Received 19 July 2023; revised 14 August 2023; accepted 26 September 2023. Date of publication 2 October 2023; date of current version 17 October 2023.

Digital Object Identifier 10.1109/OJCOMS.2023.3321405

Autonomous Flow-Based TSCH Scheduling for Heterogeneous Traffic Patterns: Challenges, Design, Simulation, and Testbed Evaluation

ANDREAS R. URKE [1,2], ØIVIND KURE [3], AND KNUT ØVSTHUS [2]

[1]Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology, 7491 Trondheim, Norway

[2]Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, 5020 Bergen, Norway

[3]Faculty of Mathematics and Natural Science, University of Oslo, 0316 Oslo, Norway

CORRESPONDING AUTHOR: A. R. URKE (e-mail: andrerur@stud.ntnu.no)

ABSTRACT The Industrial Internet of Things needs wireless communication with bounded latency and stronger robustness. Nodes employing the Time Slotted Channel Hopping (TSCH) MAC operate according to a schedule, and recent work on flow-based autonomous schedulers has shown they can guarantee dedicated resources to each flow of traffic. However, these works assume all nodes transmit toward one destination. Industrial applications such as process control require heterogeneous traffic patterns, e.g., for sensor-to-actuator. We investigate how autonomous flow-based scheduling may support heterogeneous traffic patterns. We have previously proposed the Layered scheduler that emphasized flow scheduling and spatial reuse. In this work, we extend Layered to support heterogeneous traffic patterns. The extension includes a novel mechanism where the first application traffic packet is sent in a shared cell to inherently signal the need for scheduling dedicated cells. In adapting to heterogeneous traffic patterns, we encountered seven challenges. These include, e.g., the schedule adapting to packets later found as invalid at the routing layer and MAC queues leading to packets signaling outdated routing information to neighbors. We identify a set of mitigations and key parameters to address these challenges, and we evaluate their impact using the Cooja simulator and the FIT IoT-LAB testbed. The mitigation mechanisms are essential to ensure predictable performance under all conditions. Shared cell capacity was crucial as insufficient capacity can have a detrimental impact. Lastly, the scheduler was compared to the autonomous scheduler Orchestra. In scenarios with heterogeneous traffic patterns, we found the extended Layered scheduler retained performance independent of the number of flows. However, it comes at the cost of energy per goodput. Compared to Orchestra, Layered requires approximately twice the energy to maintain the schedule, yet Layered's higher capacity allows for comparable efficiency as application traffic increases.

INDEX TERMS TSCH, autonomous scheduling, IIoT, IEEE 802.15.4, MAC.

I. INTRODUCTION

As an enabler of concepts such as Industry 4.0, the Industrial Internet of Things (IIoT) must supply reliable wireless- and IPv6-connectivity to a large number of devices. Realizing these requirements require research efforts on all parts of the network stack. For media access control (MAC), the Time Slotted Channel Hopping (TSCH) approach has shown promise as it allows for reservation-based contention-free resource allocation [1]. Nodes in a TSCH network operate according to a schedule that dictates transmission and reception opportunities. The schedule is built by a scheduler, which operates in a centralized, collaborative, or autonomous fashion, or in a combination of these [2].

Autonomous schedulers work without any dedicated communication between nodes. This improves fault tolerance and reduces overhead and complexity, yet it may be challenging to construct schedules as optimal as centralized

or collaborative schedules. Autonomous schedulers may be advantageous, e.g., during network bootstrap, as a fall-back option, or when requirements are less stringent.

Autonomous schedulers can be categorized based on whether they assign cells to nodes, links, or traffic flows. In [3], we argued that flow-based schedulers benefit industrial applications as they can guarantee resource allocation end-to-end for each traffic flow. In addition, they provide flexibility such as scheduling multiple flows from one node, and may open for flow concepts as in IETF's Deterministic Networking (DetNet) [4].

To our knowledge, all proposed autonomous flow-based schedulers are designed for convergecast traffic, i.e., where all nodes transmit to one destination, typically the network root node. Such traffic patterns are common when monitoring industrial processes. However, the pattern does not match, e.g., control applications with sensor-to-actuator traffic. In this work, we study the challenge of supporting heterogeneous traffic patterns in autonomous flow-based scheduling.

II. BACKGROUND

A. APPLICATIONS

Concepts such as Industry 4.0 and Cyber-Physical Systems envision a wide range of applications that the IIoT is required to support. These include, e.g., monitoring, process- and factory-control, asset tracking, safety alarms, etc. Typical requirements for industrial applications include reliability, bounded latency, low latency, energy efficiency, fault tolerance, scalability, resource utilization, and more [5], [6]. It is not feasible to optimize for all these requirements simultaneously. The priority and rigorousness of each requirement depend on the application, with, e.g., closed-loop control being stringent on reliability and latency, while monitoring applications may focus on energy. Our design does not target a particular application, but its objectives are for reliability and bounded latency while minimizing channel usage.

The traffic characteristics in the industrial domain are diverse. Monitoring applications are more simplistic as all sensors periodically transmit in a convergecast traffic pattern to one more data collection nodes. However, industrial applications may also include peer-to-peer traffic such as control traffic between controllers and actuators, or measurements from sensors directly to actuators [7]. The traffic intensity may also vary as periodic measurements are mixed with alarms and control actions.

Our work focuses on heterogeneous traffic patterns. The design of our scheduler permits heterogeneous traffic intensity, yet we limit our evaluation to periodic transmissions.

B. 6TISCH

To meet the requirements of the IIoT, IETF formed the working group: "IPv6 over the TSCH mode of IEEE 802.15.4e" (6TiSCH).[1] Figure 1 shows the 6TiSCH network stack,

1. https://datatracker.ietf.org/wg/6tisch

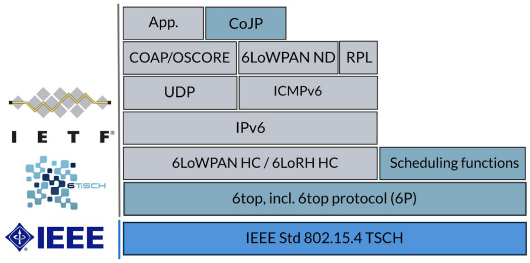


FIGURE 1. The 6TiSCH protocol stack, defined in RFC 9030 [8]. Novel 6TiSCH parts in cyan.

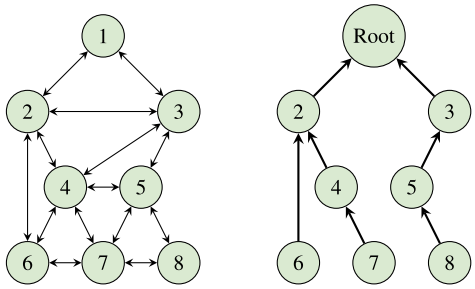


FIGURE 2. Link- and routing layer topologies. All available links on left, resulting RPL topology on right.

which combines IPv6-enabling upper layers with industrial-targeting wireless link- and physical-layers. We will describe the 802.15.4 TSCH MAC and RPL routing protocol which is crucial to our work. For further details on 6TiSCH, a tutorial may be found in [9].

C. ROUTING: RPL

The "IPv6 Routing Protocol for Low-Power and Lossy Networks" (RPL) [10] is designed for low-power and lossy networks (LLNs) where constrained nodes are interconnected by links with low reliability and data rates. RPL organizes the network into a Destination Oriented Directed Acyclic Graph (DODAG) which is rooted at an RPL root node. The DODAG is built by nodes broadcasting DODAG Information Object (DIO) messages, allowing recipients to select an appropriate preferred parent node and maintain a default route upwards. Downward routes are learned by nodes transmitting Destination Advertisement Object (DAO) messages upwards towards the root.

Figure 2 illustrates how the links (on the left) are utilized to build the RPL topology (on the right). Note how this tree requires traffic to travel upwards to a common parent before going downward to the destination. E.g., when node 6 needs to send to 4, the traffic would be transmitted via node 2.

A tutorial on RPL, and its interplay with TSCH, may be found in [11].

D. MAC: TSCH

The Time-Slotted Channel Hopping (TSCH) MAC realizes links for the network layer to utilize. Figure 3 shows a simple 4-node network and its corresponding TSCH schedule. The schedule has repeating slotframes containing cells that dictate

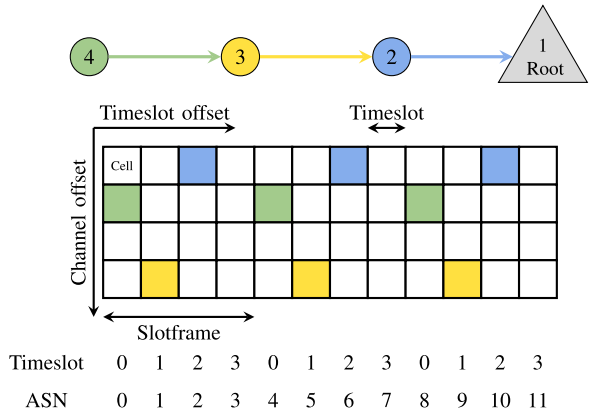


FIGURE 3. 4-node network with TSCH schedule.

transmission- and reception opportunities. A cell provides for exchanging a packet and an optional acknowledgment at a given time- and channel offset. The yellow-colored cells in Figure 3 illustrate a dedicated transmission opportunity for node 3, while the blue allows node 2 to forward a packet to the root node. Empty cells let nodes sleep, thus enabling energy-efficient operation.

TSCH is favored in industrial applications because the time-slotted approach opens for the deterministic allocation of dedicated resources. Further, its channel hopping scheme has been shown to significantly increase link reliability [12]. TSCH is widely adapted in standards, e.g., the industrial WirelessHART [13] employs a TSCH approach, while 6TiSCH selected the IEEE 802.15.4 TSCH [14].

The slotframe content is maintained by a TSCH scheduler. Schedulers typically operate in a centralized, collaborative, or autonomous fashion [2]: Centralized schedulers have a view of the entire network and may create highly optimized schedulers, yet with significant overhead when collecting information and dispersing schedules. Collaborative schedulers let nodes negotiate schedules between themselves, allowing for dynamic scheduling at the cost of increased complexity and signaling. Lastly, autonomous schedulers do not exchange information and build schedules typically by leveraging existing data from the link- and routing layer. This may be advantageous for scalability and fault tolerance since no additional signaling is required to establish and maintain links. However, since a node has limited information concerning its surroundings, it is challenging to autonomously construct an optimal schedule in terms of key metrics such as reliability, latency, and energy efficiency.

III. RELATED WORK

We proposed the Layered autonomous flow-based scheduler in [15], while Oh et al. proposed Escalator [16]. To our knowledge, these are the only flow-based autonomous schedulers. Their operation is similar, yet Escalator optimizes for short latency while Layered minimizes the number of channels needed. However, both schedulers are designed for a convergecast traffic pattern where all nodes transmit to the network root node.

Orchestra [17] is considered the state-of-the-art autonomous scheduler. It assigns a fixed amount of cells to each node in the network. As such, it inherently supports heterogeneous traffic patterns. However, since the cells are uniformly allocated in the network, the scheduler is vulnerable to traffic concentrations, such as funneling effects close to the network root. These issues are tackled by over-provisioning cells, which increases slotframe length and may waste energy - and provides only a probabilistic solution.

The autonomous ALICE [18] scheduler addresses some of the shortcomings of Orchestra by assigning cells to links instead of nodes. This improves the performance for heterogeneous traffic patterns since each link, in each direction, has a dedicated cell. However, the amount of cells is the same for each link, and ALICE thus suffers the same issues with traffic concentrations and over-provisioning as Orchestra.

Both [19] and [20] treat the limitations of schedulers that produce static schedules and survey *adaptive* autonomous schedulers. These schedulers aim to support varying topologies and/or traffic patterns by dynamically adapting the schedule. ATRIA [21] targets industrial applications and lets each node autonomously implement a schedule based on the network topology and fixed traffic intervals. The scheduling algorithm is optimized to reduce collisions, yet conflict-free cells can not be guaranteed, which may be a challenge if, e.g., bounded latency is required. Similarly, A3 [22] lets each node estimate the number of cells needed but does so solely by observing transmission attempts. As with ATRIA, conflict-free allocation cannot be guaranteed.

TESLA [23] and OST [24] aims to make Orchestra more adaptable. Using different techniques, neighboring nodes adjust their schedule in response to the current traffic load. However, to synchronize schedules between neighbors, both schedulers require nodes to signal each other by piggybacking information on acknowledgments, control traffic, or application packets. As such, we do not consider OST and TESLA strictly autonomous schedulers.

Kherbache et al. [25] provide an overview of nine different schedulers and their support for heterogeneous traffic. The survey includes Orchestra, ALICE, and OST, which were evaluated through simulations. The evaluation included nodes that generate traffic with different constant intensities, yet the traffic pattern was limited to convergecast towards the network root.

Collaborative schedulers inherently support heterogeneous traffic patterns since neighboring nodes may negotiate to adapt their schedule as traffic requirements change. A notable mention is YSF [26], which schedules for traffic flows.

The interested reader may find our comprehensive survey of TSCH schedulers in [2].

IV. PROBLEM STATEMENT AND CONTRIBUTION

In an industrial context, supporting heterogeneous traffic patterns is crucial, yet they are inherently more challenging due to their unpredictable nature. E.g., in a sensor-to-actuator

application, the location of sensors and actuators in the RPL tree is not known before deployment. Further, it may not be known to which actuator each sensor transmits or the ratio between sensors and actuators.

TSCH schedulers have to a large extent been designed for convergecast traffic patterns with constant packet rate; 70 out of the 76 schedulers surveyed in [2] were either designed or evaluated solely for convergecast. To our knowledge, no autonomous scheduler addresses heterogeneous traffic patterns while guaranteeing resources to every traffic flow. We aim to address this gap and widen the industrial applicability of autonomous scheduling.

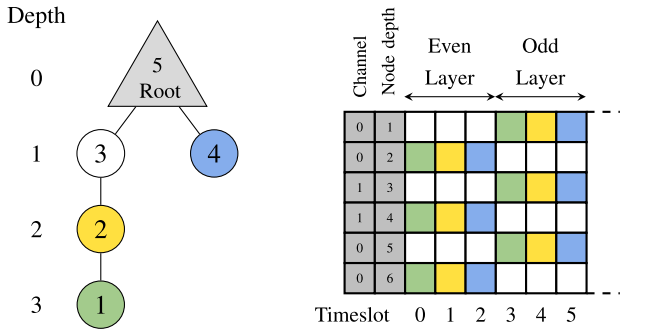
Our contribution is an investigation into the challenges facing an autonomous flow-based scheduler that supports heterogeneous traffic patterns. We expand our Layered scheduler to conduct the investigations, yet we argue our findings are valid for any scheduler meeting a short list of assumptions. Using the extended scheduler, we show how heterogeneous traffic patterns may be supported and evaluate the resulting performance. In summation, our contribution is:

1) Extended the Layered scheduler to add support for heterogeneous traffic patterns and evaluate it in the FIT IoT-LAB testbed. To our knowledge, this is the first autonomous flow-based scheduler that supports other traffic patterns than convergecast. The modifications include a novel scheduling mechanism where application traffic sent in a shared cell is used as a signal to schedule a dedicated cell for future traffic.
2) Identified and described seven challenges met when supporting heterogeneous traffic patterns in an autonomous flow-based scheduler. For each challenge, we propose mitigations and show their impact using the Cooja simulator and FIT IoT-LAB.
3) Open-sourced implementation of all proposed mitigations and the modified version of the Layered scheduler. Available at https://github.com/arurke/layered-scheduler.

The remainder of this paper is organized as follows: First, we present the original Layered scheduler since this will be used as a basis for our proposed scheduler. Next, we discuss flow-based scheduling for heterogeneous traffic patterns and describe our design and the modifications needed on the Layered scheduler. Section VII presents the challenges we identified through our work with the scheduler and discusses potential mitigations. We next move on to evaluating the scheduler and the proposed mitigations: Section VIII describes our methodology and setup, while we present and discuss our results in Section IX before concluding.

V. ORIGINAL LAYERED SCHEDULER

We proposed the autonomous Layered scheduler in [15] and conducted testbed evaluations in [3]. Layered operates in a flow-based manner by allocating cells to traffic flows that are assumed destined for the RPL root. As dedicated cells are reserved at every hop from source to destination, performance is retained independent of the network topology or competing traffic flows.



(a) Network with 5 nodes

(b) Layered schedule with all possible dedicated TX-cells. Colors denote the traffic flow being served.

FIGURE 4. 5-node network with corresponding schedule.

A. SCHEDULE STRUCTURE

Layered divides the slotframe into two layers, both containing one dedicated timeslot per traffic flow supported. Each layer is designated to be used by nodes at either odd or even depth.[2] Thus, packets in all flows are forwarded one hop after a layer has been executed. Figure 4 shows an example network where 3 sensor-nodes generate traffic flows (nodes 1, 2 and 4). The right side shows the schedule structure and all cells that may be scheduled. The example topology is only 3 hops deep, yet the Layered structure supports networks of any depth, with Figure 4(b) showcasing the first 6 hops. The green cells are reserved for the flow from node 1. Since node 1 is at the odd depth of three hops, the cell at timeslot 3 and channel 1 will be utilized when a new packet is generated. When node 2 (at depth 2) forwards onward, the green cell at timeslot 0 and channel 0 will be utilized.

Both node 1 and node 3 will utilize timeslot 3 when forwarding node 1's flow. These nodes are only two hops apart and thus use different channels since they may interfere with each other. As the number of hops between two nodes increases, it is assumed a point is reached where they will no longer interfere. Nodes at such a distance may employ spatial reuse, i.e., the same cell being used by two or more nodes simultaneously. The Layered schedule in our example employs two channels which yield spatial reuse for nodes four hops apart at depth 1 and 5, depth 2 and 6, and so on. If needed, the distance may be increased by two hops per additional channel.

B. SCHEDULING CELLS

To install the correct cells in its schedule, a node needs to know 1) Its own depth in the routing tree, which may be learned from RPL, and 2) The IDs of flows passing through

2. Number of hops from the root in the RPL tree.

the node. We assume each node in our sub-tree generates one flow each, and we let every node's unique ID serve as flow identifier.[3] The last byte of each node's IP address reflects the node ID. To learn the descendant's IP address and thus the flow identifiers, nodes inspect the originator address of RPL DAO messages that all nodes send to the root. This is a feasible approach because the DAOs will travel the same upwards path as the traffic flows.

The timeslot offset of a flow is decided by a hash function that maps the flow identifier to a timeslot. In our usage, the hash outputs the ID minus 1, such that ID 1 yields timeslot 0, ID 2 yields 1, and so on. The channel offset is decided by the node's depth and the configured number of channels as illustrated in Figure 4(b). Formal descriptions of the cell offset calculations may be found in [15].

Using Figure 4 as an example of the scheduling: When node 2 receives a DAO from node 1, it uses its own depth (2) and the DAO originator's ID (1) to install cells for node 1's traffic flow. An RX cell for traffic from node 1 is installed at timeslot 3, channel 1, and a dedicated TX cell to forward upwards is installed at timeslot 0, channel 0. This process is repeated at every hop until the root node, providing each flow with contention-free cells end-to-end.

Shared cells are inserted at fixed intervals through the slotframe, e.g., at every 7th timeslot. These have been omitted from our examples for improved clarity. Share cells serve RPL control traffic, TSCH beacons, and any downward traffic. The number of shared cells passed in the slotframe must be added when calculating the dedicated cell timeslot offset.

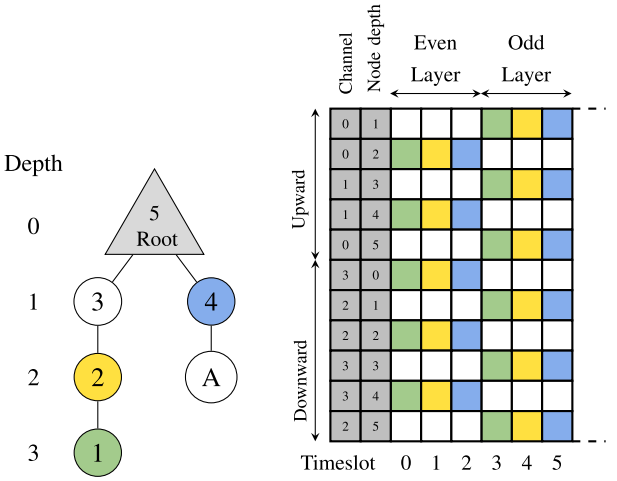
VI. DESIGN - AUTONOMOUS FLOW-BASED SCHEDULING FOR HETEROGENEOUS TRAFFIC PATTERNS

Schedulers must adapt to the flow of traffic through a network. With convergecast, the data flows upwards to the network root following the same path as RPL DAO messages. With heterogeneous traffic patterns, the flow will be upwards for some parts of the path and downwards for the remaining. There is also no network control traffic following the same path that may be exploited. Traffic concentrations depend on the routing topology and the application, e.g., the distribution of sensor and actuator nodes in the topology.

To meet the challenge of heterogeneous traffic patterns, two functions are missing in autonomous flow-based scheduling: Firstly, a *schedule structure* must be designed to accommodate cells dedicated to each flow across every link in the routing topology. Secondly, the *cell scheduling* must autonomously respond to the heterogeneous traffic pattern and schedule the correct cells.

Next, we discuss how these tasks may be solved and present a modified version of the Layered scheduler designed for heterogeneous traffic patterns.

3. Other flow identifiers may be envisioned such as the destination IP and IPv6 header flow label, see discussion in [3, Sec. III-C1].



(a) Network with 6 nodes

(b) Layered schedule with all possible dedicated TX-cells. Colors denote the traffic flow being served.

FIGURE 5. 6-node network with corresponding schedule.

A. SCHEDULE STRUCTURE

The Layered schedule structure must be modified to support a heterogeneous traffic pattern. For the upwards direction (towards the root), the existing structure can be reused without changes. To support the downward direction, we duplicate the upward schedule and create a second mirroring part. The two parts do not conflict as channels separate them: One set of channels is utilized in the upward direction and another in the downward. This leverages Layered's minimal channel usage, as the minimum amount of channels needed is doubled to four.[4]

Figure 5(a) shows an expanded topology with 3 sensor-nodes transmitting to an actuator node marked "A". The new Layered schedule structure for this network is shown in Figure 5(b). It is identical to a regular Layered schedule but includes a new downward section that utilizes channels 2 and 3, while the upward uses channels 0 and 1. Note that the upward cells for depth 0 are omitted since the root node cannot transmit upwards.

The regular Layered algorithms can be used to calculate timeslot and channel offset,[5] except for the channel offset, which needs an additional term (marked in bold below) to accommodate downward cells:

$$CH(n) = \left(\left\lfloor \frac{depth_n - 1}{L} \right\rfloor \bmod CH_{dir} \right) \mathbf{+(Dir * CH_{dir})} \quad (1)$$

where $depth_n$ is the node depth in the RPL tree, L is the number of layers, CH_{dir} is the number of channel offsets utilized per direction, and lastly Dir equals 0 for the upward direction and 1 in the downward direction. In other words,

4. The common IEEE 802.15.4 physical layer in the 2.4 GHz band offers 16 non-overlapping channels.

5. See [15, Table 1].

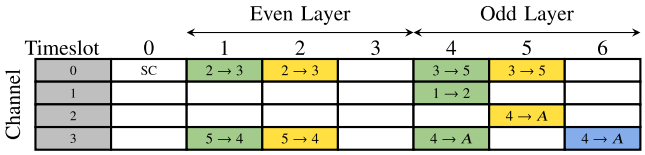


FIGURE 6. Runtime schedule for network in Figure 5(a). Colors denote the traffic flow being served. SC = Shared Cell.

when scheduling for the downward direction, channels are offset by CH_{dir}.

B. SCHEDULING CELLS

Learning which cells to schedule in networks with heterogeneous traffic patterns typically requires a form of signaling between neighbors. Collaborative schedulers may use protocols such as the 6TiSCH 6P protocol [27], which allows nodes to negotiate the cells to be scheduled. However, autonomous approaches without dedicated communication are also feasible by exploiting existing information such as the neighbor- or routing-table and other traffic. RPL DAO messages can however not be exploited, as done by Layered and Escalator, since the application traffic does not necessarily follow the same path as any control messages.

We propose a novel autonomous mechanism where the application traffic is exploited for signaling between neighbors: If a flow does not have a dedicated cell for a traffic flow, the first packet is transmitted in a shared cell. Upon receiving this packet, the neighbor reserves a dedicated RX cell. When the transmitter receives the packet acknowledgment, it reserves a corresponding dedicated TX cell. Subsequent transmissions on this link will utilize the dedicated cells. Our approach is analogous to how Ethernet switches map MAC addresses to ports: The first packet is flooded, and the reply is used to ensure the correct port/resource is used for future transmissions.

Figure 6 shows the cells that will be reserved as the network in Figure 5(a) operates. Note firstly how the first timeslot is utilized by a shared cell. When calculating the dedicated cell timeslots, one offset must therefore be added.

Using the flow from node 1 as an example: The first application packet is sent in the shared cell. As node 2 receives the packet, it uses the source IP address to learn the flow identifier. Based on its own depth, node 2 knows the transmitter is on depth 3 and that the packet is traveling upwards. Node 2 can now calculate the RX cell for node 1's flow and add it at timeslot 4 and channel 1. As node 1 receives the packet acknowledgment, it does the same calculation using its own depth and adds a TX cell.

When the packet has progressed to the root node, the direction shifts downward. Node 4 will notice this as the packet comes from its parent node. The only difference is the channel calculation where the offset described in Equation (1) must be added, yielding an RX cell at timeslot 1 and channel 3. No spatial reuse is employed in this example, as the network would need to be 5 hops deep.

Note that a traffic flow always has the same source node. If actuator A wanted to reply to any sensor nodes, that would constitute an entirely separate traffic flow that needs its own scheduling. It is not possible to re-use reservations for a flow in the reverse direction without encountering scheduling collisions.

With the original Layered, the schedule convergence is fixed to the convergence of RPL. Once the identity of the nodes downstream is known via DAO messages, the schedule is given. For the modified Layered, the schedule is built as the flow uses a shared cell once and next migrates to a dedicated cell. As the topology or traffic path changes, the process is repeated to adapt the schedule. With shared cells, e.g., every 7th timeslot, convergence for one flow is expected in less than 70 ms,[6] yet depends on the shared cell capacity, as discussed later.

Both application traffic and RPL and TSCH control traffic will utilize the same shared cells. We opted for this approach as it increases cell utilization and thus allows for fewer shared cells and a shorter slotframe, reducing latency and increasing capacity. However, it opens for application traffic to impact the network control since the transmissions may interfere. It is, therefore, critical to ensure sufficient shared cell capacity. Alternative schemes may be considered: Using a separate set of shared cells for application traffic would remove the network control coupling. However, this would reduce cell utilization and increase the slotframe length. More complex schemes could involve using specific channel offsets at different DODAG depths to reduce the collision domains.

C. ASSUMPTIONS

The scheduler outlined above rests on a small set of assumptions, and we, therefore, argue that our findings hold for any scheduler following these assumptions:

1) RPL in storing mode is utilized as the routing protocol.
2) The schedule structure can accommodate a dedicated TX and RX cell for each traffic flow across every link in the RPL DODAG.
3) Dedicated cell timeslot and channel offsets are calculated based on the packet source IP address, the current depth of a node in the DODAG, and the packet direction (upward or downward).
4) Application traffic can intermittently use shared cells to build the schedule.

VII. CHALLENGES AND MITIGATIONS

Based on the scheduler described in the previous section, we identified seven challenges. We categorize them according to their origin: 1) Common challenges that are found in most TSCH schedulers yet are exacerbated in our use-case, 2) Challenges specific to flow-based scheduling, and 3) Challenges specific to heterogeneous traffic patterns. Table 1 overviews these challenges and their mitigations, while we describe the details below.

6. Assuming 10 ms timeslots which are typical at 250 kbps datarate.

TABLE 1. Challenges and mitigations.

Origin	#	Challenge	**Mitigations** (applied mitigations marked in **bold**)
Common	A.1	Link metric volatility	**MRHOF**, **low-pass filter**, cell-agnostic metrics, cell-aware link ETX
	A.2	Shared cell capacity	**Capacity testing**, **TSCH parameter tuning**, QoS mech., adaptable schedule
	A.3	Neighbor schedule synchroniz.	**Cell timeout mechanism**, **cautious add/remove of RX cells**
Flow-based scheduling	B.1	Routing table synchronization	**Schedule changes at each routing table event**
	B.2	Forwarding outside the DODAG	**Forwarding-inconsistency mechanism**, RPL parameter tuning
Heterogeneous traffic pattern	C.1	Outdated info. in queued packets	**Schedule quarantine timer**, ingress queuing
	C.2	Network-layer processing	**Cross-layer mechanism**

A. COMMON CHALLENGES

1) LINK METRIC VOLATILITY

RPL makes decisions based on link metrics such as the estimated transmission count (ETX). Links may, however, be implemented by a mix of dedicated and shared cells with significantly different ETX. The neighbor link metrics may therefore fluctuate as the scheduler remove and add dedicated cells. These fluctuations may trigger undesirable RPL topology changes.

A straightforward solution is to use metrics that are agnostic to the particular cell, such as hop count or energy mentioned in RFC 6551 [28]. If ETX is preferred, an option is to employ the Minimum Rank with Hysteresis Objective Function (MRHOF) [29], which aims to reduce parent switches when metric changes are small. Further, applying a low-pass filter to the ETX calculations, as done with a moving average in the Contiki-NG operating system [30], may smooth out short-lived changes. Lastly, more complex solutions could be envisioned, such as making the link ETX calculation cell-aware by weighing or ignoring the ETXs of particular cells.

2) SHARED CELL CAPACITY

Estimating the appropriate shared cell capacity is difficult since, in addition to serving control traffic, the shared cells must also serve the application traffic needed to build the schedule. Unfortunately, the increased need for shared cells may be synchronized between the network control and scheduler: For example, network topology changes increase the routing protocol traffic, while the scheduler simultaneously needs shared cells to schedule dedicated cells to the new routes. Similarly, at the application-level, shared cell capacity may be strained if several nodes start transmitting simultaneously, e.g., due to events or alarms.

Ensuring appropriate shared cell capacity is a difficult balance act: Fewer cells reduce energy consumption yet increase contention and may lead to prolonged RPL, TSCH, and scheduling convergence. Conversely, adding more shared cells improves capacity yet increases energy consumption and lengthens the slotframe, increasing latency and reducing overall throughput. 6TiSCH RFC 9033 [31] recommends that broadcast traffic among a node and its neighbors not exceed one-third of the available capacity. However, identifying the amount of traffic before deployment is challenging since the radio environment unpredictably influences link topology, spurs retransmissions, and control traffic, etc.

Quality of Service (QoS) mechanisms such as traffic shaping may be applied to ensure the shared capacity is not overwhelmed. Similar techniques are mentioned in RFC 9033, pointing to the Trickle algorithm [32] used to control RPL DIO message intervals. Further, prioritization could mitigate issues by emphasizing network control and critical applications. In our flow-based queuing implementation,[7] the shared cells first serve the RPL and TSCH beacon queues before serving the application. Parameters in 802.15.4 TSCH may also be tuned, including the number of retransmissions and the maximum backoff after a failed transmission attempt.

More complex mechanisms could be envisioned where the schedule dynamically adjusts to the traffic needs. Schedulers TESLA [23] and OST [24] address this issue by changing the slotframe length based on traffic load. However, they are unable to achieve this autonomously as both schedulers require information to be embedded into packets exchanged between neighbors.

A pragmatic approach involves conducting tests as the network is deployed to identify an appropriate amount of shared cells. In our evaluations, we utilized this approach in combination with tuning TSCH parameters.

3) NEIGHBOR SCHEDULE SYNCHRONIZATION

Neighboring nodes must have corresponding TX and RX cells for transmissions to succeed. Latency, reliability, and energy consumption suffers as nodes transmit or listen in vain. Further, it may impact the routing protocol as transmission failures degrade the link's ETX. As the scheduling decisions for flow-based heterogeneous patterns are more complex than typical autonomous schedulers, care must be taken for schedules to stay synchronized.

We found it necessary to add a cell timeout: If a cell is left unused for a configurable time, it is automatically deallocated. This is needed to remove stale cells but also ensures the cleaning of erroneous reservations.

In addition, we adopted a rule of thumb to be cautious about removing RX cells. This is crucial since the neighbor

7. See [3, Sec. IV-B] for details.

is unaware of the removal, causing failed transmissions. The same guideline applies when adding cells - we are wary of restricting the scheduling of RX cells. One example is the schedule quarantine mechanism described later, which restricts scheduling TX cells, but always allows RX cells.

B. CHALLENGES SPECIFIC TO FLOW-BASED SCHEDULING

1) ROUTING TABLE SYNCHRONIZATION

Dedicated cells are scheduled for a specific traffic flow. However, a routing table change may route a traffic flow to a different next-hop neighbor. The new next-hop, unaware of the changes, does not have dedicated RX cells for the flow, and the transmission will fail if the schedule is not adapted.

We identified a set of schedule changes to be executed for each routing table event. Following our discussion on neighbor schedule synchronization, note how we restrict removing RX cells to avoid unnecessary transmission failures.

- Default route change: Remove all upwards TX cells, as all are directed to our old default route. Downward TX links are preserved since our child nodes remain unchanged.
- Route added: Remove all TX cells serving flows with the same destination but a different next-hop than the added route. This requires the scheduler to maintain an overview of the destination and next-hop of each traffic flow, which we argue is trivial.
- Route removed: Remove all TX cells serving flows with the same destination as the removed route. If this route was for a neighbor, it is no longer reachable, and all TX and RX cells toward the neighbor are removed.
- DODAG depth changed: Remove all TX cells. RX cells will be timed out or removed by the forwarding-inconsistency mechanism described below.

2) FORWARDING OUTSIDE THE DODAG

RPL allows traffic, in certain conditions, to be forwarded without following the DODAG - see Section 11.2.2.2 in RFC 6550 [10]. If the packet, e.g., crosses from one branch of the DODAG to another, it might be forwarded at the same depth twice. This may cause scheduling collisions and interference because the flow-based scheduling approach allocates one cell for each direction at *each depth*, as described in Section VI-A.

We propose a *forwarding-inconsistency mechanism* that detects and mitigates forwarding outside of the DODAG. To detect, a receiving node inspects each packet's IPv6 hop-by-hop header.[8] The header indicates the transmitter depth and the packet's intended direction (downward or upward). If this information does not correspond to the receiver's depth and direction, inconsistent forwarding is assumed. In response, the receiver poisons the link to the transmitter by removing the relevant RX cell. This will cause future transmissions to fail, increase the link ETX, and lead the transmitter to drop the erroneous route.

The inconsistency-mechanism is not without cost, as the poisoned link wastes energy on failed transmissions and may cause packet loss. The mechanism is, however, disabled for a period of time after any routing changes. This is done to avoid unnecessary poisoning in cases where 1) Ongoing topology changes cause temporary forwarding issues, and 2) To allow DIO messages with updated depth information to be dispersed.

Alternative solutions from the RPL perspective may also be considered: Shortening the route lifetime would reduce the duration of a problematic situation at the expense of energy since more frequent DAO messages are needed to refresh routes. Furthermore, RPL could be modified to be less tolerant of traffic outside the DODAG.

C. CHALLENGES SPECIFIC TO HETEROGENEOUS TRAFFIC PATTERN

1) OUTDATED INFORMATION IN QUEUED PACKETS

Since the scheduler reacts to incoming application packets, the traffic flow must reflect the current routing and scheduling state. Contiki-NG makes the routing decision before queuing the packet at the MAC layer. As queues grow, the routing decision might be outdated when the packet is forwarded. The packet thus inherently signals *outdated information* to the receiving scheduler.

This topic is discussed from the perspective of ad-hoc networks by Landmark et al. in [34]. They argue for ingress queuing, where packets are queued before any routing decision. The ingress queue is served only when there is room in the minimal egress queue at the MAC layer. Thus the routing decision is made shortly before packet transmission.

Keeping changes limited to the scheduler, we opted to add a *schedule quarantine timer*: For a configurable time after any routing changes, no TX cells may be added to the schedule. This aims to empty the queues of outdated packets before scheduling changes are allowed. Not adding TX cells increased shared cell usage during the quarantine time, causing drawbacks such as increased contention and prolonged time before the application could benefit from dedicated cells.

2) NETWORK-LAYER PROCESSING

Incoming packets are first handled at the link layer where the scheduler resides. The packet may cause scheduling changes, e.g., adding dedicated cells if an application packet is received in a shared cell. However, later processing in the network layer may reveal information relevant to the scheduling decision. Specifically, RPL may find the packet was forwarded incorrectly.[9] The scheduler should not accommodate such traffic.

8. RPL adds an IPv6 hop-by-hop header to every packet as part of the RPL loop avoidance and detection mechanism described in Section 11.2 of RFC 6550 [10], and in RFC 6553 [33].

9. See, e.g., Section 11.2.2.3 on DAO Inconsistency in RFC 6550 [10].

We addressed this issue in a direct manner by implementing a cross-layer mechanism where RPL notifies the scheduler when forwarding errors are identified. In response, schedule changes that were triggered by the erroneous packet are reverted. A general caution is needed for cross-layer mechanisms: They create couplings between functionality designed to be agnostic to each other's implementations. This opens a slippery slope where new mechanisms are required for each new protocol to be integrated, or restrictions are placed on operators regarding protocols that may be utilized. In our case, this is alleviated since employing RPL is a preexisting requirement for the scheduler to operate.

VIII. EVALUATION

A. METHODOLOGY

Reproducibility is crucial in scheduler research, and we, therefore, open-sourced our implementation and utilized openly available tools for analysis of the results. We preferred using the Cooja simulator from Contiki-NG v4.6 since simulations allow for rapid executions and detailed control. When a realistic radio environment was needed, we employed the FIT IoT-LAB [35] testbed. Further, we based our methodology on the TriScale approach [36]. TriScale aims to improve experiment replicability and provides a methodology and statistical toolchain which allows researchers to draw sound conclusions with quantifiable certainty. We briefly describe our methodology below and refer the interested reader to [36] for full details on TriScale.

Each experiment is repeated in a given number of *runs*. For each run, we calculate metrics such as latency, packet loss, and radio duty cycle. The resulting set of metrics is next used to calculate Key Performance Indicators (KPI). While the metrics describe each run, a KPI describes the underlying distribution from which the runs are drawn, i.e., the performance for *any* set of runs. The KPI is a one-sided confidence interval of a given percentile of the metric performance. In other words, the KPI provides a pessimistic[10] bound for a given percentile of the metric for any runs.

The number of runs decides the percentiles that are possible to calculate. As we target industrial applications, we aim for tail performance: For our simulations, we use 93 runs, which allows us to calculate the 95 percentile, with the worst run providing robustness against outliers. In this paper, we always use 95% confidence. Consequently, the simulation results are, with 95% confidence, at minimum, as presented, for at least 95% of any runs. Testbed experiments are time-consuming, and we, therefore, execute 14 runs. Thus, the testbed results are as presented, or better, for at least 70% of any runs.

We utilize the following metrics:

1) PDR: Packet delivery ratio. The number of packets received at the recipient application layer, to all generated packets.

10. E.g., higher for latency and energy, lower for packet delivery ratio.

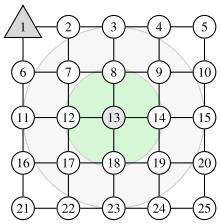


FIGURE 7. 5 x 5 grid network used in simulations. Green circle indicates transmission range, grey indicates interference range.

2) Latency: The time between a packet is generated at the sender application layer to it is received at the recipient application layer.
3) Energy: Radio duty cycle. The duration of which the radio is powered, to the experiment duration.
4) Dedicated cell ratio: The number of application data transmissions using dedicated cells, to the number of all transmissions (dedicated + shared). A high dedicated cell ratio indicates application traffic is forwarded in cells with a guaranteed capacity and without contention, as opposed to shared cells.

B. SETUP

In Cooja simulations, we employ a 5 x 5 grid network seen in Figure 7. Grids are a common topology used in scheduler evaluation and are ideal in our case as they yield a variety of routing topologies. The circles in Figure 7 illustrate the radio propagation model, where nodes within the green transmission range have perfect links if not interfered with by other transmissions.

The FIT IoT-lab testbed has different sites with differing properties, which we select according to our needs. The Grenoble site has nodes placed along hallways, typically yielding elongated line topologies. Strasbourg has nodes arranged in a grid and thus allows for more varying topologies.

We emulate a control application with sensors transmitting to actuators. This could be realized using, e.g., multicast, yet we employ unicast peer-to-peer. In a typical deployment, it will be known which nodes are sensors that transmit and which are actuators that receive. However, we aim for a more challenging situation by having all nodes transmit to random destinations. The traffic paths follow the RPL tree upwards to the common ancestor before going downward to the destination. In the simulation grid topology, the paths have a typical mean length of 5-6 hops, with a maximum length of around 10-15 hops. At boot, all nodes except the root node randomly select a destination node that it will transmit towards every 3 sec. The slotframe is typically 67 timeslots long, i.e., 0.67 sec. Thus there is more than four times over-provisioning to allow for retransmissions.

The scope of our work is limited to converged performance. The duration of transient states was found through experiments by noting when metrics reached a stable state. As such, application traffic transmissions starts after

TABLE 2. Experiment parameters.

Parameter	Value
Packet payload	24 bytes
RPL objective function	MRHOF w/ETX
TSCH packet burst	Disabled
TSCH max. backoff exponent	3
Queue size per neighbor, and flow (Layered)	16 & 8 packets
Max. links, neighbors & routes	64
FIT IoT-LAB board	M3
Physical channels Strasbourg	14, 15, 16, 17
Physical channels Section IX.E	14, 15, 20, 25, 26
Convergence wait Cooja, Grenoble, & Strasbourg	20, 12 & 30 min.
Layered shared cell spacing Grenoble, Cooja	Every 7th
Layered shared cell spacing Strasbourg	Every 5th
Layered cell timeout	2 x TX int. + 1 s

the initial TSCH and RPL convergence at 6 minutes and last throughout the experiment. The introduction of application traffic creates a new convergence period that depends on the setup and is listed in Table 2. After the convergence, data is collected for 10 minutes, with the last minute disregarded to avoid packets in flight. Note that the converged state also included RPL topology changes and scheduling adjustments. As such, the impact of any re-scheduling, such as parent switches and link failures, is included in our results. Table 2 lists all experiment parameters and all non-default settings.

C. COMPARISON WITH OTHER SCHEDULERS

We compare the performance of our modified Layered scheduler with the Orchestra scheduler briefly described in Section III. Orchestra is considered the state-of-the-art autonomous scheduler, it is widely used when comparing schedulers, and an open-source implementation is found in Contiki-NG. The original Layered scheduler is not eligible for comparison since it only supports convergecast traffic. As discussed in Section III, other autonomous schedulers aim for heterogeneous traffic patterns, yet unfortunately, no open-source implementations are available. An exception is found for ALICE,[11] yet the implementation has not been kept up-to-date with the operating system. We deem it necessary to run all schedulers on the same operating system version to ensure comparable results.

D. ISSUES IN CONTIKI-NG RPL IMPLEMENTATION

Through our work, we identified and mitigated three bugs and issues with the Contiki-NG RPL implementation. All are related to the handling of situations that are expected not to be common:

1) A sub-optimal handling of DAO inconsistency where No-Path DAOs were used to respond to forwarding errors. This is as opposed to setting the Forwarding Error 'F' bit as suggested in Section 11.2.2.3 in RFC 6550. This led to a prolonged time before a DAO inconsistency would be corrected.[12]
2) A bug in handling No-Path DAOs where the packet could be discarded erroneously. This would allow incorrect routes to remain in the routing table longer than necessary.[13]
3) A bug where No-Path DAOs would be incorrectly serialized on the transmitter. Thus, the receiver could not parse the message, and the route would remain in the routing table until timeout.[14]

In addition, we encountered and added a workaround for an RPL issue recently discovered in Contiki-NG. This could, in specific scenarios, lead to application packets being discarded although they were eligible for further processing.[15]

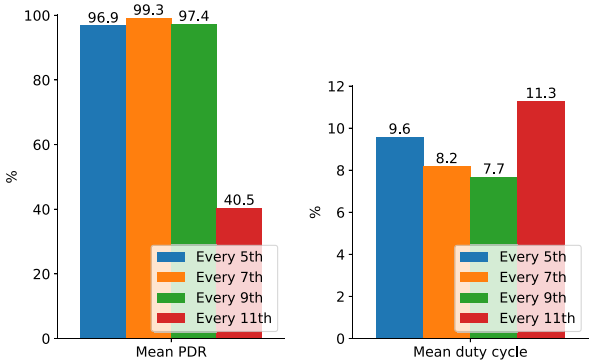


FIGURE 8. PDR and duty cycle for different shared cell intervals, in simulator.

IX. RESULTS

In the following, we evaluate the impact of our proposed mitigation mechanisms and key parameters (see Table 1). We do not investigate the volatile link metrics (challenge A.1) since it can be viewed as a general caution, nor the routing table synchronization (challenge B.1) since the mitigations were strictly necessary for the scheduler operation. We finalize this section with a brief investigation into scalability and a performance evaluation of the scheduler, comparing it to the Orchestra scheduler.

A. SHARED CELL CAPACITY

As discussed in Section VII-A2, the shared cell capacity is vital since the schedule convergence relies on it (challenge A.2).

We evaluate inserting shared cells at every 5th, 7th, 9th, or 11th timeslot. Figure 8 shows the PDR for the different intervals. Thus, with 95% confidence, for 95% of *any* runs,

11. See https://github.com/skimskimskim/ALICE.

12. See https://github.com/contiki-ng/contiki-ng/issues/2386.
13. See https://github.com/contiki-ng/contiki-ng/issues/2385.
14. See https://github.com/contiki-ng/contiki-ng/issues/2377.
15. See https://github.com/contiki-ng/contiki-ng/issues/2285.

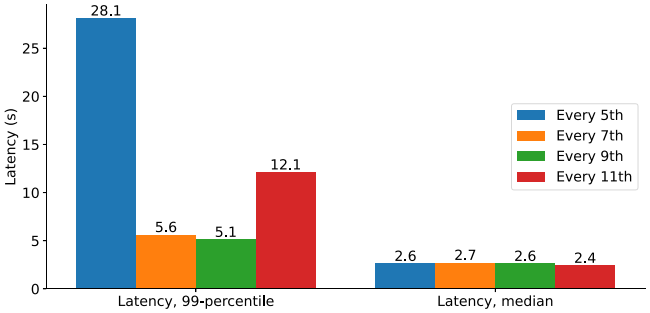


FIGURE 9. Latency for different shared cell intervals, in simulator.

the PDR is at least 99.3% with a shared cell every 7th timeslot. When decreasing to every 11th timeslot, the capacity is insufficient as the PDR drops to 40.5%. The drop is caused by a lack of scheduler and RPL convergence as transmissions of both application- and RPL-packets fail.

Figure 8 also shows the energy consumption illustrated through the radio duty cycle. Notably, the scenario with the least shared cell capacity, every 11th, yields the highest consumption. This is explained by topology instability which spurs significant control traffic. As expected, the remaining scenarios show that energy consumption decreases with the declining shared cell capacity. This illustrates the trade-off between capacity and energy consumption - increasing the shared cell interval from every 7th to every 5th timeslot increases consumption by ~17%.

Figure 9 shows how the shared cells' capacity influences the mean and 99-percentile latency. All scenarios have a median latency in line with the expectations: The slotframe is ~0.67 s long, and the mean path length is 5-6 hops. Each slotframe has two layers, with each layer advancing the packet one hop. Thus the minimum latency without retransmissions would be around 1.7 - 2 s. With shared cells every 7th and 9th timeslot, we note there is only a moderate increase in latency even at the extreme 99-percentile. Such robust performance is a desirable property in industrial applications.

However, we note a significant increase in 99-percentile latency when shared cells are inserted every 5th timeslot. This was not caused by the shared cell interval directly but by the cell timeout parameter (see Section VII-A3) being set too short. Re-running simulations with a longer timeout resolved the issue and demonstrated the importance of proper parameter settings.

In detail, we found that the link poisoning discussed in Section VII-B2 would fail. Links are poisoned by removing dedicated RX cells such that link ETX rises, and the neighbor is deemed unreachable. With a short cell timeout, a node would too quickly switch away from the poisoned dedicated cells and over to shared cells. The shared cells improve the link ETX since they had minimal interference when at every 5th timeslot. Thus, the link ETX never passes the unreachable threshold and the link loops between shared and poisoned cells until the route times out. Increasing the cell timeout reduces shared cell usage such that the link ETX increases sufficiently for the neighbor to be deemed unreachable.

Our key takeaway is the significant impact of shared cell capacity on performance. Care should be taken to ensure sufficient shared cells in the schedule. Secondly, more capacity is not always better, as seen in the increased energy consumption and latency. Lastly, it must be stressed how our specific capacity values are valid only in this setup since capacity requirements vary between deployments.

B. CELL TIMEOUT

To mitigate challenge A.3, neighbor schedule synchronization, we proposed a cell timeout mechanism in Section VII-B which removes a cell if no packets are successfully transmitted or received for a configurable timeout period. This is done to clear the schedule for no longer needed cells and to clean up any erroneous allocations. The duration of the period has several implications: A short period allows for faster cleaning up of stale allocations, thus reducing the energy consumption on idle listening. However, shorter periods also risk useful cells being removed in cases where, e.g., retransmissions earlier in a path cause a temporary halt in the traffic flow.

Since retransmissions are expected to have a significant impact, we opt to use the FIT IoT-LAB testbed, which provides a realistic radio environment. The network scale should not influence the results, and we, therefore, utilize 16 nodes at the Grenoble site.

All nodes in our experiment transmit to a random destination in 2.8-second intervals. We then vary the timeout values: The shortest timeout is at 3 seconds, i.e., only 200 ms more than the transmission interval. Thus, any retransmissions risk a cell being timed out and unscheduled along the flow path. The following scenarios increase the timeout to 4, 6, 9, and ultimately 15 seconds.

Using only the best performing channels at the Grenoble site,[16] we saw close to 100% packet reception rate (PRR)[17] across all links. In such an environment, the timeout values were of no consequence, with the PDR and dedicated cell ratio at ~100% for all values. This makes sense since cells can only time out when transmissions fail. To bring forward the timeout impact, we moved to use all available channels, yielding a PRR of around 70 - 90%. With the added volatility, tests showed an additional 8 minutes was needed for the network to converge. We also conduct 5 extra runs, which allows two runs instead of one to provide robustness against outliers.

Figure 10 shows the PDR and dedicated cell ratio for the different timeouts. The shortest timeout illustrates the impact of too aggressive removal of cells. Every retransmission will trigger a cell timeout and a corresponding use of shared

16. We identified the best channels as 18 - 21 in earlier work [3].

17. Packet Reception Rate measures the quality of a link by taking the number of received acknowledgments to the number of transmission attempts.

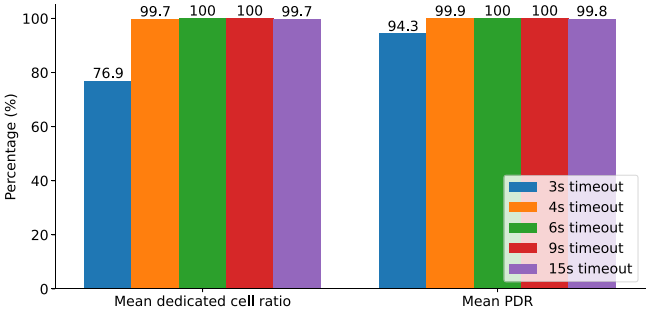


FIGURE 10. Dedicated cell ratio and PDR for different cell timeouts, in testbed.

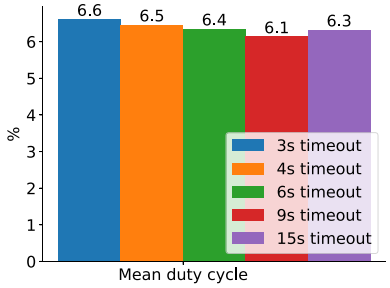


FIGURE 11. Duty cycle for different cell timeouts, in testbed.

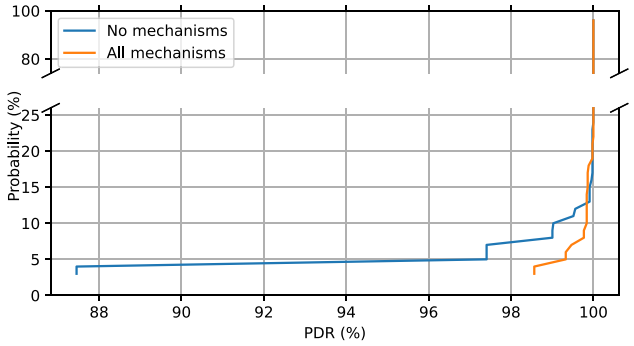


FIGURE 12. CDF of PDR, with and without mechanisms, in simulator. Note y-axis has a break to improve readability.

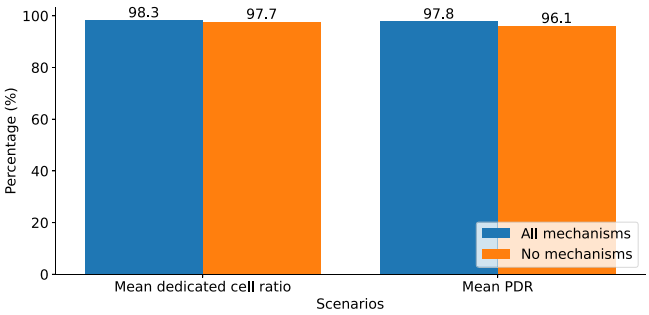


FIGURE 13. Dedicated cell ratio and PDR, with and without mechanisms, in testbed.

cells to re-schedule. As such, we see the PDR suffer, and only 76.9% of the application packet transmissions are done in dedicated cells. As the timeout increases, so does the robustness against retransmissions, and we see the PDR and dedicated cell ratio remain high. Any remaining cell timeouts are due to topology changes.

The duty cycles of each scenario are seen in Figure 11. Although the longer timeout risk stale allocations incurring idle listening, the effect is offset by the increased schedule stability, avoiding shared cell usage. This negation holds until the longest timeout, 15 s, where we see a slight increase in energy consumption. This could be explained by idle listening in lingering cell allocations.

In summation, the cell timeout does not significantly impact deployments where the PRR is close to 100%. In challenging environments with lower PRR, the timeout should be adapted to the deployment characteristics. In our setup, we found the timeout needed to be at least 1.5 times the transmission interval to ensure reliability and less than 5 times the interval to maintain energy efficiency.

C. MITIGATION MECHANISMS

In Section VII, we proposed a set of mechanisms to meet different challenges. These were 1) A *forwarding-inconsistency mechanism*, which tackles traffic being forwarded outside of the DODAG (challenge B.2), 2) The *scheduling quarantine timer*, which combats outdated routing information signaled by enqueued packets (challenge C.1), and 3) A *cross-layer mechanism* which informs the scheduler about packets deemed invalid by RPL (challenge C.2).

These mechanisms are primarily relevant for specific and rare cases. E.g., only in particular conditions may an RPL topology change cause traffic to be forwarded outside the DODAG or such that RPL recognizes packets as invalid. Therefore, the performance impact of each mechanism is difficult to quantify without designing specific experiments for each. However, when combining all mechanisms, the effect becomes more pronounced. Consequently, we compared scenarios with all mechanisms applied to those without any.

Through simulations, we found the mechanisms have minimal impact on the dedicated cell ratio yet improve the minimum PDR by ~2%. As mentioned, the mechanisms are applicable only under certain conditions, and their impact is therefore found in the more extreme percentiles. Figure 12 illustrates this point by showing the cumulative distribution function of the PDR for percentiles 3 to 97. Note how approximately 80% of runs have a PDR close to 100% regardless if mechanisms are applied or not. For the remainder, we see the mechanisms can avoid and reduce PDR degradation. Similar results were seen for latency.

The simulation has perfect radio conditions, and we, therefore, corroborate our results by employing 26 nodes at the FIT IoT-LAB Strasbourg site. The testbed radio environment has a mean PRR between 70-90% and is expected to more frequently produce situations addressed by the mechanisms. Figure 13 shows the minimum PDR and dedicated cell ratio for 70% of runs. Whereas the simulations had no difference at the 80-percentile, the testbed environment shows an ~1.7% increase already at the 70-percentile. A slight improvement in the utilization of dedicated cells was also found, indicating the scheduler spends more time converged.

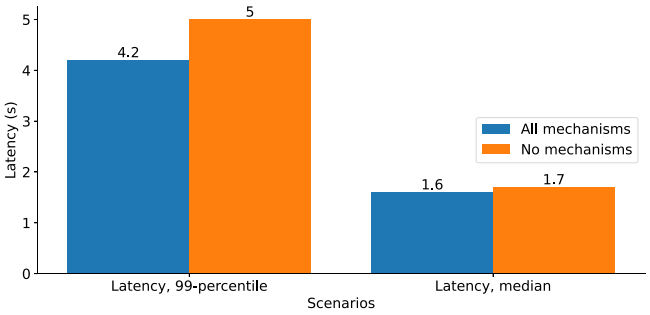


FIGURE 14. 99-percentile and mean latency, with and without mechanisms, in testbed.

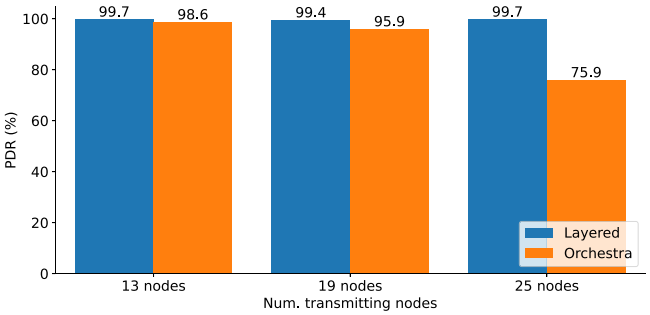


FIGURE 15. PDR for different traffic intensities, in testbed.

Figure 14 shows a similar story for the latency. Applying the mechanisms mitigates the rare cases, as demonstrated by the 16% decrease in 99-percentile latency.

The takeaway from our investigation is that the proposed mechanisms do indeed improve performance, yet the conditions for which they are applicable may be rare. As such, the mechanisms provide operators with an option for increased robustness and predictable performance. These properties are desirable in, e.g., industrial control applications where the worst-case behavior is more critical than the mean behavior. The necessity of the mechanisms depends on the deployment site since the mechanism's impact is more significant in challenging radio environments. Some mechanisms have drawbacks, such as the cross-layer mechanism, which creates couplings between different layers in the network stack. Thus, the benefit must be weighed against the disadvantages for each deployment.

D. SCALABILITY

We briefly investigate scalability by expanding our Cooja grid network to 9 x 9. The size of each layer in the Layered schedule was increased to 81 to accommodate all flows. Together with the shared cells, the slotframe length was thus 202 timeslots. Consequently, the schedule can accommodate 1 new packet every 2.02 seconds in each flow. We therefore configured each node to generate a packet every 9 seconds, yielding practically identical over-provisioning as earlier simulations.

As the network grows, the need for shared cell capacity may increase and must be considered, i.e., challenge A.2. In our setup, we needed to increase the number of shared cells from every 7th to every 5th timeslot. Secondly, we increased the max. TSCH backoff exponent from 3 to 5 to further spread retransmissions in time.

Results showed the PDR was retained at 99.7% with a mean duty cycle at 8.3%. This is comparable to the 99.3% PDR and 8.2% duty cycle seen for the smaller 5 x 5 network scenario in Section IX-A.

Latency increased significantly, with a median latency at 13.5 s and 99-percentile at 27.1 s. This is, however, expected due to the design of Layered: Firstly, the slotframe grows by 2 timeslots for every flow supported, which increases latency and reduces the throughput per flow. Secondly, Layered forwards flow packets one hop per layer. With two layers in a slotframe, each hop adds 1.01s to the latency. Larger networks with longer paths will therefore see the maximum latency increase. In our simulations, the flow path mean length grew to 11 hops, with the typical maximum length at 25 hops - corresponding to the measured latencies. This property is the cost of Layered's optimization towards minimal channel usage [15].

An option to limit the maximum latency is to segment the deployment into several networks of fewer nodes and shorter paths. Layered favors such a solution due to its minimal channel usage allowing for co-located deployments.

E. PERFORMANCE COMPARISON

We compare the performance of our modified Layered scheduler with the Orchestra autonomous scheduler. Orchestra assigns a fixed amount of cells to every node. This allows Orchestra to support any traffic patterns as long as the traffic concentration is below its capacity. We, therefore, utilize Orchestra as a baseline where applicable and do not consider its suitability to Layered. The Strasbourg site of FIT IoT-LAB is employed as it offers the most diverse and challenging deployment. Orchestra is optimally configured in its sender-based mode, with collision-free hash and a 29 timeslot long unicast slotframe.

A key property of the flow-based approach is retaining performance independent of competing flows. This should also hold with heterogeneous traffic patterns. To evaluate this claim, we create three scenarios where the number of nodes generating application traffic differs. In the first, only 13 out of the 26 nodes generate traffic. Next, we increase to 19 before, finally, all nodes except the root transmit. Packets are generated every 6 seconds to fit the capacity of Orchestra. As detailed in Section VIII-B, the destinations are randomly chosen, yielding a heterogeneous traffic pattern.

Figure 15 and 16 show the PDR and latency for the different traffic intensities. Note how Layered retains performance regardless of intensity, while Orchestra suffers as the uniform cell distribution cannot cope with traffic concentrations in the network. Orchestra has a shorter mean latency when 13 nodes are transmitting. This is due to the shorter slotframe

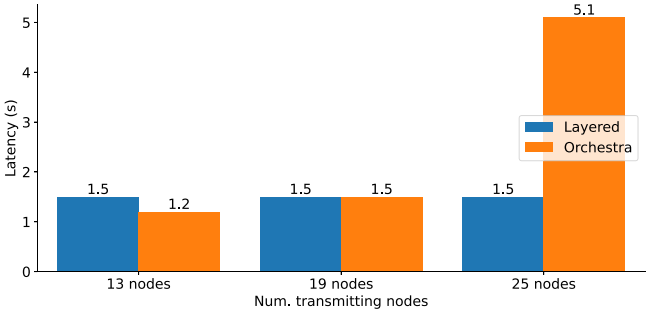


FIGURE 16. Mean latency for different traffic intensities, in testbed.

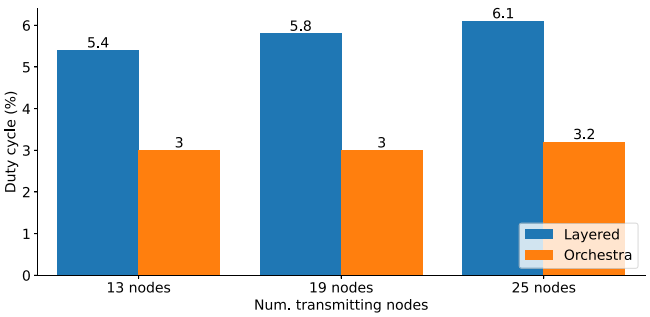


FIGURE 17. Mean duty cycle for different traffic intensities, in testbed.

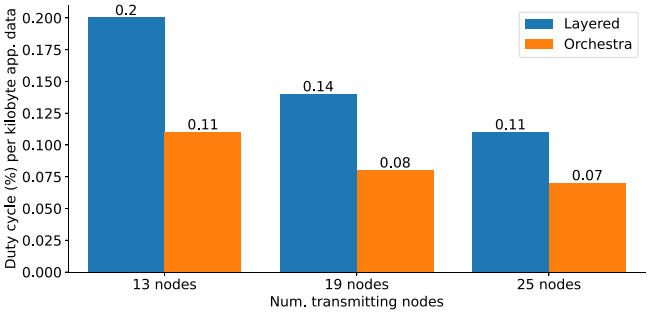


FIGURE 18. Mean duty cycle per kilobyte goodput for different traffic intensities, in testbed.

of Orchestra (29 vs. 72 timeslots) which allows for more frequent transmission opportunities for each node.

The trade-off for the Layered performance can be seen in Figure 17. It shows how Layered has approximately double the energy consumption compared to Orchestra. However, Layered has a higher goodput, i.e., application data successfully delivered. Layered utilize shared cells also for the initial part of the application traffic. As such, nodes must spend more energy listening to shared cells compared to alternative schedulers. This "energy overhead" is amortized over the data traffic, and the energy per goodput will decrease with increasing traffic intensity as seen in Figure 18. By further increasing the transmission interval, we found Layered could reach 0.06% duty cycle per kilobyte while retaining PDR and latency. Consequently, Layered requires more energy to maintain the network yet provides a schedule with higher throughput. Layered may thus be more suitable for applications with higher traffic intensity. These findings are similar to our earlier work on convergecast traffic patterns.[18]

We conclude that the extended Layered scheduler is able to support heterogeneous traffic patterns independent of the number of flows. This property is crucial for industrial use-cases. The performance comes at a significant penalty in terms of energy, with the efficiency improving as the traffic intensity increases. In our scenario, we found the duty cycle to be double that of Orchestra, yet accommodating double the throughput.

X. CONCLUSION

Existing autonomous flow-based schedulers assume a convergecast traffic pattern where all nodes transmit toward the RPL root. However, industrial applications require heterogeneous patterns, e.g., in the case of control applications employing a sensor-to-actuator pattern. We modified our autonomous flow-based Layered scheduler to support heterogeneous traffic patterns. The modifications included a novel mechanism to learn which cells to schedule autonomously. Using the scheduler, seven challenges and critical parameters were identified. For each challenge, we identified mitigations that were evaluated along with the key parameters using the Cooja simulator and FIT IoT-LAB testbed.

The expanded Layered scheduler could support heterogeneous traffic patterns while retaining reliability and latency regardless of competing traffic. However, scheduler convergence and performance cannot be achieved without adequately configured shared cell capacity. Too few cells are detrimental to convergence, while too many penalize energy efficiency and throughput. The appropriate amount differs between deployments and depends on the radio environment, network density, traffic intensity, etc. This is a common issue with autonomous schedulers, and tackling it may require a combination of approaches, such as parameter tuning and network surveying.

Supporting heterogeneous traffic in a TSCH, RPL-based network proved challenging. Some challenges were found to degrade performance during particular conditions; in the testbed, they occurred only rarely. Examples include routing changes that lead packets to be forwarded outside the RPL DODAG or in an invalid direction. We proposed mitigation mechanisms in the scheduler which negate the impact yet introduce cross-layer dependencies and increased complexity. The mechanisms may therefore be employed if robustness and predictable performance are prioritized - as is often the case in industrial applications. The mitigation benefit increases in challenging radio environments, and operators should consider each deployment against their application requirements.

As the network scales and path lengthens, the expected latency of Layered increases due to its design for minimized channel usage. To mitigate this, operators may consider

18. See [3, Sec. VII-B].

dividing a deployment into multiple smaller networks. The scheduler Orchestra was used as a baseline to compare performance. When traffic intensity increased, we found Layered could retain PDR and latency. However, the performance comes at a significant cost in energy as Layered requires more shared cells to maintain the network. The resulting schedule provides higher throughput than Orchestra. Layered's efficiency therefore improves as applications increase their traffic intensity, and in our scenarios reached 0.06% duty cycle per kilobyte goodput which is comparable to Orchestra.

REFERENCES

[1] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: Deterministic IP-enabled Industrial Internet (of Things)," *IEEE Commun. Mag.*, vol. 52, no. 12, pp. 36–41, Dec. 2014.

[2] A. R. Urke, Ø. Kure, and K. Øvsthus, "A survey of 802.15.4 TSCH schedulers for a standardized Industrial Internet of Things," *Sensors*, vol. 22, no. 1, p. 15, 2022.

[3] A. R. Urke, Ø. Kure, and K. Øvsthus, "Experimental evaluation of the layered flow-based autonomous TSCH scheduler," *IEEE Access*, vol. 11, pp. 3970–3982, 2023.

[4] N. Finn, P. Thubert, B. Varga, and J. Farkas, "Deterministic networking architecture," IETF, RFC 8655, Oct. 2019.

[5] E. Grossman, "Deterministic networking use cases," IETF, RFC 8578, May 2019.

[6] M. O. Demir, A. E. Pusane, G. Dartmann, G. Ascheid, and G. K. Kurt, "A garden of cyber physical systems: Requirements, challenges, and implementation aspects," *IEEE Internet Things Mag.*, vol. 3, no. 3, pp. 84–89, Sep. 2020.

[7] D. Baumann, F. Mager, U. Wetzker, L. Thiele, M. Zimmerling, and S. Trimpe, "Wireless control for smart manufacturing: Recent approaches and open challenges," *Proc. IEEE*, vol. 109, no. 4, pp. 441–467, Apr. 2021.

[8] P. Thubert, "An architecture for IPv6 over the time-slotted channel hopping mode of IEEE 802.15.4 (6TiSCH)," IETF, RFC 9030, May 2021.

[9] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 595–615, 1st Quart., 2020.

[10] R. Alexander et al., "RPL: IPv6 routing protocol for low-power and lossy networks," IETF, RFC 6550, Mar. 2012.

[11] O. Iova, F. Theoleyre, T. Watteyne, and T. Noel, "The love-hate relationship between IEEE 802.15.4 and RPL," *Commun. Mag.*, vol. 55, no. 1, pp. 188–194, Jan. 2017.

[12] T. Watteyne, A. Mehta, and K. Pister, "Reliability through frequency diversity: Why channel hopping makes sense," in *Proc. 6th ACM Symp. Perform. Eval. Wireless Ad Hoc Sensor Ubiquitous Netw. (PE-WASUN)*, 2009, pp. 116–123.

[13] *Industrial Networks—Wireless Communication Network and Communication Profiles—WirelessHART*, IEC Standard 62591:2016, 2016.

[14] *IEEE Standard for Low-Rate Wireless Networks*, IEEE Standard 802.15.4-2020, 2020.

[15] A. R. Urke, Ø. Kure, and K. Øvsthus, "Layered autonomous TSCH scheduler for minimal band occupancy with bounded latency," *Internet Technol. Lett.*, vol. 4, no. 2, p. e255, Oct. 2020.

[16] S. Oh, D. Hwang, K.-H. Kim, and K. Kim, "Escalator: An autonomous scheduling scheme for convergecast in TSCH," *Sensors*, vol. 18, no. 4, pp. 1–25, Apr. 2018.

[17] S. Duquennoy, B. A. Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *Proc. 13th ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, Nov. 2015, pp. 337–350.

[18] S. Kim, H. Kim, and C. Kim, "ALICE: Autonomous link-based cell scheduling for TSCH," in *Proc. 18th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2019, pp. 121–132.

[19] S. Rekik, N. Baccour, and M. Jmaiel, "Limitations of static autonomous scheduling for TSCH protocol and advances in adaptive scheduling," in *Proc. IEEE 12th Annu. Comput. Commun. Workshop Conf. (CCWC)*, 2022, pp. 1124–1129.

[20] F. Righetti, C. Vallati, A. Gavioli, and G. Anastasi, "Performance evaluation of adaptive autonomous scheduling functions for 6TiSCH networks," *IEEE Access*, vol. 9, pp. 127576–127594, 2021.

[21] X. Cheng and M. Sha, "ATRIA: Autonomous traffic-aware scheduling for industrial wireless sensor-actuator networks," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, 2021, pp. 1–12.

[22] 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. (IPSN)*, 2021, pp. 299–314.

[23] S. Jeong, J. Paek, H. Kim, and S. Bahk, "TESLA: Traffic-aware elastic slotframe adjustment in TSCH networks," *IEEE Access*, vol. 7, pp. 130468–130483, 2019.

[24] S. Jeong, H.-S. Kim, J. Paek, and S. Bahk, "OST: On-demand TSCH scheduling with traffic-awareness," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 69–78.

[25] M. Kherbache, O. Sobirov, M. Maimour, E. Rondeau, and A. Benyahia, "Decentralized TSCH scheduling protocols and heterogeneous traffic: Overview and performance evaluation," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100696.

[26] Y. Tanaka, P. Minet, M. Vučinić, X. Vilajosana, and T. Watteyne, "YSF: A 6TiSCH scheduling function minimizing latency of data gathering in IIoT," *IEEE Internet Things J.*, vol. 9, no. 11, pp. 8607–8615, Jul. 2022.

[27] Q. Wang, X. Vilajosana, and T. Watteyne, "6TiSCH operation sublayer (6top) protocol (6P)," IETF, RFC 8480, Nov. 2018.

[28] D. Barthel, J. P. Vasseur, K. Pister, M. Kim, and N. Dejean, "Routing metrics used for path calculation in low-power and lossy networks," IETF, RFC 6551, Mar. 2012.

[29] O. Gnawali and P. Levis, "The minimum rank with hysteresis objective function," IETF, RFC 6719, Sep. 2012.

[30] 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, Jun. 2022, Art. no. 101089.

[31] T. Chang, M. Vučinić, X. Vilajosana, S. Duquennoy, and D. R. Dujovne, "6TiSCH minimal scheduling function (MSF)," IETF, RFC 9033, May 2021.

[32] P. Levis, T. H. Clausen, O. Gnawali, J. Hui, and J. Ko, "The trickle algorithm," IETF, RFC 6206, Mar. 2011.

[33] J. Hui and J. P. Vasseur, "The routing protocol for low-power and lossy networks (RPL) option for carrying RPL information in data-plane datagrams," IETF, RFC 6553, Mar. 2012.

[34] L. Landmark, K. Øvsthus, and Ø. Kure, "Test-bed evaluation of ingress queuing for improved packet delivery," in *Proc. 4th Int. Conf. Netw. Services (ICNS)*, 2008, pp. 102–108.

[35] C. Adjih et al., "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Proc. IEEE 2nd World Forum Internet Things (WF-IoT)*, Dec. 2015, pp. 459–464.

[36] R. Jacob, M. Zimmerling, C. A. Boano, L. Vanbever, and L. Thiele, "Designing replicable networking experiments with triscale," *J. Syst. Res.*, vol. 1, no. 1, p. 42, Nov. 2021.

ANDREAS R. URKE received the B.S. degree in engineering and communication technology from the Western Norway University of Applied Sciences, Bergen, Norway, in 2009, and the M.S. degree in informatics and mobile communication from the University of Oslo, Oslo, Norway, in 2011. He is currently pursuing the Ph.D. degree in information security and communication technology with the Norwegian University of Science and Technology, Trondheim, Norway.

From 2016 to 2020, he was a Ph.D. Research Fellow with the Western Norway University of Applied Sciences. His research interests include the Industrial Internet of Things, scheduling algorithms for low-power short-range communication, autonomous and deterministic network operation, satellite communication, and embedded real-time software.

ØIVIND KURE received the Ph.D. degree from the University of California at Berkeley in 1988.

He is a Full Professor with the Department of Technology Systems, Section for Autonomous Systems and Sensor Technologies Research Group, University of Oslo. In 1988, he joined Telenor Research, where he worked as a Senior Researcher and a Research Manager from 1989 to 2000. His current research interests include various aspects of QoS, data communication, performance analysis, and distributed operating systems.

KNUT ØVSTHUS received the master's degree from the Norwegian University of Science and Technology in 1986, and the Ph.D. degree from the University of Oslo in 1998. He joined Telenor Research and Development in 1987 as a Research Scientist, and in 2001, he joined Norwegian Defence Research Establishment as a Research Scientist. Since 2006, he has been a Full Professor with the Western Norway University of Applied Sciences. His research interests include healthcare technology, industrial networks, conditioning-based maintenance, and CPS.