OST: On-Demand TSCH Scheduling with Traffic-Awareness

Seungbeom Jeong*, Hyung-Sin Kim†,0, Jeongyeup Paek◇, and Saewoong Bahk*
*Department of Electrical and Computer Engineering and INMC, Seoul National University, Seoul, Republic of Korea
†Nest, Google Inc., Mountain View, CA, USA
◇School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea
sbjeong@netlab.snu.ac.kr, hyungkim@google.com, jpaek@cau.ac.kr, sbahk@snu.ac.kr

***Abstract*—As the emerging Internet of Things (IoT) devices and applications flourish, demand for reliable and energy-efficient low-power wireless network protocols is surging. For this purpose, IEEE 802.15.4 standardized time-slotted channel hopping (TSCH), a promising and viable link-layer solution that has shown outstanding performance achieving over 99% reliability with low duty-cycles. However, it lacks one thing, flexibility. It is not adaptable to a wide variety of applications with varying traffic load and unpredictable routing topology due to its static timeslot scheduling. To this end, we propose *OST*, an On-demand Scheduling scheme for TSCH with traffic-awareness. In *OST*, each node dynamically self-adjusts the frequency of timeslots at *run time* according to time-varying traffic intensity. Moreover, it features on-demand resource allocation to handle bursty/queued packets in a timely manner. By doing so, *OST* aims to minimize its energy consumption while guaranteeing reliable packet delivery. We evaluate *OST* on a large-scale 72-node testbed, demonstrating that it achieves improvement of 60% in reliability and 52% in energy-efficiency compared to the state of the art.**

***Index Terms*—Low-power and Lossy Network, IEEE 802.15.4, TSCH, Dynamic Scheduling, Wireless Network Protocol**

I. Introduction

Adaptive resource scheduling in a time-synchronized wireless network (*i.e.,* TDMA) has been a long-loved research topic in the communications field. Numerous sophisticated scheduling techniques [1][2][3] have been developed considering various different aspects, such as throughput, energy efficiency, fairness, and QoS. These adaptive scheduling techniques have mainly contributed to wireless cellular systems built on very powerful base-station infrastructure. In many other cases, however, such as low power and lossy networks (LLNs), *realizing* these techniques with real embedded hardware and operating system has never been easy. Easier said than done.

LLN connects numerous low-power resource-constrained devices via multihop wireless links. With versatile applications such as environment monitoring, factory automation, and smart buildings, LLN is considered as a building block of the Internet of Things (IoT) and cyber-physical systems (CPS). Due to its strict resource constraint and time-varying topology, however, LLN has been a representative example where applying adaptive resource scheduling is difficult and inefficient. In addition to the fundamental question, "How much resource should be given to each node at each time to minimize energy expenditure while coping with traffic load?", there are several strong requirements in LLN such as simple distributed operation, collision resolution, modest control overhead, and seamless service regardless of routing topology change. Due to the non-trivial challenges, asynchronous CSMA [4][5] and (almost) fixed TDMA scheduling [6][7] have been standard practice in this regime despite their clear limitations. The latter became viable in the LLN community just recently, regarded as a breakthrough.

These simple techniques perform fairly well in traditional LLN applications which generate fixed and low-rate traffic [6][8]. As part of fast growing IoT and CPS, however, traffic demand for LLNs has increased and diversified while the energy constraint is still strict [9][10][11][12]. In addition, it has been more common that a single node runs multiple applications [13][14]. Then each LLN node should deliver different and time-varying amount of traffic depending on application behavior, physical location, node density, and routing topology. Given that neighboring technologies, such as embedded hardware [15], system architecture [16], security [17], and machine learning [18], have also evolved to support various needs of modern LLN applications, it is time to tackle the adaptive scheduling issue for LLNs to move forward. To this end, it is important to understand that, historically, success of an LLN protocol has usually come from not only a novel concept but also a fine-grained/comprehensive system design and careful implementation on resource-constrained devices.

We present a detailed design, implementation, and experimental analysis of *OST*, a novel adaptive TDMA slot scheduling technique for LLNs operating on the time-slotted channel hopping (TSCH) protocol in the latest IEEE 802.15.4 standard [19]. *OST* manages communication slots for each directional link (*i.e.,* a pair of a sender and a receiver) separately, in a distributed manner. To meet various traffic demand effectively, *OST* adopts two-step adaptive slot provisioning as follows. (1) *Periodic provisioning* allocates a periodic slot for each directional link and adapts the period according to

This research was supported by a grant to Bio-Mimetic Robot Research Center Funded by Defense Acquisition Program Administration, and by Agency for Defense Development (UD190018ID).

Saewoong Bahk and Hyung-Sin Kim are the co-corresponding authors.

[0] This work was done when Hyung-Sin Kim was in Computer Science Division at University of California, Berkeley.

the link's average traffic load. To this end, a novel *binary resource tree* is used to avoid slot collision among different directional links. (2) *On-demand provisioning* temporarily allocates additional consecutive slots for a directional link to handle an unexpected traffic burst immediately. *OST* ensures that those slots do not collide with other slots being used. Overall, *OST* timely provides 'just enough' slots for each directional link without any additional control overhead, which minimizes energy consumption without sacrificing reliability.

We implement *OST* on embedded IEEE 802.15.4 nodes using ContikiOS [20]. *OST* tightly interacts with the routing layer running RPL [21], an IPv6 standard routing protocol for LLNs, to reliably deliver routing control packets while routing topology changes dynamically. We evaluate *OST* on 72 nodes in FIT/IoT-LAB [22], a public LLN testbed. Results show that *OST* outperforms state-of-the-art TSCH scheduling techniques, improving up to 60% reliability while saving 52% energy under various traffic demands. To the best of our knowledge, *OST* is the first adaptive scheduling technique for TSCH which reliably operates with RPL routing protocol on a real-world multihop testbed.

The contributions of this work are threefold.

- We propose *OST*, the first practical and adaptive scheduling technique for TSCH demonstrated on a real-world LLN testbed. It comprises (1) periodic provisioning to adapt to time-varying average traffic load, and (2) on-demand provisioning to handle temporary traffic bursts.
- We design *OST* in a distributed manner with zero control overhead, and introduce a novel binary resource tree for collision-free slot allocation.
- We implement *OST* prototype, open the source code, and evaluate extensively on a large-scale testbed showing that *OST* outperforms state of the art in terms of reliability and energy efficiency.

II. Background

This section provides an overview of TSCH standard and the state of the art in TSCH slot scheduling.

A. IEEE 802.15.4 TSCH

As a recent link-layer protocol in IEEE 802.15.4 [19], TSCH contributes to the LLN regime in two ways: (1) It time-synchronizes the entire multihop LLN, avoiding energy waste attributed to redundant transmissions or idle listening in contrast to previous asynchronous MACs [4][5]. (2) Its frequency hopping feature makes low-power wireless links more robust to wireless interference and multipath fading [23].

As illustrated in Fig. 1, TSCH divides time into *timeslots*, each of which is typically 10 ms, long enough to exchange a data frame and an acknowledgement (ACK). Each timeslot has *absolute slot number* (ASN), representing how many timeslots have elapsed since the TSCH network began. A *slotframe* is a collection of timeslots, and is repeatedly scheduled in time. The number of timeslots in a slotframe is called slotframe size (S_{sf}). Time offset ($\mathit{offset}_{\text{time}}$) of a timeslot indicates its relative position within a slotframe (*i.e., ASN* $\%$ S_{sf}). Channel offset

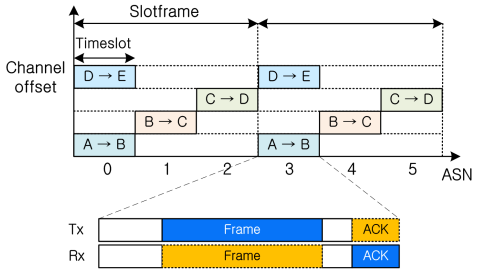


Fig. 1. An example of TSCH slotframe schedule with slotframe size of 3.

($\mathit{offset}_{\text{channel}}$) is an offset for channel selection. The channel on which a timeslot operates is calculated as,

$$\mathit{Channel} = \mathit{List}_c[(\mathit{ASN} + \mathit{offset}_{\text{channel}}) \ \% \ N_{\mathit{List}_c}] \qquad (1)$$

where List_c is a set of channels to be used and N_{List_c} is the number of its elements. As ASN progresses, each timeslot with a fixed $\mathit{offset}_{\text{channel}}$ hops over different frequency channels. Allocation of different $\mathit{offset}_{\text{channel}}$ enables distinct channels to be exploited in the same ASN, as timeslots for directional links [A→B] (*i.e.,* from sender A to receiver B) and [D→E] operate in Fig. 1. A TSCH timeslot experiences packet collision when multiple nodes try to send packets in it. In this case, CSMA backoff and retransmission mechanisms are activated to avoid collision where the unit of backoff time is timeslot.

B. The State of the Art in TSCH Timeslot Scheduling

A number of works have investigated timeslot scheduling for TSCH, which are categorized into *centralized*, *distributed*, and *autonomous* scheduling. Most centralized and distributed scheduling proposals operate adaptively at run time, but have never been successfully demonstrated on real testbeds with real routing protocols unfortunately (Section VI). We focus on autonomous scheduling techniques which are considered as the state of the art and proven to run on multihop LLN testbeds with publicly opened source codes.

Orchestra [6] is an autonomous scheduling scheme designed to operate tightly with RPL [21], regarded as a breakthrough in this regime. Orchestra decouples data transmission from control packet transmission by using multiple slotframes with dedicated channel offsets: (1) EB (Enhanced Beacon) slotframe for TSCH control packets, (2) RPL shared slotframe for RPL control packets, and (3) unicast slotframe for data packets. While timeslot schedules in the two control slotframes are globally fixed, those in the unicast slotframe change depending on neighbor relationship. In doing so, Orchestra uses neighbor information (*e.g.,* node's address) at the routing layer so that each node schedules unicast timeslots *autonomously* without any control packet exchange, which is the main difference compared to other centralized/decentralized schemes.

Orchestra provides two modes for scheduling timeslots in the unicast slotframe, sender- and receiver-based modes, where each node has only one fixed Tx or Rx timeslot per unicast slotframe, respectively, based on its MAC address as,

$$\mathit{offset}_{\text{time}} = h(\mathit{MAC}) \ \% \ S_{\text{sf}} \qquad (2)$$

where h is a shared hash function, *MAC* is the node's address, and S_{sf} is the unicast slotframe size. Since the same h is shared

throughout the network, a node can easily compute the timeslot for each neighbor using its *MAC* without control packets. As a result, each node in sender- (or receiver-) based Orchestra has one Tx (or Rx slot) with multiple Rx (or Tx) slots, as many as the number RPL neighbors, within a unicast slotframe.

ALICE [7] stepped forward. It is built on Orchestra's slotframe architecture but proposes more advanced autonomous scheduling for the unicast slotframe by allocating a separate timeslot for *each directional link* (a pair of a sender and a receiver). Thus, each ALICE node has multiple Tx and Rx slots as many as the number of RPL neighbors. ALICE also features multi-channel utilization, time-varying timeslot allocation, and early packet drop, significantly outperforming both modes of Orchestra while preserving autonomous operation.

III. Towards Practical Adaptive Scheduling

While state of the art autonomously provides a unicast timeslot with a different time offset for each node or directional link, none of them adjust *the number* of unicast timeslots at run-time, which limits their applicability in emerging LLN applications. To the best of our knowledge, the research community does not have a practical adaptive scheduling algorithm for TSCH in multihop LLNs yet (other than TESLA [24], a variant of Orchestra), which is the problem we target to tackle.

A. Need of Adaptive Scheduling in LLNs

Before going into the *OST* design, we should first answer the question: "*Why is it necessary to have an adaptive scheduling algorithm in LLNs?*" Here are some examples below:

Location: Nodes near the border router have to deliver more traffic than those at the edge due to relay burden. This asymmetry becomes more severe as the network size increases.

Node density: Physical node distribution is not necessarily even. Even with a perfectly even deployment, it is still uneven in the perspective of wireless connectivity due to various factors impacting wireless channels, such as obstacles. Nodes in a denser area are likely to deliver more traffic.

Routing topology: A routing protocol may not produce balanced routing topology even when physical node distribution is balanced. RPL is known to have load imbalance problem, which causes a few bottleneck nodes to have much more relay burden than others [25]. In addition, routing topology changes over time as wireless links fluctuate, which makes a node's traffic demand also change over time.

Application behavior: Each node in an LLN application may have different and time-varying traffic demand. In a smart building, for example, a temperature sensor generates light periodic traffic but an anemometer generates heavy continuous traffic [10]. A node's application traffic may change at run time due to event detection [12][26]. A node can run multiple applications [13][14]. A reporting strategy, *e.g.,* sending each of data packets immediately or aggregating them and sending as a batch, impacts traffic demand at run time [27].

If the number of unicast timeslots a node use for each directional link does not change flexibly according to its real-time traffic demand, the node will encounter either of the two undesirable situations: (1) wasting energy due to over-provisioned timeslots, or (2) losing data due to under-provisioned timeslots. The state-of-the-art autonomous scheduling techniques, *i.e.,* Orchestra and ALICE, are not exceptions.

B. Design Requirements for Practical Adaptive Scheduling

With the above motivation, the next question is: "*What characteristics should an adaptive scheduling algorithm have so that it can be applied to multihop LLNs practically?*" Ideally, it should provide adaptive scheduling capability while keeping all the positive characteristics of autonomous scheduling. To this end, we consider five requirements that will be used as the design elements of *OST* in Section IV.

Handling each directional link separately: A directional link is the basic unit where a transmission happens. Given that each directional link can have different, time-varying traffic demand, handling multiple links identically results in non-ideal performance. ALICE already proved the superiority of this approach under fixed-rate traffic. Adaptive timeslot scheduling should also keep this principle.

Fast response to time-varying traffic demand: An adaptive scheduling algorithm should be able to cope with unexpected time-varying traffic. When a link suddenly has many packets to send, more timeslots should be allocated as soon as possible before the sender experiences queue overflow. Otherwise, it should fast de-allocate timeslots to avoid energy expenditure.

Distributed operation with modest control overhead: The success of Orchestra and ALICE comes from their autonomous operation with zero control overhead. Adding fast and adaptive scheduling capability should not ruin this characteristic.

Timeslot collision avoidance: The flip side of distributed or autonomous scheduling is that each link's timeslot is allocated without any global information, which may incur timeslot collision among multiple links. Orchestra and ALICE exploit a hash function to reduce such a collision, which is only probabilistic and fails under high load. Adaptive scheduling should systematically manage timeslot collision.

Tight interaction with routing: It is necessary for adaptive scheduling to provide seamless service under dynamic routing topology. To this end, when a node changes its routing parent, its unicast timeslots to communicate with the new or old parent should be fast allocated or de-allocated. More importantly, for a smooth transition, control packets to setup a new parent-child relationship need to be reliably delivered, which happens *before* a timeslot is allocated for the new parent.

IV. OST Design

OST is designed to satisfy the five requirements of practical TSCH scheduling, described above. *OST* manages timeslots separately for each directional link, and *adapts* the number of unicast timeslots for each directional link. *OST* takes a two-step approach, *periodic provisioning* and *on-demand provisioning*, both of which operate in a distributed manner

with zero control overhead and nearly zero timeslot collision. In addition, the two schemes compensate each other, meeting the demand of average traffic and instantaneous traffic bursts.

A. Slotframe Architecture

OST's slotframe architecture is designed to add adaptive scheduling capability while supporting robust TSCH/RPL operation under dynamic routing topology. Its main contribution is to combine different scheduling techniques in a synergistic manner, providing a firm ground for *OST*'s adaptive operation. While maintaining the two control slotsframes in Orchestra and ALICE, *i.e.,* EB and RPL shared slotframes, with their constant channel offsets,[1] *OST* has three types of unicast slotframes for adaptive scheduling:

- **Autonomous unicast slotframe (AUS)** is for autonomous unicast with a constant periodicity and a dedicated channel offset. Each node has one AUS.
- **Periodic-provisioning Tx slotframe (PTS)** is for unicast transmission to a routing neighbor, with an elastic periodicity and channel offset. Each node has as many PTS'es as the number of its routing neighbors.
- **Periodic-provisioning Rx slotframe (PRS)** is the same as PTS except that this is used for reception.

The *autonomous* slotframe AUS has a dedicated channel offset and operates as receiver-based Orchestra,[2] which is the key for *OST*'s seamless service since it compensates weaknesses of both control slotframes and PTS/PRS. Specifically, AUS handles *unicast* TSCH/RPL control packets not only for their reliable delivery but also contention alleviation in the two control slotframes which handle all nodes' *broadcast* control packets. In addition, AUS handles data packets temporarily if a PTS-PRS pair are not yet installed for a directional link. For example, when a node changes its routing parent due to wireless environment changes, it uses AUS to send a RPL DAO packet to the new parent for setting up a new parent-child relationship, and also data packets to the parent until a PTS-PRS pair are installed for the link.

Each directional link has a PTS at the sender and a PRS at the receiver to react to its dynamic traffic demand. Specifically, an *OST* node (A) manages a PTS for a routing neighbor (B), and the PTS has one Tx slot. Node A adjusts its own PTS size dynamically according to its unicast Tx demand for its neighbor B. On the other hand, neighbor B has a PRS for node A, and the PRS has an Rx slot matching the Tx slot in the node A's PTS. Node B adjusts its PRS size according to node A's PTS size. After a parent switch in RPL, two new PTS-PRS pairs are installed for two directional links between the new parent and child, while the previous PTS-PRS pairs between the outdated parent and child are removed.

[1]As ALICE does, *OST* dedicates a channel offset for the EB slotframe but not for the RPL shared slotframe, which does not sacrifice robust operation [7].

[2]AUS inherits receiver-based Orchestra, instead of sender-based Orchestra, since the unicast slotframe of the former is more autonomous. When a node (A) needs to receive a packet from its neighbor (B), sender-based Orchestra requires A to already have B in its routing neighbor list to install an Rx slot for A. In contrast, receiver-based Orchestra enables A to receive a packet through its fixed Rx slot without even knowing B's existence.

B. Periodic Provisioning

Periodic provisioning periodically adapts unicast timeslots in a PTS and a PRS according to *average* traffic load of each directional link. By using statistical information, periodic provisioning operates relatively slowly but robustly. A sender and a receiver of a directional link take different roles, determining the *size* of the PTS/PRS and the *time offset* for a Tx/Rx slot in the PTS/PRS, respectively. The former is to meet the link's traffic demand and the latter is to avoid timeslot collision with other links. Finally, a *negotiation* procedure ensures that the two nodes agree on applying new PTS/PRS configuration. Note that the scheduling information is exchanged through existing data or ACK packets, resulting in zero additional control packets.

The operation described in the rest of the subsection applies to each of all directional links. For ease of description, we consider a directional link from a sender A to a receiver i (*i.e.,* link [$A\rightarrow i$]) where the two nodes are routing neighbors to each other, as an example.

Traffic-aware PTS/PRS Size Adaptation (Sender): PTS/PRS size of a directional link determines how much traffic the link can deliver. Given that the sender knows the exact traffic demand for the link, PTS/PRS size adaptation runs at the sender.

For a link [$A\rightarrow i$], sender A measures average traffic load towards receiver i for each period T. Let $L(A,i)$ denote the traffic load (number of unicast packets) for link [$A\rightarrow i$]. For each period T, sender A first initializes $L(A,i)$ to 0 and increases $L(A,i)$ by one whenever it enqueues a unicast packet for transmission to receiver i. At the end of the period, sender A calculates the average number of timeslots between two consecutive unicast transmissions towards receiver i, called average inter-packet slots (IPS), as

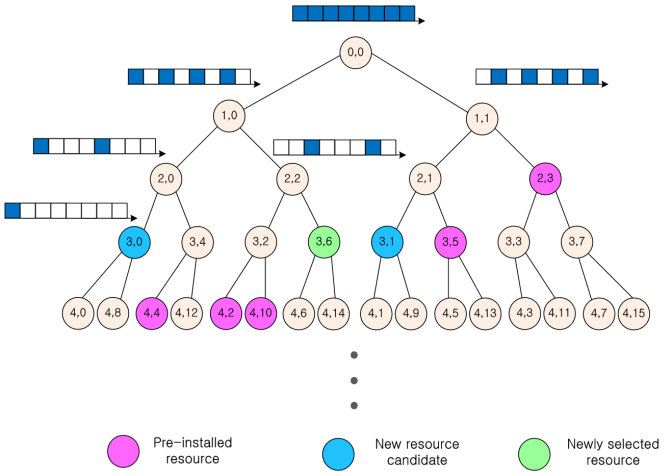
$$IPS(A,i) = n_T / L(A,i) \tag{3}$$

where n_T is the total number of timeslots in the time duration T. Then, sender A determines the size of its PTS for receiver i as $2^{N(A,i)}$, where $N(A,i)$ is a natural number satisfying $2^{N(A,i)} \leq IPS(A,i) < 2^{N(A,i)+1}$. Note that we use power of two for the PTS size to exploit a *binary resource tree*, which will be described later in this section.

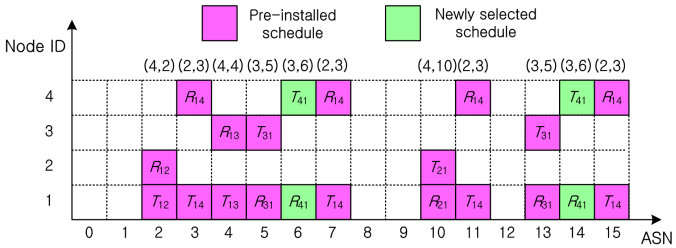
Lastly, sender A piggybacks $N(A,i)$ on unicast packets destined to receiver i in order for node i to update its PRS size for sender A accordingly.

Collision-free Time Offset Allocation (Receiver): Next, time offset ($offset_{\text{time}}$) of a Tx (or Rx) slot in a PTS (or PRS), *i.e.,* $offset_{\text{time}}(A,i)$, should be decided for the link to avoid timeslot collision. Given that packets are collided at the receiver, time offset allocation is done first by the receiver (i) by using $N(A,i)$ delivered from the sender (A).

For this purpose, each node maintains a binary resource tree as shown in Fig. 2(a), where a circle denoted with (n,t) represents a periodic timeslot with a time offset t in the slotframe of size 2^n (*i.e.,* $offset_{\text{time}}$=t and N=n). A resource (n,t) in the binary tree can be divided equally into two



(a) The binary resource tree of node 1



(b) Timeslot schedules for four nodes. T_{ab} and R_{ab} are a Tx and an Rx slot in PTS and PRS for directional link $a \rightarrow b$, respectively.

Fig. 2. An example of a binary resource tree and pre-installed schedules, where four nodes (*i.e.,* nodes 1, 2, 3, and 4) form six directional links (*i.e.,* [1→2], [1→3], [1→4], [2→1], [3→1], and [4→1]).

resources of (n+1,t) and (n+1,t+2^n). Each directional link can use one of the resource circles in the binary tree for its periodic provisioning (PTS or PRS). With the parent-child relationship of the binary tree, a resource circle always collides with any of its ancestors and partially collides with those in its subtree. Thus, even if a resource circle is not directly taken by any directional link, it is regarded *not available* when any of its ancestors or subtree circles is taken. When selecting its resource, a directional link utilizes this characteristic to avoid timeslot collision in a distributed manner.

Consider again a directional link [$A \rightarrow i$]. When receiver i detects a new $N(A,i)$ in a unicast packet sent from sender A, it selects an available resource (n,t) in its binary tree with n=$N(A,i)$, and updates its PRS with size 2^n and time offset t (=$\mathit{offset}_{\text{time}}(A,i)$). Using the resource ($n$,$t$) guarantees no timeslot collision, at least in view of the receiver i. After installing PRS with the new time offset, receiver i piggybacks $\mathit{offset}_{\text{time}}(A,i)$ on an ACK for the unicast packet from sender A.[3] Then, sender A also updates its PTS with size $2^{N(A,i)}$ and time offset $\mathit{offset}_{\text{time}}(A,i)$, which completes the periodic provisioning process for link [$A \rightarrow i$].

We exemplify this procedure using Fig. 2. In this example, we assume there are four nodes (nodes 1, 2, 3, and 4). Node 4 is the parent of node 1 whose 1-hop children are nodes 2 and 3, thus forming six directional links. Assume PTS/PRS for link [4→1] is not scheduled yet, while the other five links (*i.e.,* [1→2], [1→3], [1→4], [2→1], and [3→1]) hold the five resources, (4,2), (4,4), (2,3), (4,10), and (3,5), respectively. The pre-installed resources and their timeslot schedules are shown in Figs. 2(a) and 2(b), respectively. In Fig. 2(b), T_{Ai} (or R_{Ai}) represents a Tx (or Rx) slot in PTS (or PRS) for link [$A \rightarrow i$].

Assume node 1 gets a new $N(4,1)$=3 in a unicast packet from node 4. Then, node 1 investigates its resource tree (Fig. 2(a)) which has 2^3=8 resources with n=3. However, (3,5) is already taken by link [3→1]. Neither (3,2) nor (3,4) is available since their subtree resources (4,2), (4,4), and (4,10) are taken. In addition, (3,3) and (3,7) are not available since their common parent resource (2,3) is pre-installed. Thus, (3,0), (3,1), and (3,6) are available for the link [4→1] and (3,6) is chosen in this example. Then node 1 installs the PRS for resource (3,6) and informs node 4 of the new offset $\mathit{offset}_{\text{time}}(4,1)$=6. Lastly, node 4 installs PTS with resource (3,6), as illustrated in Fig. 2(b).

Negotiation Procedure: Since we have two adaptive mechanisms for PTS/PRS size and time offset, operating at sender A and receiver i, respectively, a *negotiation* procedure is necessary for both nodes to be on the same page. Specifically, it aims to handle two cases of failures (receiver and sender sides) of the periodic provisioning.

At the receiver side, if there is no available resource with n=$N(A,i)$ in receiver i's resource tree when receiver i detects a new $N(A,i)$ from a packet sent by sender A, it maintains the previous PRS and Rx slot (since sender A still has the previous PTS and Tx slot) and piggybacks an invalid notification on the ACK to notify sender A of the denial. Then sender A retries with $N(A,i)$ increased by one and repeats the process until receiver i accepts with a valid time offset.

Sender-side allocation failure occurs after receiver i successfully installs the PRS. Although receiver i did its best to allocate a collision-free time offset $\mathit{offset}_{\text{time}}(A,i)$ based on the current status of its resource tree, A's resource tree may have different status. Then $\mathit{offset}_{\text{time}}(A,i)$ may not be available to sender A. In this case, A removes the previous PTS and Tx slot (because i already updated PRS). Then, when sending a next unicast packet to i, A sets a bit indicating that the resource is not available, letting i select another available $\mathit{offset}_{\text{time}}(A,i)$. Importantly, this packet is sent on AUS, given that A's PTS does not exist. This negotiation process is repeated until sender A agrees on $\mathit{offset}_{\text{time}}(A,i)$ given by receiver i.

C. On-demand Provisioning

Since periodic provisioning operates based on statistical information, it cannot react to unexpected traffic bursts immediately. To fill this gap, *OST* allows a directional link to allocate additional *temporary* timeslots when there are queued packets at the sender, called on-demand provisioning.

Consider a directional link [$A \rightarrow i$] again. When sender A sends a unicast packet (p_1) on a timeslot (ASN=t_1) towards receiver i, it checks whether there is another unicast packet

[3]We designed to use Information Element (IE) field in the IEEE 802.15.4 frame to piggyback N (or $\mathit{offset}_{\text{time}}$) in unicast packets (or ACKs).

(p_2) for i in the Tx queue. If so, before transmitting p_1, A creates a subsequent timeslot schedule (STS) by looking into all its slotframes. An STS consists of $size_{\text{STS}}$ bits, where the k-th bit indicates whether A has a reserved schedule on the timeslot with ASN=t_1+k by any slotframe. If the timeslot is reserved, the k-th bit is set to 1. Otherwise, it is set to 0. Then, the STS is piggybacked on p_1 with its frame pending bit set.

If receiver i receives p_1 in ASN=t_1 and detects the frame pending bit set, it also generates its own STS starting from ASN=t_1+1, and compares it with the STS of A (piggybacked on p_1). Receiver i finds the earliest 0 bit in both STS'es of nodes A and i (namely, m-th bit). This means neither A nor i has any timeslot scheduled in ASN=t_1+m. Then, receiver i piggybacks the earliest bit's position m on the ACK for p_1, and schedules a temporary Rx slot in ASN=t_1+m. Upon receiving the ACK in ASN=t_1, sender A allocates a temporary Tx slot in ASN=t_1+m immediately, which enables the link [$A \rightarrow i$] to deliver p_2 in ASN=t_1+m without timeslot collision with any slotframe in both nodes A and i (including PTS/PRS).

This on-demand provisioning occurs recursively until sender A has no packet to send to receiver i. For example, if A has more packets for i other than p_2, it generates another STS starting from ASN=t_1+m+1 and piggybacks this when sending p_2 in ASN=t_1+m. Given that STS and m are piggybacked on unicast and ACK packets, respectively, no additional control packet is required.

Let's take Fig. 2(b) as an example to clarify the on-demand provisioning procedure. Assume, in ASN=4, node 1 is about to send a unicast packet (p_1) to node 3 while having another unicast packet (p_2) in its Tx queue for node 3. Then, it piggybacks its STS of "11100110" ($size_{\text{STS}}$ is assumed 8 bits) on p_1. Then, node 3 receives p_1 in ASN=4 and compares the piggybacked STS with its STS ("10000000"). As a result, m becomes 4, and node 3 installs a Rx slot in ASN=4+4=8. After m (=4) is delivered to node 1 on the ACK for p_1 by node 3, node 1 allocates a Tx slot in ASN=8. At last, they will send and receive p_2 in ASN=8.

D. Multi-channel Operation

Although the periodic provisioning (PP) and on-demand provisioning (ODP) avoid timeslot collision among RPL neighbors, collision among physical neighbors (not necessarily RPL neighbors) may still occur. This happens more often when traffic is heavy since *OST* almost fully utilize unicast timeslots. To enlarge network capacity, *OST* uses multiple channels for the same timeslot. For PP, the channel offset of a directional link's PTS/PRS is computed as

$$\mathit{offset}_{\text{channel,PP}} = h(\mathit{ASFN} + \mathit{MAC}_{\text{i}}) \ \% \ (N_{\mathit{List}_c} - 2). \quad (4)$$

OST uses the absolute slotframe number (*ASFN*) defined as $\lfloor \mathit{ASN}/S_{\text{PTS/PRS}} \rfloor$ [7], where $S_{\text{PTS/PRS}}$ is the slotframe size of PTS/PRS. MAC_{i} is the address of receiver i of a directional link. By hashing *ASFN*, each directional link's channel offset changes psuedo-randomly every slotframe, which prevents a specific link from suffering continuous timeslot collision.

Given that ODP does not use the slotframe concept, the channel offset for an on-demand unicast timeslot is set by hashing *ASN* instead of *ASFN* as,

$$\mathit{offset}_{\text{channel,ODP}} = h(\mathit{ASN} + \mathit{MAC}_{\text{i}}) \ \% \ (N_{\mathit{List}_c} - 2). \quad (5)$$

It is also changed every timeslot, preventing repetitive collision. Note that both equations use a modulo operator (%) with $N_{\mathit{List}_c}-2$, instead of N_{List_c}, since *OST* dedicates two channel offsets for the EB slotframe and AUS for robust operation.

V. Performance Evaluation

This section evaluates *OST* against the receiver-based Orchestra (*RB*), sender-based Orchestra (*SB*), and ALICE (*AL*) in several dimensions to show its effectiveness.

A. Methodology and Experiment Setup

We implement *OST* on ContikiOS and open the code.[4] We use Contiki-RPL at the routing layer. Lengths of the EB and RPL shared slotframes are set to 397 and 41, respectively, for all the schemes. For *OST*, AUS size is 47, T is 15 seconds, and $size_{\text{STS}}$ is 8 bits. We set the upper limit of N as 8, leading to a maximum PTS/PRS size of 256, in order for each directional link to have a Tx/Rx slot per 2.56 seconds at least.

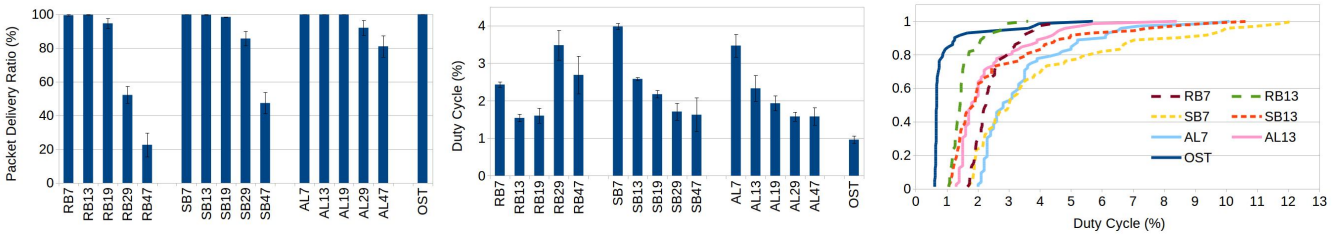
We use 72 nodes on the FIT/IoT-LAB open testbed [22] in Grenoble, which are deployed uniformly in two line linear topology. Each device uses Tx power of -17 dBm, resulting in a 8-hop network. We use four channels, known to be least interfered with Wi-Fi, for frequency hopping: 15, 20, 25, 26 (N_{List_c}=4). Each experiment lasts 1 hour, and the results are averaged over 3$\sim$5 runs for each case. An error bar represents 95% confidence interval. Each node uses a maximum of 8 retransmissions per hop, and queue size of 16 packets. Application payload is 59 bytes carried in UDP/IPv6 datagrams over 6LoWPAN, reaching frame size of 109 bytes.

B. Impact of Slotframe Size

We first investigate the impact of slotframe sizes on RB/SB/AL, and compare them with *OST*. We generate bidirectional traffic, 3/71 packets/sec for both upward and downward traffic to/from the root from/to each node. Since there are 71 non-root nodes, aggregate traffic load from/to the root is 3 packets/sec for each direction.

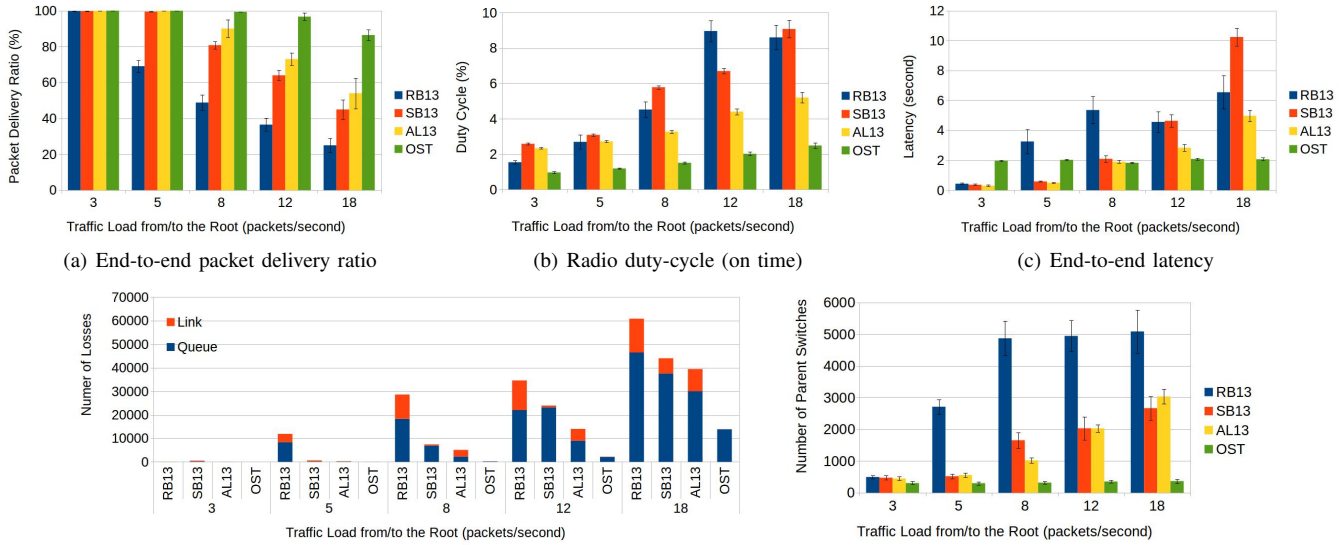
Figs. 3(a) and 3(b) plot average end-to-end packet delivery ratio (PDR) and radio duty cycle for 72 nodes, where the number presented with each scheme is the slotframe size employed. The figures show RB/SB/AL have a trade-off according to the slotframe size. PDR is degraded while duty cycle improves with the slotframe size. As an exception, duty cycle of RB decreases and increases again due to excessive channel contention with a single Rx slot within a slotframe, showing significant PDR degradation. Within a slotframe, since SB allocates more Rx slots (as many as the number of RPL neighbors) than RB with a single Rx slot, SB has better reliability with higher energy consumption than RB. AL improves PDR of RB and SB by allocating more timeslots

[4] https://github.com/seungbumz/OST



(a) End-to-end packet delivery ratio (b) Radio duty-cycle (on time) (c) Empirical CDF of duty-cycle

Fig. 3. Various experimental results according to different slotframe size.



(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 parent switch during 1-hour experiment

Fig. 4. Various experimental results according to different traffic load.

through scheduling per directional link. Nevertheless, AL achieves lower duty cycle than SB since it mitigates channel contention with time-varying and multi-channel scheduling.

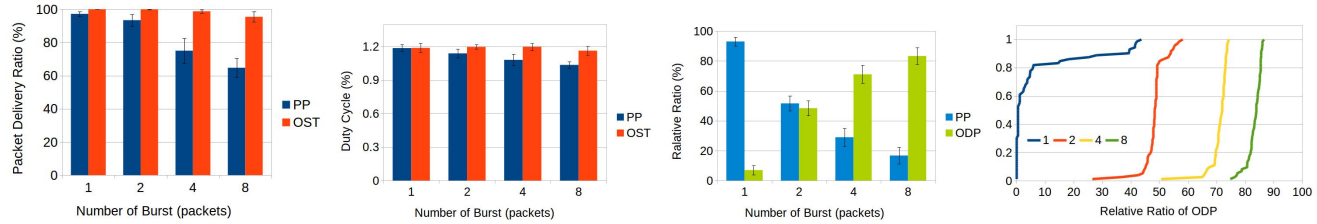
OST outperforms all the others in terms of reliability and energy efficiency for any configuration of their slotframe sizes. Fig. 3(c) shows an empirical CDF of duty cycle among all 72 nodes for *OST* and RB/SB/AL with the slotframe size of 7 and 13, all of which achieve >99% PDR. In RB/SB/AL, a longer slotframe achieves lower energy consumption, and all their nodes have duty cycle of >1%. Meanwhile, *OST* drastically reduces duty-cycles, where more than 80% of nodes have <1%. Given *OST* also has perfect PDR, the result proves that most of nodes in RB/SB/AL are over-provisioned with a short slotframe size (*e.g.,* 7 or 13) which is required only by a few bottleneck nodes. On the other hand, *OST* does not over-provision since each node adjusts the number of timeslots for each directional link based on required traffic for itself.

C. Impact of Traffic Load

We investigate the impact of traffic load by changing the aggregate rate from/to the root as in Fig. 4, from 3 to 18 packets/sec. We compare *OST* with RB13, SB13, and AL13, all of which showed perfect PDR in the previous experiment.

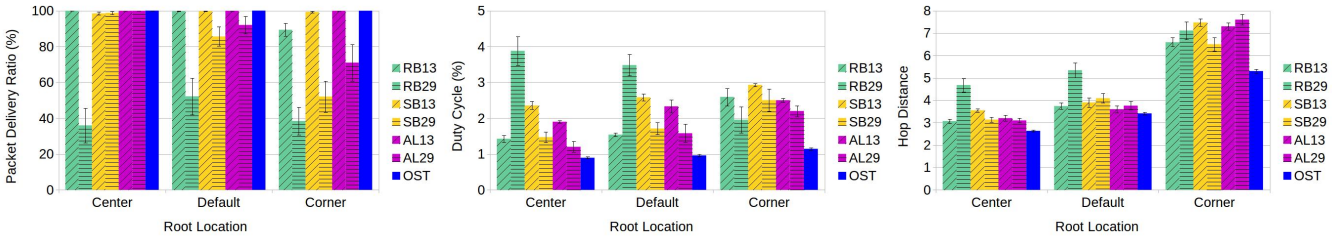
In Figs. 4(a) and 4(b), as the traffic rate increases, PDR and duty cycle for the static scheduling schemes start to degrade rapidly. On the other hand, *OST* maintains much higher PDR and lower duty cycle even under heavy traffic load. For example, compared to AL13, *OST* shows PDR improvement of 60% and energy saving of 52% in the highest traffic intensity.

To analyze reliability of the protocols, Fig. 4(d) shows the number of *link losses* and *queue losses* during the experiment. Note that each RB node has one Rx slot and multiple Tx slots within a slotframe while each SB node has one Tx slot and multiple Rx slots. This makes RB and SB provide fewer Rx and Tx opportunities, respectively. Consequently, in RB, neighbors of a bottleneck node contend for a single Rx slot, thus suffering from link losses first and then queue losses due to CSMA backoff. On the other hand, SB mainly experiences queue losses due to lack of Tx slots. ALICE allocates one Tx slot and one Rx slot within a slotframe for a directional link, thus having the same number of Tx/Rx slots. Nevertheless, as traffic gets intense, ALICE also suffers from collisions and CSMA backoff in the vicinity of bottlenecks as RB, leading to link and queue losses. Link losses make RPL network unstable, showing growth of the number of parent switches as in Fig. 4(e). Then, RPL attempts to restore the RPL topology by using more control packets, but it aggravates contention more. However, *OST* avoids channel contention by enabling each neighbor of a node to have non-overlapped dedicated Tx slots, having few link losses and stable RPL network as shown in Figs. 4(d) and 4(e). Compared to RB/SB/AL, *OST* also reduces the number of queue losses considerably by allocating more Tx/Rx slots for a directional link with heavy traffic.



(a) End-to-end packet delivery ratio (b) Radio duty-cycle (on time) (c) Average relative ratio of PP/ODP (d) Empirical CDF for ratio of ODP

Fig. 5. Performance of *OST* with burst traffic.



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

Fig. 6. Various experimental results according to different topology.

Fig. 4(c) presents the average end-to-end latency for upward and downward traffic. When the traffic load is low, *OST* exhibits the longest delay since each link has PTS/PRS with long periodicity (up to 256) to save energy. However, even though the traffic load increases, *OST* maintains the latency around 2 seconds by adopting shorter PTS/PRS and prompt on-demand allocation. Meanwhile, RB/SB/ALICE show increasing latency with traffic intensity due to lack of Tx/Rx resources and CSMA backoff. Resultingly, *OST* ends up with the lowest end-to-end delay from the traffic load of 8 packets/sec, despite not explicitly designed for latency.

D. Impact of Burst Traffic and On-Demand Provisioning

We analyze the contribution of *on-demand provisioning* in handling traffic bursts, with a burst comprising B packets where B varies from 1 to 8. The root sends downward bursts to 71 non-root nodes in a round-robin fashion with a rate of $5/B$ bursts/sec, and each node generates an upward burst with a rate of $5/(B*71)$ bursts/sec, leading to same aggregate bidirectional traffic. For example when B is 1, traffic pattern is the same with the 5 packets/sec experiment in Section V-C.

We first evaluate the effect of on-demand provisioning (ODP) by implementing *OST* without ODP (*i.e.,* periodic provisioning only, namely PP) and comparing it with *OST* (*i.e.,* PP+ODP). Fig. 5(a) plots PDR of PP and *OST* according to varying B. We found that PP alone shows quite reliable result (97.2% PDR) when B=1. However, as the traffic becomes more bursty, PDR of PP is reduced gradually. We discovered that >99% of PP's packet losses are attributed to queue overflow. As shown in Fig. 5(a), PP's reliability is restored when ODP is combined, becoming true *OST*. Since there is little energy consumption for channel contention between RPL neighbors thanks to non-overlapped timeslot scheduling in PP and *OST*, their duty-cycles in Fig. 5(b) are determined by the number of successfully forwarded packets, thus having a similar pattern with PDR in Fig. 5(a).

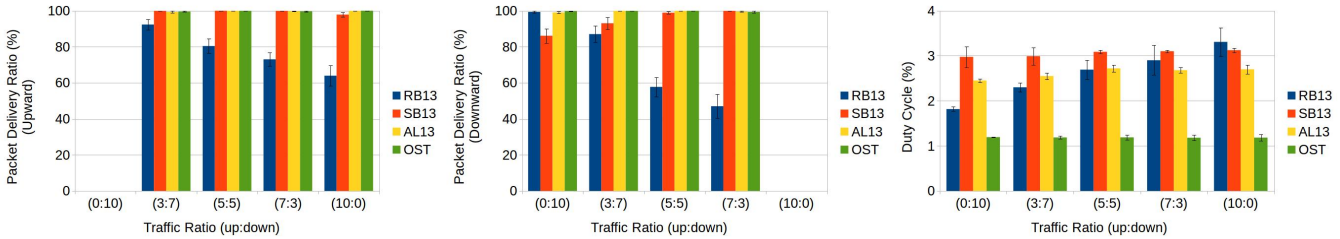
Next, we look at whether successful unicast Tx of *OST* occurred in PP or ODP. To this end, we plot two different relative ratios of successfully transmitted packets by PP and ODP in Fig. 5(c). When B=1, *OST* mainly uses PP, since few packets are queued. As B grows up, however, *OST* ends up relying on ODP to transmit queued packets. In B=8, the relative ratio of ODP is 83.3% while that of PP is just 16.7%.

Fig. 5(d) plots an empirical CDF of ODP's relative ratio among 72 nodes. Although its average relative ratio in B=1 is rather low (7%) as shown in Fig. 5(c), more than 10% of nodes have ODP's ratio of >30%. We identified that all of those are bottleneck nodes (including the root) each of which has at least 30 descendant nodes in RPL topology. Even when B is 1, they often receive a burst of traffic and forward it using ODP. As B increases, all nodes utilize ODP more by serving queued packets, recursively whenever possible.

E. Impact of Topology

In Fig. 6, we change the root position to create different topology. We use three different nodes as the root in the 72-node testbed; *Default* indicates the experiment result using the same root with all the previous experiments. *Center*/*Corner* represent the experiments using alternative nodes at the center/corner of the testbed as the root, respectively. We set aggregate traffic rate from/to the root to 3 packets/sec which is used in Fig. 3. In this experiment, *OST* is compared against RB/SB/AL with slotframe sizes 13 and 29.

RB/SB/AL still show a trade-off between reliability and energy efficiency in Figs. 6(a) and 6(b); High PDR is achieved with high duty cycle by using a short slotframe size. However, *OST* always maintains >99% PDR with remarkably low duty cycle regardless of network topology. As the root changes its location from *Center* to *Corner*, average hop distance increases as depicted in Fig. 6(c). However, *OST* has shorter hop distance in RPL network than any other scheme. This is because *OST* keeps the RPL network stable, even for



(a) End-to-end 'upward' packet delivery ratio (b) End-to-end 'downward' packet delivery ratio (c) Radio duty-cycle (on time)

Fig. 7. Various experimental results according to different ratio of bidirectional traffic.

bottleneck nodes, by avoiding timeslot collisions between RPL neighbors and mitigating collisions between non-RPL neighbors with time-varying channel offsets. Consequently, each node freely chooses a neighboring node at the optimal routing cost (*i.e.,* the shortest hop) as its routing parent.

F. Impact of Bidirectional Traffic Ratio

Throughout the previous evaluation, we have used equal traffic rates for upward and downward traffic. Now, we use 5 different traffic ratios for upward and downward traffic, (0:10), (3:7), (5:5), (7:3), and (10:0). Each element denotes aggregate traffic rate in packets/sec for each direction. For example, (7:3) represents an experiment with 7 and 3 packets/sec of aggregate upward and downward traffic, respectively. Even though we vary the ratio, total traffic in the network remains unchanged.

The results are summarized in Fig. 7. SB shows the lowest downward PDR in (0:10) since it suffers from queue overflow due to lack of Tx slots, mainly in the root node which needs to forward all the downward traffic. As the portion of upward packets increases, both directional PDRs in RB are degraded because of channel contention with insufficient Rx slots. Meanwhile, AL and *OST* always show perfect bidirectional PDRs thanks to their directional-link based scheduling. However, *OST* always has the lowest duty cycle. Interestingly, *OST*'s duty cycle remains constant with different traffic ratios, since it is affected by not traffic pattern (upward-centric or downward-centric) but total amount of traffic for both directions based on timeslot assignment with traffic awareness.

VI. Related Work

We summarize representative centralized/decentralized TSCH scheduling methods, which proves why inflexible autonomous scheduling has been state of the art in this regime.

Centralized TSCH scheduling attempts to construct an optimal schedule with a global view of the network. AMUS [28] assumes nodes with shorter hop distance from the root have more traffic to forward, and allocates Tx slots accordingly. However, since traffic load is also affected by other factors, it brings out inefficient scheduling. Traffic-aware scheduling algorithm (TASA) [29] builds a schedule based on traffic load offered by each node. It allocates timeslots and channel offsets based on network topology to maximize parallel Tx. However, Sempere-Paya *et al.* [30] showed its poor performance on a real multihop testbed due to its slow schedule adaptation and significant control overhead. Elsts *et al.* [31] proposed a hybrid use of dedicated and shared timeslots. However, their scheme needs larger number of channels than the network diameter, which is not applicable to a channel limited network. Hashimoto *et al.* [32] proposed a method to add and share timeslots for retransmissions under the constraints of reliability and delay. However, it handles neither collision by multiple contenders nor energy waste due to excessive idle listening.

As a decentralized TSCH scheduling, SF0 [33], defined by 6TiSCH [34], adapts the number of slots according to the current allocation and demands from the neighbors. SF0 uses the 6top protocol (6P) [35] to add/delete timeslots but does not specify which timeslots should be reallocated. As a SF0-based scheme, low latency scheduling function (LLFS) [36] daisy-chains the timeslots in a multi-hop path to reduce end-to-end latency. However, SF0-based approaches require significant 6P overhead to schedule each link. 6TiSCH also defines TSCH minimal configuration [37], a simple autonomous scheduling where all nodes share an identical and fixed slotframe with a single timeslot for both Tx/Rx. Vallati *et al.* [38] improves it by adapting the slotframe size according to the number of packets. However, it handles a single schedule for all links, suffering packet collisions and redundant overhearing. In DeTAS [39], each node aggregates its own bandwidth request with those from its children, and delivers it to its parent recursively. Then, it schedules timeslots sequentially from the sink to leaf nodes. However, if a packet is lost, all the subsequent schedules are wasted. In some algorithms [40][41][42][43], nodes locally reallocates colliding schedules between interfering radio links. However, negotiation overhead increases substantially with node density, and they have been evaluated only in small scale.

In contrast, success of *OST* comes from enabling adaptive scheduling without sacrificing any desirable characteristic of autonomous scheduling through a novel two-step scheduling concept and distributed protocol design.

VII. Conclusion

We presented *OST*, an on-demand TSCH scheduling scheme with traffic awareness. Contrary to static scheduling, in *OST*, each node dynamically self-adjusts the period of timeslots at *run time* according to time-varying traffic intensity. Moreover, *OST* provides on-demand allocation to handle packet bursts in a timely manner without queue overflow. We demonstrated that *OST* achieves up to 60% of reliability improvement with 52% of energy saving, compared to the state of the art. With its open implementation, and as the first adaptive scheduling verified on a large-scale testbed, *OST* can serve as a new *de facto* timeslot scheduling for TSCH in LLNs.

REFERENCES

[1] D. Gesbert, S. G. Kiani, A. Gjendemsjo, and G. E. Oien, "Adaptation, coordination, and distributed resource allocation in interference-limited wireless networks," *Proceedings of the IEEE*, vol. 95, no. 12, pp. 2393–2409, 2007.

[2] G. Schembra, "A resource management strategy for multimedia adaptive-rate traffic in a wireless network with TDMA access," *IEEE transactions on wireless communications*, vol. 4, no. 1, pp. 65–78, 2005.

[3] G. Holland, N. Vaidya, and P. Bahl, "A rate-adaptive MAC protocol for multi-hop wireless networks," in *International conference on Mobile computing and networking*, 2001, pp. 236–251.

[4] 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.

[5] A. Dunkels, "The contikimac radio duty cycling protocol," Swedish Institute of Computer Science, Tech. Rep., 2011.

[6] 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.

[7] S. Kim, H.-S. Kim, and C. Kim, "ALICE: autonomous link-based cell scheduling for TSCH," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2019.

[8] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *ACM conference on embedded networked sensor systems (SenSys)*, 2009.

[9] 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.

[10] S. Kumar, M. P. Andersen, H.-S. Kim, and D. E. Culler, "Performant TCP for Low-Power Wireless Networks," in *NSDI*, 2020.

[11] D. Jung, Z. Zhang, and M. Winslett, "Vibration analysis for iot enabled predictive maintenance," in *IEEE International Conference on Data Engineering (ICDE)*, 2017, pp. 1271–1282.

[12] 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.

[13] A. Levy, B. Campbell, B. Ghena, D. B. Giffin, P. Pannuto, P. Dutta, and P. Levis, "Multiprogramming a 64kb computer safely and efficiently," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[14] J. Adkins, B. Ghena, N. Jackson, P. Pannuto, S. Rohrer, B. Campbell, and P. Dutta, "The signpost platform for city-scale sensing," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2018.

[15] N. Jackson, J. Adkins, and P. Dutta, "Capacity over capacitance for reliable energy harvesting sensors," in *ACM International Conference on Information Processing in Sensor Networks (IPSN)*, 2019.

[16] H.-S. Kim, M. P. Andersen, K. Chen, S. Kumar, W. J. Zhao, K. Ma, and D. E. Culler, "System architecture directions for post-soc/32-bit networked sensors," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2018.

[17] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT," in *USENIX Security Symposium*, Aug. 2019.

[18] M. Meyer, T. Farei-Campagna, A. Pasztor, R. Da Forno, T. Gsell, J. Faillettaz, A. Vieli, S. Weber, J. Beutel, and L. Thiele, "Event-triggered natural hazard monitoring with convolutional neural networks on the edge," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2019.

[19] "IEEE Standard for Low-Rate Wireless Networks," April 2016.

[20] 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.

[21] 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.

[22] 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.

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

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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 *International Conference on Communication Systems and Networks (COMSNETS)*, 2016.

[33] 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.

[34] 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.

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

[36] 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.

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

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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