Slot-size Adaptation and Utility-based Packet Aggregation for IEEE 802.15.4e Time-Slotted Communication Networks

Hongchan Kim, Geonhee Lee, Juhun Shin, Jeongyeup Paek, and Saewoong Bahk

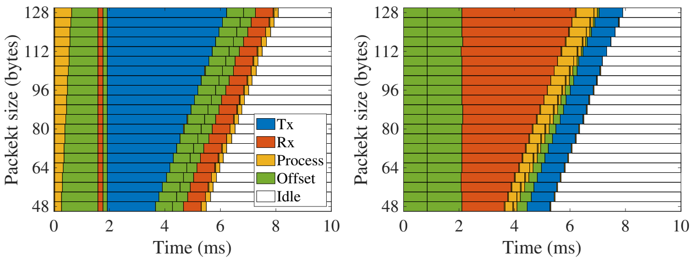
***Abstract*—Time-slotted communication is used in countless protocols and systems. IEEE 802.15.4e time-slotted channel hopping (TSCH) is one of those examples which has shown remarkable performances in the literature. However, time-slotted systems have one fundamental drawback: *a slot* is predefined to be sufficiently long enough to accommodate one exchange of a maximum-sized packet and an acknowledgment. If most packets in the system are far smaller than the maximum, a significant amount of residue time within each slot is wasted, leading to corresponding loss in effective data rate. To address this fundamental challenge, we propose *utility-based adaptation of slot-size and aggregation of packets* (*ASAP*) which reduces wasted time in slotted systems to improve throughput and latency. *ASAP* consists of two orthogonal approaches: *slot-length adaptation* (*SLA*) dynamically adapts timeslot length to actual packet size distribution, and *utility-based packet aggregation* (*UPA*) transmits aggregated packets in multiple consecutive slots to maximize slot utility. We case-study *ASAP* in the context of TSCH. We implement *ASAP* on real embedded devices, and evaluate on large-scale testbeds using state-of-the-art schedulers to demonstrate a 2.21x improvement in throughput as well as a 78.7% reduction in latency.**

***Index Terms*—Time-slotted communication, residue time, throughput, latency, time-slotted channel hopping (TSCH)**

I. Introduction

Time-slotted communication has been long-loved by various communication protocols and systems. It synchronizes the network, divides time into slots, and a communication transaction occurs within each timeslot. In contrast to asynchronous random access approaches such as pure ALOHA and carrier sense multiple access (CSMA) [1], time-slotted communication can easily coordinate communication or better allocate resources between devices given that there is a coordinator/master to manage the synchronization. Therefore, it can improve reliability and throughput by preventing collisions and interference due to uncoordinated transmissions, and also reduce energy waste attributed to redundant rendezvous attempts or idle listening.

IEEE 802.15.4e time-slotted channel hopping (TSCH) [2], a MAC protocol for low-power and lossy network (LLN), is



(a) TSCH Tx slot **(b)** TSCH Rx slot

Fig. 1: Time usage breakdown of a regular TSCH Tx and Rx slot (of 10 ms) according to packet size, including ACK in the opposite direction. There are a lot of *idle time* (in white color) within a slot.

one of those examples. It has been designed to satisfy the growing demand for more reliable and energy-efficient LLNs in emerging Internet of Things (IoT) applications such as industrial IoT [3]–[8], in-vehicle IoT [9], [10], environmental monitoring [11]–[14], home IoT [15], and health IoT [16]–[18]. TSCH brings the benefits of time-slotted communication to LLN, and its channel hopping allows the network to become more robust to external interference or multi-path fading through frequency diversity. As such, TSCH has shown remarkable performance in the literature [19]–[26].

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

A naive approach would be to shorten the time-slot length. But to what size? Obviously, a size smaller than the packets would break the basic assumptions of slotted operation. What if the system has mix of packet sizes from small to big? Furthermore, what if the application running on the network (and thus the packet sizes) changes after configuring the slot size? or if multiple applications are running concurrently?

H. Kim, G. Lee, J. Shin, and S. Bahk are with the Department of Electrical and Computer Engineering and INMC, Seoul National University, Seoul 08826, Republic of Korea (e-mail: {hckim, ghlee, jhshin}@netlab.snu.ac.kr; sbahk@snu.ac.kr) J. Paek is with the Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea (e-mail: jpaek@cau.ac.kr). S. Bahk and J. Paek are the co-corresponding authors.

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2024-2021-0-02048) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation), also by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A5A1027646), and also by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A4A5034130).

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

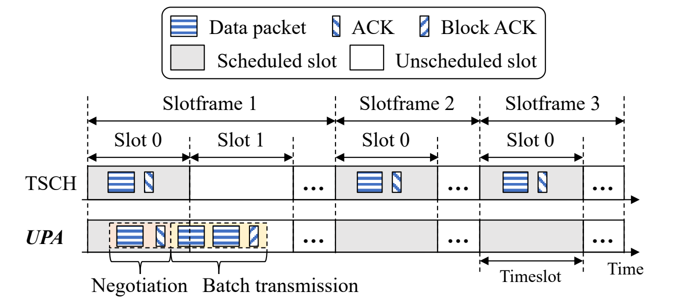
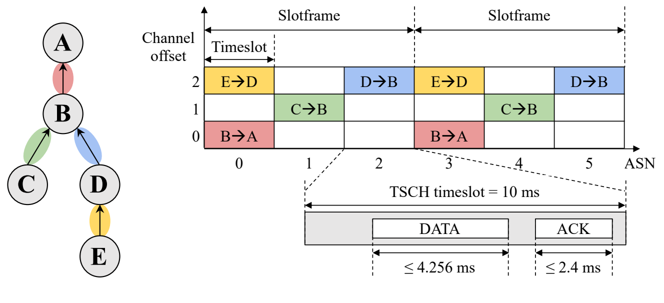


Fig. 2: Example of *UPA*'s packet aggregation and batch transmission (bottom) compared to the default slotted operation of TSCH (top).



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

Fig. 3: Example of TSCH operation.

These questions cannot be answered using a pre-configured fixed size slot.

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

Fig. 2 exemplifies how *UPA* operates compared to the default TSCH. We define *'slot utility'* as the ratio of number of packets transmitted to the number of slots used for those transmissions. Assume that a Tx node has three packets to send to a Rx node, and one slot is scheduled for the link in each slotframe as in Fig. 2. The default TSCH behaviour would be to send three packets using three slots (slot utility of one) over three slotframes[2]. Instead in *UPA*, the Tx and Rx nodes first exchange one packet and an ACK through which they negotiate whether additional batch transmission is doable and can be advantageous in terms of slot utility. If batch transmission is determined to be beneficial, then the Tx node sends the remaining two packets immediately in a batch, and the Rx node acknowledges the result with a *block ACK*. In this way, *UPA* finishes transmissions of three packets in two slots within a slotframe, thus increasing the slot utility from 1 to 1.5 (3 packets over 2 slots) and reducing latency to less than half (within one slotframe, see Fig. 2). By reducing the number of slots required per packet, this increase in slot utility allows acquiring additional resources for other transmissions and delivering multiple packets faster in terms of both datarate (less residue time) and latency (less slotframes).

We case-study *ASAP* in the context of TSCH. We implement *ASAP* on real embedded IEEE 802.15.4 devices using Contiki-OS [27], and evaluate on multiple large-scale topologies in the FIT/IoT-LAB public LLN testbed [28] with various state-of-the-art TSCH schedulers. Results show that *ASAP* improves throughput and reduces latency of TSCH network by up to 2.21x and 78.7% respectively. Given that our approaches are not limited to TSCH but can be applied to other time-slotted communication protocols and systems, we believe *ASAP* can be a generic solution to the fundamental challenge of time-slotted communication.

[2]We provide background on TSCH and its terminology in §II-A.

Our contributions can be summarized as follows.

- We present an analysis of time usage breakdown in TSCH through real measurements to demonstrate the time wastage in slotted communication.
- We propose *ASAP*, consisting of *SLA* and *UPA*, to address the problem in the context of TSCH. *SLA* adjusts timeslot length network-wide according to the distribution of packet sizes observed in the network, and *UPA* aggregates packets and transmits them in a batch over multiple consecutive slots based on slot-utility to minimize time waste due to slotted communication.
- We evaluate *ASAP* via a proof-of-concept implementation on real embedded devices in multiple sizeable public testbeds, and compare it against recent state-of-the-art approaches to demonstrate significant improvement in throughput and latency.

The remainder of this paper is organized as follows. We present the background and motivation in §II, and discuss related work in §III. We present the design of *ASAP* in §IV, and evaluate *ASAP* in §V. Finally, §VI concludes the paper.

II. Background and Motivation

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

A. Time-Slotted Channel Hopping (TSCH)

TSCH is a MAC protocol standardized in IEEE 802.15.4e [2] that combines time-slotted communication and channel hopping. TSCH synchronizes the network, and devices communicate in a time-slotted manner to improve reliability and energy efficiency. Channel hopping enables TSCH to be robust to external interference and fading through channel diversity.

As Fig. 3 illustrates, TSCH divides time into *timeslots*. The length of a timeslot is typically set to 10 ms, sufficiently long enough for exchanging a maximum-sized (128 Bytes) frame and an ACK of up to 70 Bytes. Each timeslot has an *absolute slot number (ASN)*, which is initialized to zero at the beginning of the network and then sequentially incremented. A set of timeslots constructs a *slotframe*, which is repeated in time and functions as a unit of TSCH schedule. The number of timeslots in a slotframe is called slotframe length (L_{SF}). Then, *time offset* (t_o) is a relative position of a specific timeslot within a slotframe calculated as,

$$t_o = mod(ASN, L_{\mathrm{SF}}). \quad (1)$$

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

$$Channel = List_{\mathrm{c}}[mod(ASN + c_{\mathrm{o}}, sizeof(List_{\mathrm{c}}))] \quad (2)$$

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

B. TSCH scheduling

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

Centralized schedulers [29]–[33] use global network information (e.g., topology, link quality, etc.) to construct a schedule, and distribute it to the network for each node to use. Although they can potentially optimize the schedule based on a global view of the network, collecting network information and disseminating the schedule requires a huge communication overhead. With **distributed schedulers** [22]–[24], [34]–[36], every node has a scheduling function that determines its schedule based on local information or negotiation with neighboring nodes. Although distributed schedulers can lower control overhead compared to centralized schedulers, they still suffer from non-negligible overhead. Lastly, **autonomous schedulers** [19], [21], [25] determine a schedule according to predetermined rules (e.g., a hash function) and self-obtainable information (e.g., node ID) in each node. Therefore, autonomous schedulers have the advantage of requiring no additional control overhead. However, since each node schedules itself autonomously and independently, autonomous schedulers inherently cannot consider the schedules or situations of neighboring nodes [37].

It is important to note that the problem we are trying to solve is orthogonal to how the schedulers operate. Regardless of the type of the scheduler, residue time may always exist under fixed-size time-slotted operation. Therefore, without loss of generality, we select ALICE [21], a state-of-the-art autonomous scheduler among many, to case-study *ASAP*. In ALICE, nodes autonomously determine their own unicast schedule by utilizing routing information (i.e., the node IDs of neighboring nodes) and a hash function that is shared between all nodes in the network. For instance, the time offset of a unicast schedule for a directional link from node *A* to node *B* can be calculated as;

$$t_o(A, B) = mod(Hash(\alpha{\cdot}ID(A){+}ID(B){+}ASFN), L_{\mathrm{SF}}). \quad (3)$$

The coefficient α is used to differentiate traffic direction, while $ID(x)$ denotes the node ID of x. *ASFN* stands for *absolute slotframe number*, which is initialized to zero at the beginning of the network and then incremented as the slotframe progresses. *ASFN* makes the outcome of the hash function distinctive every slotframe, leading to time-varying resource assignment. This time-varying resource assignment prevents repetitive overlaps between resources or repeated disruption from interference that can occur with fixed location of resources. Since each node can obtain the node ID of its neighboring nodes through the routing layer, ALICE no longer requires the exchange of control messages for scheduling.

C. Problem and Motivation

Here we further analyze the time usage breakdown of a TSCH slot in Fig. 1. To obtain the result, we measured the execution time of various TSCH operations within a slot while exchanging packets between two IEEE 802.15.4 M3 devices with varying packet sizes from 48 to 128 bytes. ACK length is set to 20 bytes, the typically used size in Contiki-OS. We enabled the clear channel assessment (CCA) feature before packet transmission on the Tx side, and classified the operation times into five categories as follows:

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

Intuitively, smaller packet size leads to decrease in Tx/Rx times, and thus an increase in idle time within a slot (Fig. 1). Idle time ratio reaches almost 50% when the packet size is at its minimum. Even with the largest packet size, $\sim$20% of the timeslot is still wasted. This is because the TSCH timeslot length is set to accommodate a maximum-sized frame and also a maximum-sized ACK. Since the typical ACK size is far smaller than the maximum, significant idle time exists even when a max-sized packet is sent. In summary, $20-50\%$ of bandwidth is wasted in TSCH due to the fixed slot length, and therefore, reducing the idle/residue time is crucial to increase network throughput. This is the problem that we aim to address in this work. Finally, although we case-study in the context of TSCH, the problem we solve is not limited to TSCH but is a common problem of time-slotted systems. This observation motivated us to the design of *ASAP*, which we believe will apply as a general solution to many time-slotted systems.

III. Related Work

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

One approach is to utilize temporary resources in addition to the ones allocated by the scheduler. TSCH standard [2] defines the *default burst transmission* (DBT) method that allows using an additional slot temporarily allocated between a sender and a receiver if the sender has non-zero packets in its queue toward the receiver and there is no schedule in the next slot for both nodes. On-demand provisioning in OST [23] is similar to DBT, but it checks the schedule of a predetermined number of subsequent slots and uses the earliest available slot among them to enable additional transmission.

Another approach is to adjust the amount of resources adaptively based on traffic load. In OST [23], each node measures the traffic load on its links, and allocates non-overlapping exclusive resources for each link accordingly through negotiation between nodes. A3 [25] divides a slotframe into multiple zones and assigns resources to each zone, with the number of active zones adjusted adaptively according to the measured traffic load on the links. However, none of the aforementioned approaches address the residue time problem within a timeslot.

There are other approaches that utilize residue time rather than reducing it. In [38], [39], methods are proposed to mitigate collisions in shared slots and improve throughput by performing additional collision avoidance during the residue time. However, these approaches are only effective in shared slots and the benefits may be limited when traffic load is not heavy enough to cause collisions.

Attempts have been made to aggregate application layer payloads from multiple nodes into a single frame in order to increase throughput, taking advantage of the fact that packet sizes in TSCH networks are typically short [9], [30], [40]–[42]. However, aggregation size is limited to a single frame and is applicable only for the same application-layer destination whereas *ASAP* is a link-layer solution that allows aggregating larger number of frames and slots. The idea of frame aggregation for throughput enhancement has been used in other domains as well. For example in Wi-Fi [43], aggregated MAC protocol data unit (A-MPDU) is a well-known approach that combines multiple data frames into a larger frame, improving transmission efficiency. Similar attempts have also been made in Bluetooth [44]. However, their focus is on reducing the header/control overhead for transmission efficiency rather than reducing residue time within a timeslot.

IV. *ASAP* Design

ASAP consists of two independent and orthogonal methods, *slot length adaptation* (*SLA*) and *utility-based packet aggregation* (*UPA*), that can be enabled individually. *SLA* is responsible for reducing residue time by dynamically adjusting the slot length based on the current packet size distribution. *UPA* further reduces residue time by aggregating and transmitting multiple packets over consecutive slots if beneficial in terms of slot utility. *SLA* and *UPA* complement each other, achieving better time efficiency for slotted communication.

TABLE I: Definitions of symbols for *SLA*

Symbol	Definition
T_{det}	Period at which *SLA* coordinator determines slot length
T_{adv}	Advertisement duration of new slot length
t_{dec}	Moment when *SLA* coordinator decides to change slot length
t_{act}	Moment when the new slot length is activated and applied
$T^{\text{slot}}_{\text{default}}$	Default slot size
T^{slot}_{SLA}	Slot size adjusted by *SLA* coordinator
$T^{\text{TXN}}_{\text{default}}$	Default transaction time
$T^{\text{TXN}}_{\text{UC}}$	Transaction time for a unicast packet (including ACK)
$T^{\text{TXN}}_{\text{BC}}$	Transaction time for a broadcast packet
B_{UC}	Size of a unicast packet in bytes
B_{BC}	Size of a broadcast packet in bytes
B_{ACK}	Size of an ACK in bytes
R	Data rate of the PHY layer
k	Percentile at which the *SLA* coordinator determines slot length
$T^{\text{TXN}}_{\text{UC,k}}$	Transaction time for a k-th percentile sized unicast packet
$T^{\text{TXN}}_{\text{BC,k}}$	Transaction time for a k-th percentile sized broadcasat packet
T^{TXN}_{Δ}	Difference between $T^{\text{slot}}_{\text{default}}$ and T^{slot}_{SLA} (reducible slot length)
α, β	Coefficients to determine T_{adv}
h	The network depth
L_{EB}	EB slotframe size

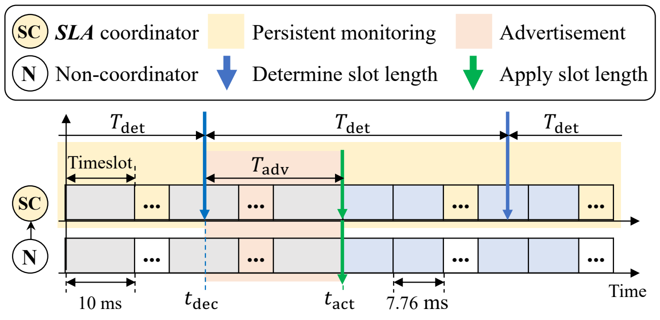


Fig. 4: *SLA*'s slot length adaptation process.

A. *SLA* Design

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

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

By repeating this procedure, *SLA* adapts the slot length to the packet size distribution in the network at run-time to reduce wasted residue time. It leverages the fact that any time-slotted system would require some form of coordinator or master that synchronizes the network. We describe the details of *SLA* using an illustration in Fig. 4. Table I lists the definitions of symbols used in *SLA*.

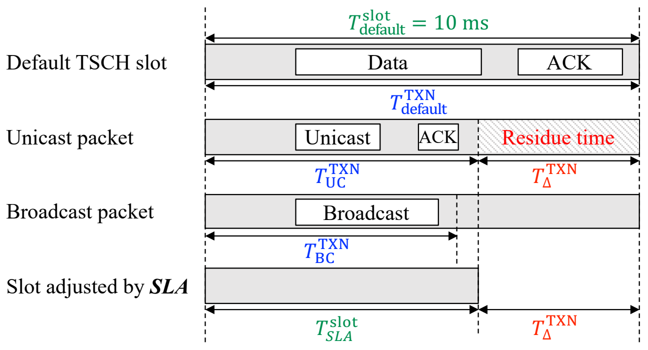

$T^{\text{slot}}_{\text{default}} = 10$ ms
Default TSCH slot
Data
ACK
$T^{\text{TXN}}_{\text{default}}$
Unicast packet
Unicast
ACK
Residue time
$T^{\text{TXN}}_{\text{UC}}$
T^{TXN}_{Δ}
Broadcast packet
Broadcast
$T^{\text{TXN}}_{\text{BC}}$
Slot adjusted by ***SLA***
T^{slot}_{SLA}
T^{TXN}_{Δ}

Fig. 5: Illustration of *SLA*'s slot length decision for adjustment.

Persistent monitoring: To determine an appropriate slot length for the entire network, the *SLA* coordinator needs to know the distribution of packet sizes generated throughout the network. For this purpose, we take advantage of the fact that on a multi-hop TSCH network, RPL (*IPv6 Routing Protocol for LLN*) [45], [46] is the de-facto standard routing protocol, and the TSCH coordinator operates as an RPL root[3]. Almost all types of packets flow in and out through the RPL root under most scenarios (e.g., data collection, command dissemination, DIO/DAO, etc.), and thus the TSCH coordinator can acquire the packet size distribution of the entire network by observing the packets it sends and receives. Therefore, we assign the role of *SLA* coordinator to the TSCH coordinator and have it monitor the packet size distribution necessary for determining the slot length.

In our *SLA* design, the *SLA* coordinator quantizes the observed packet sizes into 8-byte interval bins, and uses the longest length within each bin as a representative value. This is to manage the packet size distribution in a memory-efficient manner and to prevent excessively fine-grained adjustment of the slot length. To determine for how long (T_{adv}) to advertise the new slot length and when to apply it (t_{act} in Fig. 4), the *SLA* coordinator also needs to know the depth of the multi-hop network (details explained shortly). For this purpose, our *SLA* design utilizes the time-to-live (TTL) field in the IPv6 header. The *SLA* coordinator monitors the number of hops each packet traverses to derive the maximum hop distance, and use it as the network depth. However, it is also possible to derive it directly from RPL's routing table.

Determination of slot length: Based on the packet size distribution collected from persistent monitoring, *SLA* coordinator determines an appropriate slot length periodically every T_{det}. Fig. 5 illustrates how this is done. An appropriate slot length is determined based on the estimated *transaction time* required for sending and receiving a packet. The default transaction time ($T^{\text{TXN}}_{\text{default}}$) in TSCH is 10 ms, long enough for exchanging a pair of maximum-sized packet and ACK. However, the actual required transaction time varies depending on the type (i.e., unicast or broadcast) and length of the packet. For instance, a unicast packet requires a transaction time of $T^{\text{TXN}}_{\text{UC}}$ for both packet and ACK, whereas a broadcast packet requires $T^{\text{TXN}}_{\text{BC}}$ without an ACK, calculated as,

$$T^{\text{TXN}}_{\text{UC}} = T_{\text{proc}} + T_{\text{offset}} + (B_{\text{UC}} + B_{\text{ACK}})/R \qquad (4)$$
$$T^{\text{TXN}}_{\text{BC}} = T_{\text{proc}} + T_{\text{offset}} + B_{\text{BC}}/R \qquad (5)$$

where B_{UC}, B_{BC}, and B_{ACK} represent the total bytes in each corresponding packet type, including both the packet body and control fields such as the preamble. R is the data rate of the PHY layer, which is 250 kbps for IEEE 802.15.4 PHY. T_{proc} and T_{offset} denote the total process and the total offset time within a slot, respectively, as defined in §II-C. We note that the sum of T_{proc} and T_{offset} is nearly constant (Fig. 1) regardless of whether *SLA* is used. Therefore, the transaction time can be determined by adjusting B_{UC} and B_{ACK}, or B_{BC} according to the desired packet size.

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

Advertising and applying the new slot length: When the *SLA* coordinator decides to change the slot length, the new slot length must be advertised and applied throughout the network. *SLA* updates the slot length of all nodes *at once*. This design choice considers that synchronization in a TSCH network occurs across multiple hops. *Gradually changing the slot length may break the synchronization, as nodes may have to maintain synchronization with nodes with different slot lengths at the same time.* For this simultaneous activation, the *SLA* coordinator must determine when the new slot length should be applied. As illustrated in Fig. 4, if the *SLA* coordinator decides to change the slot length at t_{dec} and determines the advertisement duration T_{adv}, then the endpoint of this advertising period becomes the activation time as, $t_{\text{act}} = t_{\text{dec}} + T_{\text{adv}}$. Then, the new slot length and its activation time are advertised throughout the entire network during T_{adv}, and all nodes apply the new slot length simultaneously when t_{act} is reached.

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

[3]DODAG root in RPL's terminology

TABLE II: Definitions of symbols for *UPA*

Symbol	Definition
T_{N}	Time taken for negotiation phase
$T^{\text{N}}_{\text{proc}}$	Process time (T_{proc}) in negotiation phase
$T^{\text{N}}_{\text{offset}}$	Offset time (T_{offset}) in negotiation phase
T_{B}	Time taken for batch transmission phase
$T^{\text{UC}}_{\text{B}}(i)$	Time taken to send the i-th unicast packet in batch transmission phase
$T^{\text{B,UC}}_{\text{proc}}(i)$	Process time (T_{proc}) for the i-th unicast packet transmission in batch transmission phase
$T^{\text{B,UC}}_{\text{offset}}(i)$	Offset time (T_{offset}) for the i-th unicast packet transmission in batch transmission phase
T^{BA}_{B}	Time taken to send the block ACK in batch transmission phase
$T^{\text{B,BA}}_{\text{proc}}$	Process time (T_{proc}) for block ACK in batch transmission phase
$T^{\text{B,BA}}_{\text{offset}}$	Offset time (T_{offset}) for block ACK in batch transmission phase
$B_{\text{UC}}(i)$	Size of the i-th unicast packet in bytes
B_{ACK}	Size of the ACK for the first negotiation packet in bytes
B_{BA}	Size of the block ACK in bytes
R	Data rate of the PHY layer
n	Number of packets in Tx node's queue
$T_{UPA}(n)$	Time required for transmitting entire n packets
$S_{UPA}(n)$	Slots required for transmitting entire n packets

length and the activation time via EBs also transmit EBs including this information during T_{adv}.

Considering this advertisement procedure, *SLA* coordinator determines T_{adv} to be long enough to ensure that all nodes in the network can acquire the information. Specifically, the *SLA* coordinator uses the network depth obtained during the persistent monitoring. Assume that the network depth is h hops and every TSCH node has resources to send EBs at a period of L_{EB} slots. Then, considering that all nodes transmit EB at every resource for EB transmission during the advertisement period, all nodes can receive at least one EB after $L_{\text{EB}} \cdot h$ slots. Based on these observations, the *SLA* coordinator calculates the required advertisement duration T_{adv} as,

$$T_{\text{adv}} = (\alpha \cdot L_{\text{EB}} \cdot h + \beta) \times (\textit{current slot length}) \text{ [ms]} \quad (6)$$

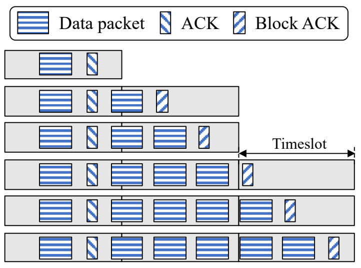
where α and β are coefficients considering re-transmissions [4].

B. UPA Design

UPA presents another approach to minimize residue time by aggregating multiple packets and transmitting them over multiple (but fewer) consecutive slots to fully utilize the time within the slots. fewer slots, it would be possible to effectively reuse the previously wasted residue time in each slot and increase effective data rate. To realize this, *UPA* requires two phases as shown in the earlier example Fig. 2: (1) utility-based negotiation and (2) batch transmission. Table II lists the definitions of symbols used in *UPA*.

Slot utility-based negotiation: *UPA* aggregates packets only when there is a gain in terms of slot utility. To achieve this, the

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



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

Fig. 6: Example of impact of *UPA*'s packet aggregation on slot utility and bit sequence.

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

Specifically, the Tx node constructs a *slot-utility information block* (SIB) to represent the slot utility according to the number of packets to be aggregated as exemplified in Fig. 6[5]. SIB consists of two bit sequences (the two 8-bit sequences shown on the right table in Fig. 6). The first 8-bit sequence in blue contains the total number of packets pending in the Tx node, and the second 8-bit sequence in red represents *the increments in the number of required slots according to the number of aggregated packets*. For example, when the number of queued packets is six in Fig. 6 (bottom row), the first 8-bits in blue simply encode six in binary (00000110_2). The second 8-bit sequence in red (01010000) is interpreted from left-to-right:

- '0' at the first bit means that no additional slot is needed to send one packet. (one slot total, utility=1).
- '1' at the second bit means that one additional slot is needed to send two packets (two slots total, utility=1).
- '0' at the third bit means that no additional slot is needed to send three packets (two slots total, utility=1.5).
- '1' at the fourth bit means that one additional slot is needed to send four packets (three slots total, utility=1.33).
- '0' at the fifth bit means that no additional slot is needed to send five packets (three slots total, utility=1.67).
- '0' at the sixth bit means that no additional slot is needed to send six packets (three slots total, utility=2).

This encoding allows the receiver to calculate the slot utility for all cases and choose the number of packets that maximizes slot utility subject to it's RX buffer size limit (or whatever policy that it wishes to enforce).

[5]The example in Fig. 6 is simplified to a uniform packet size, but the actual implementation reflects the individual (possibly distinct) packet sizes during slot utility calculation

In detail, the increment in the number of required slots (the second 8-bit sequence in red) according to the number of aggregated packets and their packet sizes is calculated as follows: Assume total of n packets in the Tx node's queue. The time it takes for the negotiation to complete (T_{N}), i.e., the time duration until the Tx node receives an ACK from the Rx node for the first packet, is calculated as,

$$T_{\mathrm{N}} = T^{\mathrm{N}}_{\mathrm{proc}} + T^{\mathrm{N}}_{\mathrm{offset}} + (B_{\mathrm{UC}}(1) + B_{\mathrm{ACK}})/R \tag{7}$$

where $T^{\mathrm{N}}_{\mathrm{proc}}$ and $T^{\mathrm{N}}_{\mathrm{offset}}$ denote the total process and offset times within the negotiation phase. $B_{\mathrm{UC}}(i)$ represent the total bytes of i-th packet, including both the packet body and the control fields such as preamble.

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

$$T^{\mathrm{UC}}_{\mathrm{B}}(i) = T^{\mathrm{B,UC}}_{\mathrm{proc}} + T^{\mathrm{B,UC}}_{\mathrm{offset}} + B_{\mathrm{UC}}(i)/R \tag{8}$$

where $T^{\mathrm{B,UC}}_{\mathrm{proc}}$ and $T^{\mathrm{B,UC}}_{\mathrm{offset}}$ denote the total process and offset times while sending the i-th packet.

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

$$T^{\mathrm{BA}}_{\mathrm{B}} = T^{\mathrm{B,BA}}_{\mathrm{proc}} + T^{\mathrm{B,BA}}_{\mathrm{offset}} + B_{\mathrm{BA}}/R \tag{9}$$

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

$$T_{\mathit{UPA}}(n) = T_{\mathrm{N}} + T_{\mathrm{B}} = T_{\mathrm{N}} + \sum_{i=2}^{n} T^{\mathrm{UC}}_{\mathrm{B}}(i) + T^{\mathrm{BA}}_{\mathrm{B}} \tag{10}$$

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

$$S_{\mathit{UPA}}(n) = \lceil T_{\mathit{UPA}}(n)/(\mathit{current\ slot\ length})\rceil. \tag{11}$$

We note that B_{ACK}, B_{BA}, and all the process and offset times in Eqs. (7) to (11) are predefined constants. Therefore, the Tx node can estimate the slot utility according to the number of aggregated packets only by considering $B_{\mathrm{UC}}(i)$. When aggregating packets, if the number of required slots (i.e., $S_{\mathit{UPA}}(n)$) increases, the bit of the second bit sequence corresponding to the packet count is set to one; otherwise, it is set to zero. Then, SIB is sent by piggybacking on the triggering (first) unicast packet.

In the example of Fig. 6, five packets are pending in addition to the triggering packet, and the number of slots required increases by one when sending one or three additional packets. Accordingly, Tx node can construct a bit sequence representing the increase in the number of slots as indicated by the red bits. Then the number of aggregated packets is represented in the first bit sequence as shown by the blue numbers.

Upon receiving the SIB, the Rx node can calculate the slot utility for each number of aggregated packets. Rx node restores $S_{\mathit{UPA}}(n)$ for each value of n from the received SIB, and calculates the slot utility as $n/S_{\mathit{UPA}}(n)$. Then, the Rx node selects the number of packets that maximizes the slot utility within its buffer limit (l), by solving the following problem:

$$\underset{2\le n\le l}{\operatorname{argmax}} \frac{n}{S_{\mathit{UPA}}(n)} \tag{12}$$

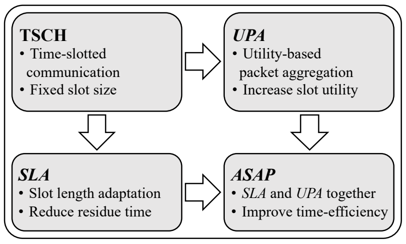
The Rx node chooses zero as a means of denying the request for packet aggregation if there is no way to improve slot utility or when there are insufficient space available in the Rx buffer to accommodate the aggregated packets. In such circumstances, the Rx node selects zero without the need to solve Eq. (12). Then the selected number of packets to aggregate is conveyed to the Tx node through ACK. Since the negotiation information are piggybacked on unicast and ACK packets, no additional control packet is required.

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

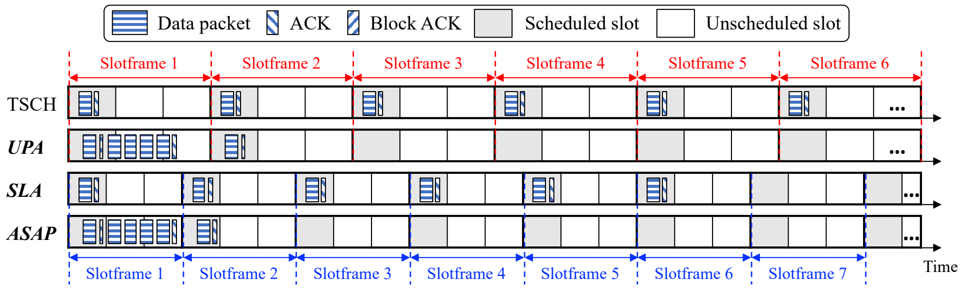
However, since the batch transmission of *UPA* is performed beyond the slot originally scheduled for the Tx and Rx nodes of *UPA*, it may overlap with the schedules of other links. If the overlapped schedules are executed simultaneously, a collision will occur. The collision can reduce the number of packets successfully delivered by *UPA*, decreasing the slot utility. Therefore, we enable non-participants to avoid collision with *UPA*'s batch transmission through CCA. Specifically, we modify and apply the CCA operation proposed in [47] to fit the operation procedure and timing of *UPA*.

C. ASAP: SLA and UPA Together

Now we explain how the two independent and orthogonal methods, *SLA* and *UPA*, coexist and function together as *ASAP*. We use Fig. 7 for this purpose. As indicated in Fig. 7a, *SLA* and *UPA* can be independently applied to TSCH. Specifically, *SLA* reduces residue time through slot length adjustments, while *UPA* focuses on enhancing slot utility through utility-based packet aggregation. Moreover, when both methods are simultaneously implemented in *ASAP*, they enhance the overall time efficiency of the network. Fig. 7b demonstrates the transmission of six packets between a Tx node and an Rx node through TSCH, *SLA*, *UPA*, and *ASAP*. We assume that one slot is scheduled for the link in each slotframe. The slot size is 10 ms in TSCH and *UPA*, while it is adjusted to 8 ms by *SLA* in *SLA* and *ASAP*. The Tx node has six packets in its queue, and we assume the Rx node determines to accept five in a batch considering its Rx buffer availability. In the case of TSCH, the transmission of these packets spans six slotframes. *SLA* also requires six slotframes



(a) Functional flow of *ASAP*



(b) An example of transmitting six packets in TSCH, *UPA*, *SLA*, and *ASAP*

Fig. 7: Examples of the operation of TSCH, *UPA*, *SLA*, and *ASAP*.

to transmit all packets, similar to TSCH. However, the use of shorter slot length in *SLA* results in a reduced overall transmission time compared to TSCH. On the other hand, *UPA* efficiently aggregates five packets in the first slotframe, achieving a slot utility of 1.67 (five packets transmitted over three slots). The sixth packet is then transmitted in the second slotframe. In the context of *ASAP*, it follows a similar approach to *UPA*, aggregating the first five packets within the initial slotframe, resulting in a slot utility of 1.67. It then completes the transmission of the sixth packet in the second slotframe. However, *ASAP* achieves even faster packet delivery compared to *UPA* due to the smaller slot size managed by *SLA*. We note that the tiny delay introduced before the first packet in both *UPA* and *ASAP* are due to the offset time required for CCA.

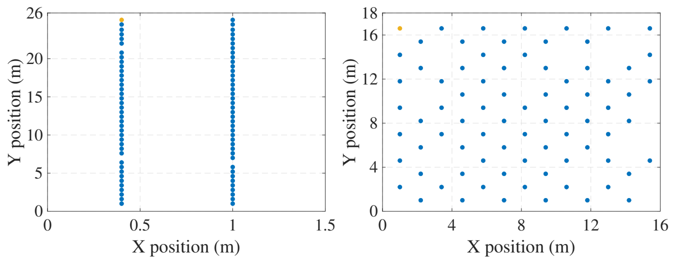
V. Evaluation

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

A. Implementation and experiment setup

We implement *ASAP*[6] on M3 board using Contiki-NG[7] [27]. For the comparison schemes, we use the publicly available implementations of ALICE[8], DBT[9], and A3[10], which are also implemented in Contiki-NG. The slot length is set to 10 ms by default, and we set the slotframe lengths of EB, broadcast, and unicast slotframes to 397, 17, and 20 slots, respectively, referring to the configuration in the A3 paper [25]. For channel hopping, we utilize the four IEEE 802.15.4 channels 15, 20, 25, 26. For the routing layer, we use RPL storing mode [48] and MRHOF with ETX [49] for the objective function in RPL. We enable the DAO-ACK option in RPL to promote reliable downward routing and resource scheduling. We set the transmission power of each node to -17 dBm.

[6]https://github.com/Hongchan-Kim/ASAP
[7]https://github.com/iot-lab/iot-lab-contiki-ng
[8]https://github.com/skimskimskim/ALICE
[9]It is included in the implementation of Contiki-NG.
[10]https://github.com/skimskimskim/A3



(a) Grenoble testbed (b) Lille testbed

Fig. 8: Node deployment topology at Grenoble and Lille testbeds. In both testbeds, the node marked in yellow (located in the upper left corner for both testbeds) serves as the root.

We conduct experiments on the FIT/IoT-LAB testbed [28], a large-scale public testbed, at three sites: Lyon, Grenoble, and Lille. Lyon testbed was used for the preliminary study in Fig. 1 and the evaluation of *UPA* in §V-C. Grenoble and Lille testbeds were used to assess the performance of *ASAP* in a multi-hop topology utilizing 79 M3 nodes at each site. Fig. 8 depicts the physical deployment topology of the 79 nodes at Grenoble and Lille. At Grenoble, the nodes are arranged in two narrow and long rows, while at Lille, the nodes are distributed almost evenly in a rectangular grid shape. Approximately 8-hop routing topology is formed at Grenoble, while a 3-4 hop topology is formed at Lille. Through experiments in these two sites with very distinct characteristics, we comprehensively verify the performance of *ASAP*.

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

Here we list the key performance metrics for evaluation:

- **App. layer goodput** is the end-to-end per-minute packet goodput for each node excluding losses and retransmissions.

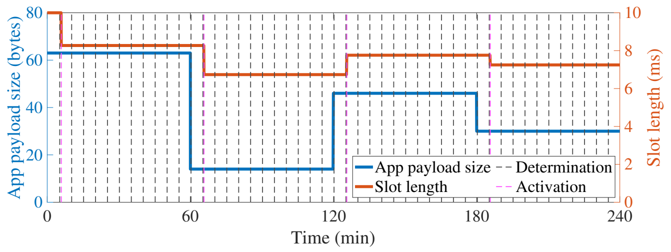
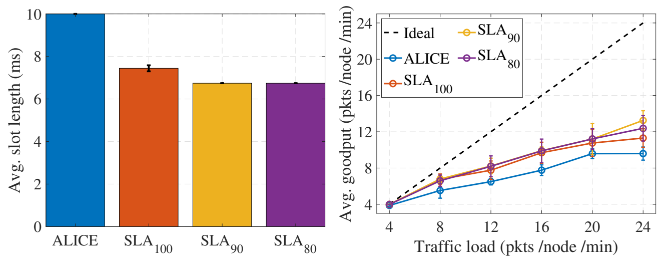


Fig. 9: Real-time operation of *SLA*: Evolution of slot length over time in accordance to payload (packet) size changes.



(a) TSCH slot length **(b)** App layer goodput

Fig. 10: Performance of *SLA* with varying k-percentile values vs. ALICE, at the Grenoble testbed, for data collection scenario.

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

B. Performance of SLA

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

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

Fig. 9 plots the time-evolution of slot length in accordance to payload size changes over time, together with the timing of *SLA*'s slot length determination and activation actions. *SLA* coordinator continuously monitors the packet size distribution and periodically determines the appropriate slot length, as indicated by the black dashed line. Whenever there is a change in the distribution, the *SLA* coordinator detects it and adjusts the slot length accordingly. The *SLA* coordinator also determines the activation time, represented by the pink dashed line. After all nodes in the network advertise the new slot length and activation time, the new slot length is applied simultaneously at the designated time. The overall result demonstrates that *SLA* successfully adjusts the slot length whenever the packet size decreases or increases. We note that at the first determination point of *SLA*, the slot length decreases even though the packet length remains the same. This happens because the original slot length of 10 ms is adjusted to fit the data packets of 63 bytes payload.

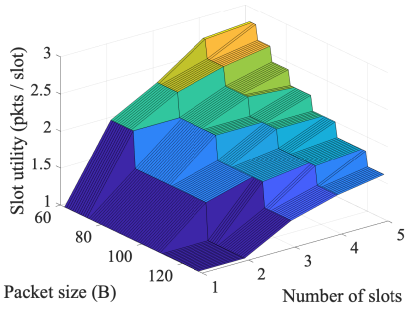
Choice of an appropriate k for the k-th percentile policy: The k-th percentile policy serves as a reference for adjusting the slot length based on the packet size distribution, and impacts the performance of *SLA*. Therefore, choosing an appropriate k value is crucial. In most of our experiments, a single-size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

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

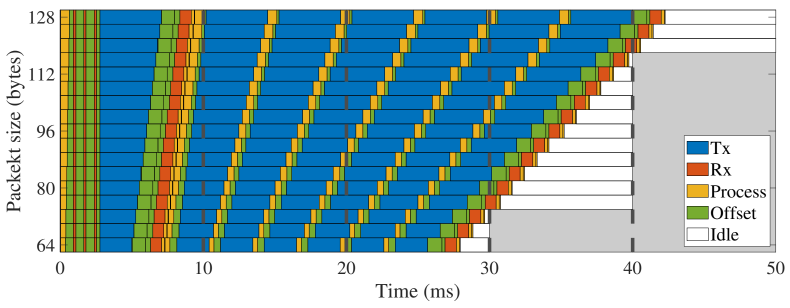
C. Performance of UPA

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

Fig. 11a plots the maximum achievable slot utility of *UPA* as a function of packet size and number of slots used for aggregation. For example, if *UPA* utilizes five consecutive slots, slot utility can go up to 2.8 by aggregating 14 minimum-

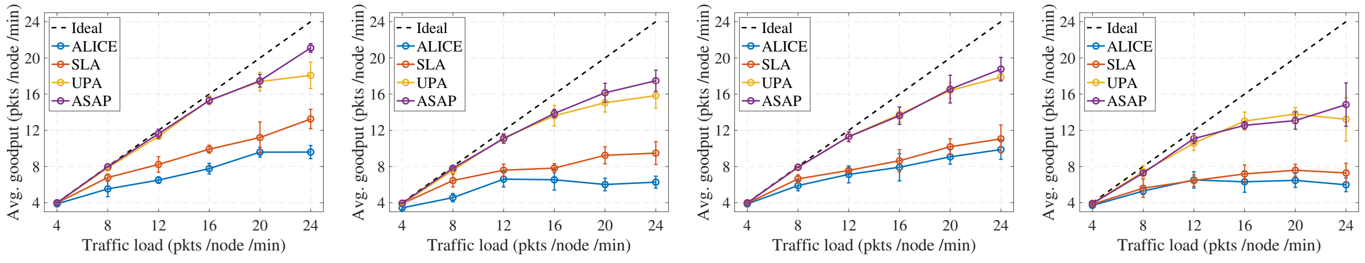


(a) Slot utility vs. packet size and num. of slots

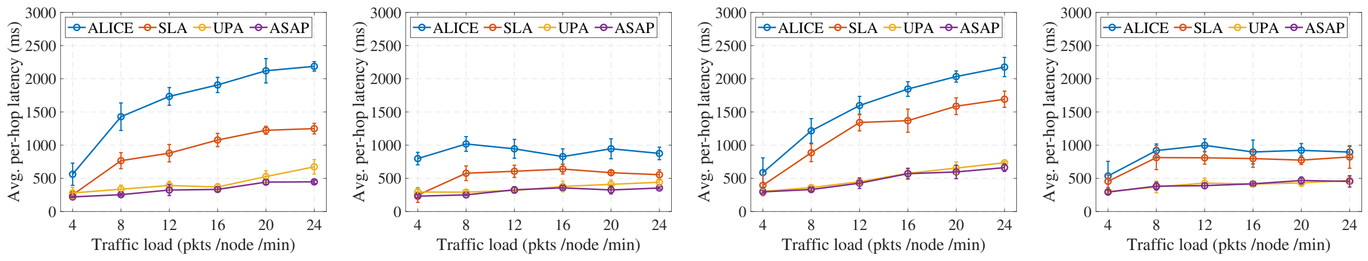


(b) Aggregating seven packets in *UPA*. Seven packets can be sent in three–five slots.

Fig. 11: Performance of *UPA*: maximum slot utility and an example of aggregating seven packets to reduce residue time.



(a) Goodput, 14 bytes, upward **(b)** Goodput, 14 bytes, downward **(c)** Goodput, 63 bytes, upward **(d)** Goodput, 63 bytes, downward



(e) Latency, 14 bytes, upward **(f)** Latency, 14 bytes, downward **(g)** Latency, 63 bytes, upward **(h)** Latency, 63 bytes, downward

Fig. 12: Ablation study on *ASAP*: Goodput and latency results for upward vs. downward scenarios and 14 vs. 63 byte payloads.

sized packets. This figure can guide *UPA* in terms of when and how many packets to aggregate. It shows that *UPA* can achieve slot utility of 1.5∼2.8 in most cases, and smaller packets provide more opportunity for higher improvement. This analysis is corroborated by the time usage breakdown of aggregating seven packets as illustrated in Fig. 11b. Following the triggering packet and ACK, six packets are transmitted in a batch with very short intervals, reducing the number of slots required from seven to three∼five depending on the packet length. For example, for the seven minimum-sized packets, *UPA* can improve the slot utility to 2.33 by utilizing 3 consecutive slots.

D. Performance of ASAP: an ablation study

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

Application layer goodput: Fig. 12a plots the goodput in the data collection (upward) scenario with the minimum payload length (14 bytes). ALICE performs the worst beyond traffic load of 4 pkts/min/node, and fails to deliver a significant number of packets as the traffic load increases. This is because, the traffic load was higher than ALICE's effective data rate at the bottleneck nodes in most cases. *SLA* improves over ALICE, demonstrating that *SLA* has successfully increased the effective data rate by adjusting the slot length. *UPA* exhibits even more improvement, showing that *UPA* has successfully increased the effective data rate through packet aggregation. *UPA* achieves higher improvement than *SLA* because *UPA* can flush the bottleneck node's queue backlog in a burst within a slotframe whereas *SLA* still requires same number of slotframes as the number of packets. The collaboration of *SLA* and *UPA* within *ASAP* leads to further improvement outperforming *UPA* as the traffic load increases.

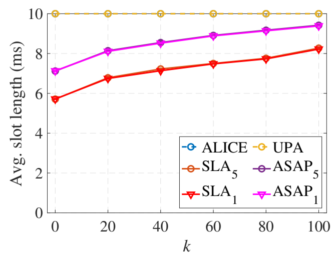
Per-hop latency: Fig. 12e plots the latency result in the collection (upward) scenario with the smallest payload length (14 bytes). *SLA* achieves lower latency than ALICE by adapting the slot length to be shorter. *UPA* further reduces latency since its packet aggregation allows each packet to be transmitted significantly earlier than the baseline schedule. By harmonizing the effects of these two techniques, *ASAP* achieves the lowest latency.

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

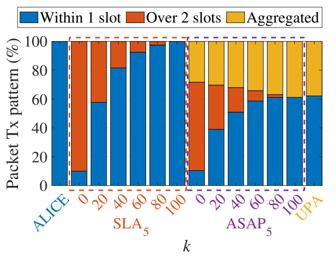
Impact of traffic direction: In the downward traffic scenario, the root node generates and disseminates downward traffic throughout the network. Since packets are distributed to multiple nodes, aggregation opportunity in each link may decrease, reducing the slot utility. Furthermore, the root is the main bottleneck of throughput, resulting in an overall decrease in goodput across all schemes as shown in Fig. 12b. Nevertheless, *SLA*, *UPA*, and *ASAP* still outperforms ALICE for the same reasons discussed earlier, with *ASAP* having the highest performance. Latency improves for the downward scenario as well, as shown in Fig. 12f. However, in contrast to the upward scenario, downward traffic has relatively constant per-hop latency. This is because the root node is the bottleneck, and the nodes below the root forward fewer packets than the root and rarely experience congestion. Thus, the per-hop latency does not change significantly per traffic load.

Impact of packet length: We lastly explore the impact of packet length on the performance of *ASAP* using experiments with a payload size of 63 bytes. The results are plotted in Figs. 12c, 12d, 12g and 12h. We observe that the trends for goodput and latency remain unchanged from the experiments with 14-byte payload, in both upward and downward traffic scenarios. In the case of ALICE, the goodput and latency remain almost the same because the packet length difference does not affect ALICE's operation. However, for *SLA*, *UPA*, and *ASAP*, the goodput and latency degrade slightly as the packet size increases. This is because, with longer packets, *SLA* cannot shorten the slot length as much as with shorter packets, and *UPA* cannot aggregate as many packets. Nevertheless, *SLA* and *UPA* still achieve significantly better performance than ALICE, and so does *ASAP*.

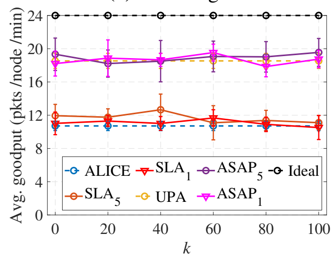
Summary: Overall, the results validate the superiority of *ASAP*. Across all scenarios, irrespective of traffic direction and packet size, the proposed schemes improve effective data rate by addressing the inefficiencies of ALICE resulting from wasted residue time, leading to enhanced goodput and latency.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



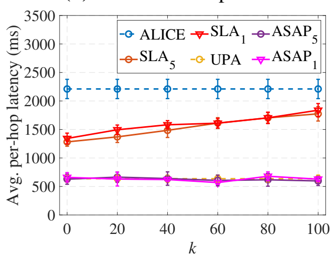
(a) Slot length



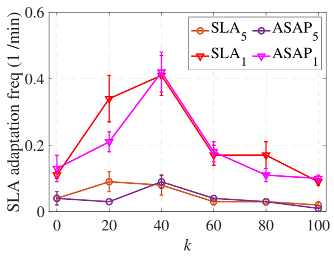
(b) Transmission pattern



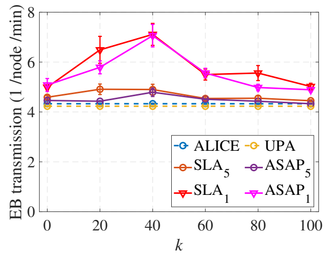
(c) Goodput



(d) Latency



(e) *SLA* adaptation frequency



(f) EB transmission frequency

Fig. 13: Performance of *SLA*, *UPA*, and *ASAP* with varying *'random'* payload sizes (uniform random between 14-63B), depending on k and T_{det} configurations. The subscripts 1 and 5 in *SLA* and *ASAP* denote that T_{det} is 1 minute and 5 minutes, respectively.

E. Discussion on SLA, UPA, and ASAP

So far, we examined the performance of *SLA*, *UPA*, and *ASAP* using fixed-size data packets while varying the traffic load and direction. Here we extend our evaluation to variable packet sizes and discuss the limitations of the proposed schemes.

To introduce variability in packet sizes, we let each node transmit data packets with payload lengths uniform randomly distributed between 14 to 63 bytes. We also vary the k parameter in *SLA*'s k-th percentile policy from 0 to 100%, and adjust the *SLA*'s adaptation interval T_{det} between 1 and 5 minutes. We maintain the traffic load of 24 packets per node per minute in upward direction to keep the network saturated. Fig. 13 presents the results for ALICE, *SLA*, and *ASAP*.

Impact of k in *SLA*'s k-th percentile policy: Fig. 13a plots slot lengths for ALICE, *UPA*, *SLA*, and *ASAP* according to the k value. ALICE and *UPA* has fixed slot lengths regardless of the k values since there is no adaptation. In contrast, *SLA* and *ASAP* reduce their slot sizes as k decreases, as smaller k prompt *SLA* to adapt the slot length to smaller packet sizes. These differently adjusted slot sizes result in distinct transmission patterns as shown in Fig. 13b. ALICE transmits one packet in one slot for all packets. *UPA* aggregates some packets and sends them in batches while other packets are still sent individually per slot. In *SLA* and *ASAP*, smaller k values may lead to a larger proportion of packets being larger than a slot, thus transmitted over the slot boundaries. However, this minimally affects average goodput, as shown in Fig. 13c. This is because in both *SLA* and *ASAP*, CCA effectively handles over-the-boundary packets, resolving collisions. Furthermore, in *ASAP*, *UPA* gains additional efficiency by aggregating long packets with short ones, achieving higher slot utility and improved goodput compared to *SLA*. Fig. 13d demonstrates that *ASAP* maintains consistent latency across different k values, while *SLA*'s latency decreases with smaller k. *ASAP*'s low latency is primarily due to *UPA*'s packet aggregation, with k having minimal impact. Conversely, in *SLA*, the k value directly affects *SLA*'s latency, as shorter slots lead to more frequent resource availability.

One of the challenges in implementing *SLA* is choosing the appropriate value for k. The reduction in slot length determined by k decreases the residue time for packets below the k-th percentile but may lead to transmissions beyond the slot boundary for packets above the 100-k^{th} percentile. *SLA* allows these transmissions, causing the next slot to be skipped. However, this behavior is undesirable as it can result in another residue time issue within the subsequent slot. Therefore, selecting the right k value requires finding a balance between reducing residue time and preventing overflow. Our experiments demonstrate that choosing an appropriate k effectively addresses these issues while achieving the desired performance. For instance, selecting $k \geq 80$ mitigates such problems, enhancing goodput and reducing latency compared to TSCH, as supported by comprehensive experimental results. Nevertheless, the optimal k value may vary depending on network characteristics or application requirements.

Impact of *SLA*'s adaptation interval T_{det}: As depicted in Fig. 13e, a smaller T_{det} value results in more frequent slot length adaptations, leading to more frequent EB transmissions as shown in Fig. 13f. However, altering T_{det} does not significantly affect the adjusted slot size for both *SLA* and *ASAP* as plotted in Fig. 13a, and it has no significant impact on goodput nor latency as presented in Figs. 13c and 13d. This stability can be attributed to several reasons. Firstly, in our experiment, although individual packet sizes vary dynamically, the overall packet size distribution remains the same (i.e., uniform random). Secondly, the minute-scale granularity of T_{det} is significantly larger than the packet latency of seconds or the transmission interval of milliseconds[11]. Therefore, when T_{det} is small (e.g., 1 minute), *SLA* may respond more frequently to temporary packet length changes compared to when T_{det} is large (e.g., 5 minutes), but the long-term performance remains almost the same across different T_{det} values.

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

[11]78 nodes each transmit 24 packets per minute, resulting in an average inter-packet interval of 31.2 milliseconds.

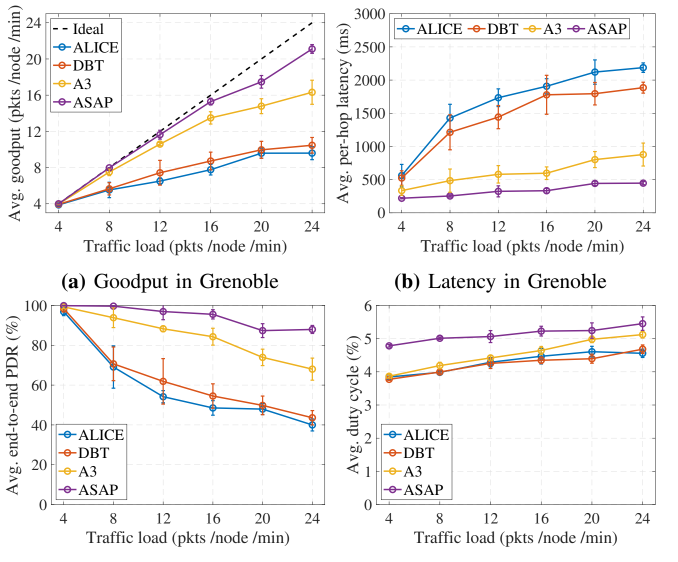
Impact of packet aggregation in *UPA*: As shown in Fig. 13b, a significant proportion of packets are aggregated and transmitted when *UPA* is applied (i.e., *UPA* and *ASAP*). For example, even when k is set to zero resulting in the smallest slot size and minimal attempt for packet aggregation, *ASAP* transmits more than 28% of the packets through aggregation. As we have already mentioned in §IV-B, batch transmission of *UPA* is performed beyond the originally scheduled slot. Thus it may overlap with the schedules of other links, possibly resulting in a collision. To address this problem, we enabled non-participants to avoid collision with *UPA*'s batch transmission through CCA, which may lead non-participating nodes to yield their transmissions. However, this effectively prioritizes communication that can transmit more packets within the same slots, aligning with *UPA*'s design goal to enhance the network's time efficiency. Furthermore, comprehensive experimental results demonstrate that despite the potential for collisions, *UPA* achieves improved goodput, latency, and PDR compared to when *UPA* is not applied.

The second challenge in implementing *UPA* is the overhead introduced by packet aggregation. During *UPA* operation, all frames add an extra 6 bytes. In our proof-of-concept implementation, these 6 bytes include the field header/termination for the IEEE 802.15.4e information element (IE) to comply with the IEEE 802.15.4 header format. It is important to note that these extra 6 bytes represent only about 5-10% of the packet size and just 1.92% of the default 10 ms timeslot duration. However, *UPA* reduces the number of slots required for packet transmission, which offsets this overhead. In our evaluations, these extra bytes had no noticeable impact on goodput, latency, nor PDR. The overhead is therefore considered acceptable and effectively contributes to improving the time efficiency of the network.

F. Performance of ASAP: a comparative study

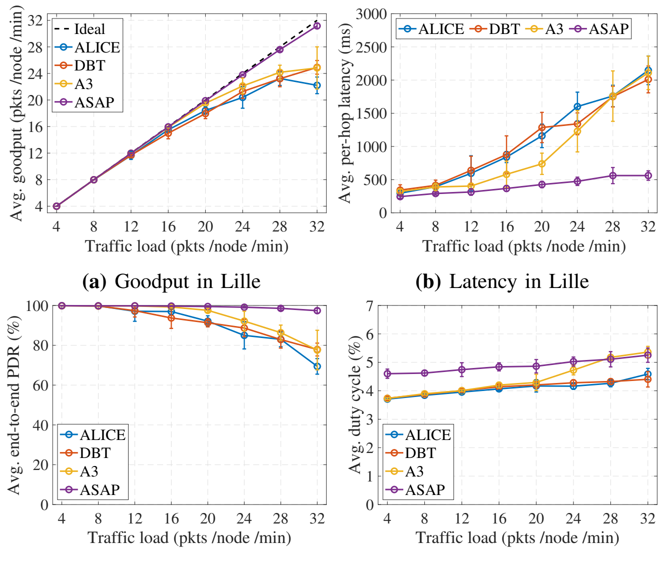
We now compare the performance of *ASAP* against other state-of-the-art schedulers, namely ALICE, DBT, and A3. For this purpose, we run experiments for the data collection scenario (i.e., upward traffic) at both Grenoble and Lille sites. While varying the traffic load, we measure the application layer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

Overall performance: As shown in Fig. 14, *ASAP* significantly outperforms all three baseline schemes in terms of



(a) Goodput in Grenoble **(b)** Latency in Grenoble **(c)** PDR in Grenoble **(d)** Duty cycle in Grenoble

Fig. 14: Performance comparison of *ASAP*, ALICE, DBT, and A3 at Grenoble with dual-linear topology. *ASAP* achieves better goodput, latency, and PDR with only a slight increase in duty-cycle (<1%).



(a) Goodput in Lille **(b)** Latency in Lille **(c)** PDR in Lille **(d)** Duty cycle in Lille

Fig. 15: Performance comparison of *ASAP*, ALICE, DBT, and A3 at Lille testbed with grid-like topology. *ASAP* achieves better goodput, latency, and PDR with only a slight increase in duty-cycle (<1%).

goodput, latency, and PDR. *ASAP* enhances not only goodput and latency but also PDR for the following reasons. In multihop TSCH networks, packet loss rate is primarily influenced by queue losses, i.e., the number of packets lost in the queues of forwarding nodes. *ASAP* achieves better PDR due to its reduction in queue losses by reducing residue time and increasing the effective data rate, thereby alleviating congestion at bottleneck nodes. DBT performs similarly to ALICE, with only minor improvements in some cases. A3, on the other hand, exhibits notably better performance than ALICE and DBT. Nevertheless, *ASAP* has significantly better performance especially at higher traffic load. However, *ASAP* exhibit a slightly higher duty cycle than other schemes. This is because *ASAP* delivers more packets successfully; That is, the radio on time is used for useful work, significantly improving throughput.

Comparison with DBT: It is worth noting that DBT attempts to send additional packets by recursively allocating additional resource when a queue backlog occurs and there is no schedule in the subsequent slot. This approach does provide DBT with more opportunities to send packets than ALICE. However, it does not cope with residue time within each timeslot. Moreover, when an existing schedule is present in the subsequent slot, DBT prioritizes it instead of allocating additional resource, resulting in limited improvements in reducing queue backog or alleviating congestion at the bottleneck compared to ALICE. In contrast, *ASAP* exhibits superior performance compared to DBT (Figs. 14a to 14c), primarily due to its *UPA* mechanism; i.e., it enables more packets to be transmitted in fewer slots than DBT. Moreover, *ASAP* prioritizes transmissions with high slot utility over the existing schedule, which significantly enhances network time-efficiency.

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

G. *Performance of ASAP in different environment*

Finally, we validate whether *ASAP* maintains good performance in a different environment with drastically different physical topology. For this purpose, we evaluate ALICE, DBT, A3, and *ASAP* at the Lille testbed which has a compact grid deployment (Fig. 8b) unlike the long dual-linear topology of the Grenoble site (Fig. 8a). The network topology characteristics of the two locations are markedly different. At Lille, the hop distance is reduced by about half compared to Grenoble, and the bottleneck problem becomes less severe, but more nodes are connected to the root directly. Considering this topological difference, we increase the traffic load up to 32 pkts/min/node at Lille from that of 24 at Grenoble. By comparing Fig. 14 and Fig. 15, we can analyze the impact of topology on *ASAP* and other compared schemes.

As shown in Fig. 15, the performance trends among ALICE, DBT, A3, and *ASAP* at the Lille site are similar to those observed at the Grenoble site; *ASAP* outperforms all other schemes. However, at the Lille site, ALICE, DBT, and A3 perform well even at much higher traffic loads than those at the Grenoble site. This is due to the shallower topology where many nodes can directly deliver packets to the root node, thus have less resource shortage issues. Nonetheless, both ALICE and DBT experience a decrease in goodput and an increase in latency at a traffic load of 20, while A3 experiences these issues at traffic load of 24. In contrast, *ASAP* maintains goodput close to ideal and minimizes latency increase even at traffic load of 32. Regarding the duty cycle, similar trend but slightly lower energy consumption is observed at the Lille testbed for the same reason of shallower network depth. To sum up, *ASAP* has shown successful performances in both linear and grid shape topologies. Based on the results, we believe that *ASAP* can generalize to other topologies and perform well in various environments.

VI. Conclusion

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

References

[1] L. Li, Y. Dong, C. Pan, and P. Fan, “Timeliness of Wireless Sensor Networks With Random Multiple Access,” *Journal of Communications and Networks*, vol. 25, no. 3, pp. 405–418, 6 2023.

[2] R. Heile, R. Alfvin, P. Kinney, J. Gilb, and C. Chaplin, “IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer,” IEEE, Tech. Rep., 2012.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] S. Kim, H.-S. Kim, and C. Kim, “ALICE: Autonomous Link-based Cell Scheduling for TSCH,” in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks (IPSN)*, 2019, pp. 121–132.

[22] 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.

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

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

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

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

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

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

[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, pp. 327–332.

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

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

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

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

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

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

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

[37] J. Shin, H. Kim, J. Paek, and S. Bahk, "Eca: Exclusive cell allocation for autonomous scheduling in time-slotted channel hopping," in *Proceedings of The IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*. IEEE, Sep. 2023.

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

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

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

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

[42] C. Michaelides, T. Adame, and B. Bellalta, "ECTS: Enhanced Centralized TSCH Scheduling with Packet Aggregation for Industrial IoT," in *IEEE Conference on Standards for Communications and Networking (CSCN)*, 2021, pp. 40–45.

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

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

[45] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012.

[46] H.-S. Kim, J. Ko, D. E. Culler, and J. Paek, "Challenging the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL): A Survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2502–2525, Sep. 2017.

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

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

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

Hongchan Kim is currently a Ph.D. candidate at the School of Electrical and Computer Engineering, Seoul National University, Seoul, Republic of Korea. He received his B.S. degree in electrical engineering from Seoul National University, in 2015. His research interests include designing network protocols for IoT and constructing mobile IoT systems. He received the National Research Foundation (NRF) Global Ph.D. Fellowship, in 2016.

Geonhee Lee is currently a Ph.D. student at the School of Electrical and Computer Engineering, Seoul National University, Seoul, Republic of Korea. He received his B.E. degree in electrical engineering from Yonsei University, in 2021. His research interests include designing MAC protocols for LoRa LPWAN.

Juhun Shin received his B.S. degree in Electrical and Electronics Engineering from Chung Ang University, Seoul, South Korea, in 2022, and he is currently pursuing a Ph.D. degree with the Department of Electrical and Computer Engineering from Seoul National University, Seoul, South Korea. His research interests include the Internet of Things and network protocols for low-power and lossy networks.

Jeongyeup Paek is a professor at Chung-Ang University, Department of Computer Science and Engineering since 2015. He received his B.S. degree from Seoul National University in 2003 and his M.S.degree from University of Southern California in 2005, both in Electrical Engineering. He then received his Ph.D. degree in Computer Science from the University of Southern California in 2010. He joined Cisco Systems Inc. in 2011 where he was a Technical Leader in the Internet of Things Group (IoTG). In 2014, he was with the Hongik University, Department of Computer Information Communication as an assistant professor.

Saewoong Bahk received B.S. and M.S. degrees in electrical engineering from Seoul National University (SNU), in 1984 and 1986, respectively, and 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 at SNU. He has been leading many industrial projects on 3G/4G/5G and the IoT. He has published more than 300 technical articles and holds more than 100 patents. He is a member of the National Academy of Engineering of Korea (NAEK). He was President of the Korean Institute of Communications and Information Sciences (KICS). He has been serving as Chief Information Officer (CIO) of SNU. He was General Chair of the IEEE WCNC 2020, IEEE ICCE 2020, and IEEE DySPAN 2018. He was Director of the Asia–Pacific Region of the IEEE ComSoc. He is an Editor of the IEEE Network Magazine and IEEE Transactions on Vehicular Technology. He was Co-Editor-in-Chief of the Journal of Communications and Networks (JCN), and on the Editorial Board of Computer Networks Journal and the IEEE Tran. on Wireless Communications.