016.

[89] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, “The trickle algorithm,” Tech. Rep., 2011.

초 록

IoT (Internet of Things)는 저전력 임베디드 기기들로 구성되는 새로운 네트워크 시대를 개척하였다. 산업 IoT 네트워크에서는 수 많은 센서와 구동기가 배치되어 시스템을 모니터링하고 원격 제어한다. 스마트홈에서부터 스마트시티까지, 전기관리, 보안, 건강관리, 스마트그리드 등과 같은 다양한 어플리케이션들이 등장하고 있다. 이렇게 IoT 어플리케이션이 다양해짐에 따라, 이러한 네트워크를 제어/관리하기 위해 신뢰성 높고, 에너지 소모가 적고, 유연한 동작을 할 수 있는 네트워크 프로토콜에 대한 수요가 급증하고 있다. 이를 위해 본 논문에서는 저전력 네트워크를 위한 3가지 각기 다른 적응적 프로토콜을 제안한다.

먼저 우리는 이동성 저전력 네트워크에 초점을 맞춘다. 최근 등장하는 IoT 어플리케이션에서는 이동성이 매우 중요한 부분을 차지하고 있다. 하지만 그동안 대부분의 저전력 프로토콜은 저전력 디바이스의 하드웨어적 한계를 이유로 이동성에 초점을 맞추지 않았다. 몇몇 연구에서는 네트워크의 이동성 지원을 위한 프로토콜을 제안하였지만, 그들은 듀티사이클링, 제어 오버헤드, 메모리 사용 등과 같은 저전력 디바이스에서 필수적인 요소를 반영하지 못하였다. 이러한 문제를 해결하고자 우리는 *MAPLE* 을 설계한다. *MAPLE* 은 고전력 전송 전력을 사용할 수 있는 보더 라우터를 사용하여 비대칭 전력 네트워크를 구성한다. 이를 통해 저전력 네트워크에서 듀티사이클링을 통해 에너지 소비를 절감함과 동시에 네트워크의 이동성을 지원할 수 있게 된다. 우리는 *MAPLE* 을 실제 저전력 임베디드 플랫폼에 구현을 하고 31개의 저전력 ZigBee 노드를 사용하여 구축한 테스트베드에서 성능을 검증한다. 이를 통해 결과적으로 *MAPLE* 은 기존 기법 대비 27.2%/55.7%만큼 상향링크/하향링크

패킷수신율을 개선하였고, 그와 더불어 17.9%의 에너지 절감을 보여주었다. 다음으로 우리는 최근 IEEE 802.15.4-2015에서 표준화된 TSCH (Time Slotted Channel Hopping) 프로토콜에 주목한다. 기존 비동기 방식의 MAC 프로토콜에 비해, TSCH는 더욱 높은 신뢰도와 저전력 동작을 보장한다. 이러한 연유로 최근 다양한 TSCH 스케줄링 기법이 제안되었다. 하지만 그들은 "고정된" 슬랏프레임 길이를 사용함으로써, 예측할 수 없는 트래픽과 라우팅 토폴로지를 가진 다양한 어플리케이션과 서비스를 지원하기 위한 유연한 동작을 보장하지 못한다. 따라서 이를 해결하기 위한 TSCH 스케줄링 프로토콜로서 *TESLA* 를 제안한다. *TESLA* 는 트래픽 양을 측정하여 실시간으로 슬랏프레임을 적응적으로 조절할 수 있다. 이를 통해 높은 패킷수신율을 보장함과 동시에 에너지 소모를 최소화할 수 있다. 우리는 110개/77개의 노드로 구성된 대규모 테스트베드에서 *TESLA* 의 성능을 검증한다. 그 결과 기존 기법 대비 99%의 신뢰성을 유지하면서 동시어 70.2%의 라디오 에너지 사용을 감축한다는 사실을 확인하였다.

마지막 연구는 우리가 발견한 *TESLA* 의 한계점에서 시작한다. *TESLA* 에서 하나의 수신 슬랏은 여러 명의 송신 노드에 의해 공유가 된다. 그들 간의 충돌을 막기 위해 *TESLA* 는 어쩔 수 없이 실제 필요한 수신 슬랏의 갯수보다 많은 양을 할당하게 된다. 이러한 자원의 낭비를 줄이고자 우리는 *OST* 라는 온디맨드 방식의 TSCH 스케줄링 기법을 제안한다. *TESLA* 와 마찬가지로 *OST* 는 트래픽 로드에 맞춰 적응적 동작을 하지만, 이를 더욱 개선하여 각 스케줄링은 여러 명의 송신 노드에게 공유되지 않고 단 하나의 노드에게만 할당된다. 더욱이, *OST* 는 순간적인 트래픽 버스트에도 대처하기 위한 온디맨드 방식의 추가적인 자원할당을 포함한다. 우리는 *OST* 를 ContikiOS를 통해 구현하고 72개의 노드로 구성된 다중 홉 테스트베드에서 성능 검증을 진행함으로써 *OST* 의 우수성을 보여준다.

주요어: 저전력 네트워크, IEEE 802.15.4, 이동성, 비대칭 전력 네트워크, TSCH, 동적 스케줄링, 무선 네트워크 기법

학번: 2013-20877