IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 23, 2019, accepted August 2, 2019, date of publication September 11, 2019, date of current version September 24, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2940457*

# TESLA: Traffic-Aware Elastic Slotframe Adjustment in TSCH Networks

**SEUNGBEOM JEONG[1], JEONGYEUP PAEK[2], (Senior Member, IEEE), HYUNG-SIN KIM[3], AND SAEWOONG BAHK[1], (Senior Member, IEEE)**

[1]Department of Electrical and Computer Engineering, INMC, Seoul National University Seoul 08826, South Korea
[2]School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea
[3]Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720, USA

Corresponding authors: Jeongyeup Paek (jpaek@cau.ac.kr) and Saewoong Bahk (sbahk@snu.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning under Grant 2017R1E1A1A01074358, in part by the grant to Bio-Mimetic Robot Research Center funded by the Defense Acquisition Program Administration, and by the Agency for Defense Development under Grant UD160027ID, and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2017R1D1A1B03031348.

**ABSTRACT** Low-power wireless network for the emerging Internet of Things (IoT) should be reliable enough to satisfy the application requirements, and also energy-efficient for embedded devices to remain battery powered. Synchronized communication methods such as Time Slotted Channel Hopping (TSCH) have shown promising results for these purposes, achieving end-to-end reliability over 99% with low duty-cycles. 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.

**INDEX TERMS** Low-power and lossy network, IEEE 802.15.4, TSCH, dynamic scheduling, wireless network, MAC protocol.

## I. 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 [4] 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 [5]–[7], 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 [8]. 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* [9] of 6TiSCH [10] and Orchestra [11] (Section II-B and II-C).

*Challenge:* Any well designed protocol can end up with miserable performance if its parameters are not set appropriately [12]–[14]. 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.

The associate editor coordinating the review of this manuscript and approving it for publication was Bo Li.

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 [15], forwarding traffic intensity [16], [17], and application behaviors [2], [18].

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 III). 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 IV). *TESLA* inherits and extends the Orchestra's receiver-based scheduler [11] 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 and extensive evaluation on two distinct testbeds with 110 nodes and 79 nodes (Section V), showing that *TESLA* outperforms the state-of-the-art in terms of reliability and energy-efficiency using distributed dynamic scheduling.

## II. BACKGROUND

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

### A. IEEE 802.15.4 TSCH

TSCH is a time-synchronous MAC specified in the IEEE 802.15.4-2015 standard [4]. Its synchronous operation saves energy by reducing redundant transmissions or idle listening compared to asynchronous MACs [6], [7], [19], and its channel hopping enables resilient operation over narrow-band interference and multipath fading [8].

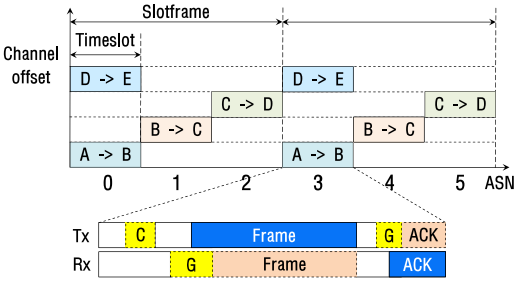



**FIGURE 1. An example of TSCH slotframe schedule and timeslot with slotframe size of 3.**

TSCH network is globally time-synchronized, and time is divided into *timeslots* as in Figure 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 Figure 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

$$Channel = List_c[(ASN + offset_{\text{channel}}) \% N_{List_c}] \quad (1)$$

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, and others in Section VI as related work. 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.

### B. 6TiSCH AND ITS MINIMAL CONFIGURATION

In 2013, IETF Working Group 6TiSCH [10] was established for the purpose of designing IPv6 support on top of TSCH. 6TiSCH defines a TSCH *minimal configuration* [9], 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.

### C. ORCHESTRA

Orchestra [11] provides autonomous TSCH scheduling together with the RPL routing layer.[1] For the construction of TSCH and RPL network, Orchestra employs two types 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,

$$offset_{\text{time}} = h(\text{MAC}) \ \% \ S_{\text{sf}} \tag{2}$$

[1]RPL is the standard IPv6 routing protocol for LLNs. The detailed description and related work for RPL are in [20] and [21], which is out of the scope of this paper.

where $h$ is a hash function shared in the network, MAC is the hardware address of the node, and $S_{\text{sf}}$ 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.

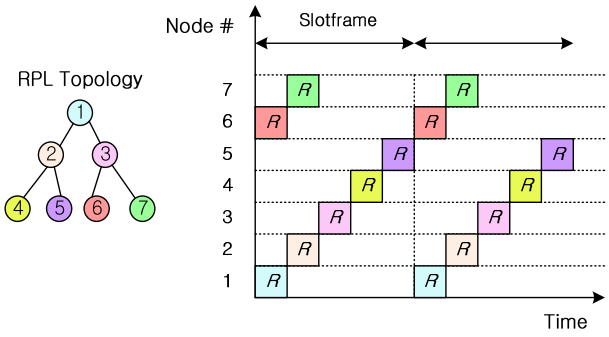


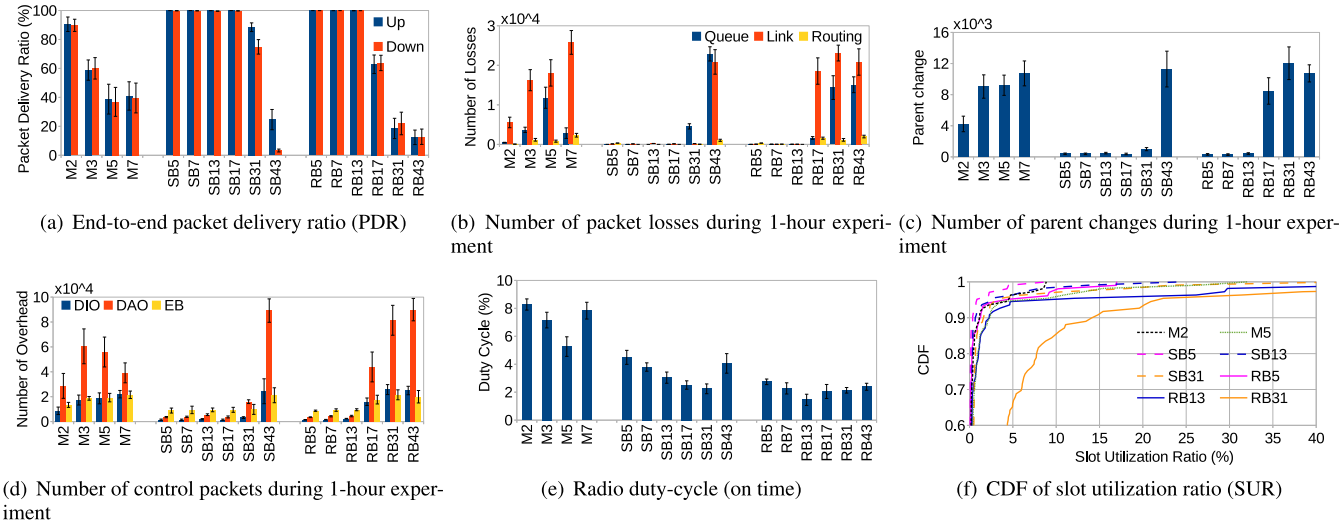

**FIGURE 2.** An example of TSCH scheduling in receiver-based Orchestra.

Figure 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 $offset_{\text{time}}$ of $R$ using its ID as output of $h$(MAC) where $S_{\text{sf}}$ is 5. For example, node 1 or 7 has the $offset_{\text{time}}$ 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 other hand, in sender-based scheduling, a node has a fixed Tx slot and multiple Rx slots.

## III. 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




(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.** **Various performance metrics in Orchestra and 6TiSCH minimal configuration with different slotframe sizes.**

configuration [9], sender-based and receiver-based Orchestra [11], implemented on ContikiOS [22].

### A. METHODOLOGY

We evaluate the three schemes on FIT/IoT-LAB testbed [23] 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 × 109) seconds to equal the bidirectional traffic load. Detailed explanations of the experimental settings will be provided in Section V-A.

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 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.

### B. EXPERIMENTAL RESULTS

Figure 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. Figure 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. (2). SB and RB achieve PDR of >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, Figure 3(b) plots the number of three types of packet losses: *queue loss*, *link loss*, and *routing loss*. Figure 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 [15]. For example, when RB employed a slotframe size of 31, 85% of lost packets disappeared at just two bottleneck nodes.

However, Figure 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(c) and 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(b) and 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 [13].

Figure 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 III-A, 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 Figure 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 Figure 3(a). The minimal configuration cases exhibit low PDR with low SUR, an undesirable performance characteristic.

### C. 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 [16]. A node's application traffic can change at run time due to emergency detection [18], [24], device control [25], [26], 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 [17]. This motivates *TESLA*, a technique for dynamic and local adjustment of slotframe size according to traffic load.

## IV. 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 schedule 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.

### A. 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 [11]. 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*. Figure 4 shows an example of *TESLA* scheduling for the same routing topology as Figure 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.

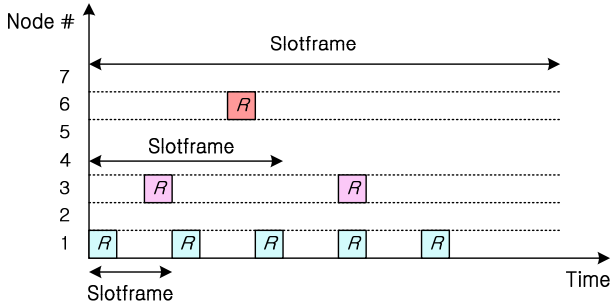



**FIGURE 4.** An example of *TESLA* scheduling.

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

$$offset_{\text{time}}(t) = h_t(\text{MAC}) \ \% \ S_{\text{rsf}}(t). \tag{3}$$

Differently from Orchestra, the RSF size ($S_{\text{rsf}}$) changes over time ($t$) and so does $offset_{\text{time}}$. 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.

### B. Rx SLOTFRAME SIZE ADAPTATION

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

#### 1) 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 III, node $i$ calculates the traffic load $L(i, A)$ by adding two elements, as

$$L(i, A) = M(i, A) + Q(i, A). \tag{4}$$

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 $A$.

#### 2) 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 $T_{\text{adapt}}$): (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 $T_{\text{adapt}}$). $L(i, A)$ has been accumulated since node $A$'s last RSF size change (*i.e.,* during last $t_{\text{update}}(A)$, longer than or equal to $T_{\text{adapt}}$ because the adaptation procedure may not always change the RSF size). Thus, the traffic load from node $i$ to node $A$ during recent $T_{\text{adapt}}$, namely $L_{\Delta}(i, A)$, is

$$L_{\Delta}(i, A) = L(i, A) - L_{\text{last}}(i, A). \tag{5}$$

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 $T_{\text{adapt}}$, the normalized total incoming traffic load at node $A$, $L_{\Delta,\text{n}}(A)$, is computed as

$$L_{\Delta,\text{n}}(A) = \sum_{i \in \mathbb{N}(A)} \frac{L_{\Delta}(i, A)}{W}. \tag{6}$$

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

Figure 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 $T_{\text{adapt}}$, *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 $T_{\text{adapt}}$.

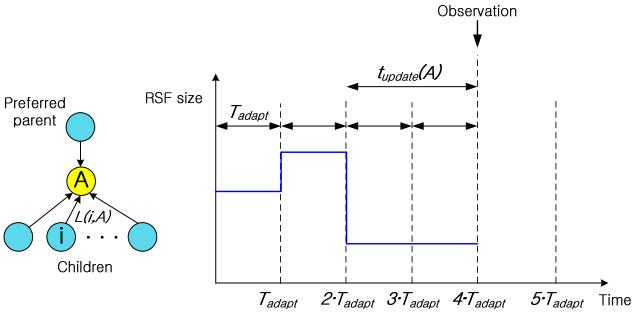



**FIGURE 5.** Topology of node *A* and its RPL routing neighbors, and an example of RSF size adaptation.

Given that node $A$ has $W$ Rx slots for last $T_{\text{adapt}}$, 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

$$PRR_c(i,A) = \prod_{k(\neq i)\in\mathbb{N}(A)} (1 - \frac{L_\Delta(k,A)}{W}). \tag{7}$$

Then, node $A$ uses $PRR_{c,\min}(A)$, 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 $PRR_{c,\min}(A)$, are mutually complementary. For example, when traffic is heavy but comes from the only one neighbor node, $PRR_{c,\min}(A)$ 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, $PRR_{c,\min}(A)$ 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.

### 3) 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. (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 ($S_{\text{rsf}}$) is a prime number, which increases frequency diversity. As an example, in Figure 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. (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 IV-A, 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.

### 4) TRAFFIC-AWARE Rx SLOTFRAME SIZE ADAPTATION

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

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

**Algorithm 1** How to Decrease Rx Slotframe (RSF) Size

```
2  index ← FindIndex(S_rsf, ℙ);
4  W_new ← W;
6  while (PRR_{c,min}(A) < PRR_{th,low}) || (L_{Δ,n}(A) > L_th) do
8      S_rsf,new ← ℙ[--index];
10     W_new ← (W·S_rsf) / S_rsf,new;
12 S_rsf ← S_rsf,new;
```

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 ($S_{\text{rsf}}$). For example, $index = 1$ when $S_{\text{rsf}} = 2$ and $index = 3$ when $S_{\text{rsf}} = 5$. Next, $W_{\text{new}}$ is defined as the expected number of Rx slots during next $T_{\text{adapt}}$ with a new RSF size, and initialized to $W$, the number of Rx slots with the current RSF size in last $T_{\text{adapt}}$. Then in the loop on line 3, the two traffic load metrics, $PRR_{c,\min}(A)$ and $L_{\Delta,\text{n}}(A)$, are calculated using $W_{\text{new}}$ 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) $PRR_{c,\min}(A)$ is worse than a lower bound ($PRR_{\text{th,low}}$).
2) $L_{\Delta,\text{n}}(A)$ exceeds a threshold ($L_{\text{th}}$).

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 $S_{\text{rsf}}$ (line 4). In the following $T_{\text{adapt}}$ with the new RSF size, node $A$ is expected to wake up $\frac{S_{\text{rsf}}}{S_{\text{rsf,new}}}$ times more. Thus, $W_{\text{new}}$ goes up accordingly (line 5), which increases $PRR_{c,\min}(A)$ 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 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.

**Algorithm 2** How to Increase Rx Slotframe (RSF) Size

```
2  index ← FindIndex(S_rsf, ℙ);
4  W_new ← W;
6  while (PRR_{c,min}(A) > PRR_{th,up}) && (L_{Δ,n}(A) < L_th) do
8      S_rsf,new ← ℙ[++index];
10     W_new ← (W·S_rsf) / S_rsf,new;
12     if S_rsf,new / S_rsf > ε then
14         break;
16 S_rsf ← S_rsf,new;
```

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 ($L_{\text{th}}$).

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 $\varepsilon$. Specifically, if the ratio of $S_{\text{rsf,new}}$ to $S_{\text{rsf}}$ exceeds $\varepsilon$, it breaks the loop and stops increasing the RSF size (lines 6 and 7).

### C. 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.

#### 1) 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 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 $A$.

#### 2) 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.

### D. 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,

$$offset_{\text{channel}} = h_c(\text{MAC}) \ \% \ (N_{List_c} - 2). \tag{8}$$

Note that $offset_{\text{channel}}$ uses a modulo operator with $N_{List_c} - 2$, 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.

### E. COLLABORATION WITH RPL

A practical embedded network is typically designed as a vertical silo where multiple layers intimately collaborate [11], [21]. To this end, we discuss how *TESLA* jointly operates

[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.

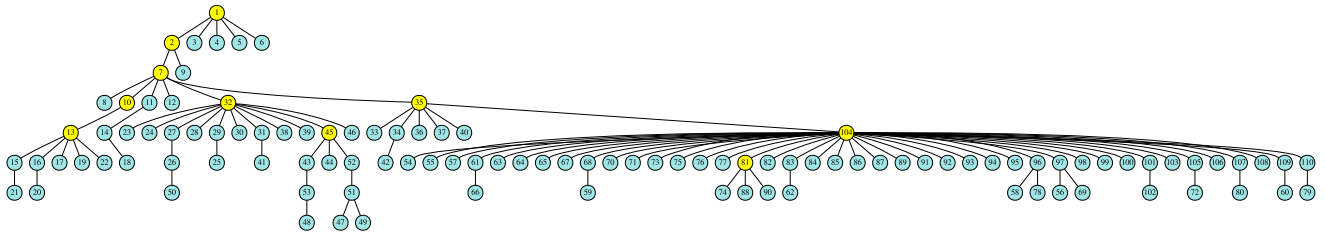


**FIGURE 6.** A snapshot of RPL topology for 110 nodes with −17 dBm of Tx power on FIT/IoT-LAB testbed in Lille.

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.

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 reinitialization [27], 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.

## V. 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.

### A. METHODOLOGY AND EXPERIMENT SETUP

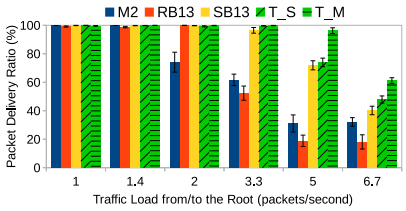
We implement *TESLA* on ContikiOS and compare it against 6TiSCH minimal scheduling (*M*), sender-based Orchestra (*SB*), and receiver-based Orchestra (*RB*), using 110 and 79 nodes on the FIT/IoT-LAB testbeds [23] in Lille and Grenoble, France, respectively. Each node features a 32-bit ARM Cortex-M3 microcontroller (STM32F103REY) and an AT86RF231 IEEE 802.15.4 radio chip. This node is representative of today's state-of-the art IoT devices [23]. 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 III, 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 IV-E.

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∼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.

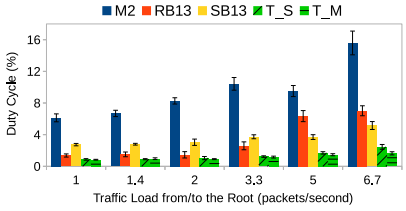
For *TESLA*, we use RSF adaptation period $T_{\text{adapt}}$ of 15 seconds, and limitation factor $\varepsilon$ is set to 1.5 to increase RSF size conservatively. In other words, RSF size cannot increase by more than 50% every $T_{\text{adapt}}$. The load threshold $L_{\text{th}}$ is 50%, and $PRR_{\text{th,low}}$ 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).

### B. IMPACT OF TRAFFIC LOAD

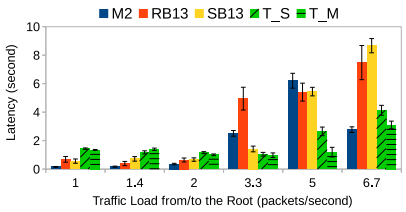
We first investigate the performance of the state-of-the-art TSCH schedules and *TESLA* with various traffic intensities on Lille testbed consisting of 110 nodes. We use slotframe size of 2 for the minimal configuration which had the highest PDR in Section III. Both receiver/sender-based Orchestra use a unicast slotframe size of 13. Each node uses transmission power of −17 dBm. Figure 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



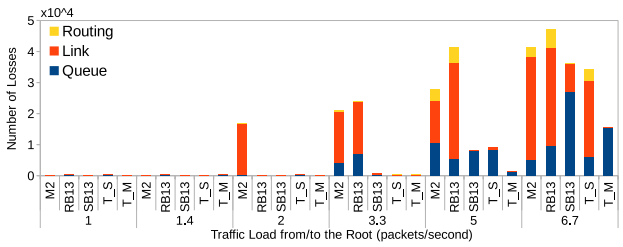

(a) End-to-end packet delivery ratio



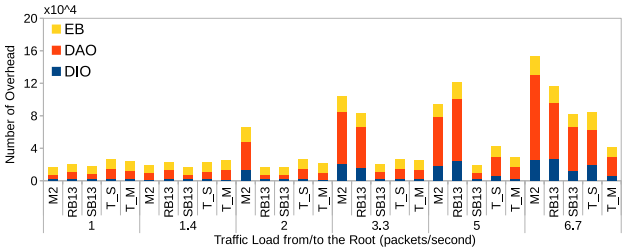

(b) Radio duty-cycle (on time)



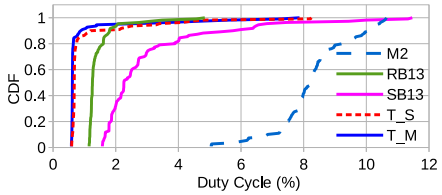

(c) End-to-end latency



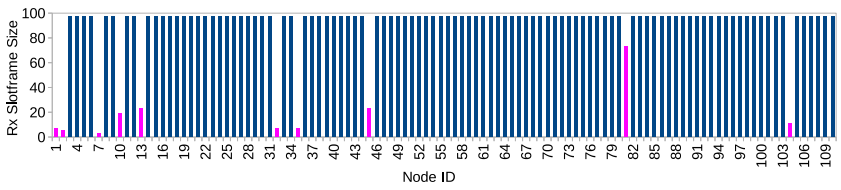

(d) Number of packet losses during 1-hour experiment




(e) Number of overhead during 1-hour experiment




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




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

**FIGURE 7. Various experimental results according to different traffic load.**

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.

### 1) RELIABILITY

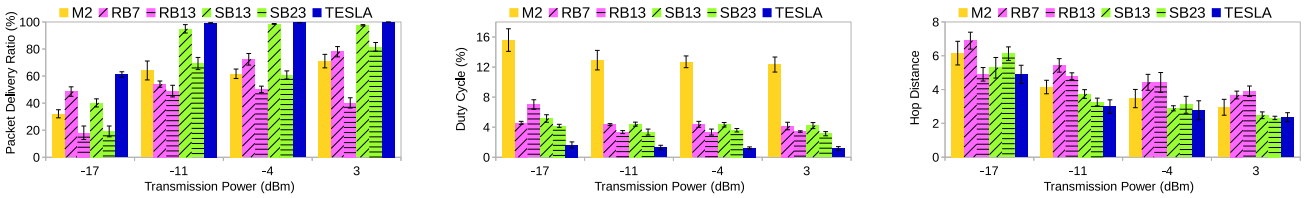
Figure 7(a) shows the average bidirectional end-to-end PDR under different traffic load. Under light traffic, all the protocols perform well with PDR over 99%. However, when the traffic load from/to the root is 2 or 3.3 packets/second, M, RB, and SB start to show performance degradation. On the other hand, *TESLA* is still capable of accommodating traffic by reducing slotframe sizes adaptively. For example, Figure 7(g) 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 Figure 6 use much smaller RSF sizes than the other nodes to resolve contention. Figure 7(a) also shows that *T_M* improves PDR of *T_S* by up to 30.1%

As presented in Figure 7(d), 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* successfully does its role at the link layer: rescuing bottleneck nodes from the contention hell, as much as possible.

### 2) ENERGY CONSUMPTION

Figure 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, 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 Figure 7(a), *T_M* reduces duty-cycle by 67.1% on average.

Figure 7(f) 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 Figure 7(b), but *TESLA* performs even better, enabling more than 90% of nodes to save their energy by 50% compared




(a) End-to-end packet delivery ratio (b) Radio duty-cycle (on time) (c) Hop distance

**FIGURE 8.** Impact of transmission power.

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. Figure 7(g) 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 Figure 7(f), *TESLA* does increase the duty-cycles of <5% bottleneck nodes compared to RB13, because they select RSF sizes shorter than 13 as shown in Figure 7(g).

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.

#### 3) END-TO-END LATENCY

Figure 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.,* ∼30% PDR). Overall, although *TESLA* is not explicitly designed for latency improvement, its contention alleviation ends up with better latency.

#### 4) 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 IV-C.1. Figure 7(e) 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 Figure 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 Figure 7(b). Figure 7(e) 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.

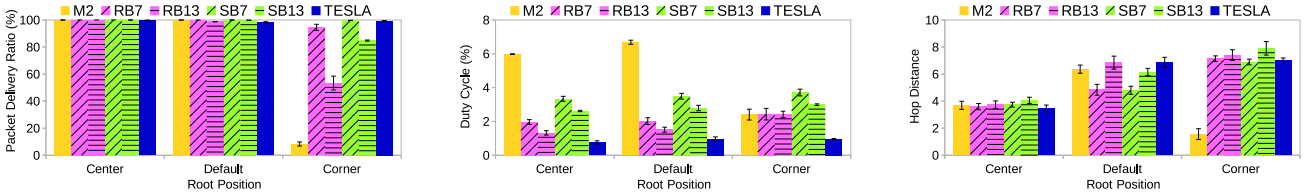
### C. IMPACT OF NETWORK TOPOLOGY

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.

#### 1) 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 V-B anticipating that higher Tx power will result in better performance, and compare *TESLA* with M2, RB7, RB13, SB13, and SB23.

Figure 8 plots end-to-end PDR, duty-cycle of radio, and average hop distance of RPL topology. As Tx power increases, Figure 8(a) shows that all the schemes except RB13 provide better PDR and Figure 8(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 Figure 8(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




(a) End-to-end packet delivery ratio (b) Radio duty-cycle (on time) (c) Hop distance

**FIGURE 9.** **Impact of root location.**

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.

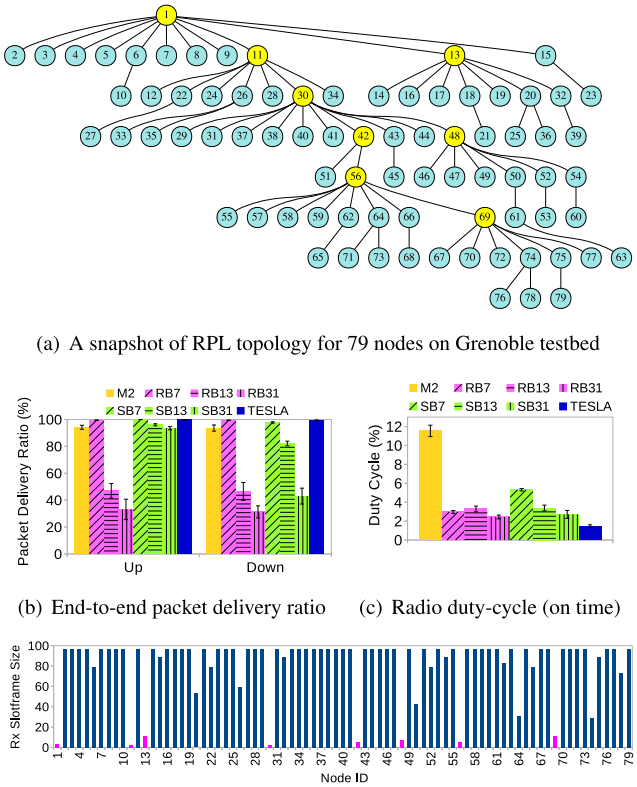
### 2) DIFFERENT POSITIONS OF THE ROOT

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 >99% PDR in Section V-B. In this experiment, *TESLA* is compared with M2, RB7, RB13, SB7, and SB13, and the results are shown in Figures 9(a) through 9(c).

When we use the center and default as the root positions, all the schemes achieve >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.

### 3) 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 of 2 packets/second for each of bidirectional traffic, and Tx power of −17 dBm.




(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 10.** **Results on the 79-node IoT-LAB Grenoble testbed.**

The experiment results are summarized in Figure 10(a). Compared to Figure 6, the routing topology of Grenoble testbed illustrated in Figure 10(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, Figure 10(b) plots upward and downward PDR, and Figure 10(c) plots radio duty-cycle. By adaptively controlling RSF sizes as illustrated in Figure 10(d), *TESLA* shows the best performance in both aspects. Specifically, Figure 10(d) and Figure 7(g) 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 V-B and V-C 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 V-A), but *TESLA* has always presented the best performances nevertheless.

### D. 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 Figure 11.

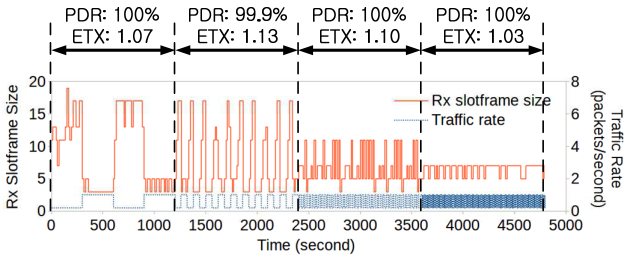



**FIGURE 11.** Time vs. Rx slotframe size.

Figure 11 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. 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 Figure 11. 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.

### E. 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 Figure 3(a) which used the same traffic load, we chose a common slotframe size '13' for sender-based and receiver-based Orchestra, which achieves >99% reliability for both, and a slotframe size '2' for the minimal schedule, which provides the highest PDR. We evaluate *TESLA* with different pairs of ($PRR_{th,low}$, $PRR_{th,up}$) and different RSF size upper bounds. For instance, T-(80%,90%)-200 indicates *TESLA* with 80% of $PRR_{th,low}$, 90% of $PRR_{th,up}$, and the upper bound of 200.

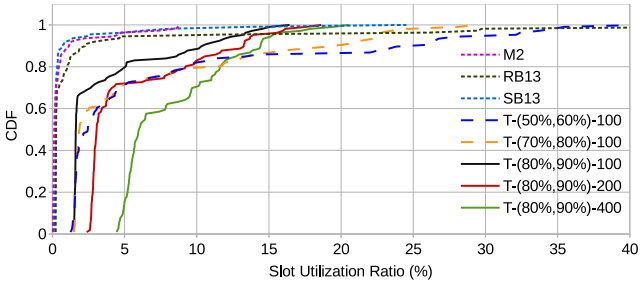



**FIGURE 12.** Slot utilization ratio with different PRR thresholds and upper bound of slotframe lengths.

Figure 12 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 >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, $PRR_{th,low}$ and $PRR_{th,up}$ 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 $PRR_{th,low}$ and $PRR_{th,up}$ 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 configuration for all the previous experiments.

## VI. RELATED WORK

Numerous duty-cycling MACs have been proposed for LLNs. Among those, asynchronous approaches [5]–[7], [19], [28], [29] 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 [8], such as TSCH, which opens the scheduling problem. Below we summarize prior work on TSCH

scheduling other than that already presented in Section II, which is categorized into centralized and decentralized approaches.

### A. CENTRALIZED TSCH SCHEDULING

Centralized scheduling is often employed in industrial scenarios [30]–[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. Palattella *et al.* [34] and [35] 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.

### B. 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]–[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]–[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].

### C. WHY TESLA?

Both centralized and decentralized approaches have not shown robust operation in real LLNs, which is the reason why Orchestra [11], an *autonomous* scheduling mechanism has been the state-of-the-art. Autonomous scheduling, however, significantly sacrifices flexibility to avoid any additional control overhead, operating inefficiently in dynamic LLN environments. *TESLA*'s novelty is that it operates on autonomous scheduling, but adds the flavor of distributed scheduling slightly and very carefully to fully realize TSCH's potential. The careful combination makes *TESLA* robust as autonomous scheduling and flexible as distributed scheduling.

## VII. 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.

## REFERENCES

[1] J. Adkins, B. Ghena, N. Jackson, P. Pannuto, S. Rohrer, B. Campbell, and P. Dutta, "The signpost platform for city-scale sensing," in *Proc. 17th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, Apr. 2018, pp. 188–199.

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

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

[4] *IEEE Standard for Low-Rate Wireless Networks*, Standard 802.15.4-2015, Apr. 2016.

[5] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. 2nd Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, Nov. 2004, pp. 95–107.

[6] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: A short preamble MAC Protocol for duty-cycled wireless sensor networks," in *Proc. ACM Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, Oct. 2006, pp. 307–320.

[7] 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 *Proc. ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, Nov. 2008, pp. 1–14.

[8] T. Watteyne, A. Mehta, and K. Pister, "Reliability through frequency diversity: Why channel hopping makes sense," in *Proc. 6th ACM Symp. Perform. Eval. Wireless AD HOC, Sensor, Ubiquitous Netw.*, Oct. 2009, pp. 116–123.

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

[10] P. Thubert, T. Watteyne, R. Struik, and M. Richardson, *An Architecture for IPv6 Over the TSCH Mode of IEEE 802.15. 4*, document draft-ietf-6tisch-architecture-10, IETF Draft, Jun. 2016, pp. 1–49.

[11] S. Duquennoy, B. A. Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *Proc. ACM Conf. Embedded Networked Sensor Syst. (SenSys)*, Nov. 2015, pp. 337–350.

[12] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "ptunes: Runtime parameter adaptation for low-power MAC Protocols," in *Proc. 11th ACM Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2012, pp. 173–184.

[13] H.-S. Kim, J. Paek, D. E. Culler, and S. Bahk, "Do not lose bandwidth: Adaptive transmission power and multihop topology control," in *Proc. 13th Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Jun. 2017, pp. 99–108.

[14] T. Lee, J. Han, M.-S. Lee, H.-S. Kim, and S. Bahk, "CABLE: Connection interval adaptation for BLE in dynamic wireless environments," in *Proc. 14th IEEE Int. Conf. Sens., Commun., Netw. (SECON)*, Jun. 2017, pp. 1–9.

[15] H.-S. Kim, H. Kim, J. Paek, and S. Bahk, "Load balancing under heavy traffic in RPL routing protocol for low power and lossy networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 4, pp. 964–979, Apr. 2017.

[16] 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," Nov. 2016, *arXiv:1811.02721*. [Online]. Available: https://arxiv.org/abs/1811.02721

[17] 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. 20558–20571, 2018.

[18] J. Ko, J. H. Lim, Y. Chen, R. Musăloiu-E, A. Terzis, G. M. Masson, T. Gao, W. Destler, L. Selavo, and R. P. Dutton, "Medisn: Medical emergency detection in sensor networks," *ACM Trans. Embedded Comput. Syst.*, vol. 10, no. 1, Aug. 2010, Art. no. 11.

[19] A. Dunkels, "The contikimac radio duty cycling protocol," SICS, Tech. Rep. T2011:13, 2011.

[20] T. Winter, *RPL: IPV6 Routing Protocol for Low-Power and Lossy Networks*, document RFC 6550, 2012.

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

[22] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. IEEE Int. Conf. Local Comput. Netw.*, Nov. 2004, pp. 455–462.

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

[24] D. Reina, M. Askalani, S. Toral, F. Barrero, E. Asimakopoulou, and N. Bessis, "A survey on multihop ad hoc networks for disaster response scenarios," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 10, Oct. 2015, Art. no. 647037.

[25] M. Erdelj and M. Król, and E. Natalizio, "Wireless sensor networks and multi-uav systems for natural disaster management," *Comput. Netw.*, vol. 124, pp. 72–86, Sep. 2017.

[26] 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," *Proc. IEEE*, vol. 104, no. 5, pp. 1013–1024, May 2016.

[27] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, *The Trickle Algorithm*, document IETF RFC 6206, Mar. 2011.

[28] D. Moss and P. Levis, "BoX-MACs: Exploiting physical and link layer boundaries in low-power networking," *Comput. Syst. Lab. Stanford Univ.*, vol. 64, no. 66, p. 120, 2008.

[29] S. Duquennoy, O. Landsiedel, and T. Voigt, "Let the tree Bloom: Scalable opportunistic routing with ORPL," in *Proc. ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, Nov. 2013, Art. no. 2.

[30] J. Lee, T. Kwon, and J. Song, "Group connectivity model for industrial wireless sensor networks," *IEEE Trans. Ind. Electron.*, vol. 57, no. 5, pp. 1835–1844, May 2010.

[31] M. Nobre, I. Silva, and L. A. Guedes, "Routing and scheduling algorithms for wireless HART networks: A survey," *Sensors*, vol. 15, no. 5, pp. 9703–9740, May 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," *Comput. Commun.*, vol. 114, pp. 84–105, Dec. 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 *Proc. IEEE Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2012, pp. 327–332.

[34] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic-aware time-critical scheduling in heavily duty-cycled IEEE 802.15.4e for an industrial IoT," in *Proc. IEEE Sensors*, 2012, pp. 1–4.

[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 IEEE802.15.4e TSCH," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3655–3666, Oct. 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 *Proc. IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 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 *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 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 sensor networks," in *Proc. Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2016, pp. 1–8.

[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 *Proc. IEEE Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Jun. 2017, pp. 3–10.

[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*. Cham, Switzerland: 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 Trans. Mobile Comput.*, vol. 8, no. 10, pp. 1384–1396, Oct. 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 Trans. Netw.*, vol. 16, no. 3, pp. 511–524, Jun. 2008.
[43] R. Soua, P. Minet, and E. Livolant, "DiSCA: A distributed scheduling for convergecast in multichannel wireless sensor networks," in *Proc. 14th IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2015, pp. 156–164.
[44] A. Aijaz and U. Raza, "Deamon: A decentralized adaptive multi-hop scheduling protocol for 6TiSCH wireless networks," *IEEE Sensors J.*, vol. 17, no. 20, pp. 6825–6836, Oct. 2017.
[45] C. Vallati, S. Brienza, G. Anastasi, and S. K. Das, "Improving network formation in 6TiSCH networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 1, pp. 98–110, Jan. 2019.
[46] D. Dujovne, L. A. Grieco, M. R. Palattella, and N. Accettura, *6TiSCH 6top Scheduling function Zero (SF0)*, document draft-ietf-6tisch-6top-sf0-01, Internet Engineering Task Force, 2016.
[47] Q. Wang, X. Vilajosana, and T. Watteyne, *6top Protocol (6P)*, document Internet-Draft draft-ietf-6tisch-6top-protocol-02, Internet Engineering Task Force, 2016.
[48] T. Chang, T. Watteyne, Q. Wang, and X. Vilajosana, "LLSF: Low latency scheduling function for 6TiSCH networks," in *Proc. IEEE Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, May 2016, pp. 93–95.
[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 Things J.*, vol. 2, no. 6, pp. 455–470, Dec. 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 Netw.*, vol. 26, pp. 88–102, Mar. 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 J.*, vol. 16, no. 2, pp. 550–560, Jan. 2016.
[52] T. P. Duy, T. Dinh, and Y. Kim, "Distributed cell selection for scheduling function in 6TiSCH networks," *Comput. Standards Inter.*, vol. 53, pp. 80–88, Aug. 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 *Proc. 19th ACM Int. Conf. Modeling, Anal. Simulation Wireless Mobile Syst.*, Nov. 2016, pp. 102–110.

**SEUNGBEOM JEONG** the B.S. degree from Seoul National University (SNU), Seoul, South Korea, in 2013, and the Ph.D. degree with the School of Electrical and Computer Engineering, SNU, in August 2019. His research interests include network and communication protocols for the mobile IoT systems and low-power wireless networks.

**JEONGYEUP PAEK** received the B.S. degree from in electrical engineering Seoul National University, in 2003, the M.S. degree in electrical engineering from the University of Southern California, in 2005, and the Ph.D. degree in computer science from the University of Southern California (USC), in 2010. He was with R&D Labs, Deutsche Telekom, Inc., USA, as a Research Intern, in 2010, and then joined Cisco Systems, Inc., in 2011, where he was a Technical Leader with the Internet of Things Group (IoTG), Connected Energy Networks Business Unit (CENBU), [formerly the Smart Grid BU]. In 2014, he was with the Department of Computer Information Communication, Hongik University, as an Assistant Professor. He has been an Associate Professor with the School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea, since 2015.

**HYUNG-SIN KIM** received the B.S. degree in electrical engineering from Seoul National University (SNU), Seoul, South Korea, in 2009, and the M.S. and Ph.D. degrees in electrical engineering and computer science (EECS) from SNU, in 2011 and 2016, respectively, all with outstanding thesis awards, where he was a Postdoctoral Researcher with the Network Laboratory (NETLAB), from March 2016 to August 2016. He is currently a Postdoctoral Researcher of EECS with the University of California at Berkeley and working at the Building Energy Transportation Systems (BETS) Group led by Prof. D. E. Culler. He received the Qualcomm Fellowship, in 2011, and the National Research Foundation (NRF) Global Ph.D. Fellowship and Postdoctoral Fellowship, in 2011 and 2016, respectively. His research interest includes the development of embedded networked systems for the Internet of Things and cyber-physical systems.

**SAEWOONG BAHK** received the B.S. and M.S. degrees in electrical engineering from Seoul National University (SNU), in 1984 and 1986, respectively, and the Ph.D. degree from the University of Pennsylvania, in 1991. He was with AT&T Bell Laboratories as a Member of Technical Staff, from 1991 to 1994, where he had worked on network management. From 2009 to 2011, he served as the Director of the Institute of New Media and Communications. He is currently a Professor with SNU. He has been leading many industrial projects on 3G/4G/5G and the IoT connectivity supported by Korean industry. He has published more than 200 technical articles and holds more than 100 patents. He is a member of the National Academy of Engineering of Korea (NAEK) and of Whos *Who Professional in Science and Engineering*. He was a recipient of the KICS Haedong Scholar Award, in 2012. He has been serving as a Chief Information Officer (CIO) for SNU, a General Chair for the IEEE DySPAN 2018 (Dynamic Spectrum Access and Networks) and for the IEEE WCNC 2020 (Wireless Communication and Networking Conference), and the Director of the Asia–Pacific Region of the IEEE ComSoc. He is the president-elect of the Korean Institute of Communications and Information Sciences (KICS), a TPC Chair of the IEEE VTC-Spring 2014, and a General Chair of JCCI 2015. He is Co-Editor-in-Chief of the *Journal of Communications and Networks* (JCN) and an Editor of the *IEEE Network Magazine*. He was on the Editorial Board of *Computer Networks Journal* (COMNET) and the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS (TWireless).