# On-the-Fly Autonomous Slot Allocation in 6TiSCH-Based Industrial IoT Networks

Alakesh Kalita, *Member, IEEE*, and Mohan Gurusamy, *Senior Member, IEEE*

***Abstract*—The IPv6 over time-slotted channel hopping mode of IEEE 802.15.4e (6TiSCH) wireless protocol stack is released to offer high-throughput, low and bounded latency, energy efficient, and reliable communication in industrial Internet of Things (IoT). However, scheduling communication *cell* among the nodes for exchanging sensory data is not trivial in 6TiSCH networks when the network traffic is highly dynamic and unpredictable. The existing autonomous scheduling schemes suffer from static allocation, high end-to-end latency, and high energy consumption. To address the abovementioned problems, in this work, we propose on-the-fly autonomous slot allocation (OASA) scheme to schedule slots for adaptive traffic in 6TiSCH networks autonomously and immediately. OASA also enables a low *radio-duty-cycle* of the nodes when network traffic is less, which is not considered by any existing adaptive autonomous schedulers. To validate the effectiveness of OASA, we implemented it on Contiki-NG and performed testbed experiments on FIT IoT-LAB. The testbed experiment results demonstrate the effectiveness of OASA in terms of latency, packet delivery ratio, and energy consumption compared to the existing autonomous scheduling schemes.**

***Index Terms*—Autonomous scheduling, IEEE 802.15.4e, IPv6 over the time-slotted channel hopping (TSCH) mode of IEEE 802.15.4e (6TiSCH), Industrial Internet of Things (IIoT), time-slotted channel hopping (TSCH).**

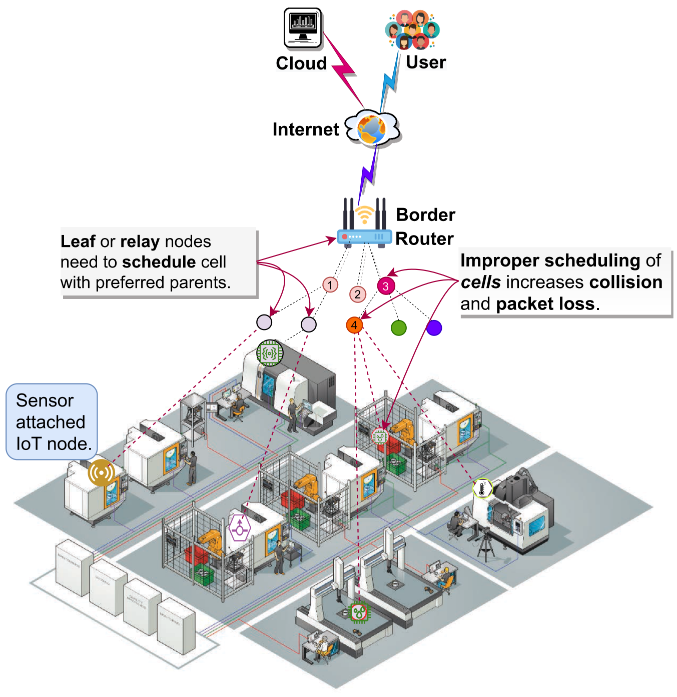
Cloud
User
Internet
Border Router
Leaf or relay nodes need to schedule cell with preferred parents.
Improper scheduling of cells increases collision and packet loss.
1
2
3
4
Sensor attached IoT node.

Fig. 1. 6TiSCH IIoT network connected to the Internet.

## I. Introduction

THE IPv6 over the time-slotted channel hopping (TSCH) mode of IEEE 802.15.4e (6TiSCH) wireless network protocol enables wireless communication in Industrial Internet of Things (IoT) (IIoT) and critical IoT applications [1]. 6TiSCH networks are built on the top of IEEE 802.15.4e standard, which is a low-power wireless communication standard that introduced TSCH medium access control (MAC) mode to provide stringent requirements, such as high reliability, higher throughput, delay-bounded, and energy-efficient communication in resource-constrained node-based IoT networks. Particularly, 6TiSCH networks can be adopted in scenarios requiring low-power and highly reliable communication, making them suitable for applications, such as industrial automation, smart grids, and healthcare systems [2]. In brief, 6TiSCH networks are a robust choice for time-sensitive and mission-critical applications. For example, in IIoT, as shown in Fig. 1, 6TiSCH can be used to enable real-time monitoring and controlling of industrial operations, such as assessing temperature, humidity, and other environmental factors, where sensors are placed all over a factory. The sensors attached to IoT nodes deliver real-time data to the root node [aka border router (BR)] using the routing protocol for low power and lossy network (RPL) [3] of the 6TiSCH protocol stack. The BR can process such data by itself or transmit it to the cloud using the Internet for further processing.

To transmit a sensory data packet from a given node, i.e., either leaf or intermediate/relay node to the BR, the node must schedule a unique “cell,” which is a combination of a *timeslot* (aka *slot*) and a physical *channel*, with its’ preferred RPL routing parent. However, the IEEE 802.15.4e standard does not provide information about managing/scheduling (i.e., allocation

Manuscript received 8 January 2024; revised 11 March 2024; accepted 19 March 2024. Paper no. TII-24-0120. *(Corresponding author: Alakesh Kalita.)*

The authors are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077 (e-mail: alakesh.kalita1025@gmail.com; gmohan@nus.edu.sg).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2024.3385117.

Digital Object Identifier 10.1109/TII.2024.3385117

and deallocation) such data transmitting cells. Scheduling of transmission cells is very challenging in dynamic IoT networks because assigning more cells unnecessarily would increase the energy consumption of the nodes by forcing them to activate their radios in the assigned slots. On the other hand, fewer assigned cells would severely degrade the network performance. Therefore, a conflict- and collision-free, and appropriate cell scheduling scheme is expected to improve the performance of multihop 6TiSCH-based IIoT networks.

For scheduling cells in 6TiSCH networks, researchers used either centralized [4], distributed [5], [6], or autonomous scheduling approaches, such as Orchestra [7]. However, autonomous scheduling has the advantage of zero control packet overhead in the network, and so, can save nodes' energy. Recently, several autonomous schedulers have been proposed, such as Orchestra [7], autonomous link-based cell scheduling (ALICE) [8], OST [9], and a cube (A3) [10]. However, both Orchestra and ALICE suffer from static and limited allocation of transmission and reception slots. Hereafter, we denote the transmission and reception slots by `Tx` and `Rx` slots, respectively. On the other hand, even though OST and A3 provide adaptive slot allocation, i.e., depending on networks' slot requirement, they suffer from high energy consumption when there is less traffic in the network. In addition, OST and A3 allocate slots based on traffic estimation, which is not immediate. Consequently, it contributes to an increase in packet delivery latency. Atis et al. [11] extensively studied some of the existing autonomous scheduling methods using testbed experiments and suggested to increase the number of `Rx` slots at the root node. However, increasing `Rx` slots of the root node only help the immediate neighbor nodes and, significant performance improvement of the other nodes, i.e. two or more hop distance nodes cannot be expected.

Therefore, to deal with the problems of existing autonomous scheduling schemes, we propose the on-the-fly autonomous slot allocation (OASA) scheme in this work. The proposed OASA is highly adaptive to the network traffic compared to the existing adaptive autonomous scheduling schemes, i.e., OST and A3. In brief, OASA enables very low radio-duty-cycle (RDC) based communication when traffic is very low. Still, it quickly adds more slots during sudden and burst traffic transmission (e.g., when an event occurs). Using OASA, nodes do not need to estimate traffic load before allocating slots, unlike A3 and OST. In brief, allocation by OASA does not increase packet latency. OASA autonomously allocates and schedules slots *on-the-fly* to handle adaptive traffic in 6TiSCH networks. The main contributions of this work are as follows.

1) We propose OASA scheme to autonomously and immediately allocate data-transmitting cells for adaptive traffic within 6TiSCH networks.
2) Addressing a gap in existing adaptive autonomous schedulers, OASA enhances the efficiency of nodes by enabling a low-duty cycle when network traffic is minimal.
3) We implement OASA in the widely-used IoT operating system, Contiki-NG [12] and perform testbed experiments on open-source FIT IoT-LAB [13] to validate the effectiveness of the proposed OASA.

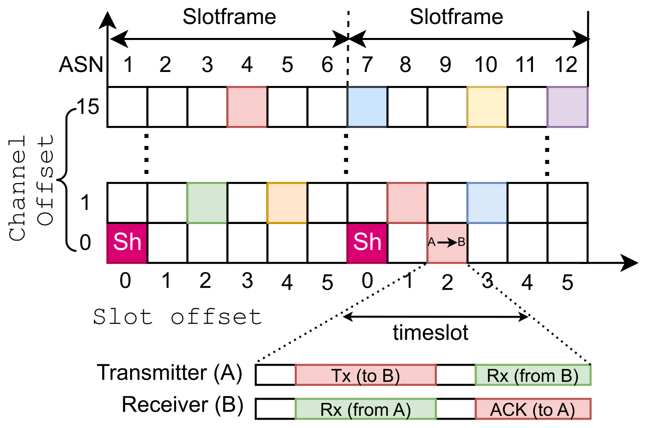

Fig. 2. Scheduling of TSCH slots with slotframe size of 6.

The rest of this article is organized as follows. Section II provides a brief overview of 6TiSCH network and discusses the existing scheduling schemes for it. We briefly discuss the design of our proposed scheme in Section IV, and provide comparison-based testbed results in Section V. Finally, Section VI concludes this article.

## II. Background and Related Work

In this section, we briefly discuss IEEE 802.15.4e TSCH and existing scheduling schemes for 6TiSCH networks.

### A. Overview of IEEE 802.15.4e TSCH

IEEE 802.15.4e TSCH MAC's time-slotted channel access feature provides guaranteed, bounded, deterministic, and collision-free packet delivery. On the other hand, the channel hopping feature helps in getting rid of interference and multipath fading issues on a single channel. To provide time-slotted channel access, as illustrated in Fig. 2, the time is divided into fixed-duration *timeslots*, which are long enough [usually, 10 ms to transmit a packet and receive its acknowledgement (ACK)]. Several timeslots constitute one *slotframe*, which repeats one after another. `Slot Offset` ($x$-axis of Fig. 2) is used to denote the relative position of a timeslot within a slotframe. A pair of nodes also need to decide on a channel to transmit a packet in a given timeslot, and the combination of timeslot and channel is called "cell." The `Channel Offset` ($y$-axis of Fig. 2) denotes the offset for channel selection. The sender and receiver pairs need to schedule their cells efficiently for collision-free communication. A well-designed scheduling scheme can efficiently use the cells to achieve high network performance. For exchanging data packets, in a given timeslot, both the sender and receiver should be on the same physical channel. For such physical channel selection, the following equation is used [1]:

$$\text{channel} = \text{FHS}\left[(\text{ASN} + \text{channeloffset}) \bmod N_c\right]. \quad (1)$$

where channel is the `channel index` associated with a dedicated frequency, absolute slot number (ASN) denotes the number of timeslots elapsed from the starting of the network, FHS is the frequency hopping sequence, $N_c = ||\text{FHS}||$ denotes the number of physical channels used in the network and each channel identified by `Channel Offset` ($y$-axis of Fig. 2). An entire 6TiSCH network uses the same ASN number, which is broadcasted by the enhanced beacon (EB) control frame. As the value of ASN increased by one after each timeslot, (1) gives different values of channel for the same *channel offset* within FHS, which enables the channel hopping feature of TSCH. For example, let FHS of a TSCH network is 11, 19, 15, 13. Then, the FHS of channel offset 0 will be 11, 19, 15, 13 that of channel offset 1 will be 19, 15, 13, 11, and so on. In Fig. 2, we use $N_c = 16$.

## B. Related Works

In this section, we briefly discuss the existing scheduling algorithms used in 6TiSCH network. In 6TiSCH network, cells are scheduled by centralized, distributive, or autonomous approaches. However, the centralized and distributed schemes have the issues of single node failure and control packet overhead, respectively. Therefore, autonomous approaches gain more attention from the researchers compared to the other two approaches. Traffic aware scheduling algorithm (TASA) [4] and MODESSA [14] are two well-known centralized scheduling schemes for the 6TiSCH network. TASA leverages the global tree topology and nodes' traffic load to minimize latency and RDC. On the other hand, MODESSA emphasizes load balancing by distributing loads on different channels. However, both schemes have severe control packet overhead and exhibit slow scheduling. The work in [15] mentioned a centralized approach to use SDN in 6TiSCH-based IoT networks, whereas the work in [16] proposed a Layer-2 slicing scheme to reduce SDN control traffic in TSCH network.

Distributed approaches utilize handshaking approaches, i.e., exchanging of control packets for several rounds for scheduling cells between the sender and receiver. Decentralized traffic aware scheduling (DeTAS) [17] is a distributed scheduling approach based on TASA, where the parent nodes gather the information about the traffic load of their child nodes and, accordingly, schedule cells. On the fly (OTF) [6] allocates cells based on previous transmission statistics and adjusts the number of cells in use. In a decentralized slot reservation policy (ASAP) [18], the authors analyzed 6TiSCH network during its formation, considering the real-use cases of 6TiSCH networks with varied numbers of nodes. ASAP adds two slots for each link, i.e., for each parent and child pair by exchanging control packets. Recently minimal scheduling function (MSF) standard [5] was released for dynamically allocating data transmission cells in 6TiSCH networks. However, this standard uses a distributed scheduling mechanism and has 6P control packet overhead.

To overcome the issues of centralized and distributed scheduling, autonomous schedulers allow the nodes to apply *hash function* either on its' own ID (i.e., `EUI64` or MAC address), neighbor's ID, or on both IDs to independently determine the dedicated timeslots and channels for exchanging packets. To join a 6TiSCH network, a new node should get an EB frame and a DODAG information object (DIO) packet from one of the already joined nodes, which is considered as the parent of the new node. From the headers of these control packets, the new node

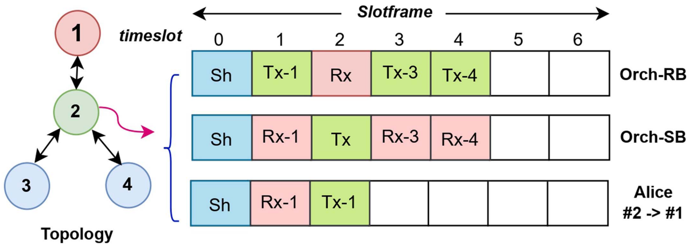

Fig. 3. Slot scheduling by node #2 with existing schemes.

comes to know about the `EUI64` address of the parent node. On the other hand, when the new node transmits data packets or other RPL's control packets, such as destination advertisement object (DAO) to the parent node, the parent node comes to know about the `EUI64` address of the new node. Thus, the parent and the new node come to know about each other `EUI64` addresses and exchange packets. When network topology changes, the autonomous scheduler of the nodes gets updated with the new neighbor information from the same control packets.

Orchestra, the first autonomous scheduling scheme proposed by Duquennoy et al. [7] provides two different modes of operation: receiver-based (Orch-RB) and sender-based (Orch-SB). However, only one mode can be used in a network at a given time. All the nodes assign a unicast cell for themselves using their ID, which operates as an `Rx` slot in Orch-RB and `Tx` slot in Orch-SB. On the other hand, the neighboring nodes consider such unicast cells as `Tx` and `Rx` in Orch-RB and Orch-SB, respectively. So, in Orch-SB, nodes transmit their packet by applying hashing on their IDs, as shown in Fig. 3. The main disadvantage of Orchestra scheduling is that Orch-RB faces high contention as all neighboring nodes transmit in the same `Rx` slot. On the other hand, Orch-SB suffers from queuing delay as a node can transmit only one packet to its' routing parent per slotframe. Furthermore, both the modes of Orchestra use static (i.e., fixed) slot allocation, which does not change with network slot requirements.

To address the limitations of Orchestra concerning the limited and static number of slots allocation per slotframe, Kim et al.[8] proposed ALICE in 2019. ALICE differs from the Orchestra in several ways. ALICE employs link-based slot scheduling, i.e., two separate slots for each RPL parent and child pair (shown in Fig. 3), unlike the node-based Orchestra, which schedules only one slot per slotframe for such pair. It also uses link-based `Channel Offset`, unlike a single `Channel Offset` as in Orchestra. Another key difference in ALICE is that it regularly reassigns/reallocates all the unicast cells by feeding the hash function with a time-dependent input. However, ALICE also suffers from the problem of limited slots allocation per slotframe (i.e., one for uplink towards the root and one for downlink toward the leaf) for the networks with high traffic loads, as experimentally shown by the works OST [9] and A3 [10]. For example, a parent node has two child nodes, and each of them needs to transmit one packet per slotframe to the root. However, as ALICE allows one uplink slot toward the root, the parent node can send only one packet per slotframe, which is insufficient. It requires three uplink slots per slotframe. The slot allocation by ALICE is also static, which is two slots per slotframe. Again, nodes need to keep their radios active once in a slotframe for each neighbor node irrespective of data transmission, which increases energy consumption in low traffic-based networks. Recently, to address the problem of insufficient slot allocation by Orchestra and ALICE, two adaptive scheduling schemes, OST and A3 were proposed by the same authors Kim et al. [9] and [10]. In OST, the nodes piggybacked their buffer occupancy information with their link-layer frames, and based on that, the receiver nodes estimate the traffic load from their neighbors and, subsequently, allocate slots. However, allocating slots after getting the traffic load information increases the packet latency as slot allocation and packet transmission are not done immediately, i.e., *on-the-fly*. Even though, in A3, nodes can estimate the traffic from their neighbor nodes without knowing their buffer occupancy information, unlike OST, it also has some disadvantages. A3 may be unable to deal with sudden burst traffic as traffic estimation and slot allocation take some time, which can result in packet loss, increasing packet latency. Similarly, slot deallocation also takes some time, which increases nodes' energy consumption. Most importantly, using OST and A3, nodes must keep their radios active at least once per slotframe for each neighbor, like ALICE, increasing the nodes' energy consumption in low-traffic networks.

The works in [19] and [20] autonomously allocate a shared minimal cell for control packet transmission in 6TiSCH networks. The authors did not consider the transmission of data traffic in their works. The work in [21] proposed a scheme for mobile-TSCH networks that has control packet overhead. On the other hand, the work in [22] dealt with the problem of missing ACKs in TSCH networks but was limited to single-hop networks. The work in [23] proposed the scheme Auto-Sched, which presents a fully autonomous scheduling approach, ensuring reliable uplink and downlink traffic scheduling while providing network robustness against failures. By assigning retransmission time slots based on link reliability and employing pipeline-like communication schedules, the Auto-Sched minimizes collisions. However, Auto-sched reserves some slots for control packets, such as DAO which may create problems when the slotframe length (SF) is small. In summary, even though autonomous scheduling is better than distributed scheduling algorithms, static autonomous scheduling algorithms, such as Orchestra and ALICE suffer from insufficient bandwidth allocation. On the other hand, OST and A3 can increase end-to-end packet latency and energy consumption of the nodes because of traffic estimation and longer convergence time of scheduling decisions. Therefore, to address these problems, we propose an adaptive scheduling technique for 6TiSCH networks that immediately allocates slots based on traffic demands, i.e., *on-the-fly*. In Table I, we summarize the existing works along with our proposed work considering different parameters.

## III. Study on Limitations of the Existing Schemes

In this section, we investigate the Orchestra and A3 autonomous scheduling schemes on FIT IoT-LAB using 60 `M3` IoT nodes (MCU: *ARM Cortex M3*, *32-bits*, *72* Mhz, *64* kB *RAM*, radio communication: *802.15.4 PHY standard*, *2.4* Ghz.), where all the leaf nodes and relay nodes generate and transmit packets toward the root node. Therefore, the utilized network topology can function as a sparse network under low traffic rates and as a dense network under high traffic rates. Note that the core mechanism of A3 is to estimate the traffic load, it can be used with any autonomous scheduler, such as Orchestra and ALICE, as an independent module. In our evaluation, we use A3 along with ALICE. During our experiments, the nodes are organized by RPL routing protocol as shown in Fig. 4. We used RPL storing mode for routing and minimum rank with hysteresis objective function (MRHOF) [26] with ETX for the objective function. In our experiments, we assess the performance of each scheme by comparing the end-to-end packet delivery ratio (PDR) and RDC. The SF is varied from 7 to 101 *timeslots*. In addition, we evaluate the packet acknowledgment ratio (PAR), queue losses, and parent changes across different SFs. The other experimental settings are listed in Table II. We consider standardized network configurations/settings as well as Contiki-NG's default experimental configurations for our evaluation. We run each experiment for 60 min, out of which in the initial 20 mins, we

TABLE I
Existing Scheduling Schemes

| Approach | Scheme | Cell/Traffic allocation | Convergence time | Data packet collision | Control packet Overhead |
|---|---|---|---|---|---|
| Centralized | TASA [4] | Adaptive | Moderate | Moderate | High |
| | MODESSA [14] | Adaptive | Moderate | Moderate | High |
| Distributed | MSF [5] | Adaptive | Moderate | Moderate | High |
| | ASAP [18] | Static | Moderate | Moderate | Less |
| | DeTAS [17] | Adaptive | Moderate | Moderate | High |
| | OTF [6] | Adaptive | Low | Moderate | High |
| | e-OTF [24] | Adaptive | Low | Moderate | High |
| Autonomous | Orchestra SB [7] | Static | – | High | No |
| | Orchestra RB [7] | Static | – | High | No |
| | ALICE [8] | Static | – | High | No |
| | OST [9] | Adaptive | High | Less | Less |
| | 6TiSCH-MC [25] | Static | – | – | No |
| | TACTILE [19] | Adaptive | – | – | No |
| | TRGB [20] | Adaptive | – | – | No |
| | A3 [10] | Adaptive | High | Less | No |
| | This work | Adaptive | Low | Less | No |

"–" denotes that a specific metric is not considered or not applicable to the corresponding scheme.

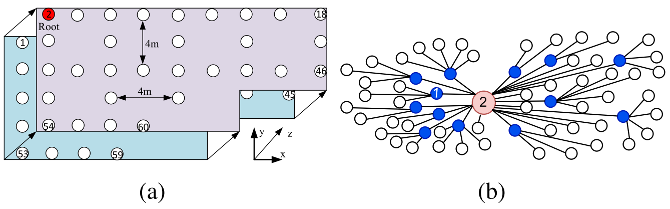

Fig. 4. Experimental 6TiSCH network topology from Strasbourg's. (a) Physical topology. (b) Routing topology.

TABLE II
Experimental Settings

| Parameter | Value |
|---|---|
| Operating system and testbed | Contiki-NG, FIT IoT-LAB |
| RPL version | RPL with MRHOF |
| Timeslot length | 10 ms |
| Unicast SF size | 7, 19, 67, 101 slots |
| EB SF size | 397 slots |
| Application traffic rate | 4 pkt/min |
| Queue size | 64 pkt |
| Number of channels | 4 |
| Payload size | 127 Bytes |
| Application warm-up time | 20 mins |
| Data collection time | 40 mins |

allow the nodes to join in the network, and in the rest of the 40 mins, we collect our data. In IIoT, plenty of nodes can be attached with a single BR; therefore, considering such a dense IIoT network, we evaluate Orchestra and A3 with high traffic rate, i.e., 4 packets(pkt)/min toward the root node.

In our evaluation, the following metrics are used.

1) *PDR:* Ratio between the packets received by the BR and transmitted by a sender. PDR of a sender is calculated at the BR.
2) *PAR:* Ratio between layer-2 ACKs and transmitted packets. Each node calculates PAR by itself. Note that layer 2 ACK is sent by the receiver irrespective of its queue capacity.
3) *Queue Loss:* Number of packets lost due to buffer overflow, calculated by the nodes themselves.
4) *RDC:* Percentage of the radio active time (i.e., `Rx`, `Tx`) against total experimental time. High RDC denotes higher energy consumption and vice versa.
5) *Latency:* Average time taken by the packets to reach root.
6) *Parent Change:* Number of times a node changes its parent. It is calculated at the node itself. More parent change denotes less stability in the network.

Our preliminary experimental results are shown using 95% confidence interval in Fig. 5 by running each experiment at least 25 times. In Fig. 5(a), we can see that the performance of Orch-SB and Orch-RB modes in terms of PDR degrades with the increasing value of SF due to insufficient provisioning of bandwidth, i.e., data transmission cells per slotframe to handle such high traffic load, 4 pkt/min. Even though SB mode provides better PDR using shorter SF, its performance drastically degrades when SF is 101 slots. The results shown in Fig. 5(b) denote that only a few packets were lost due to external interference or packet collision. As the nodes do not get enough slots to transmit their packets and the packets received from their child nodes when SF length is more, nodes keep the packets in their buffer for a longer time. Later, this results in discarding new incoming packets when the buffer becomes full, leading to more queue losses, as shown in Fig. 5(c). Note that both SB and RB show comparatively better results with short SF. However, the adverse effect of having short SF can be seen in Fig. 5(d). When the network uses short SF, nodes frequently allocate `Rx` and `Tx`, so the RDC of nodes increases, leading to more energy consumption. So, short SF is not desirable in 6TiSCH networks.

Furthermore, when using Orchestra, nodes need to retain data packets for longer due to limited slot allocation, increasing packet delivery latency for the nodes, as shown in Fig. 5(e). In addition, insufficient allocation of slots per slotframe forces the nodes to change their parent very frequently, as shown in Fig. 5(f), which creates unstable networks [10].

From these different results, we can claim that Orchestra does not provide sufficient slots per slotframe to transmit high traffic. As slot allocation by ALICE is also static and uses one slot per slotframe for each direction of a link, using ALICE, similar results can be expected as Orch-SB. Even though A3 can adaptively provide sufficient slots per slotframe for transmitting

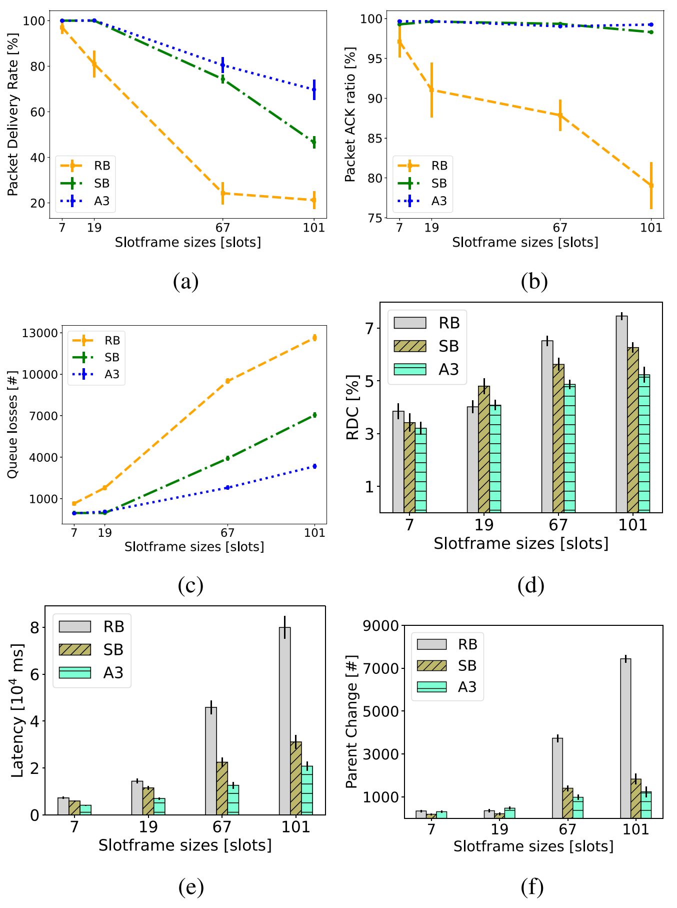

Fig. 5. Testbed experimental results of some existing autonomous schemes using application data rate 4 pkt/min. (a) PDR. (b) Packet ACK ratio. (c) Queue losses. (d) RDC. (e) End-to-end packet latency. (f) Number of parent changes.

such massive network traffic, it takes time to estimate the traffic load in the network. Because of the convergence of traffic estimation, both packet latency and energy consumption of the nodes increased. These experimental results from the testbed for existing autonomous schemes motivate us to design an adaptive and autonomous scheme that can dynamically allocate slots within a given slotframe based on the slot requirements of a 6TiSCH network. In addition, it should also maintain a low RDC for the nodes when network traffic is low. In the following sections, we briefly discuss our proposed approach.

## IV. Proposed Approach

In 6TiSCH networks, nodes change their physical transmission channels after each transmission. Therefore, apart from scheduling the packet in the same timeslot, it is important for both the transmitter and receiver to be active on the same channel. Considering this synchronization requirement, we briefly discuss our proposed autonomous and adaptive *on-the-fly* slot scheduling scheme in the following section.

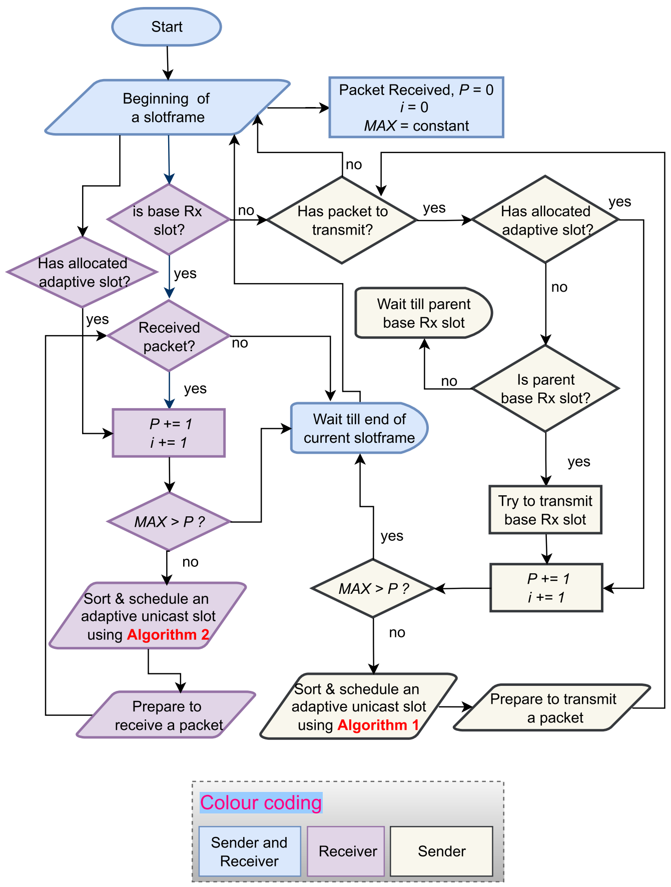
Start
Beginning of a slotframe
Packet Received, P = 0
i = 0
MAX = constant
is base Rx slot?
no
Has packet to transmit?
yes
Has allocated adaptive slot?
yes
no
Has allocated adaptive slot?
yes
yes
Received packet?
no
yes
Wait till parent base Rx slot
no
Is parent base Rx slot?
yes
P += 1
i += 1
Wait till end of current slotframe
Try to transmit base Rx slot
MAX > P ?
no
yes
P += 1
i += 1
MAX > P ?
no
Sort & schedule an adaptive unicast slot using Algorithm 2
Sort & schedule an adaptive unicast slot using Algorithm 1
Prepare to transmit a packet
Prepare to receive a packet
Colour coding
Sender and Receiver
Receiver
Sender

Fig. 6. Working of proposed OASA.

## A. On-the-Fly Autonomous Slot Allocation

Our proposed OASA uses the node's unique EUI64 address to independently and autonomously schedule cells like other autonomous schedulers, such as A3 and OST. Note that OASA works with other unique layer 2 address, such as MAC. Furthermore, using OASA, each node maintains a dedicated *broadcast slot* and a shared *common slot* for transmitting EB and RPL control packets, respectively, like the exiting autonomous schemes. However, for transmitting data packets, OASA works differently as discussed in the next paragraph. Fig. 6 illustrates the procedural steps employed by a sender–receiver pair in the implementation of our proposed scheme OASA. The ensuing paragraphs provide a detailed discussion of each step.

*Upward traffic:* It denotes the traffic from the sensor-attached IoT nodes to the root node of 6TiSCH networks. The root node later transmits the collected traffic to the cloud for analysis. For example, periodic data transmission by home monitoring system. For such upward traffic, each node maintains two types of unicast slots using OASA. The first slot is the shared Rx slot (hereafter referred to as ① base-Rx slot), which is used to receive packets from any child node like RB-Orch. The timeslot ($T_b$) and the channel ($C_b$) for the base-Rx cell of a receiver node are determined by applying hash-ing to the receiver's ($\mathbf{R}$) EUI64 address and then taking the modulo of the unicast SF, as follows:

$$T_b = \text{mod}(\texttt{hash}(\texttt{EUI64}(\mathbf{R}) + \text{ASFN}),\ \text{SF})$$

$$C_b = \text{mod}(\texttt{hash}(\texttt{EUI64}(\mathbf{R}) + \text{ASFN}),\ N_c - 1) + 1$$

here, ASFN denotes absolute slotframe number (calculated as $\lfloor \text{ASN}/\text{SF} \rfloor$ to reduce repeated collisions as in ALICE [8], and $N_c$ denotes the number of channels used in the network. We use a simple *32-bit integer mix* function for hashing as used by ALICE [8]. Note that both sender and receiver should use the same methods to calculate the $T_b$ and $C_b$, but they would be in opposite radio states, i.e., the base-Rx slot of the receiver should be the base-Tx slot for the sender. Otherwise, nodes will face radio-conflict. The other type of slot is ② adaptive unicast slot, which is used to transmit more than one packet in a slotframe. A sender–receiver pair allocates multiple adaptive unicast slots within a slotframe depending on the number of outstanding packets in the sender buffer (briefly discussed in the next two paragraphs). The first ($i = 0$) adaptive unicast cells's timeslot $T_{<a,i=0>}$ and channel $C_{<a,i=0>}$ are calculated using both the sender ($\mathbf{S}$) and receiver ($\mathbf{R}$) EUI64 addresses as follows.

$$T_{<a,i=0>} = \texttt{mod}\ (\alpha\ \texttt{hash}(\texttt{EUI64}(\mathbf{R}) + \texttt{EUI64}(\mathbf{S}) + \text{SHIFT}[i] + \text{ASFN}),\ \text{SF}),$$

$$C_{<a,i=0>} = \texttt{mod}\ (\alpha\ \texttt{hash}(\texttt{EUI64}(\mathbf{R}) + \texttt{EUI64}(\mathbf{S}) + \text{SHIFT}[i] + \text{ASFN}),\ N_c - 1) + 1.$$

A *SHIFT-array* and $\alpha$ (both are discussed later) are used to allocate successive adaptive unicast slots and differentiate the traffic directions, respectively. Note that if any of the calculated timeslot, $T_{<a,i>}$ for adaptive unicast slots conflict with base-Rx slot of the parent, then next timeslot is used, i.e., $T_{<a,i>} = T_{<a,i>} + 1$. The sender and receiver use these slots as follows.

*Sender:* When a child node wants to transmit more than one packet to its RPL parent (i.e., upward traffic toward the root) in a slotframe, it tries to transmit its first packet on the base-Rx slot of the parent as the adaptive unicast slots are not allocated yet. As the base-Rx slot is shared, all the child nodes (senders) attempt to transmit their first packet following the standard TSCH backoff approach. When the sender successfully transmits its first packet and receives ACK, it allocates adaptive unicast Tx slot(s) using the SHIFT[MAX] *array* to transmit subsequent packets as follows.

The SHIFT-array is used to shift the adaptive slots from the first adaptive slot within the current slotframe. The values of *SHIFT-array* can be obtained as follows:

$$\text{SHIFT}[i] = i * \left\lfloor \frac{\text{SF}}{\text{MAX}} \right\rfloor, \quad i = 0, 1, \ldots, \text{MAX} - 1.$$

For instance, consider a scenario where the SF consists of 101 timeslots, and the given value of MAX is 4. In this case, an additional three adaptive slots would be added, starting from the first adaptive unicast cell $T_{<a,i=0>}$ and $C_{<a,i=0>}$, with each of these new slots positioned at intervals of [25, 50, 75] slots apart. However, all the adaptive unicast cells are calculated and sorted

**Algorithm 1:** OASA-Sender.

```
1: INPUT: Outgoing packet, SHIFT=[], $flag = 0$, $MAX$
2: OUTPUT: Schedule transmission Tx slot(s)
3: if new $slotframe$ starts then
4:     Initialize $i = 0$
5: end if
6: if $flag$ is not set then
7:     Schedule and transmit in base-Tx slot using parent
       EUI64 address
8:     Set the $flag$ when transmission is success
9: end if
10: if buffer is not empty AND $flag$ is set AND $i$ is less
    than $MAX$ then
11:     Sort and schedule adaptive Tx slot with SHIFT[i]
12:     Increase $i$ by unity
13: end if
14: if buffer is empty then
15:     Unset the flag
16: end if
17: if not received ACK in scheduled adaptive Tx slot then
18:     Re-transmit in the next adaptive Tx slot
19: end if
```

**Algorithm 2:** OASA-Receiver.

```
1: INPUT: SHIFT=[], $i = 0$,
2: OUTPUT: Schedule Rx slot(s) for sender $k$
3: if new $slotframe$ starts then
4:     Initialize $i = 0$
5: end if
6: if received a packet in base-Rx slot then
7:     Remove already allocated adaptive Rx slot(s)
8:     Allocate new adaptive Rx slot with SHIFT[i]
9: end if
10: if received a packet in $C_k$-based adaptive Rx slot AND $i$
    is less than $MAX$ then
11:     Sort and schedule adaptive Rx slot(s) with SHIFT[i]
12: end if
13: Increase $i$ by unity
14: if not receive packet in current adaptive Rx slot then
15:     Remove all the scheduled adaptive Rx slots for $k$
16: end if
```

based on the relative position of the associated timeslots, $T_{<a,i>}$ from the beginning of a slotframe and scheduled accordingly.

Using many adaptive unicast slots within a slotframe would increase the RDC of the parent node. Therefore, we use MAX threshold to restrict the used number of slots per slotframe. The value of MAX threshold can be broadcasted in the information element (IE) of an EB frame. Therefore, SHIFT[MAX] can be autonomously calculated by the nodes (i.e., sender and receiver) by themselves as the same and known value of SF is used in the network. To reduce contention in the base-Rx slot, we impose a restriction preventing a node from transmitting a packet in the base-Rx slot of the subsequent slotframes, if the node has already installed adaptive Tx slot(s). As different parent-child pairs transmit their subsequent packets using different adaptive slots based on their EUI64 addresses, there will be no collisions in such allocated autonomous slots (assuming there is no hashing conflict and mitigating hashing conflict out of the scope of this article). Algorithm 1 and Fig. 6 show the steps of a sender to transmit its packet using OASA.

*Receiver:* When a node receives a packet in its base-Rx slot, it immediately schedules an adaptive unicast slot for the corresponding sender. If the node receives one more packet in the allocated adaptive unicast slot, it allocates one more adaptive unicast slot, and it continues till MAX is reached. Note that the receiver always allocates an adaptive slot when it receives a packet from the sender, and the sender may not have a packet to transmit in that allocated adaptive slot. When the receiver does not receive any frame in any of the allocated adaptive slots, it removes all the subsequent allocated adaptive slots, considering the sender does not have packets to transmit. The receiver needs to unnecessarily listen in the allocated adaptive slot in such events. However, energy consumption by the receiver for such *idle*-listening (i.e., 128 $\mu$S) is much less compared to energy consumption for packet reception and retransmission. Note that there could be a situation in which a receiver sends an ACK, but the sender does not receive it. In such a case, the sender tries to retransmit the packet on the next adaptive unicast slot. Receivers also need to sort the adaptive unicast cells before scheduling like the senders. Algorithm 2 and Fig. 6 show the steps of a receiver to receive packets from the sender $k$ using OASA. Note that a node uses either Algorithm 1 or Algorithm 2 at a given timeslot as it can be either sender or receiver.

*Downward Traffic:* It denotes the command or query from the cloud or root node to the actuator or sensor-attached IoT nodes. For example, getting the value of the temperature sensor or switching OFF the electric motor. For such downward traffic, i.e., query from RPL root (BR) to leaf, OASA can be used efficiently, where RPL parent nodes behave as the senders, and the child nodes behave as the receivers. Note that because of the usage of the coefficient "$\alpha$" for calculating $T_{<a,i>}$ and $C_{<a,i>}$, different adaptive cells will be allocated for downward traffic. In brief, coefficient "$\alpha$" distinguishes traffic direction.

*Advantages over OST and A3:* Some of the advantages of OASA over OST and A3 as follows.

1) Allocation is done *on-the-fly*, unlike OST and A3, which allocate after collecting and estimating traffic load, respectively. This helps in reducing packet latency.
2) In OST and A3, Rx and Tx-slots are allocated in every slotframe for each directional link irrespective of traffic load, which increases RDC. For example, if a node has three child nodes, the node allocates three Rx-slots in every slotframe in both OST and A3. However, in OASA, the node would allocate only one base-Rx-slot. So, OASA has the advantage over OST and A3 in terms of RDC even though, by default, it assigns one adaptive cell upon receiving a packet on base-Rx slot.

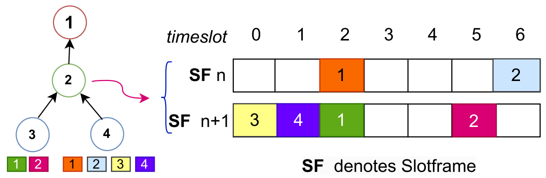

Fig. 7. Example of OASA-based scheduling. Different packets are denoted by different colors and sequence numbers.

3) Implementation of OASA uses only a few *flag* variables. In contrast, traffic estimation in A3 requires some computation, and OST incurs packet overhead.

## B. Working Example of OASA

In Fig. 7, we illustrate the scheduling of slots for nodes #3 and #4 toward node #2 across two slotframes using OASA. For this, we assume the values for MAX, ASFN, and SF*size* as 4, 0, and 7, respectively, and we use only one channel for packet transmission. Initially, nodes #3 and #4 both attempt to transmit their first packet during the base-Rx slot of node #2 (i.e., `slot 2`). Assuming node #4 successfully transmits its first packet, nodes #4 and #2 schedule their next adaptive slot at `slot 6`, i.e., $(\mathrm{id}(2) + \mathrm{id}(4) + \mathrm{SHIFT}[i] + \mathrm{ASFN})\%SF = 6$, where $i = 0$, $\mathrm{SHIFT}[i] = 0$. Subsequently, both nodes schedule sorted slots in the next slotframe (so, ASFN $= 1$) using the SHIFT values 0 and 1, respectively, at `slot 0` (i.e., $\mathrm{id}(2) + \mathrm{id}(4) + 0 + 1)\%7 = 0$) and `slot 1` (i.e., $\mathrm{id}(2) + \mathrm{id}(4) + 1 + 1)\%7 = 1$). As node #3 can transmit its first packet during base-Rx slot of node #2, nodes #3 and #2 coordinate to schedule their adaptive slot at `slot 5`. Thus, OASA autonomously allocates slots in real time.

## V. Experimental Results

We implement OASA on Contiki-NG and perform testbed experiments on FIT IoT-LAB using the same network configurations as discussed in Section III. The testbed results are shown in Fig. 8. We compare our proposed OASA with Orch-SB, OST, and A3. For fair evaluation, we consider the same value for *number of zones* in A3 and *MAX-threshold* of OASA, which is 4. Note that the number of zones and MAX denote the maximum number of Rx/Tx slots per slotframe. The other existing autonomous schemes, such as [19], [20], [25], primarily focus on control packet transmission. Consequently, we omit comparison of these schemes with OASA, as the latter specifically addresses data packet transmission, aligning more closely with methodologies like Orchestra, OST, and A3. As Orch-SB uses only one Rx/Tx slots per slotframe, therefore, the performance of Orch-SB is significantly poor compared to OST, A3, and OASA when the used SF is more, i.e., 67 and 101 timeslots, as shown in Fig. 8(a) and (b). It is because both the Rx- and Tx-slots appear less frequently (i.e., after 101 timeslots when SF $= 101$). Such infrequent availability of transmission slots degrades the performance of the nodes near the BR (i.e., nodes #2, #3, and #4) when the leaf and other intermediate nodes generate a large amount of traffic. In addition, because of the unavailability of

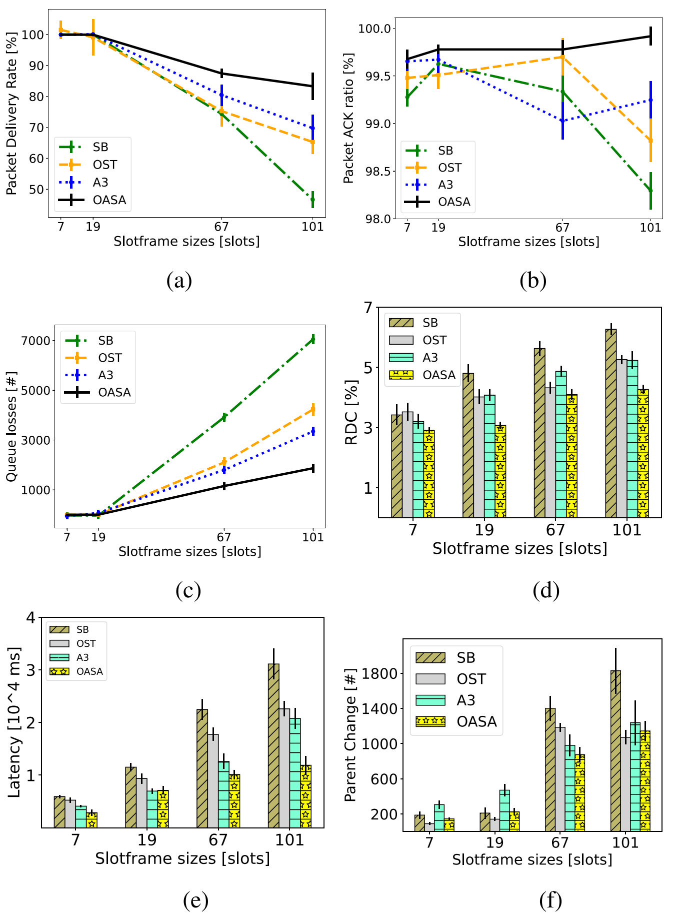

Fig. 8. Testbed experimental results of the proposed schemes using application data rate 4 pkts/min. (a) PDR. (b) Packet ACK ratio. (c) Queue losses. (d) RDC. (e) End-to-end packet latency. (f) Parent changes.

sufficient slots per slotframe using Orch-SB, nodes (specifically, again, nodes #2, #3, and #4) need to drop their packets from their buffer due to overflow, as shown in Fig. 8(c). Finally, it affects the PDR and the reliability of the IoT application. However, better performance is observed with short SF, i.e., 7 and 19 timeslots using all scheduling schemes as all the nodes get enough slots to transmit their packets. But short SF affects the RDC of nodes, i.e., energy consumption of the nodes. To deal with the problems of Orchestra due to static allocation, OST, A3, and OASA dynamically vary the number of slots per slotframe depending on network requirements. Therefore, all the three schemes perform better than Orch-SB. As we use the same value for *number of zones* of A3 and *MAX-threshold* of OASA, both A3 and OASA show almost similar results using all the used SF in terms of all the performance metrics. However, OASA performs better than A3 regarding RDC and latency. It is because, using OASA, the receiver does not allocate separate Rx slots for each neighbor in every slotframe, unlike ALICE, OST, and A3. In addition, using OASA, nodes do not need to wait for traffic load estimation to allocate Rx/Tx slots per slotframe, unlike OST and A3, which reduces the packet latency of the transmitted packet. Hence, OASA performs better compared to OST and A3 in terms of RDC and latency. Furthermore, because of its simplicity, OASA

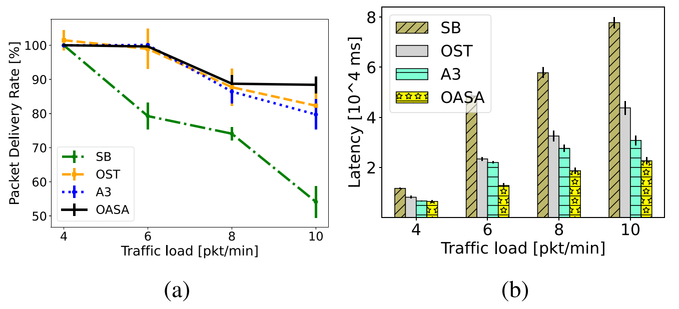

Fig. 9. Testbed results using fixed SF equals to 19 timeslots. (a) PDR. (b) End-to-end packet latency.

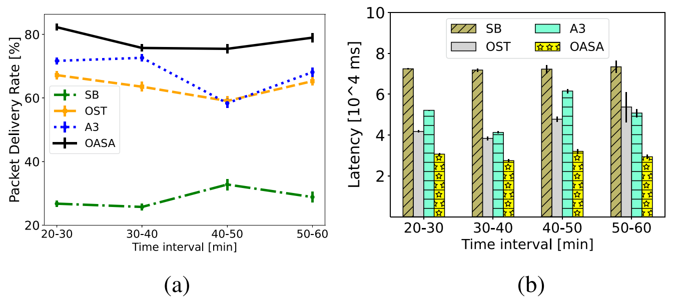

Fig. 10. Testbed results are presented at different time intervals throughout the total experimental duration. (a) PDR. (b) End-to-end packet latency.

requires less computational resources than A3. In brief, OASA is easy to implement. In Fig. 9, we show testbed experimental results by varying the packet transmission rate while keeping the same SF equal to 19 timeslots to show how our proposed scheme behaves with different network configurations. When the traffic load is less, i.e., 4 pkt/min and 6 pkt/min in the network, all the schemes perform in terms of PDR and latency. This is because the allocated number of slots by all the schemes is sufficient to handle such network traffic load. However, when the traffic load increases, Orch-SB fails to handle it because of uses only one slot per neighbor per slotframe. Even though both OST and A3 increase slots per slotframe, that is not immediate. Therefore, some of the packets get dropped using the estimation of the traffic load. However, using our proposed scheme OASA, nodes do not need to estimate traffic load, and allocation is done *on-the-fly*. Therefore, OASA improves the PDR and latency compared to all the existing schemes.

In Fig. 10, we present the results regarding PDR and packet latency at different time intervals. To observe the behavior of the OASA under a stable network and during network convergence, we temporarily stopped packet transmission from all nodes after 40 min of experiments and resumed it after the next 5 min. We use SF equal to 19 timeslots and packet transmission rate equal to 10 pkts/min. The results indicate that the existing scheme SB and OASA remain unaffected by network convergence. Conversely, as OST and A3 manage cells after collecting and estimating traffic load, respectively, both took some time to allocate cells for packet transmission after resuming. Therefore, their PDF and latency increased during the time interval 40–50 min. Hence, both PDR and packet latency are affected due to the convergence time of A3 and OST. We also run experiments with a very low traffic rate, i.e., 1 pkt/10 min, where OASA shows $\approx 41\%$ improvements of RDC over A3.

## VI. Conclusion

In this work, we developed a cell scheduling scheme called OASA to efficiently and autonomously allocate slots in real time in 6TiSCH networks. OASA scheme can potentially improve the reliability of 6TiSCH networks, particularly in situations where the network traffic is highly dynamic and unpredictable. OASA also enables a low energy consumption of the nodes when there is less traffic load in the 6TiSCH networks. In brief, by dynamically allocating and deallocating the communication slots *on-the-fly* for both upward and downward routing, OASA can adapt to real-time traffic demand. We implemented and tested OASA on Contiki-NG, and FIT IoT-LAB testbed, respectively. The results show that OASA performs better than the existing adaptive autonomous schemes, indicating the potential advantages of adopting OASA in practical IoT applications. In future, we aim to enhance the adaptability of OASA by considering different requirements of 6TiSCH networks and configurations of 6TiSCH protocol stacks.

## References

[1] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A tutorial," *IEEE Commun. Surveys Tut.*, vol. 22, no. 1, pp. 595–615, First Quarter 2020.
[2] X. Vilajosana, T. Watteyne, M. Vučinić, T. Chang, and K. Pister, "6TiSCH: Industrial performance for IPv6 Internet-of-Things networks," *Proc. IEEE*, vol. 107, no. 6, pp. 1153–1165, Jun. 2019.
[3] T. Winter et al., "RPL: IPv6 routing protocol for low-power and lossy networks," Internet Engineering Task Force, Wilmington, DE USA, Rep. RFC 6550, Mar. 2012.
[4] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic aware scheduling algorithm for reliable low-power multi-hop IEEE 802.15.4e networks," in *Proc. IEEE 23rd Int. Symp. Pers. Indoor Mobile Radio Commun.*, 2012, pp. 327–332.
[5] T. Chang, M. Vučinić, X. Vilajosana, S. Duquennoy, and D. R. Dujovne, "6TiSCH minimal scheduling function (MSF)," Internet Engineering Task Force, Wilmington, DE USA, Rep. RFC 9033, May 2021.
[6] M. R. Palattella et al., "On-the-fly bandwidth reservation for 6TiSCH wireless industrial networks," *IEEE Sensors J.*, vol. 16, no. 2, pp. 550–560, Jan. 2016.
[7] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *Proc. 13th ACM Conf. Embedded Networked Sensor Syst.*, 2015, pp. 337–350.
[8] S. Kim, H.-S. Kim, and C. Kim, "ALICE: Autonomous link-based cell scheduling for TSCH," in *Proc. 18th Int. Conf. Info. Process. Sensor Netw.*, 2019, pp. 121–132.
[9] S. Jeong, H.-S. Kim, J. Paek, and S. Bahk, "OST: On-demand TSCH scheduling with traffic-awareness," in *Proc. IEEE Conf. Comp. Commun.*, 2020, pp. 69–78.
[10] S. Kim, H.-S. Kim, and C.-K. Kim, "A3: Adaptive autonomous allocation of TSCH slots," in *Proc. 20th Int. Conf. Inf. Process. Sensor Netw.*, 2021, pp. 299–314.
[11] A. Elsts, S. Kim, H.-S. Kim, and C. Kim, "An empirical survey of autonomous scheduling methods for TSCH," *IEEE Access*, vol. 8, pp. 67147–67165, 2020.
[12] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. IEEE 29th Annu. Int. Conf. Local Comput. Netw.*, 2004, pp. 455–462.
[13] C. Adjih et al., "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Proc. IEEE 2nd World Forum Internet Things*, 2015, pp. 459–464.

[14] R. Soua, P. Minet, and E. Livolant, "MODESA: An optimized multichannel slot assignment for raw data convergecast in wireless sensor networks," in *Proc. IEEE 31st Int. Perform. Comput. Commun. Conf.*, 2012, pp. 91–100.
[15] P. Thubert, M. R. Palattella, and T. Engel, "6TiSCH centralized scheduling: When SDN meet IoT," in *Proc. IEEE Conf. Standards Commun. Netw.*, 2015, pp. 42–47.
[16] M. Baddeley, R. Nejabati, G. Oikonomou, S. Gormus, M. Sooriyabandara, and D. Simeonidou, "Isolating SDN control traffic with layer-2 slicing in 6TiSCH industrial IoT networks," in *Proc. IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw.*, 2017, pp. 247–251.
[17] N. Accettura, M. R. Palattella, G. Boggia, L. A. Grieco, and M. Dohler, "Decentralized traffic aware scheduling for multi-hop low power lossy networks in the Internet of Things," in *Proc. IEEE 14th Int. Symp. World Wirel., Mobile Multimedia Netw*, 2013, pp. 1–6.
[18] G. Micoli et al., "ASAP: A decentralized slot reservation policy for dynamic 6TiSCH networks in industrial IoT," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2019, pp. 1–6.
[19] A. Kalita and M. Khatua, "Autonomous allocation and scheduling of minimal cell in 6TiSCH network," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 12242–12250, Aug. 2021.
[20] A. Kalita and M. Khatua, "Time-variant RGB model for minimal cell allocation and scheduling in 6TiSCH networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1803–1814, Feb. 2024.
[21] W. Jerbi, O. Cheikhrouhou, A. Guermazi, and H. Trabelsi, "MSU-TSCH: A mobile scheduling updated algorithm for TSCH in the Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 19, no. 7, pp. 7978–7985, Jul. 2023.
[22] S. Scanzio, G. Cena, and A. Valenzano, "Enhanced energy-saving mechanisms in TSCH networks for the IIoT: The PRIL approach," *IEEE Trans. Ind. Informat.*, vol. 19, no. 6, pp. 7445–7455, Jun. 2023.
[23] A. Darbandi and M.-K. Kim, "Autonomous scheduling for reliable transmissions in industrial wireless sensor networks," *Energies*, vol. 16, no. 20, 2023, Art. no. 7039.
[24] F. Righetti, C. Vallati, G. Anastasi, and S. K. Das, "Analysis and improvement of the on-the-fly bandwidth reservation algorithm for 6TiSCH," in *Proc. IEEE 19th Int. Symp. World Wireless, Mobile Multimedia Netw.*, 2018, pp. 1–9.
[25] X. Vilajosana, K. Pister, and T. Watteyne, "Minimal IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) configuration," Internet Engineering Task Force, Wilmington, DE USA, Rep. RFC 8180, May 2017.
[26] O. Gnawali and P. Levis, "The minimum rank with hysteresis objective function," Internet Engineering Task Force, Wilmington, DE USA, Rep. RFC 6719, Sep. 2012.

**Alakesh Kalita** (Member, IEEE) received the B.Tech. degree in computer science and engineering from Assam Don Bosco University, Guwahati, India, in 2012, the M.Tech. degree in computer science and engineering from Assam University, Silchar, India, in 2016, and the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Guwahati, Guwahati, India, in 2022.

He is currently a Postdoctoral Research Fellow with the National University of Singapore, Singapore. His research interests include Internet of Things and edge/cloud computing.

Dr. Kalita was the recipient of Best Project (Thesis) Award for his Ph.D. thesis from Indian National Academy of Engineering (INAE) in 2022.

**Mohan Gurusamy** (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Madras, Chennai, India, in 2000.

In 2000, he joined the National University of Singapore, Singapore, where he is currently an Associate Professor with the Department of Electrical and Computer Engineering. He has about 220 publications to his credit, including two books and three book chapters in the area of optical networks. His research experience and interests are in the areas of Internet of Things, 5G networks, software-defined networks, network function virtualization, cloud computing, data center networks, and optical networks.

Dr. Mohan was a TPC Co-Chair for several conferences and was an Editor for IEEE TRANSACTIONS ON CLOUD COMPUTING and is serving on the editorial board for *Computer Networks* (Elsevier) and *Photonic Network Communications* (Springer).