IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ADVANCES ON HIGH PERFORMANCE WIRELESS NETWORKS FOR AUTOMATION AND IIOT

Received 14 December 2022, accepted 26 December 2022, date of publication 9 January 2023, date of current version 13 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3235267

RESEARCH ARTICLE

Experimental Evaluation of the Layered Flow-Based Autonomous TSCH Scheduler

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: Andreas R. Urke (andrerur@stud.ntnu.no)

ABSTRACT The Industrial Internet of Things (IIoT) requires wireless connectivity that meets the strict industrial requirements on metrics such as reliability and latency. Promising approaches include Time Slotted Channel Hopping (TSCH) media access, where nodes operate according to a schedule. Autonomously built schedules typically rely on shared resources, where reliability and latency may suffer depending on traffic scenarios and topologies. We have earlier proposed the Layered scheduler, which belongs to a new category of autonomous schedulers: Flow-based scheduling. Layered allocates resources to traffic flows, and as opposed to typical autonomous schedulers, dedicated resources are guaranteed to be scheduled at every hop from source to destination in a convergecast scenario. In addition, Layered minimizes the number of channels required through the novel employment of autonomous spatial reuse. We extend earlier theoretical analysis and simulations by evaluating Layered using the FIT IoT-LAB testbed and compare it to Orchestra and 6TiSCH Minimal scheduler. The experiments demonstrate the feasibility of spatial reuse and that Layered retains performance independent of network topology and traffic intensity - a desirable feature in industrial scenarios. The performance comes at the expense of energy consumption, which in the worst case is 75 % higher compared to Orchestra and Minimal. We also present lessons learned, such as the impact of TSCH configuration on RPL convergence, the benefits of black-listing on performance, and how co-located TSCH networks could be divided by channel offsets as opposed to physical channels. Lastly, we discuss flow-based scheduling in general, its properties, and future research areas.

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

I. INTRODUCTION

As the Internet of Things (IoT) has matured, attention has expanded to new areas of application. Among these, the industrial paradigm is one of the most promising. However, the Industrial Internet of Things (IIoT) places strict requirements on the network, including high reliability, bounded latency, energy-efficient operation, etc. [1] These properties are typically challenging in wireless networks, and they require significant research and development across the entire network stack.

A contribution in this direction is the IETF 6TiSCH, which defines an IPv6-enabled network stack for industrial wireless low-power networks [2]. It combines common IoT protocols such as IPv6, UDP, 6LoWPAN, and RPL [3], with a MAC based on Time Slotted Channel Hopping (TSCH). TSCH has several desirable properties from an industrial perspective. It combines channel hopping which increases reliability, with time-slotted access, which allows for energy efficiency and contention-free utilization of wireless links. The industrial standard WirelessHART [4] utilizes a TSCH approach, and IEEE has standardized a TSCH MAC in 802.15.4 [5], which 6TiSCH employs.

Nodes in a TSCH network operate according to a schedule that dictates when each node may transmit or receive. The schedule is thus of critical importance to the network performance. Schedules are built by a scheduler, which operates mainly in a centralized, collaborative, or autonomous fashion [6].

Autonomous schedulers operate without dedicated signaling between nodes and do not introduce additional convergence or overhead in signaling. They are also typically less

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

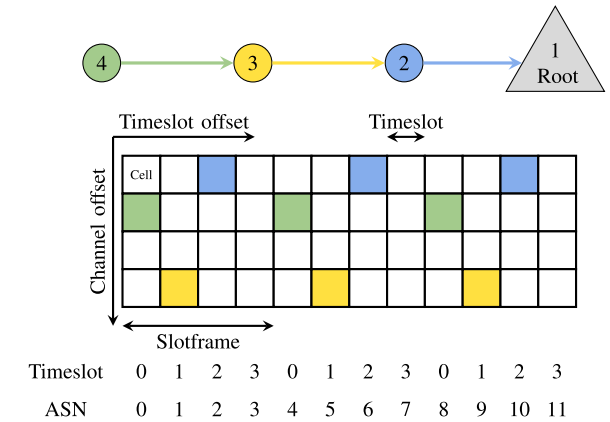

FIGURE 1. Simple 4-node network with corresponding TSCH schedule.

complex in operation and simple to understand a priori. Collaborative schedulers depend on nodes to exchange information, allowing for more dynamic and optimized schedules. Lastly, centralized schedulers typically collect information to gain a network-wide view enabling the creation of optimized schedules, yet with increased overhead and loss of flexibility. Autonomous schedulers are typically utilized for applications without stringent requirements, during network bootstrapping, or as a fallback if a more sophisticated scheduler fails. In this work, we contribute towards expanding the capabilities of autonomous scheduling, making it more viable for industrial applications.

A. TIME SLOTTED CHANNEL HOPPING (TSCH)

A simple topology and corresponding TSCH schedule are shown in Figure 1. The schedule divides time into *timeslots* horizontally and channel offsets vertically. A *cell* is identified by its timeslot and channel offset, and its duration allows transmitting one packet and receiving an acknowledgment. A cell can be shared between nodes e.g. for broadcasts, or it can be dedicated, allowing contention-free communication. As empty cells allow devices to sleep, TSCH schedules can yield energy-efficient operation.

A collection of timeslots repeats in periods called *slotframes*. The green-colored cell in Figure 1 indicates a dedicated transmission opportunity for node 4, while the blue cell allows node 2 to transmit toward the RPL root node. The physical channel to be used for transmission is calculated based on the channel offset and the Absolute Slot Number (ASN). The ASN always increases; thus, the physical channel may differ each time the cell is executed. The resulting frequency-hopping is key to mitigating frequency-dependent effects such as multi-path fading and external interference.

Further details on TSCH and the IEEE 802.15.4 standard may be found in [7].

B. PROBLEM STATEMENT AND CONTRIBUTION

Monitoring is a common industrial application where sensors send data toward one destination, typically forwarded through other sensors. This yields a convergecast traffic pattern where traffic intensity increases in a *funneling effect* close to the destination. To fulfill the industrial requirements on reliability and latency, this effect must be taken into account by the TSCH scheduler to avoid excessive queuing and packet loss. Existing autonomous schedulers typically distribute resources uniformly and rely on contended resources to absorb traffic heterogeneity, yielding non-deterministic performance [8].

Furthermore, radio channels in industrial environments may be scarce: Co-located networks must be expected, where channels are reserved to avoid fate sharing or to ease planning and configuration. Additionally, channels may need to be black-listed due to a challenging RF environment.

In [9], we proposed the autonomous Layered scheduler, which employs flow-based scheduling to address the funneling effect using dedicated resources, while minimizing the band occupancy through spatial reuse. Together with a detailed description, we also conducted theoretical analysis and simulations. The work confirmed the theoretical feasibility of the Layered scheduler design. However, these investigations were done assuming optimal RF conditions since realistic radio environments are challenging to model in a simulator. We acknowledged this gap and noted testbed evaluation as future work.

Complementing with a testbed evaluation is crucial to get a realistic and fuller understanding of a scheduler. This is especially important for Layered, as it relies on spatial reuse, i.e. multiple nodes transmitting simultaneously. The performance of spatial reuse depends on the radio environment, and an evaluation can only be done under realistic RF conditions, such as those found in testbeds. Therefore, this work's main contribution is an experimental evaluation of Layered on the popular FIT IoT-LAB testbed [10]. This work complements the earlier analysis and simulations in [9], which will be summarized in Section V.

Running a scheduler on actual node hardware is also necessary to verify the implementation and uncover issues not visible in a simulator, such as interoperability issues or unrealistic processing and memory usage. As such, our testbed experiments also brought about additional contributions including, amongst others, optimization of queuing for traffic flows. These optimizations and the Layered implementation are open-sourced to benefit reproducibility and future research.

From our testbed experiments, we contribute the following:

- Confirmed the feasibility of autonomous spatial reuse in realistic RF conditions. The novel utilization of spatial reuse is a cornerstone of Layered design and was found to have a negligible impact on performance.
- Evaluation and comparison of Layered with the 6TiSCH minimal schedule and state-of-the-art autonomous scheduler Orchestra. We show how Layered can retain performance regardless of traffic and topology at the expense of energy.
- Discussion and lessons learned throughout our testbed experiments. These include strategies for exploiting channel resources, insight into the relationship between

TSCH and RPL, and the importance of channel surveys and black-listing.

Furthermore, we contribute:

- To our knowledge, the first description and treatment of flow-based autonomous scheduling, a new category of TSCH scheduling to which the Layered scheduler belongs.
- Open-sourced implementation of the Layered autonomous flow-based scheduler, together with an adaption of the Contiki-NG TSCH queuing mechanism to support flow-based scheduling. Available at https://github.com/arurke/layered-scheduler.

The remainder of this work is organized as follows: First, we present related work before describing the Layered scheduler in Section III. We include a discussion of the new flow-based autonomous scheduling category in Section III-C. The description of Layered is finalized in Section IV, where we discuss the Layered implementation, queuing implications, as well as changes to the Contiki-NG operating system. Section V summarizes earlier theoretical and simulation work conducted in [9], and discusses their limitations. Next, we present our experimental evaluation in the FIT IoT-LAB testbed, including a comparison with Orchestra and the 6TiSCH minimal scheduler. Finally, the last sections discuss the results and conclude our work.

II. RELATED WORK

6TiSCH defines a *minimal mode* intended to be used for network bootstrapping [11]. It includes a TSCH schedule with 1 shared cell and typically 6 unscheduled timeslots. This schedule is often used as a benchmark in research on new schedulers.

Orchestra [12] is the current state-of-the-art and most prominent autonomous scheduler [6]. It aims at IoT applications and is node-based, i.e. it assigns cells to particular nodes based on their ID. Expanding on Orchestra, ALICE [13] is link-based, i.e. it assigns cells to specific links based on the neighbor's IDs and link direction (upwards or downwards). However, both schedulers allocate a fixed number of cells to each node or link in the network. Since the number of cells needed depends on the topology and traffic intensity, neither Orchestra nor ALICE can guarantee sufficient resources, and their performance is thus susceptible to e.g. the funneling effect.

OSCAR [14] optimizes Orchestra for convergecast traffic by assigning additional cells to nodes closer to the network root. The approach is best-effort as the number of cells is fixed and does not consider traffic.

New developments in *adaptive* autonomous schedulers are surveyed in [15] and [16]. These approaches aim to adapt the schedule to e.g. traffic or topology autonomously. This is typically achieved by nodes independently trying to estimate the required changes to a schedule. The performance, therefore, relies heavily on the accuracy and stability of the estimate. An approach targeting industrial applications may be found in [17].

Escalator [18] is an autonomous scheduler aimed at industrial applications and is the closest match to Layered in terms of functionality and approach. It is designed for convergecast traffic and optimizes for bounded and low latency. By allocating dedicated cells at each hop, all nodes can, without retransmissions, have their packet delivered to the root within one slotframe. However, the low latency is traded off for band occupancy: Escalator requires one channel per two hops of network depth[1] supported.

Several collaborative schedulers target bounded latency. They differ fundamentally from autonomous schedulers as cell allocation is based on neighbors negotiating. This enables the creation of dynamic schedules that adapts to changes in the network, traffic pattern, application requirements, etc. However, this adds signaling overhead, introduces scheduling convergence, and increases complexity. A notable collaborative scheduler example can be found in [19], which, similar to Layered, utilizes the node depth and spatial reuse to optimize for delivery within one slotframe. Lastly, YSF [20] schedules for traffic flows and uses autonomous scheduling to handle topology changes and accommodate negotiations for additional cells.

A comprehensive survey of TSCH schedulers conducted by the authors may be found in [6].

Testbed evaluations are typically complex and time-consuming, yet may be needed for a proper understanding and profiling of a scheduler. The FIT IoT-LAB [10] is a widely used testbed for IoT research. Critically, its open access allows for increased replicability of experimental results and comparison of results between different research efforts. FIT IoT-LAB was used to evaluate Orchestra and ALICE in [8], and OSCAR in [14], while Escalator was assessed using an ad-hoc testbed in [18]. A treatment of performance evaluation for low-power lossy networks, and available testbeds, may be found in [21].

III. LAYERED SCHEDULER

We propose the autonomous Layered scheduler, which guarantees resources end-to-end while minimizing the number of required channels. The flow-based approach in Layered guarantees dedicated cells to each flow at every hop from source to destination. The latency of a flow, without retransmissions, is thus bounded by the number of nodes and maximum network depth. This property is retained regardless of topology or other traffic in the network, which is key for industrial applications. In particular, Layered does not suffer from the funneling effect seen in e.g. monitoring applications where all nodes transmit towards the root node in a convergecast traffic pattern. Such scenarios are challenging for node- and link-based autonomous schedulers since the resources are allocated uniformly across the network. Contrarily, Layered allocates resources uniformly to flows, i.e. each flow gets the same amount of cells. Thus for Layered, the most optimal use-cases in terms of energy efficiency and throughput

[1] Hops from root.

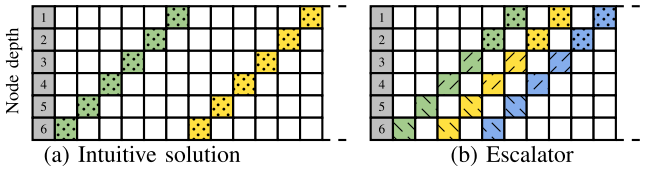

FIGURE 2. Solutions for funneling effect. Colors denote from which node traffic originated. Pattern indicates the channel. Depth = hops from root.

include e.g. monitoring, where all nodes generate data with the same fixed interval.

Layered is designed to minimize the number of channels occupied through spatial reuse of TSCH cells in a novel autonomous fashion. Minimizing band occupancy is especially desirable in an industrial scenario where channel conditions may be challenging due to co-existing networks, metal surfaces, industrial equipment, etc. [6] Thus, Layered support techniques such as aggressive black-listing may be easier employed to e.g. avoid interference from co-located Wi-Fi networks. Furthermore, limited band occupancy increases operator flexibility by permitting multi-network or -scheduler deployments.

Layered only requires basic routing topology information and a method for identifying the flows. Topology information is provided by the RPL routing protocol [3], while the source node ID identifies flows.

A. BACKGROUND

Performance independent of network traffic intensity and topology can only be achieved if sufficient resources are allocated for every node's traffic from source to destination. A simple and intuitive solution is illustrated in Figure 2: A cell is allocated at every hop, and is dedicated to a particular node's traffic. Each resulting string of cells constitutes a path of contention-free resources from source to destination. Assuming perfect links, the path guarantees delivery within one slotframe. However, this approach yields long slotframes which do not scale well since the length equals the number of supported nodes times the maximum allowed hop counts.

The strategy to reduce the slotframe length is characteristic of Layered, and it is the key difference compared to the similar autonomous scheduler Escalator. Before describing the Layered approach in the next section, we first look at Escalator: It reduces the slotframe length by placing the strings of cells on top of each other, as illustrated in Figure 2b. Interference is avoided by employing a different channel for every two hops, as indicated by the patterned cells. However, this approach *significantly increases* the band occupancy: For every two hops supported, one additional channel is occupied.

B. DESCRIPTION

The Layered scheduler solves the slotframe length problem by dividing the string of cells into pieces based on depth. All cells on a depth are then allocated to a *layer*. This is done in an alternating fashion, such that cells from every other depth overlap, significantly reducing the slotframe length. This is illustrated in Figure 3. Thus, as a layer is passed, all packets have advanced one hop.

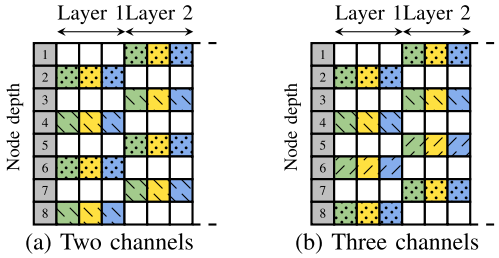

FIGURE 3. Transmission-cells in Layered schedule. Colors denote from which node traffic originated. Pattern indicates the channel.

TABLE 1. Autonomous strategies.

Type	Example schedulers	Cell calculation input
Node-based	Orchestra	Sender or receiver ID
Link-based	ALICE	Sender & receiver ID, direction
Flow-based	Escalator, Layered	Source ID, depth, direction

All cells belonging to one node's traffic are in the same timeslot, i.e. *traffic belonging to one node is being forwarded simultaneously at multiple hops*. This spatial reuse introduces possible interference between nodes transmitting in the same timeslot and channel. It is assumed that as the distance between nodes in terms of hops increases, a threshold is reached where they no longer interfere. This threshold depends on factors such as the routing protocol objective function and local RF conditions. Layered therefore permits configurable spatial reuse: Utilizing one more channel adds two additional hops of distance between nodes using the same cell. Figures 3a and 3b illustrate this as spatial reuse occurs at 4 and 6 hops distance, respectively. Thus, Layered offers trading band occupancy for increased resilience.[2]

To build the schedule on a node, Layered requires knowledge of the 1) Current depth in the routing tree, and 2) IDs of nodes in its sub-tree. This information can be provided by the RPL routing protocol found in the 6TiSCH stack. Lastly, Layered assumes each node generates one flow, which allows a flow to be identified by the source node ID. Therefore, a collision-free hash function is required, which maps node IDs to timeslots. Collisions are avoided by limiting the maximum number of nodes supported in a network, which we argue is reasonable in an industrial scenario. All of these requirements are also found in Escalator.

RPL traffic and other broadcasts are handled using common slots (CS) inserted at configurable intervals throughout the slotframe. Further details on Layered, including specifics on TSCH beacons, RPL traffic, a formal description of cell coordinate calculations, and examples of how the schedule is built, may be found in [9].

C. FLOW-BASED AUTONOMOUS SCHEDULING

Elst et al. [8] categorize autonomous schedulers according to how cells are allocated, and identify *node- or link-based*

[2]It is possible for Layered to operate using only one channel. However, this yields a sub-optimal distribution of cells in terms of latency; see [9] for details.

strategies. We propose a third strategy in the Layered scheduler, namely *flow-based* scheduling. Flow-based schedulers allocate resources to particular traffic flows in the network. Table 1 overviews the properties of the three strategies.

To our knowledge, we present the first treatment of flow-based autonomous scheduling, and Layered is the second autonomous scheduler operating according to its principles. Escalator also operates in a flow-based manner, yet [18] did not explicitly treat this aspect.

With flow-based being a new area in autonomous scheduling, we highlight some of its key properties and possible future areas of research:

1) BOUNDED PERFORMANCE

Dedicated resources can be guaranteed end-to-end for a particular flow by allocating cells to flows instead of links- or nodes. Layered utilizes this to provide bounded latency for flows when there are no retransmissions, regardless of competing traffic or network topology - a desirable property in an industrial scenario.

2) SCHEDULING FLEXIBILITY AND NEW USE-CASES

Layered and Escalator identifies a flow by the originating node ID. This caters to a common scenario where each node originates one flow. However, other identifiers may be envisioned. These include the Flow Label in the IPv6 header, the destination IP, or the received cell (similar to GMPLS [22]). This allows for flexibility in the schedule design and opens up for use-cases such as 1) Creating shared flows, e.g. a set of monitoring nodes that send alarms via a common flow, or 2) Allocating an arbitrary number of flows to each node, e.g. allowing one sensor node to run multiple applications simultaneously, or transmitting data to several different actuators at the same time. Furthermore, flow-based scheduling opens up for quality of service (QoS) efforts since resources can be dedicated to particular flows and designed to meet the application's QoS requirements.

3) OPTIMIZED THROUGHPUT AND ENERGY EFFICIENCY

With flow-based scheduling, cells are allocated in the slotframe to cater to the given flows. Thus, the slotframe length depends on the number of flows, not the number of nodes, as in link- and node-based schedulers. This is beneficial when a network has nodes that do not originate flows, e.g. nodes that are utilized to extend the range of a network, or nodes intended only to receive data, such as actuators and data sinks. Such nodes would not increase slotframe length, as opposed to a typical link- or node-based schedule. This allows flow-based schedulers to keep the slotframe short, yielding a higher throughput.

Secondly, nodes reserve cells only when they are in the path of a flow. Nodes outside the preferred path may therefore end up with a minimal schedule, only catering control traffic. This prolongs their lifetime and may thus contribute to increased network lifetime.

4) AUTONOMOUS SUPPORT FOR FLOW CONCEPTS

The concept of traffic flows is found in several standardization efforts. One example is Deterministic Networking (DetNet) [23] which provides upper layers with deterministic flows across diverse networks. Among these, we find 6TiSCH networks, which are envisioned to support DetNet flows by employing a centralized scheduler [24]. Autonomous flow-based schedulers have minimal overhead and may thus play a role in this regard, e.g. for fallback in case of failure, providing secondary paths when packet duplication is applied, serving as an option when bandwidth- or computation-resources are severely constrained, or catering to simple networks and scenarios.

IV. IMPLEMENTATION

We implemented the Layered scheduler on the Contiki-NG v4.6 [25] operating system. In addition to the scheduler, we made changes to the Contiki-NG TSCH queue module to support the flow-based approach employed in the Layered scheduler. Both implementations are available at https://github.com/arurke/layered-scheduler.

A. LAYERED SCHEDULER

Layered is designed for constrained devices, and the resources required in terms of code size, memory, and processing is considered lightweight: Its operation requires only simple arithmetic, and the line of codes is around 1000, which is comparable to Orchestra.

The scheduler is implemented as a service in Contiki-NG. It interfaces with the routing layer, which notifies Layered about current node depth, and routes added, removed, or altered. This information is sufficient for Layered to generate and maintain a schedule by utilizing functions offered by the TSCH module. A hash function maps node addresses to timeslots. As with Orchestra, a simple hash is provided where the last byte of the node address equals the timeslot.

To increase the stability of the RPL network, we utilize the RPL MR-HOF objective function [26] and keep the ETX metric (default in Contiki-NG). Since Layered needs to know the node depth in the RPL network, we add the hop count as an RPL metric container.[3] A similar approach is used in the collaborative scheduler by Hosni [27]. Note that this is not required if using hop count as a metric.

B. QUEUING FOR FLOW-BASED SCHEDULING

We found the existing queuing mechanisms in Contiki-NG not applicable to a flow-based scheduling approach. The Contiki-NG TSCH module uses a *per-neighbor* packet queue model, where queues packets according to their next-hop address. This model fits schedulers assigning cells to particular neighbors: When TSCH executes a TX cell, it picks the first packet in the corresponding queue.

[3] A standardized way to add additional metrics to DIO and DAO messages, see [3].

However, with flow-based schedulers such as Layered, cells are assigned to flows instead of neighbors. This is incompatible with the per-neighbor queue model: Multiple flows may pass through the same neighbor (in a convergecast scenario, all flows pass through the parent). Consequently, packets from different flows arrive at the same neighbor queue. When executing a Layered TX cell, we request a packet belonging to the corresponding flow. However, inspecting the head of all neighbor queues may yield no results, as packets for other flows may be in front. The resulting head-of-line blocking causes the cell to go unused and increases the flow latency.

One option is to keep the principle of a per-neighbor model while allowing packets to be fetched out of order: We modified the Contiki-NG TSCH module to iterate the entire neighbor queue when searching for a packet belonging to the flow. If found, the packet was extracted, and the queue was re-arranged to accommodate the removal.[4]

However, as expected, testing with increasing traffic showed that this approach does not scale with a flow-based scheduler: The time spent searching and manipulating the queues depends on the queue lengths. As queues grew, we experienced transmission failures as the time constraints in TSCH were increasingly violated.[5]

A proper solution for flow-based scheduling calls for a *per-flow* queue model where queues are maintained for each flow. We implemented such a model in Contiki-NG and combined it with the per-neighbor model: Packets belonging to a flow are added to their respective flow queue. All other packets, such as RPL and TSCH beacons, are handled in per-neighbor queues. Thus, when TSCH executes a TX flow cell, it simply picks from the front of the corresponding flow queue, while regular TX cells are served from neighbor queues. Consequently, eliminating any blocking or time-consuming manipulations.

V. THEORETICAL PROPERTIES AND SIMULATIONS

Table 2 summarizes the theoretical properties of Layered, Orchestra, and Escalator as presented in our earlier work [9]. This provided theoretical bounds on key properties such as latency. Using the Contiki-NG COOJA simulator [25], we verified the feasibility of Layered and confirmed these properties in optimal conditions for a range of network sizes, topologies, and traffic intensities. Lastly, autonomous spatial reuse was showcased by optimally configured transmission- and interference-ranges in the simulator. This proved the *theoretical feasibility* of the technique. Further details on this work can be found in [9].

A realistic RF environment is challenging to model in a simulator, and we therefore opted to simulate optimal conditions. This served its purpose, yet made it impossible to investigate Layered's autonomous spatial reuse or real-world performance. We conduct an experimental campaign using an open real-world testbed to fill this gap.

Testbeds are also crucial to understand implementation aspects since the simulator may not accurately capture the characteristics of the node hardware. Our simulations did for example not reveal the excessive computing involved in handling traffic per-flows using the regular Contiki-NG per-neighbor queuing approach. The efforts spurred from these findings are discussed in Section IV-B.

In the following sections we describe our testbed experiments.

VI. TESTBED EVALUATION

We employ the Grenoble and Strasbourg sites of the FIT IoT-LAB [10] to run our experiments. The ARM-m3 wireless nodes are running Contiki-NG v4.6, with the modifications detailed in Section IV when executing the Layered scheduler. The Contiki-NG configuration is left at default, except for the parameters listed in Table 3. The table also contains experiment configuration details such as channels utilized and convergence time.

A. METHODOLOGY

We believe reproducibility is important in the evaluation of scheduler performance. This is reflected in the performance indicators collected, the description of the testbed, and the availability of our code as open source. However, using a public testbed means that each test will have different radio environments with different background traffic. The performance indicators collected must be robust enough in varying environments for the experiment to be replicable. Therefore, we based our experiments' design and the statistical analysis on the TriScale approach described by Jacob et al. in [28]. TriScale aims to improve the replicability of experiments and enable researchers to draw sound conclusions with quantified uncertainties. It describes a statistical approach and how experiments can be designed to cater to it.

TriScale uses the *run* and *series* concepts, where an experiment is repeated across several runs, which together constitute a series. From each run, we calculate metrics, e.g. the Packet Delivery Ratio (PDR). The metric value from each run is utilized to calculate Key Performance Indicators (KPIs) for the series as a whole. A KPI is an estimate of the metric performance if we were to execute an infinite number of runs. In other words, it is the statistics for the underlying distribution from which our runs are sampling. The KPIs are expressed as a one-sided confidence interval for a given percentile of the runs. In our experiments, we execute 30 runs in each series, which allows us to calculate tail performance such as the 80-percentile, with the two worst runs providing robustness. Secondly, we always use a 95 % confidence. Consequently, *with 95 % confidence, the results we present are at minimum as stated, for at least 80 % of runs*. Please refer to [28] for further details on TriScale.

[4]Removing a packet from the middle of a Contiki-NG TSCH queue is non-trivial as they are implemented as a ring buffer, which natively only supports adding or removing at its ends.

[5]TSCH requires operations such as TX, RX, ACK, etc. to follow strict timings.

TABLE 2. Comparison of autonomous schedulers (N = number of nodes, CS = Number of common slots).

	Orchestra	Escalator	Layered
Traffic type	Any	Convergecast	Convergecast
Max. hops supported	No restriction	32	No restriction
Min. num. channels needed	1	$\frac{hops_{max}}{2}$	1
Slotframe length (SF_{len})	N	$N*2$	$N*2+CS$
Max. latency in timeslots without retransmissions	N/A	$(SF_{len}+hops_{max})*2$	$SF_{len}+(hops_{max}-1)*N+\left\lceil\frac{hops_{max}}{2}\right\rceil*CS$
Retransmission penalty	SF_{len}	SF_{len}	SF_{len}
Cells occupied per node/flow	1	32	$2*Channels$

TABLE 3. Experiment parameters.

Parameter	Value
Packet payload	24 bytes
RPL objective function	MRHOF w/ETX
TSCH packet burst	Disabled
Queue size per flow (Layered)	8 packets
Queue size per neighbor	16 packets
Max. links, neighbors & routes	64
TSCH max. backoff exponent	3
Physical channels Grenoble, Section VII-A	18, 19, 20, 21
Physical channels Grenoble, Section VII-B	19, 20, 21
Physical channels Strasbourg	14, 15
Wait for convergence Grenoble, Section VII-A	12 min.
Wait for convergence Grenoble, Section VII-B	24 min.
Wait for convergence Strasbourg	30 min.

The execution of a run is as follows if not otherwise stated:

1) 0 min.: All nodes boot
2) 6 min.: Nodes start transmissions towards root
3) X min.: Start measuring metrics.
4) X + 8 min.: End experiment

The time to wait for convergence, X, depends on the experiment and is specified in Table 3. These timings and settings were found experimentally, with aid from the TriScale toolchain. The last minute of measurements was ignored when analyzing to remove any effects from packets in flight.

When an experiment has multiple scenarios, such as comparing two settings or schedulers, the different scenarios are tested in an alternating fashion:

- 1st run for scenario A
- 1st run for scenario B
- 2nd run for scenario A
- ...

This scheme ensures that any time-dependent effects, such as office hours at the testbed locations, impact the scenarios as evenly as possible.

We utilize the following metrics:

- Packet Delivery Ratio (PDR): The ratio of received packets at destination nodes, to the total number of generated packets at source nodes.

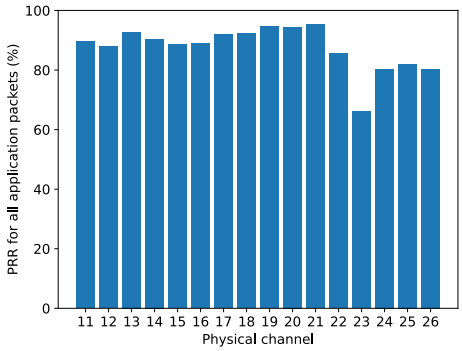

FIGURE 4. Survey of channel performance in Grenoble.

- Latency: Time from a packet is generated at the source node application layer until it is received at the destination node application layer.
- Packet Reception Ratio (PRR): The ratio of successful transmissions, to the total number of transmissions. A transmission in this case is an attempt at the physical layer of transmitting to a neighboring node.
- Energy: The ratio of time the radio has been powered, referred to as the duty cycle.

B. TESTBED ENVIRONMENT

Most FIT IoT-LAB nodes are located in an academic campus environment. Although not directly comparable to an industrial environment, it includes co-existing technologies such as Wi-Fi, and IEEE 802.15.4 deployments in the form of other experiments being executed simultaneously on the same premise. We thus observed significant variability in RF properties when running experiments. Examples include particular channels with catastrophic performance. This is discussed in more detail in Section VIII-A. Variability in the RF impacts the routing layer, as RPL topology may change. This, in turn, affects the autonomous scheduler since the schedule must accommodate the new topology.

We observed that, in particular during poor RF conditions, RPL struggled to converge. This causes severe performance degradation. Such events could probably be mitigated by optimizing RPL parameters. However, this was outside our scope. We instead opted to conduct an informal RF survey by measuring the packet reception ratio (PRR) on every channel.

Results for the Grenoble site are seen in Figure 4. To attain the highest RPL stability, we chose to utilize only the best-performing channels, 18-21.[6]

Channel black-listing substantially improved the RPL convergence, but topology changes still occurred. Runs with such topology changes were not excluded from the results unless otherwise noted.

C. COMPARISON WITH OTHER SCHEDULERS

We compare Layered to the 6TiSCH minimal mode [11], which often is used to benchmark scheduler performance, and Orchestra [12], the state-of-the-art and most common autonomous scheduler [6]. Only open-source implementations of Orchestra and 6TiSCH minimal mode scheduler were used to avoid any bias due to coding errors. The ones implemented in Contiki-NG are well tested and used in other experiments. Unfortunately, a direct comparison with the Escalator [18] scheduler is not possible since there is no open-source implementation of the scheduler. We did not deem it realistic to re-implement Escalator due to the risk of discrepancies with the original publication, which would yield our results invalid. In general, open-source implementations of TSCH schedulers are unfortunately scarce. Orchestra and 6TiSCH minimal are however the most commonly used for comparison when evaluating autonomous schedulers. They are e.g. employed by ALICE [13], Escalator [18], OSCAR [14], and Phung et al. [29].

VII. TESTBED EVALUATION - RESULTS

Our evaluation is organized as follows. We first evaluate a fundamental assumption behind Layered: The feasibility of spatial reuse in an RPL network. Next, we look at the performance of Layered across different scenarios: As Layered reserves resources for every flow at each hop, its key promise is for performance to be maintained independently of network traffic intensity and topology. To evaluate this, we first utilize a line-topology network with differing traffic intensities. Next, we investigate different network scenarios by increasing the scale and differing the node density and topology. The final key aspect of Layered is its minimal band occupancy which we discuss in Section VIII.

A. FEASIBILITY OF SPATIAL REUSE

A crucial part of Layered is the employment of spatial reuse. It is assumed that nodes will not interfere when the hop distance between them is sufficient. In this section, we evaluate the feasibility of this assumption. The aim is to measure the degree of interference introduced by spatial reuse in a testbed network. The results cannot be generalized to all deployments but will indicate whether the approach is feasible.

To measure the impact of spatial reuse, we require a network of at least 5 hops depth, as this is the minimum depth at which Layered employs spatial reuse: As described in Section III-B, when two channel offsets[7] are employed, spatial reuse may occur at depths 1 and 5, 2 and 6, and so on.

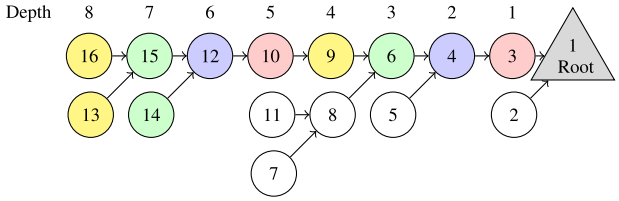

FIGURE 5. Example RPL topology from FIT IoT-LAB. Nodes of same color may be sharing cells, i.e. spatial reuse.

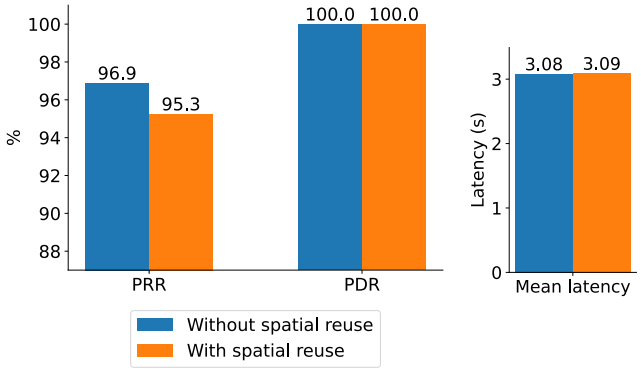

FIGURE 6. 50-percentile of PRR, PDR, and latency for spatial vs. non-spatial reuse scenarios.

We employ 16 nodes at the FIT IoT-LAB Grenoble site, which is especially suitable for generating deep networks, since nodes are physically placed along a hallway. By adjusting radio parameters,[8] we cannot design exact routing topologies, but we can ensure the necessary network depth, which is our only requirement. Figure 5 shows an example routing topology from one of the 30 runs in our 16-node network. The 4 deepest nodes[9] transmit one packet every other second slotframe towards the root.

In the first scenario, we configure Layered with four channel offsets, so there is no spatial reuse in our network. In the second scenario, we apply the minimal configuration of two channel offsets. This yields spatial reuse on depths 1 and 5, 2 and 6, and so on, which is the closest possible distance in hops for which Layered employs spatial reuse. An example can be seen in Figure 5: Nodes with the same color at different depths are prone to spatial reuse, e.g. the nodes 3 and 10, and the nodes 4 and 12.

If spatial reuse introduces interference, we expect decreased PRR and increased latency as more retransmissions are needed. Since we are interested in the impact of interference, we removed any run in which RPL was not converged (2 out of 30 runs in each scenario). This ensures the PRR is not impacted by e.g. scheduler mismatch between nodes during re-convergence.

[6] Counter-intuitively, most of the best performing channels overlap with 802.11 Wi-FI channels. A common approach, and the default in Contiki-NG, is to use channels 15, 20, 25, and 26 to avoid typical Wi-Fi channels. This showcases the importance of assessing and adapting to local conditions.

[7] Note the difference between physical channels and channel offsets. We utilize the same physical channels in all these experiments.

[8] TX power -17 dBm, RX sensitivity -78 dBm.

[9] In the physical topology.

How these two scenarios compare in terms of PRR, PDR, and latency can be seen in Figure 6. We found that the PRR is ∼1.7 % lower when spatial reuse is present. The reduced PRR is insufficient to impact the PDR or the latency. Thus, this deployment has a small and arguably negligible penalty when employing spatial reuse across four hops.

We finalize by increasing the hop distance between nodes with spatial reuse: From the previous 4 hops, we now utilize an additional channel and thus increase the distance to 6 hops. To ensure sufficient depth in the network, we also increase the number of nodes to 25. With the increased distance, we found the PRR decreases by ∼1 %, significantly less than the ∼1.7 % reduction seen at 4 hops distance.

These results indicate autonomous spatial reuse is feasible, even when Layered is configured to employ spatial reuse at its minimal distance of 4 hops. However, we must stress that RF properties are highly environment-dependent, and the network topology is influenced by the routing protocol and its objective function. Other deployments may therefore see significantly different results. This evaluation aims to establish the feasibility of spatial reuse in an RPL-based network without requiring departure from standardized settings. In deployments experiencing interference, Layered offers a trade-off with band occupancy: For each additional channel employed, the distance between nodes using spatial reuse increases by two hops.

B. PERFORMANCE FOR VARYING TOTAL TRAFFIC

A key property of Layered is that the performance of a flow is independent of other traffic, and it remains bounded regardless of network topology. This section focuses on traffic, while the next section investigates topologies. We compare the performance of Layered to 1) The minimal mode schedule described in 6TiSCH with the default configuration of 1 shared and 6 sleep slots, and 2) Orchestra, the state-of-the-art autonomous scheduler, which we optimally configure in sender-based mode with a collision-free hash.

The experiment uses the same Grenoble setup with 16 nodes as in the previous section. Application packets destined for the root are generated at a fixed interval of 2 seconds. However, in each scenario, the number of nodes generating traffic is different: In the first scenario, the 3 deepest nodes transmit, while next increasing to 7 nodes, 11 nodes, and lastly, 15 nodes. As the number of transmitting nodes increases, this creates an ever more substantial funneling effect toward the root. This is especially pronounced in the Grenoble line topology since the network is deep and sparse.

Orchestra and Layered are configured to support 17 nodes/flows in all scenarios. In Layered, common slots for RPL and broadcast traffic are added every 9 timeslots to ensure RPL convergence. The resulting slotframe length yields significant over-provisioning to allow for retransmissions.

The PDR and latency for the different traffic scenarios and schedulers can be seen in Figures 7 and 8, respectively. Notably, Layered retains a high PDR and low latency

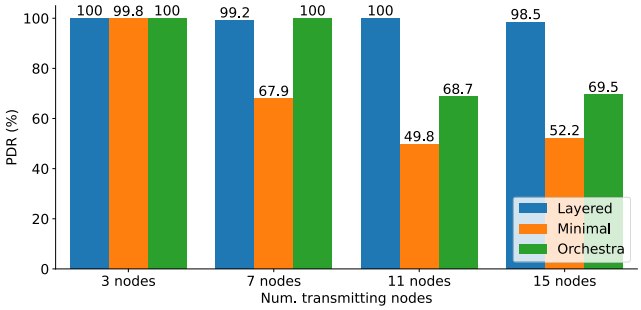

FIGURE 7. Packet delivery ratio.

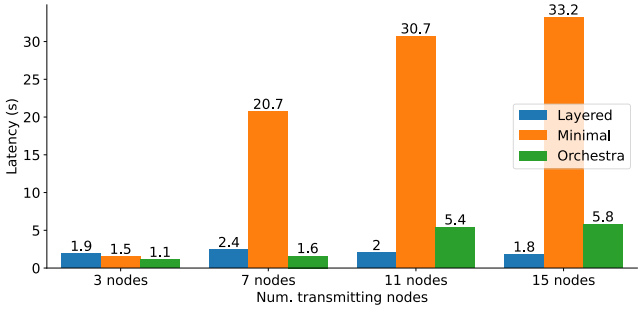

FIGURE 8. 99-percentile of latency.

regardless of the traffic load. A reduction of 1.5 % PDR is seen when all 15 nodes are transmitting. This is correlated with a slight increase in RPL parent switches and might be explained by the additional traffic creating more volatile ETX estimations for the parents.

Orchestra can retain a high PDR and short latency as long as 7 or fewer nodes are transmitting. With even more nodes, the funneling effect exceeds the throughput: In Orchestra, all nodes have a unicast slotframe which is 17 timeslots long, with one dedicated TX cell. This allows for one transmission approximately every 0.17 seconds - compared to the 2-second transmission interval. However, this cell is shared by all traffic flows and is thus insufficient at nodes close to the destination. Note, however, that the short slotframe allows for short latencies: Orchestra has almost half the latency of Layered, with 1.1 s and 1.6 s compared to Layered 1.9 s and 2.4 s. This is expected since the flow-based approach in Layered requires more than twice the slotframe length compared to Orchestra, as was highlighted in Table 2.

Like Orchestra, the 6TiSCH Minimal schedule also has a short slotframe (7 timeslots). It thus provides short latency, 1.5 s compared to Layered 1.9 s, as long as the capacity is not exceeded. When 7 or more nodes are transmitting, the capacity is exceeded, and the results become less relevant for our analysis.

Figure 9 shows the mean duty cycle for the different traffic intensities. Layered has a 61 - 74 % higher duty cycle than Orchestra. Similarly, it is 70 % higher compared to Minimal, which has 1 active timeslot per 6 inactive timeslots.

A key contributor to the increased energy consumption stems from the frequent shared broadcast cells in the Layered schedules. These cells carry all RPL traffic, both broadcast and unicast. This is as opposed to Orchestra, which has far

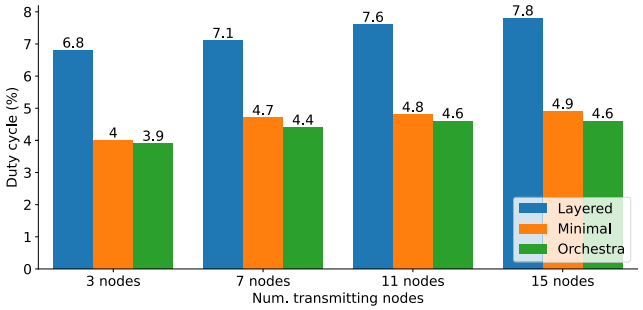

FIGURE 9. Mean duty cycle.

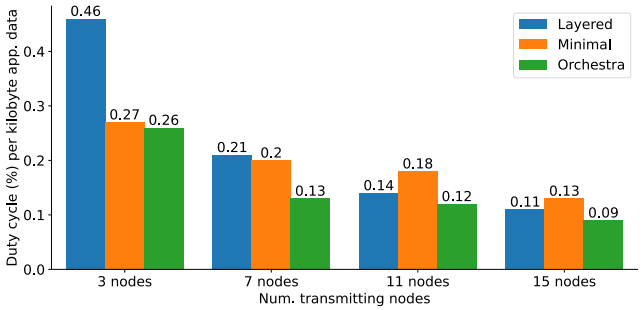

FIGURE 10. Mean duty cycle per kilobyte received application data.

fewer shared cells because RPL unicast packets are transmitted in the dedicated cells used by application traffic. Any shared cells must also be over-provisioned to accommodate sudden increases in RPL traffic or handle retransmissions. Thus to ensure RPL convergence with Layered, frequent shared cells are needed, especially for denser networks.

A second contribution is a consequence of the flow-based approach: Layered allocates all resources per flow, including any over-provisioned cells needed to handle retransmissions. This allows the flows to stay independent - retransmissions in one flow do not impact others. However, the approach decreases energy efficiency as over-provisioned cells are dedicated to each and every flow. Conversely, Orchestra allocates resources and any over-provisioning per node. This is energy-efficient as all flows can draw on the same smaller pool of resources. However, it introduces fate-sharing as retransmissions needed by one traffic flow might impact other flows.

We have seen Layered have a higher duty cycle. However, the Layered schedule also has significantly higher capacity than Orchestra and Minimal. To capture this, Figure 10 shows the duty cycle per kilobyte of received application data. Since Layered has a higher duty cycle independent of traffic, it is also the least effective in the first scenarios where traffic intensity is low. However, when the schedulers are approaching their maximum capacity, as seen by the PDR, the energy efficiency is similar: Layered has a duty cycle of 0.11 % per kilobyte when all nodes are transmitting. Similarly, Orchestra has 0.13 % per kilobyte in the 7-node scenario - the last scenario where its capacity is not exceeded. Minimal is less effective at 0.27 %.

We conclude from the duty cycle measurements that Layered requires more energy to maintain a network, yet the resulting schedules allow for significantly higher capacity. Therefore, the energy efficiency per throughput improves as the traffic intensity increases. Layered may thus be more beneficial for applications with higher traffic intensity.

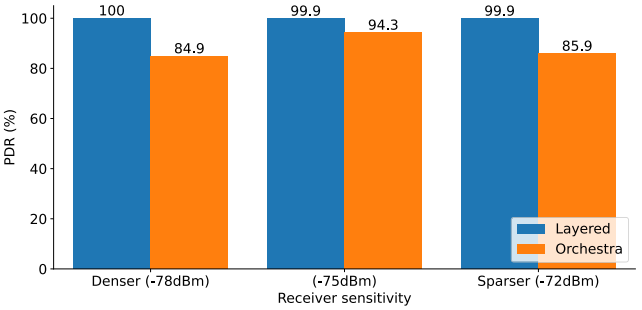

FIGURE 11. Packet delivery ratio.

C. PERFORMANCE FOR VARYING TOPOLOGIES

We utilize the Strasbourg site to evaluate the performance across differing network topologies. In Strasbourg, nodes are placed in a grid, allowing for more versatile topologies than the elongated hallways at the Grenoble site. We increase the number of nodes employed to 26 and let one node in a corner act as the root. The remaining nodes transmit in the direction of the root every 4 seconds.

The transmission interval was selected to be close to the maximum capacity for Orchestra in order to analyze any impact from the topology. Layered has a higher capacity, thus, the intention is to show the properties of each scheduler individually and not to compare directly. The results for the minimal schedule are omitted, as the traffic in all scenarios is far beyond its capacity.

Experience showed that "designing" particular topologies is challenging due to the unpredictable and dynamic properties of the RF environment. Therefore, to produce different topologies, we adjust the receiver sensitivity[10] of the nodes: Increasing sensitivity increases the range of a node, leading to more neighbors and a denser and shallower network. Conversely, decreasing sensitivity produces sparser and deeper topologies. We employ three settings: −78 dBm, −75 dBm, and −72 dBm, where −78 dBm is the most sensitive, i.e. producing the densest and shallowest network, typically at 4 hops depth and an average of 17 neighbors per node. −72 dBm is the least sensitive, yielding networks around 6 hops deep, with an average of 10 neighbors per node.

Orchestra and Layered are configured to support 29 nodes/flows, while Layered common slots are added every 7 timeslots to ensure RPL convergence.

The measured PDR can be seen in Figure 11. Layered is found to retain PDR at 99.9 - 100 %, independent of the differing topologies. Orchestra is impacted by the topology, as the PDR varies between 84.9 % and 94.3 %. Latency indicates the same behavior and is omitted for brevity.

[10]Receiver sensitivity allows for more granular control of the FIT IoT-LAB m3 nodes, as opposed to transmission power which has limited resolution, especially in the lower ranges.

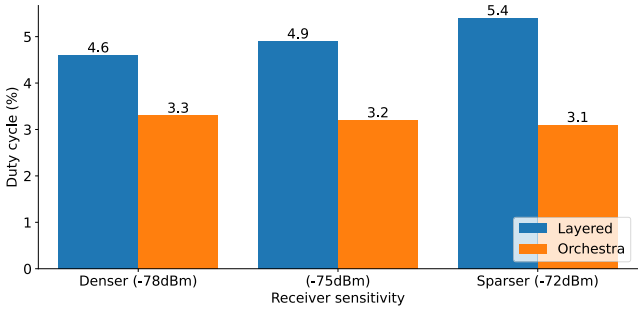

FIGURE 12. Mean duty cycle.

Figure 12 shows the mean duty cycle. As in the traffic intensity experiment, Layered has a significantly higher duty cycle than Orchestra. Notably, the Layered duty cycle depends on the topology: As the network becomes deeper, packets must be transmitted across more hops, which require additional cells and transmissions, increasing the duty cycle. The dependence on topology is less pronounced in Orchestra since each node always has the same amount of cells.

VIII. DISCUSSION

A. LAYERED MINIMAL BAND OCCUPANCY

Layered aims for minimal channel occupancy through spatial reuse, as opposed to the similar Escalator scheduler, which requires one channel per two hops supported. This makes Layered beneficial in scenarios where frequencies are scarce, significant black-listing is needed, or in multi-network or -scheduler deployments. Exploiting these benefits depends on the deployment, yet these lessons can be outlined from our experiments:

1) BLACK-LISTING

As described in Section VI-B, we identified several physical channels with significantly poorer performance than others. With the limited channels required by Layered, we were able to aggressively black-list channels without risking scheduling collisions. Such black-listing may not be possible in Escalator as it requires one channel per supported depth (in hops) in the network.

To illustrate the benefit of black-listing, we re-run the varying traffic experiment[11] in Section VII-B. We focus on the scenario with Layered and all 15 nodes transmitting. First, we enable black-listing, which leaves 4 channels utilized, and second, we employ all 16 channels.

The results can be seen in Figures 13. Utilizing all channels reduces the PRR by ∼4 %, which is not significant enough to impact the PDR. However, the increased amount of retransmissions raises the latency and duty cycle by 37 % and 13 %, respectively, compared to when aggressive black-listing is applied.

2) CO-LOCATED NETWORKS

The limited usage of channels in Layered also simplifies the co-location of networks. With fewer channels needed,

[11]The only difference being 4-second transmission interval instead of 2 seconds.

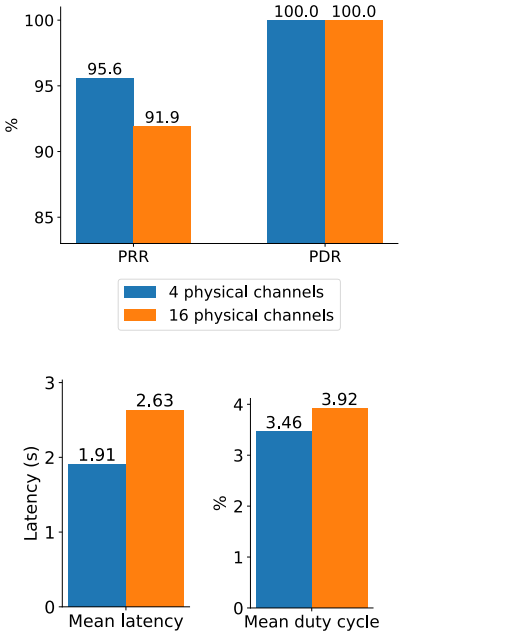

FIGURE 13. 50-percentile of PRR, PDR, latency, and duty cycle for 4- vs. 16-channel scenarios.

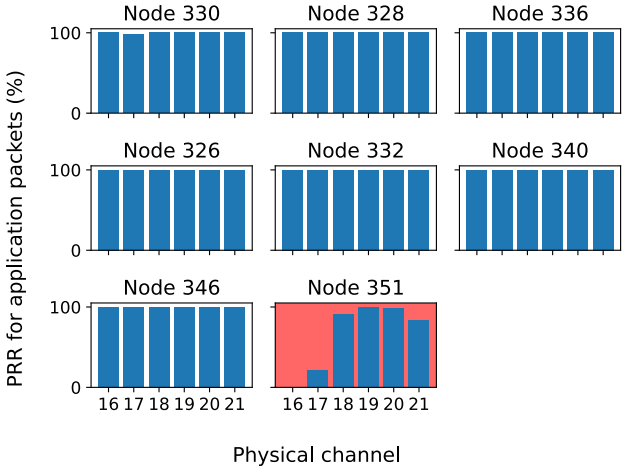

FIGURE 14. PRR per node for all physical channels utilized during one run.

reserving physical channels to co-located networks and technologies may be done with greater ease and flexibility. When TSCH networks are co-located, it may be an option to synchronize them in time such that the frequency hopping follows the same pattern. In these cases, it might be advantageous to rather separate networks by *channel offsets* in the schedules as opposed to physical channels:

Our measurements in the FIT IoT-LAB testbed show that a physical channel's performance may be highly volatile between nodes. In particular, we observed some channels practically unusable between some nodes. An illustration of this can be seen in Figure 14. It shows the PRR per physical channel for a subset of nodes in an experiment run. Note how the performance of channels 16 and 17 for node 351 is severely degraded compared to all other channels and nodes. We found that the channels and nodes affected were time-dependent without any identifiable pattern. To mitigate this

effect, we typically opted to use a hopping scheme with more channels (3-4) than strictly needed (2). Consequently, in this particular deployment, it is not advisable to reserve physical channels per network - one would be better off reserving *channel offsets* in a TSCH schedule. This allows the co-located networks to hop between more physical channels, thus minimizing the impact of any particular channel performing poorly.

B. RELATIONSHIP BETWEEN ROUTING AND TSCH MAC

Our experimental campaign found a critical relationship between the routing protocol, TSCH, and the schedule. If these parts are not properly managed and tuned, performance may be significantly degraded. Such problems were often experienced regardless of the scheduler employed and must thus be addressed for any deployment. Issues were typically manifested as a lack of RPL convergence due to missing or delayed routing information. We found the following factors to be vital in avoiding and mitigating these effects:

The TSCH schedule must provide sufficient capacity to forward control traffic. With Layered, this is emphasized, as all RPL traffic is transmitted in shared cells, whereas Orchestra uses dedicated cells for unicast traffic. In all cases, the topology density must also be taken into account to ensure that collisions in shared cells are kept at an acceptable level.

The timeliness of information must also be considered when configuring queue sizes in TSCH: If queues are too long, the information in an RPL message may become outdated before transmission. This may significantly impact RPL convergence. Chasing the same objective, the TSCH backoff mechanism should be tuned to balance between avoiding collisions and prompt transmission of information.

Furthermore, experiments must include sufficient time for initial convergence - a parameter that depends heavily on the network size. And lastly, as mentioned in Section VIII-A1, performance may greatly benefit by surveying the channel characteristics and employing a black-list based on the results.

For most of the mentioned considerations, the performance is traded off for energy consumption, which must also be taken into account. Further treatment of the relationship between TSCH and RPL may be found in [30] and [31].

C. HANDLING TRANSMISSION FAILURES

As observed, wireless transmissions are unpredictable and inclined to fail. One approach to handle failed transmissions is to retransmit. This is suitable for applications that value end-to-end reliability and tolerate the added latency. Retransmitting requires over-provisioning in the schedule and is the approach applied in our experiments, in which the transmission intervals were longer than the slotframe length. However, over-provisioning reduces available capacity and energy efficiency as unused cells induce idle listening. Such effects are especially pronounced in Layered due to the flow-based approach where each flow has dedicated over-provisioned resources. This is opposed to e.g. node-based schedules where the over-provisioning is shared by all traffic going through a node. Lastly, retransmissions inevitably increase latency. This effect is particularly visible in schedulers such as Escalator and Layered, where retransmission increases the latency by one slotframe length (see Table 2).

Therefore, it may be advantageous to combine a Layered schedule with approaches that handle transmission failures without retransmissions. One source for such alternatives is the ongoing work at the IETF Reliable and Available Wireless (RAW) working group.[12] Ultimately, the appropriate solution would depend on the application and its requirements.

IX. CONCLUSION

The Layered scheduler operates in a flow-based manner which is a novel category of autonomous TSCH schedulers. It allocates resources to traffic flows and guarantees dedicated cells on every hop from source to destination. Additionally, it minimizes the channels required by employing autonomous spatial reuse. We found that flow-based scheduling has favorable properties such as bounded performance, scheduling flexibility, and optimized throughput.

Layered was implemented on the Contiki-NG operating system, and we found the queuing mechanism needed to be adopted to support the flow-based approach. Both implementations are open-sourced at https://github.com/arurke/layered-scheduler. Experiments in the FIT IoT-LAB [10] showed how this allowed performance of each flow to be retained independent of traffic and topology. This is different from Orchestra and 6TiSCH minimal schedulers, which rely on shared resources and showed varying performance for the differing scenarios. The resilience in Layered comes at the expense of up to a 74 % increase in energy consumption. However, the Layered schedules allow for increased capacity. When fully utilizing its capacity, the energy spent per kilobyte of application data received is comparable to Orchestra. Layered may thus be more beneficial for applications with higher traffic intensity.

Layered relies on novel autonomous spatial reuse to reduce the number of channels occupied. The feasibility of this approach was demonstrated in the testbed, where PRR decreased by only 1.7 % at the minimum distance of 4 hops. Lessons learned from the experimental evaluation include the crucial relationship between TSCH and RPL convergence. It is governed by parameters such as queue size and schedule capacity, the importance of channel surveys and black-listing, and how co-located TSCH networks may be divided according to channel offsets as opposed to physical channels.

ACKNOWLEDGMENT

The authors would like to thank FIT IoT-LAB and its staff for keeping such an invaluable tool running and accessible for all. They also would like to thank to the authors of TriScale

[12] https://datatracker.ietf.org/wg/raw

for making the toolchain available, and for the assistance provided during their use.

REFERENCES

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.
[2] X. Vilajosana, T. Watteyne, T. Chang, M. Vucinic, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 595–615, 1st Quart., 2020.
[3] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 routing protocol for low-power and lossy networks," RFC Editor, Tech. Rep., RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt
[4] *Industrial Networks—Wireless Communication Network and Communication Profiles—WirelessHART (IEC 62591:2016)*, document IEC, 3:1–1043, International Electrotechnical Commission, 2016.
[5] *IEEE Standard for Low-Rate Wireless Networks*, IEEE Standard 802.15.4-2020 (Revision of IEEE Standard 802.15.4-2015), pp. 1–800, 2020.
[6] 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, Dec. 2021.
[7] D. De Guglielmo, S. Brienza, and G. Anastasi, "IEEE 802.15.4e: A survey," *Comput. Commun.*, vol. 88, pp. 1–24, Aug. 2016.
[8] 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.
[9] 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, Mar. 2021.
[10] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Proc. IEEE 2nd World Forum Internet Things (WF-IoT)*, Dec. 2015, pp. 459–464.
[11] X. Vilajosana, K. Pister, and T. Watteyne, "Minimal IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) configuration," RFC Editor, Tech. Rep., RFC 8180, May 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8180.txt
[12] S. Duquennoy, B. A. Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled TSCH," in *Proc. ACM Conf. Embedded Networked Sensor Syst. (SenSys)*, New York, NY, USA, Nov. 2015, pp. 337–350.
[13] S. Kim, H.-S. Kim, and C. Kim, "ALICE: Autonomous link-based cell scheduling for TSCH," in *Proc. 18th Int. Conf. Inf. Process. Sensor Netw.*, Apr. 2019, pp. 121–132.
[14] M. Osman and F. Nabki, "OSCAR: An optimized scheduling cell allocation algorithm for convergecast in IEEE 802.15.4e TSCH networks," *Sensors*, vol. 21, no. 7, p. 2493, Apr. 2021.
[15] 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.
[16] 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)*, Jan. 2022, pp. 1124–1129.
[17] 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)*, Nov. 2021, pp. 1–12.
[18] S. Oh, D. Hwang, K.-H. Kim, and K. Kim, "Escalator: An autonomous scheduling scheme for convergecast in TSCH," *Sensors*, vol. 18, no. 4, p. 1209, Apr. 2018.
[19] I. Hosni and F. Théoleyre, "Self-healing distributed scheduling for end-to-end delay optimization in multihop wireless networks with 6TiSCh," *Comput. Commun.*, vol. 110, pp. 103–119, Sep. 2017.
[20] Y. Tanaka, P. Minet, M. Vucinic, 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, Jun. 2022.
[21] K. Kritsis, G. Z. Papadopoulos, A. Gallais, P. Chatzimisios, and F. Theoleyre, "A tutorial on performance evaluation and validation methodology for low-power and lossy networks," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 1799–1825, 2018.
[22] E. Mannie, "Generalized multi-protocol label switching (GMPLS) architecture," RFC Editor, Tech. Rep., RFC 3945, Nov. 2004. [Online]. Available: https://www.rfc-editor.org/info/rfc3945
[23] N. Finn, P. Thubert, B. Varga, and J. Farkas, "Deterministic networking architecture," RFC Editor, Tech. Rep., RFC 8655, Oct. 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8655.txt
[24] P. Thubert, "An architecture for ipv6 over the time-slotted channel hopping mode of IEEE 802.15.4 (6TiSCH)," RFC Editor, Tech. Rep., RFC 9030, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9030.txt
[25] 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.
[26] O. Gnawali and P. Levis, "The minimum rank with hysteresis objective function," RFC Editor, Tech. Rep., RFC 6719, Sep. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6719
[27] I. Hosni, "Distributed scheduling with efficient collision detection for end-to-end delay optimization in 6TiSCH multi-hop wireless networks," *Ann. Telecommun.*, vol. 74, nos. 5–6, pp. 239–255, Jun. 2019.
[28] 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, pp. 1–28, Sep. 2021.
[29] K.-H. Phung, T. T. Huong, D. K. Dung, V. X. Tuong, T. Pham, T. Nguyen, and K. Steenhaut, "A scheduler for time slotted channel hopping networks supporting QoS differentiated services," in *Proc. Int. Conf. Adv. Technol. Commun. (ATC)*, Oct. 2018, pp. 232–236.
[30] O. Iova, F. Theoleyre, T. Watteyne, and T. Noel, "The love-hate relationship between IEEE 802.15.4 and RPL," *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 188–194, Jan. 2017.
[31] S. Duquennoy, J. Eriksson, and T. Voigt, "Five-nines reliable downward routing in RPL," 2017, *arXiv:1710.02324*.

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 at the Western Norway University of Applied Sciences. His research interests include the Industrial Internet of Things (IIoT), scheduling algorithms for low-power short-range communication, autonomous and deterministic network operation, satellite communication, and embedded real-time software development.

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

He is currently a Full Professor at the Sensor Technologies Research Group, Department of Technology Systems, Section for Autonomous Systems, University of Oslo. He joined Telenor after received his Ph.D. where he has 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 (NTNU), in 1986, and the Ph.D. degree from the University of Oslo (UiO), in 1998. In 1987, he joined Telenor Research and Development as a Research Scientist. In 2001, he joined the Norwegian Defence Research Establishment as a Research Scientist. Since 2006, he has been a Full Professor at the Western Norway University of Applied Sciences. His research interests include healthcare technology, industrial networks, conditioning-based maintenance, and CPS.