Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH

Simon Duquennoy[1]
simonduq@sics.se

Beshr Al Nahas[2]
beshr@chalmers.se

Olaf Landsiedel[2]
olafl@chalmers.se

Thomas Watteyne[3]
thomas.watteyne@inria.fr

[1]SICS Swedish ICT, Sweden
[2]Chalmers University of Technology, Sweden
[3]Inria, France

ABSTRACT

Time slotted operation is a well-proven approach to achieve highly-reliable low-power networking through scheduling and channel hopping. It is, however, difficult to apply time slotting to dynamic networks as envisioned in the Internet of Things. Commonly, these applications do not have predefined periodic traffic patterns and nodes can be added or removed dynamically.

This paper addresses the challenge of bringing TSCH (Time Slotted Channel Hopping MAC) to such dynamic networks. We focus on low-power IPv6 and RPL networks, and introduce Orchestra. In Orchestra, nodes autonomously compute their own, local schedules. They maintain multiple schedules, each allocated to a particular traffic plane (application, routing, MAC), and updated automatically as the topology evolves. Orchestra (re)computes local schedules without signaling overhead, and does not require any central or distributed scheduler. Instead, it relies on the existing network stack information to maintain the schedules. This scheme allows Orchestra to build non-deterministic networks while exploiting the robustness of TSCH.

We demonstrate the practicality of Orchestra and quantify its benefits through extensive evaluation in two testbeds, on two hardware platforms. Orchestra reduces, or even eliminates, network contention. In long running experiments of up to 72 h we show that Orchestra achieves end-to-end delivery ratios of over 99.99%. Compared to RPL in asynchronous low-power listening networks, Orchestra improves reliability by two orders of magnitude, while achieving a similar latency-energy balance.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless Communication

Keywords

TSCH, RPL, Scheduling, Wireless Sensor Network

SenSys'15, November 1–4, 2015, Seoul, South Korea..

ACM 978-1-4503-3631-4/15/11 ...$15.00.
DOI: http://dx.doi.org/10.1145/2809695.2809714.

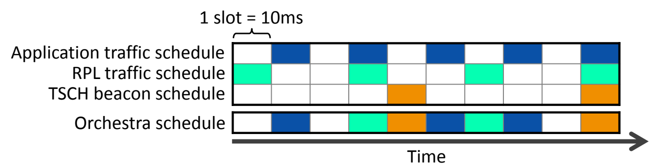


Figure 1: In Orchestra, several schedules repeat at different periods (here 2, 3, and 5), with slots allocated to specific traffic planes (resp. Application, RPL, TSCH). Slots are skipped whenever overlapped by a slot in a higher priority schedule (priority increasing from top to bottom in the figure).

1. INTRODUCTION

Context. As the Internet of Things (IoT) is emerging, there is an increasing need for low-power communication solutions that are both flexible (*i.e.*, are easy to use and able to to satisfy a variety of often dynamic application requirements) and robust (*i.e.*, work reliably). Example applications range from smart homes to smart cities, including wearable consumer devices. In these scenarios, short-range, low-power mesh networking is envisioned as a candidate technology to achieve both energy-efficiency and reliable large-scale operation.

Challenge. Flexibility and reliability are opposing goals. Asynchronous low-power mesh networks (including low-power IPv6) are flexible and support non-deterministic applications, but are best-effort. State-of-the art solutions have loss rates in the range of one percent [15, 19, 21, 11]. In the absence of end-to-end reliability, i.e., transport layer re-transmissions, such a loss rate is too high for most applications. With end-to-end reliability, losses trigger costly re-transmissions which often come in burst and result in jittery performance. At the other end of the spectrum, deterministic networks running TDMA and scheduled traffic can achieve 2 or 3 orders of magnitude fewer losses (*i.e.*, up to one loss per 10.000 packets or more) [8, 2, 37, 12, 28]. We investigate how to achieve such high level of reliability in non-deterministic scenarios.

Approach and Distinction. In this paper, we make a case for autonomous TSCH (Time Slotted Channel Hopping [1]) scheduling in non-deterministic low-power RPL and IPv6 networks. We show that even though it requires global synchronization, TSCH is practical in sparse traffic scenarios, and helps achieve high reliability in networks running a distributed routing protocol such as RPL [40]. The key challenge we address is that of creating TSCH schedules without hindering any of the flexibility of RPL networks and matching the requirements of non-deterministic applications. Orchestra achieves this with local, autonomous scheduling, and requires neither a centralized nor a distributed scheduler.

This is radically different from traditional TSCH scheduling solutions such as WirelessHART [14] and ISA100.11a [17], which rely on a centralized scheduling entity. This is also different from the standards being developed in the IETF 6TiSCH working group [33], which employ schedule negotiation between neighbor nodes.

In Orchestra, nodes employ simple periodic schedules and update the schedules automatically and instantly as the routing topology evolves. A TSCH schedule in Orchestra consists of a set of over-provisioned communication slots dedicated each to a specific communication plane: MAC, routing and application (as illustrated in Figure 1). As a result, Orchestra allows building a generic, flexible, low-power routing backbone using RPL while benefiting from the robustness of TSCH. Our schedules allow to reduce contention drastically, or even eliminate it altogether in certain cases.

Results. We implement Orchestra and TSCH in Contiki [10], and experiment in two testbeds with 98 and 25 nodes, each with a different hardware platform. In total, our evaluation bases on 219 individual experiments, up to 72 hours long, and a total of 1,178,601 UDP packets routed from source to destination. We show that Orchestra enables autonomous TSCH scheduling in RPL networks, and achieves end-to-end delivery ratios over 99.99%. This is an improvement of 1 or 2 orders of magnitude over state-of-the-art asynchronous solutions such as RPL with ContikiMAC. We show that Orchestra achieves this strong reliability while keeping energy and latency close to the state of the art.

Contribution. The main contribution of this paper is Orchestra, a system that allows TSCH nodes to maintain their schedules autonomously, driven by the state of the routing protocol. Orchestra operates without a centralized scheduler, and without inter-node schedule negotiation nor path reservation. We demonstrate experimentally that Orchestra is practical, scalable, and achieves end-to-end loss rates two orders of magnitude below low-power listening.

Outline. The remainder of this paper is organized as follows. §2 gives necessary background on TSCH and RPL, before introducing the basic concepts of Orchestra. §3 characterizes the potential benefits of TSCH as an alternative to asynchronous MAC layers. §4 discusses the design of Orchestra and §5 details implementation aspects. §6 discusses the results of our thorough experimental evaluation in two different testbeds as well as in controlled simulation. We discuss related work in §7 and conclude in §8.

2. OVERVIEW

This section introduces necessary background and gives a brief overview of our system, Orchestra.

2.1 TSCH

The IEEE802.15.4e-2012 [1] standard defines a number of MAC protocols for IEEE802.15.4. In this paper, we focus on TSCH (Time Slotted Channel Hopping), which inherits from WirelessHART and ISA100.11a.

TSCH nodes form a globally synchronized low-power mesh network. Nodes may join the network after hearing an Enhanced Beacon (EB) from another node. Time synchronization trickles from the PAN coordinator down to leaf nodes along a Directed Acyclic Graph (DAG) structure. Time is cut into timeslots; timeslots are grouped into one or more slotframes. A timeslot, typically 10 ms long, is long enough for a node to send a frame and for the receiver to acknowledge it. A TSCH schedule indicates to a node what to do in each timeslot: transmit, receive or sleep. A timeslot in a slotframe is identified by its time offset (when in the slotframe it occurs), its channel offset (denoting the frequency to communicate on), and a set of properties: whether it is to be used for transmission, reception, time synchronization, etc. Slots can be dedicated or shared, *i.e.*, contention-free or contention-based with CSMA back-off.

TSCH networks use channel hopping: the same slot in the schedule translates into a different frequency at each iteration of the slotframe. The result is that successive packets exchanged between neighbor nodes are communicated at different frequencies. In case a transmission fails because of external interference or multi-path fading, its retransmission happens on a different frequency, often with a better probability of succeeding than using the same frequency again [37].

How the communication schedule in the TSCH network is built and maintained is out of the scope of the established standards. The traditional way to scheduling (used in WirelessHART, ISA100.11a, and one of the modes in 6TiSCH) is to use a centralized entity which gathers information from the network, computes a schedule centrally, and disseminates routes and schedules to the nodes. Since late 2013, the IETF 6TiSCH working group is defining mechanisms to support decentralized scheduling. In this approach, a node negotiates with its neighbor to add/remove a slot to their local schedule.

2.2 RPL

RPL [40] is the routing protocol for low-power IPv6 networks standardized by the IETF ROLL working group. It is an oriented distance-vector routing protocol that organizes nodes in a Destination-Oriented DAG (DODAG) structure. The DODAG is rooted at the border router node (Internet access point). Each node is attached a *rank*, *i.e.*, its distance to the root using some cost function (*e.g.*, the ETX metric).

A node sends a packet towards the root by forwarding it to a neighbor node with a smaller rank. Routing from the root to one of the nodes is done by using the reverse links. In this paper, we focus on the *storing mode* of RPL: each node maintains a routing table towards its routing children, *i.e.*, the nodes further away from the root than itself. RPL uses unicast and broadcast signaling messages; namely, DIO (to disseminate the metrics), DIS (to request DODAG information) and DAO (to disseminate routes) messages. Routing from any node to another is done by first routing *up* to a common ancestor (along decreasing ranks), then *down* to the destination (following the routing tables).

2.3 Orchestra in a Nutshell

This paper introduces Orchestra, a new approach to scheduling in TSCH+RPL networks[1]. Orchestra is radically different from existing scheduling solutions in that it does not involve any extra central entity, negotiation, signaling, nor multi-hop path reservation among nodes. In-

[1]The approach developed in this paper is general enough to be applied to other scheduled MAC layers and any routing protocol. The focus of this paper is on TSCH and RPL.

stead, nodes maintain their own schedule locally and autonomously, based on their RPL neighbors and parents. As a result, Orchestra makes TSCH as flexible as asynchronous MAC layers, and able to support random-access traffic.

An Orchestra schedule contains different slotframes of different lengths. Each slotframe is dedicated to a particular type of traffic: TSCH beacons, RPL signaling traffic or application data. Nodes select slots using scheduling rules which reduces contention drastically, or in certain cases eliminates contention (see §4.2). This makes Orchestra particularly appealing for low-power IPv6 scenarios where different applications generate event-based data, without any predefined (*e.g.*, periodic) traffic pattern.

A concrete example Orchestra schedule contains:

- A dedicated broadcast slot from every node to its children for TSCH beacons, repeating every X slots;
- A slot common for all nodes in the network for broadcast+unicast for RPL signaling (DIO, DIS, DAO), repeating every Y slots;
- A dedicated unicast slot from every node to its RPL preferred parent, repeating every Z' slots;
- N dedicated unicast slots from every node to each of its children, repeating every Z" slots.

Orchestra uses slotframe lengths which are mutually prime, ensuring the slots overlap each other evenly, without unintended synchronization effects. The key is that we select the time and channel offset of the every slot as a function of the sender's or the receiver's identifier (MAC address or a unique network node ID). Depending on the scheduling rules, Orchestra can either attain very low levels of contention, or operate contention-free.

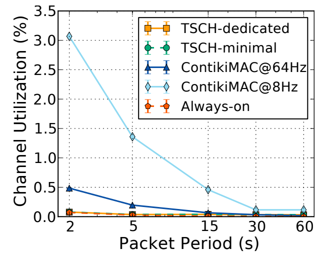
3. A CASE FOR TSCH IN LOW-POWER MESH

Before moving to the detailed design of Orchestra, we discuss and characterize the potential benefits of TSCH over asynchronous solutions.

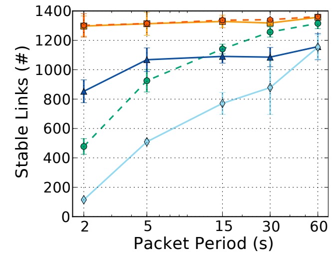
Cost of Global Synchronization. TDMA protocols are often regarded as impractical in random-access or sparse traffic scenarios, because of the overhead of global synchronization. In TSCH, nodes keep synchronized to one or several time sources, re-adjusting their clock whenever receiving a data of acknowledgment packet from it. In IEEE802.15.4, the maximum clock drift allowed is 40 ppm (parts per million), *i.e.*, max 80 ppm among two nodes. Assuming a guard time of ± 1 ms (the default value in TSCH), a node needs to re-synchronize to its time source neighbor every 12.5 s. Resynchronizing involves sending a short data packet, and receiving a short acknowledgment, which accounts for around 6 ms of radio on-time. Under these assumptions, resynchronization results in an additional radio duty cycle of 6 $ms/12.5\ s = 0.048\%$.

In practice, this baseline cost can be pushed even lower. In our testbeds (see §6.1) we measured an average drift between nodes in the range of 10–20 ppm, way below the 80 ppm assumed above. It is possible, in addition, to have nodes characterize their drift at runtime and adjust their clock dynamically, further reducing the cost of synchronization [4].

The numbers above show that synchronization can be obtained at a very low cost, in fact insignificant when compared to the typical duty cycle of mesh networks, in the order of percent or tens of a percent [15, 19, 21, 11] (*e.g.*, ContikiMAC in its default settings has a 0.6 % baseline duty cycle [9]). We therefore argue that TSCH is practical even in sparse traffic scenarios.



(a) Channel utilization



(b) Total number of stable links

Figure 2: Periodic Broadcast Probing Experiment. ContikiMAC has much higher channel utilization than TSCH and *Always-on*, resulting in fewer stable links as high traffic loads. TSCH with dedicated slots, being completely contention-free, yields performance similar to the baseline *Always-on*.

Scheduled vs. Asynchronous. We run an initial set of experiments to characterize link properties when using different MAC layers. We use the Indriya testbed [6], containing 98 TelosB nodes. We implement TSCH for the Contiki OS, and compare TSCH against ContikiMAC and the *Always-on* MAC, where nodes listen all the time and transmit using CSMA (Contiki's Nullrdc+Csma with link-layer ACK). The latter two are contention-based, asynchronous MAC layers. ContikiMAC [9] is a state-of-the-art low-power listening protocol that builds upon well established mechanisms [23, 16] described next. In ContikiMAC, nodes transmit their packet repeatedly for one period (*e.g.*, 125 ms) until the receiver wakes up and acknowledges it. Nodes are loosely synchronized through a phase-lock mechanism, which reduces strobe length towards already known neighbors. Although *Always-on* is generally impractical in low-power scenarios, we include it as a baseline approach.

In the experiment, every node transmits a broadcast packet at a given period with added jitter, and all packet receptions are logged. We use 3 different asynchronous MAC layers: *Always-on*, ContikiMAC at 8 Hz (default setting, wakeup period of 125 ms), ContikiMAC at 64 Hz (wakeup period of 15.6 ms). We use 2 different configurations for TSCH: TSCH-minimal, based on the 6TiSCH minimal configuration [35], where every node has a single shared communication slot for any traffic (here we use a slotframe of 1 slot, *i.e.*, all slots are active), and TSCH-dedicated where we use the nodes' unique node ID in the testbed to allocate a dedicated transmission slot to every node, ruling out all contention. For fairness with *Always-on* and ContikiMAC, and to focus on scheduling rather than channel hopping, we run TSCH on a single channel (channel 26, among the best channels in Indriya) in this specific experiment.

Figure 2 summarizes the results of this experiment. We look at two metrics. First, channel utilization is the average portion of time spent by a node with its radio transmitting. Second, the number of stable links, refers to the total number of links with PRR (Packet Reception Rate) above 90%.

ContikiMAC leads to high channel utilization with packet strobing, *i.e.*, up to 3% per node, which, in the 98 node testbed, corresponds to on average 3 nodes transmitting at any point in time. In comparison, TSCH and *Always-on* nodes have a channel utilization below 0.08%, a 37× factor improvement. This results in lower contention, and in turn a higher number of stable links.

Channel Hopping. An essential benefit of TSCH is its channel hopping nature. Channel hopping is known to effectively combat external interference and multi-path fading [38, 37], thereby increasing channel capacity and reducing the energy spent in packet retransmissions. Channel hopping can achieve similar benefits in asynchronous MACs, but at the cost of extra synchronization overhead [32, 24].

Orchestra aims at making TSCH as flexible as asynchronous MACs, while enjoying reduced contention (through scheduling) and robustness (through channel hopping).

4. Orchestra DESIGN

We introduce Orchestra, a system for routing-aware, autonomous slot allocation in random-access TSCH networks.

4.1 Big Picture

In Orchestra, nodes adapt their schedule by exploiting information from the RPL topology, and following a set of *scheduling rules*. This results in periodic activity patterns, with slotframes and slots assigned to different traffic planes such as TSCH beacons, RPL signaling, or application data.

Network Bootstrap. When switched on, a node joins the TSCH network by listening until it receives a Enchanced Beacon (EB), either from the PAN coordinator or another node. After synchronizing to that EB, the node runs Orchestra. A viable Orchestra setup requires slots for sending and receiving packets to/from any neighbor. This emulates an always-on link to all neighbors, allowing RPL nodes to discover their neighbors and build a topology.

TSCH-RPL Topology Mapping. Orchestra consistently uses the node's RPL preferred parent as its TSCH time source neighbor[2]. As the RPL topology evolves and parent switches occur, nodes update their TSCH time source accordingly. This yields a loop-free timing structure (a tree in this case), taking advantage of RPL's built-in loop avoidance mechanism. Furthermore, we use the RPL rank as TSCH join priority, as defined in the 6TiSCH architecture [33]. Doing so, we also take advantage of the RPL mechanisms for gradient convergence and stability. In case a node loses synchronization, it also leaves the RPL network, ensuring a clean slate bootstrap after re-joining.

TSCH Time Synchronization. TSCH time synchronization happens on any packet (or ACK) from the time source neighbor. In Orchestra, time synchronization happens primarily through periodic TSCH and RPL beacon transmission. This is efficient as a single broadcast message allows all children to update their clocks. Whenever a node has not communicated with its time source neighbor for a given duration (we use 12 s), it sends a unicast keep-alive message. The packet is re-sent until acknowledged, and re-synchronization is done by using the timing information embedded in the IEEE802.15.4e enhanced ACK.

[2]Note TSCH supports multiple time sources, but in our current design, Orchestra sticks to a single time source.

Routing-aware Scheduling. Throughout the network lifetime, Orchestra installs and updates TSCH schedules by using information from the routing layer. RPL runs unmodified, with slots being set up automatically as the topology evolves, ensuring network connectivity and allowing upper layers to run transparently. A basic example is where Orchestra maintains a dedicated slot for parent to child communication, repeating at a fixed period (*e.g.*, 1 s), at a time offset and channel offset selected from the parent's unique node ID. Whenever a child switches parent, it updates its slot to match the new parent's node ID.

4.2 Scheduling

Orchestra runs deployment-specific scheduling rules that describes how to maintain TSCH slotframes and slots as a function of the routing topology.

4.2.1 Orchestra Slots

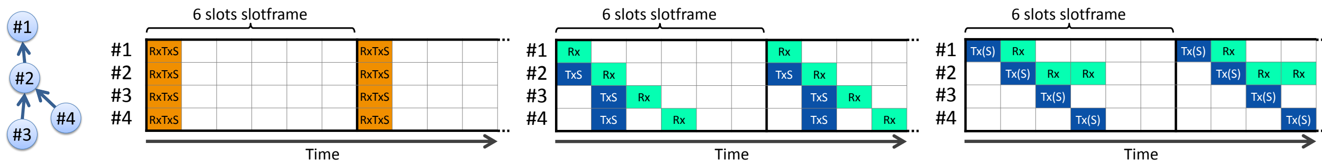
We identify four main types of slots in Orchestra: common shared slots, receiver-based shared slots, sender-based shared slots, and sender-based dedicated slots. Orchestra are dynamically mapped at runtime into 0, 1, or multiple TSCH slots. The different types of Orchestra slots are illustrated in Figure 3, in a 4-node network (Figure 3a) and showing only the slots for child-to-parent traffic.

Common Shared Orchestra Slots (CS). CS Orchestra slots consist in one shared slot used by all nodes in the network for both Rx (reception) and Tx (transmission), as illustrated in Figure 3b. The slot is installed at fixed coordinates (time and channel offset), resulting in a behavior similar to slotted ALOHA. This emulates an always-on link, allowing RPL to discover neighbors and run seamlessly. Note that TSCH uses an exponential back-off to resolve contention in shared slots, triggered whenever a unicast transmission is unacknowledged.

Receiver-based Shared Orchestra Slots (RBS). RBS are assigned for communication between two neighbors, at coordinates (time and channel offset) derived from properties of the receiver. At every node, a RBS Orchestra slot results in one Rx slot (coordinates based on the node), and one Tx slot per neighbor (coordinates based on the neighbor). To calculate slot coordinates, one can use a hash of the node's MAC address, modulo the slotframe length, or exploit unique node identifiers when available.

A typical example is for child-to-parent communication: nodes listen for any traffic in one slot, and children maintain a transmit slot towards their parent. As nodes switch parent, they update their transmit slot autonomously. Because several nodes may install slots towards the same receiver, contention may arise in such slots. For instance in Figure 3c, #3 and #4 contend to send to their parent #2, using standard TSCH back-off.

Sender-based Shared Orchestra Slots (SBS). SBS are similar to RBS, except that the slot coordinates are obtained from properties of the sender node rather than the receiver. At every node, a SBS Orchestra slot results in one Rx slot per neighbor (coordinates based on the neighbor) and a single Tx slot (coordinates based on sender node). This results in higher energy consumption than RBS (Tx slots cost nothing when there is no traffic, whereas Rx slots always require a wakeup), but can also help decrease contention by avoiding per-receiver slot assignement.



(a) Topology (b) Common Shared Slot (c) Receiver-based Shared Slot (d) Sender-based (Shared) Slot

Figure 3: Illustration of the different Orchestra slot types, in a 4-node network, showing child-to-parent slots. Slot properties are also shown: Reception (Rx), Transition (Tx), Shared (S). In common shared slots, all nodes wakeup up simultaneously to receive or transmit in a contention-based manner. In receiver-based slots, nodes have their own receive slot (here based on node ID), and their children contend when sending to them. In sender-based slots, nodes have their own transmit slot, and their parent wakes up to receive from them.

For instance, for child-to-parent communication, nodes have one fixed Tx slot, and parents maintain a Rx slot for each of their children. Whenever switching parent, the child does not need to update its transmit slot, but the old parent must remove a listen slot and the new parent install a new one. In Figure 3d, #2 has two listen slots, one for each of its children #3 and #4. Here again, contentions are resolved with the back-off from standard TSCH shared slots.

Sender-based Dedicated Orchestra Slots (SBD). In a slotframe long enough to accommodate unique transmit slots to every node, and assuming unique node identifiers are available, contention-free communication is possible. Orchestra achieves contention-free communication with SBD, which are similar to SBS except they use dedicated TSCH slots instead of shared. Note that with TSCH dedicated slots, lost packets are re-sent without a back-off (using the next slot towards the same neighbor). Unique node IDs can be hard-coded at deployment time, or obtained at runtime from a network manager.

4.2.2 *Orchestra Slotframes*

Orchestra manages several slotframes at every node, each of which is assigned to a particular traffic plane, *e.g.*, TSCH beaconing, routing traffic, application. Slotframes consist of a set of slots, with properties defined by simple scheduling rules. The slotframes repeat at periods that are mutually prime, ensuring they cycle independently. In case slots from different slotframes overlap, the slot in the highest priority slotframe takes precedence[3].

The length of a slotframe introduces a trade-off in traffic capacity, network latency and energy consumption.

Traffic Capacity. Shorter slotframes have their slots repeat more often, resulting in higher traffic capacity. Orchestra's approach is to over-provision TSCH in order to support non-deterministic traffic, and the slotframe length is the primary way to control the amount of over-provisioning for a given traffic plane.

Network Latency. The per-hop latency on a given traffic plane is basically proportional to the length of the slotframe for this particular traffic plane.

[3]Note that this is a slight departure from the standard TSCH slotframe priority rules, where *Tx* slots take precedence over other slots independent of slotframe handles. This departure is not strictly required but makes Orchestra slot interaction easier to comprehend.

Energy Consumption. Similarly, the shorter the slotframe, the more often nodes have to wake up to listen or transmit, resulting in higher energy baseline.

4.2.3 *Scheduling Rules*

Orchestra maintains its schedules using simple *scheduling rules*, described in this section. Scheduling rules are a set of TSCH slotframes and slots enhanced with a number of Orchestra-specific properties. Some of the slotframe and slot properties are per IEEE802.15.4e (labeled *std*), other include extensions to standard properties (*ext*), or are introduced by Orchestra (*new*).

The properties of an Orchestra slotframe S are:

Handle *(std)***.** A unique positive integer for both identification and priority. The smaller it is, the higher the priority.

Length *(ext)***.** The number of slots in the slotframe. Must be mutually prime with all other slotframe lengths in the network.

Traffic Filter *(new)***.** The traffic plane the slotframe is intended for. Filters packet properties (*e.g.*, *unicast*, *broadcast*) and protocols (*e.g.*, *TSCH*, *RPL*).

Slotframes are made of Orchestra slots, each mapped into 0, 1 or multiple TSCH slots depending on the current TSCH and RPL state. An Orchestra slot can for instance be reserved for communication with all TSCH time sources, RPL children, or the current RPL preferred parent. The properties of a slot are:

Neighbors *(new)***.** The neighbor or set of neighbors the Orchestra slot is to be instantiated for, such as the RPL preferred parent or all RPL children. The resulting TSCH slots are updated automatically whenever changes occur in the TSCH or RPL state.

Coordinates *(ext)***.** The time and channel offset within the slotframe. Can either be fixed or a variable such as a node ID a hash of the neighbor MAC address.

Options *(std)***.** Standard TSCH options. Includes: *Rx* (reception), *Tx* (transmission), *S* (shared), defining what the slot can be used for, and if it is shared or dedicated.

Although today the Orchestra rules are statically programmed in the nodes, one could design a CoAP-based management interface to define new rules at runtime. Once the slotframes and slots are installed, Orchestra executes TSCH according to standard IEEE802.15.4e, except for transmit slots. For transmit slots, in addition to matching the packet and slot address fields, Orchestra checks the packet against the traffic filter of the current slot's slotframe.

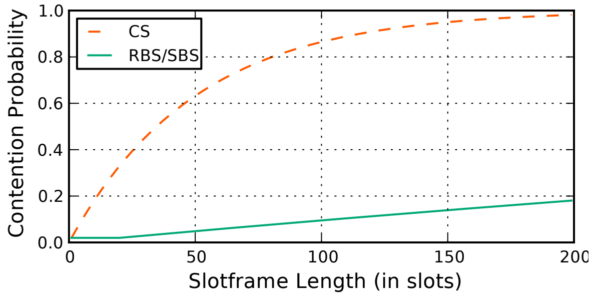


Figure 4: Analytical contention probability for CS slots vs. RBS and SBS slots, for 20-node clique and a traffic load of one packet per 500 ms. RBS and SBS do not eliminate contention but reduce it drastically. SBD are not in the figure as they are contention-free.

4.3 Performance Analysis

This section analyzes the contention rates obtained with different types of Orchestra slots. It then formulates guarantees on the frequency of overlap among different slotframes, and derives bounds for the nodes' radio duty cycle.

4.3.1 Contention Rate

Orchestra slots repeat at a constant period to serve a particular traffic plane. This is a behavior equivalent to slotted ALOHA for each traffic plane. In slotted ALOHA, the probability for any transmission to face contention is [41]:

$$p(cont_{slottedALOHA}) = 1 - e^{-T} \tag{1}$$

Where T is the average traffic load on the slot, with traffic following a Poisson distribution. For instance, if there is on average one packet sent every two slot, $T = \frac{1}{2}$.

Let us consider a simple case with a network of N nodes, all connected to each other (a clique). In a setup with one slotframe of length L with a single CS slot, the load on every slot is $T \times L$, and the contention probability is:

$$p(cont_{CS}) = 1 - e^{-TL} \tag{2}$$

In a case with RBS or SBS slots, the traffic is spread over all slots in the slotframe. If the slotframe is longer than or equal to the network size, the traffic is spread evenly across all nodes, decreasing the traffic load by a factor N. Otherwise, all slots are shared equally among nodes, decreasing the traffic load by a factor of only L. As a result, the contention probability is:

$$p(cont_{RBS}) = p(cont_{SBS}) = \begin{cases} 1 - e^{-\frac{TL}{N}} & \text{if } L \geq N \\ 1 - e^{-T} & \text{otherwise.} \end{cases} \tag{3}$$

Finally, sender-based dedicated slots (SBD) are by design contention-free:

$$p(cont_{SBD}) = 0 \tag{4}$$

Figure 4 shows $p(cont_{CS})$, $p(cont_{RBS})$ and $p(cont_{SBS})$ for a 20-node network, with an overall traffic load of one packet per 500 ms and 10 ms slots ($T = \frac{1}{50}$). In all cases, contention increases for longer slotframe as a result of sparser slot repetition. At any slotframe length, RBS and SBS decrease the level of contention by an important factor. They are therefore advisable in any scenario where a common rendez-vous slot is not required.

4.3.2 Slotframe Overlap

Every slotframe repeats with at a given period (its length in slots). Let B_{slots} be the number of slots in slotframe B of length B_{len}. Let $coll_B$ denote the event of a given slot colliding with any slot in B. The probability for any slot to collide with B is:

$$p(coll_B) = \frac{1}{B_{len}/B_{slots}} \tag{5}$$

When such slot collision occur, the slot from the slotframe with smaller handle takes precedence, all other slots are skipped. We denote SF the set of all slotframes in the system. The probability for A (handle denoted as A_h) to be skipped due to a slot collision with any other slotframe is:

$$p(skip_A) = 1 - \left(\prod_{\forall B \in SF,\ B_h < A_h} 1 - p(coll_{A,B}) \right) \tag{6}$$

4.3.3 Duty Cycle Bounds

Let $rxMinDc$ be the radio duty cycle of a Rx slot when no communication occurs, defined as $rxMinDc = \frac{rxGuardTime}{slotLength}$ where $rxGuardTime$ denotes the TSCH Rx guard time and $slotLength$ the TSCH slot duration. $A_{dcRxBase}$ is the listening cost of slotframe A, in the absence of communication:

$$A_{dcRxBase} = (1 - p(skip_A)) \times \frac{A_{rxSlots} \times rxMinDc}{A_{len}} \tag{7}$$

Here, $A_{rxSlots}$ is the number of slots with Rx flag in slotframe A.

In the absence of data to send, a node does not switch on its radio in transmit slots. Therefore, the lower bound duty cycle of slotframe A is $A_{dcLower} = A_{dcRxBase}$.

Because of reception guard times, Rx slots result in a maximum duty cycle (denoted $rxMaxDc$) higher than Tx slots (denoted $txMaxDc$). The upper bound duty cycle is reached when a full-sized packet is received (resp. sent) at every Rx slot (resp. Tx-only slot):

$$A_{dcUpper} = (1 - p(skip_A)) \times \frac{A_{rxSlots} \times rxMaxDc + A_{txOnlySlots} \times txMaxDc}{A_{len}} \tag{8}$$

Where $A_{txOnlySlots}$ is the number of slots in slotframe A with Tx flag but not Rx flag.

The system-wide lower and upper bound duty cycle are denoted respectively $dcLower = \sum_{\forall A \in SF} A_{dcLower}$ and $dcUpper = \sum_{\forall A \in SF} A_{dcUpper}$.

4.4 Example Orchestra Schedules

We introduce a number of example Orchestra setups used throughout the paper for discussion and evaluation.

4.4.1 6TiSCH Minimal Schedule

A simple example is the schedule defined by the 6TiSCH minimal configuration [35]. It consists of a single slotframe with a single common shared (CS) slot. This configuration is a very practical one, as it establishes basic connectivity between every node, for any traffic type. However, all slots are shared in the entire network, resulting in a purely contention-based scenario. We refer to such a setup as *TSCH-min-X*, where X is the length of the slotframe.

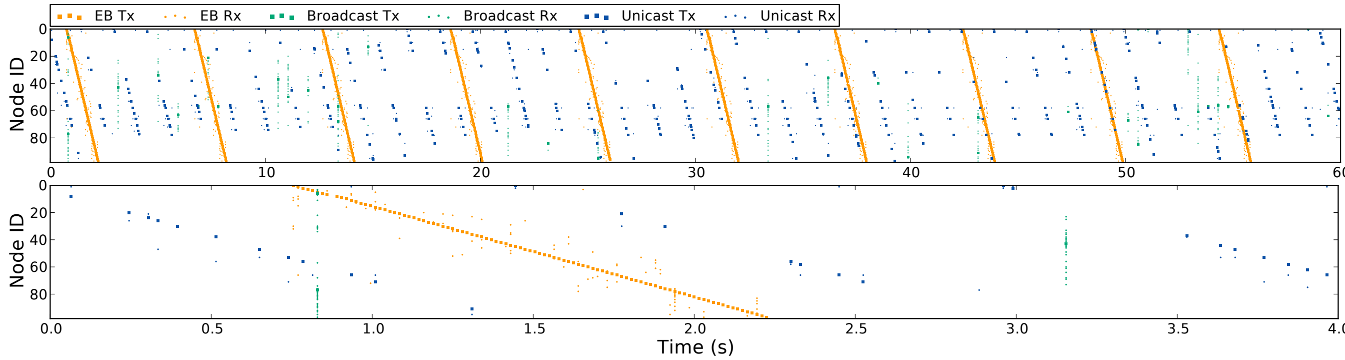


Figure 5: A 60 minutes slice of Orchestra running in Indriya (top) and a zoomed view on 4 seconds (bottom). The EB and unicast slotframes result in diagonal lines of activity as they contain a single *Tx* slot each, with time offset equal to node ID. The broadcast slotframe has a single common shared slot at time offset 0, and therefore results in vertical lines, as all nodes are listening simultaneously. All slotframes repeat with a different periodicity. The duty cycle is 0.47%.

4.4.2 Setup with Receiver-based Unicast Slots

We describe a more advanced setup made of three slotframes, including one receiver-based slotframe for unicast. We refer to such a setup as *TSCH-RB-X-Y-Z*, where X, Y, Z are the length of the three slotframes S0, S1, S2.

EB Slotframe. The first slotframe (S0) is dedicated to EB (TSCH Enhanced Beacon) transmissions, used for TSCH association and child-parent synchronization. The slotframe is longer than the number of nodes in the network, and consists of one sender-based dedicated (SBD) slot. As a result, every node has one Tx slot, and one Rx slot to listen for EBs from its time source. All transmissions in S0 are contention-free. We use $X = 397$ slots as a default length for S0.

Broadcast Slotframe. We add a slotframe (S1) with one common shared (CS) slot for RPL broadcast messages. As described in §4.3, S1 periodically collides with S0, as the latter has a higher priority. We use $Y = 31$ slots as a default length for S1. The probability for this slot to collide with S0 and be skipped is:

$$p(skip_{S1}) = 1 - (1 - p(coll_{S0})) = \frac{1}{S0_{len}/S0_{slots}} \approx 0.005 \quad (9)$$

Receiver-based Unicast Slotframe. We finally add a slotframe (S2) for unicast traffic with the RPL parent and all children, through a receiver-based shared (RBS) slot. In this setup, every node wakes up at a time offset derived from its own MAC address, listening for incoming traffic. We assign time offset $hash(MAC)\%Z$ to every node, where Z is the slotframe length. The probability for any slot in S2 to be skipped due to a collision with either of the previously described slotframes is:

$$p(skip_{S2}) = 1 - (1 - p(coll_{S0})) \times (1 - p(coll_{S1}))) \approx 0.037 \quad (10)$$

Which means that unicast transmissions to a given neighbor will occur with a 96.3% probability, and be postponed to the next slot otherwise.

4.4.3 Setup with Sender-based Unicast Slots

We introduce a variation of the setup above, where unicast transmissions take place in a sender-based shared (SBS) slot. S0 and S1 are the same as for TSCH-RB, but S2 is modified to use SBS instead of RBS. We refer to this setup as *TSCH-SB-X-Y-Z*, where X, Y, Z are the length of S0, S1, and S2.

In this setup, every sender has a transmit slot assigned in S2, and every node listens to their RPL parent and children's slot. The main benefit of TSCH-SB is that it reduces contention in comparison to TSCH-RB where all transmissions to a given node take place in the same slot. The downside is a higher energy baseline, as nodes need to wake up and listen at each of their or parents and children's slots. In scenarios where all nodes have a unique ID, and where the slotframe length Z is greater than the network size, one can ensure contention-free transmissions, and replace the shared *Tx* slot with a dedicated one.

Figure 5 shows Orchestra running with this setup with unicast slotframe of 101 slots (TSCH-SB-101) on the Indriya testbed's 98 TelosB nodes (*c.f.*, §6.1). In this specific run, we use the testbed node IDs to define the time offsets, resulting in cascaded transmissions. The figure shows the periodic transmissions of EBs (orange, cascaded), broadcast slots (green, all nodes aligned for contention-based communication) and unicast (blue, cascaded).

5. SYSTEM INTEGRATION

We discuss here a number of modifications we applied to Contiki's RPL implementation in order to gear it towards high reliability, and we introduce our implementation of TSCH and Orchestra.

Reliable RPL. Because we are aiming for high reliability, and to make sure TSCH nodes always have a reachable time source neighbor, we fine-tune RPL as follows.

First, we noticed that the ETX metric builds best-effort rather than reliable routes. For instance, a 56% PRR hop (ETX=1.8) is considered better than two perfect hops (ETX=1+1=2). When reliability matters, the latter should be clearly preferred. We use the squared ETX value as the link's cost in order to favor good links while preserving the gradient nature of RPL.

Second, we have to make RPL less aggressive in switching parents to avoid switching to a neighbor with which we do not have good statistics yet. To this end, we implement a simple probing mechanism: every node transmits a unicast probe to its best or second best parent at a given interval (we use 4 min in this paper's experiments). This allows a node to maintain up-to-date link estimates, ensuring that it always has some knowledge about the link quality to its backup (second best) parent. We also use the RSSI from received DIOs to calculate an initial estimate of the ETX towards the sender of this DIO.

Third, we noticed that, although RPL manages to repair routing loops eventually, a number of packets is always dropped in the process. To avoid this, we add a mechanism where, whenever a routing loop is detected, the receiver node transmits a unicast DIO message to the sender, thereby forcing the immediate update of both nodes' routing state.

We found that these enhancements greatly improve the end-to-end delivery ratio, not only with TSCH and Orchestra but also with asynchronous MAC layers (*e.g.*, up to 99.8% delivery with ContikiMAC+RPL data collection, the best results we are aware of in the literature).

Implementation. We implement[4] TSCH and Orchestra for the Contiki OS. This implementation supports two platforms, both evaluated in the next section: TelosB (MSP430, 10kB RAM, 48kB flash, external CC2420 radio) and NXP JN5168 (SoC with 32-bit RISC CPU, 32kB RAM, 256kB flash, 802.15.4 radio) [25]. In both implementations, all TSCH operations are governed by a timer interrupt (32kHz clock for TelosB, 16MHz for JN5168), and radio interrupts are disabled. TSCH is entirely responsible for managing the radio and reading out data whenever appropriate, as well as for generating and parsing IEEE802.15.4e enhanced ACKs and beacons. Runtime schedule modifications are done as follows: first wait for the end of the current slot, disable TSCH, proceed during next slot, resume TSCH operation.

TSCH (with Orchestra TSCH-min setup and queue space for 16 packets), when compiled for TelosB, has a memory footprint of 10kB flash and 2kB RAM. Due to the limited memory available on the TelosB, we were unable to run RPL with support for downwards routing in addition to TSCH, only upwards traffic is supported. All available Contiki RPL features are supported on the JN5168.

6. EVALUATION

In this section, we first present extensive experiments in two different testbeds, demonstrating Orchestra's superiority against state-of-the-art asynchronous MAC layers in reducing contention and achieving high reliability. Second, we run simulations to compare against a centralized scheduler, where we quantify the cost in latency and energy of our autonomous scheduling approach, and demonstrate Orchestra's adaptability to varying link conditions. Table 1 summarizes our testbed evaluation results.

6.1 Setup

Simulation and Testbeds. We use three different environments. First, we run experiments in the Indriya testbed [6], featuring 98 TelosB nodes in a three-floor office building in Singapore. We use node #2, on the top floor, as root of the network. An experiment in Indriya lasts 1h, repeated between 3 and 10 times. Results shown are averages with standard deviation.

Second, to overcome the memory restrictions of the TelosB platform, we use a testbed (JN-IoT) of 25 JN5168 nodes deployed in an office building. We use node #1, in a corner, as root. Results on the JN-IoT testbed are from a single experiment for each configuration, each experiment lasting between 16h and 72h. The results we report are gathered through a total of 219 testbed experiments and we routed 1,178,601 UDP packets from source to destination.

Third, to compare Orchestra against static scheduling with full control on the network conditions, we use Cooja, Contiki's network simulator. Cooja emulates TelosB nodes running compiled MSP430 firmware. Cooja allows us to have full control over network conditions and emulates varying connectivity in a repeatable manner.

Testbed	Traffic	Protocol	PDR [%] (loss rate)
Indriya Nodes: TelosB Size: 98 nodes ∅ hops: 4.2 ∅ density: 16	*Upwards* 60s interval	*Always-on*	99.91 (1 / 1139)
		ContikiMAC-64	99.8 (1 / 482)
		ContikiMAC-8	99.0 (1 / 98)
		TSCH-min-3	99.87 (1 / 779)
		TSCH-min-5	99.2 (1 / 127)
		TSCH-RB-7	99.996 (1 / 27,044)
		TSCH-SB-7	99.996 (1 / 25,450)
		TSCH-SB-47	99.997 (1 / 35,700)
JN-IoT Nodes: JN5168 Size: 25 nodes ∅ hops: 3 ∅ density: 13	*Upwards* 30s interval	*Always-on*	99.97 (1 / 2893)
		TSCH-min-5	99.86 (1 / 715)
		TSCH-SB-29	99.991 (1 / 11,160)
	Down-up 50s interval	*Always-on*	99.92 (1 / 1152)
		TSCH-min-5	99.85 (1 / 674)
		TSCH-SB-29	99.98 (1 / 5607)

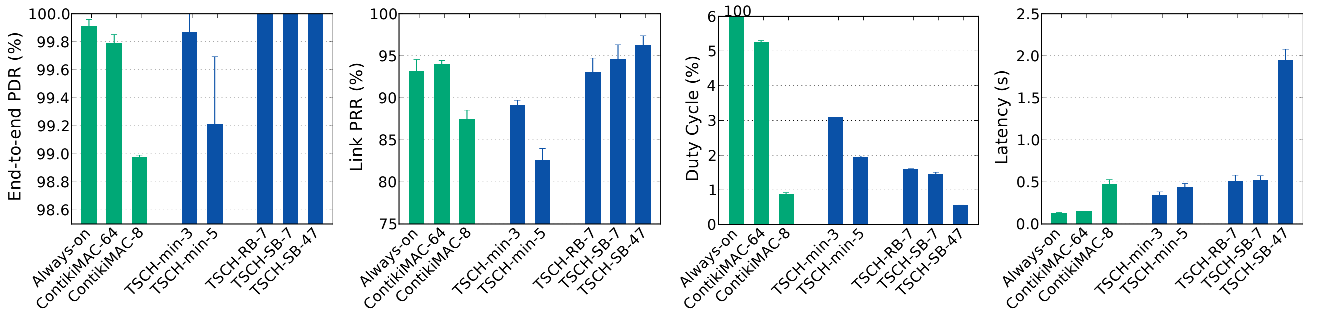
Table 1: Testbed experiments summary. Orchestra TSCH-RB and TSCH-SB consistently achieve the lowest loss rates, outperforming the low-power alternative ContikiMAC by two orders of magnitude. Through contention avoidance and channel hopping, Orchestra also beats the *Always-on* MAC, with 4 to 35 times fewer losses.

Protocols. We use Orchestra with all three configurations from §4.4. We compare Orchestra against a centralized, static scheduler (described in §6.6) and the asynchronous MAC layers *Always-on* and ContikiMAC at 8Hz and 64Hz (*c.f.*, §3). All protocols use a maximum of 8 retransmissions per hop, and a maximum of 16 packets in the queue. As for the TSCH slot timing, on TelosB, we use 15 ms slots and a guard time of ±0.6 ms. On JN5168, we use 10 ms slots and a ±0.25 ms guard time. Finally, we run *Always-on* and ContikiMAC over the best channel available, 26, and TSCH over the four best channels: 15, 20, 25, 26.

Application Scenarios. We run two different application scenarios: *upwards* routing and *down-up* routing. In *upwards*, nodes transmit a packet to the network root at a given average interval, with added jitter to emulate non-deterministic traffic. This is done with RPL downwards routing disabled. In *down-up*, the network root picks a node in the network at random and transmits a request to it. The destination answers immediately by sending a response back. This is a classic traffic pattern in IoT scenarios, *e.g.*, in RESTful architectures with CoAP. In all cases but centralized scheduling, nodes run RPL, 6LoWPAN, and all application traffic is raw UDP with a 16 bytes payload. The first 15 minutes of every run are always excluded, to allow the network to form and RPL to converge to a stable topology.

Metrics. The three main metrics we focus on are the end-to-end packet delivery ratio (PDR), end-to-end latency, and radio duty cycle. The PDR is the portion of packets sent at the application layer which make it to their final destination, possibly over multiple hops. The end-to-end latency

[4]The code is available at `https://github.com/simonduq/orchestra`.



(a) (End-to-end) PDR (b) Unicast PRR (c) Duty Cycle (d) Latency

Figure 6: *Upwards* Experiments in Indriya. Orchestra results in the highest delivery ratios, here between 99.996% and 99.997%. This is partly explained by higher link PRR, reaching as high as 97%, in comparison with ContikiMAC's 94% (twice as many losses in the latter case). ContikiMAC, however, offers the best latency-energy balance. Note that the y-axis of the PDR and PRR plots does not begin a zero.

is measured between the initial application's intention to send a packet and its reception at the final destination. The duty cycle is the portion of time spent with the radio on (either transmitting, listening or receiving), and is used as a platform-independent measure of energy consumption. In addition, we look at the link packet reception rate (PRR), which the per-hop, per-transmission attempt success rate.

6.2 Comparison with Asynchronous MACs

We run Orchestra, *Always-on* and ContikiMAC in Indriya, with *upwards* traffic generated at every node at an average 60 s interval. Figure 6 summarizes our results.

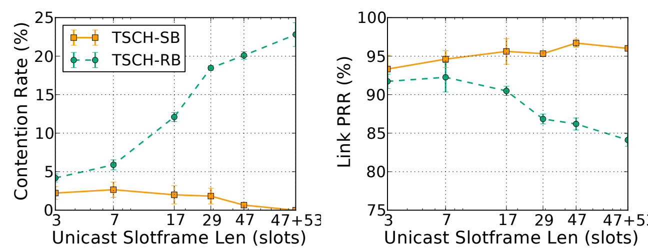
All Orchestra configurations achieve the highest PDRs, above 99.99%, *i.e.*, less than one end-to-end loss per 10k packets or 10^{-4} loss rate. The best asynchronous results are with *Always-on*, reaching a PDR of 99.9% *i.e.*, a loss rate of 10^{-3}, one order of magnitude behind Orchestra. ContikiMAC is an other order of magnitude below, with 99% or a loss rate of 10^{-2}.

Figure 6b shows the MAC success rate, *i.e.*, the link quality achieved by each protocol. For all protocols, there is a clear correlation between link quality and end-to-end PDR, which means the overall performance is mostly limited by medium access (rather than routing or queue drops). Orchestra achieves the highest MAC success rates, *e.g.*, 97% with TSCH-SB-47 against 93% for *Always-on*. We attribute this mostly to Orchestra's ability to reduce contentions. TSCH-min, which is fully contention-based, results in the lowest success rates.

ContikiMAC obtains a loss rate which is two orders of magnitude above Orchestra, but achieves the best latency-energy balance. For instance, ContikiMAC@8Hz yields a 0.5s latency for duty cycle of 0.8%, while TSCH-SB-7 has a duty cycle of 1.4% for the same latency. *Always-on* results by design in 100% duty cycle, and also achieves the lowest latency results, as nodes never have to wait for their neighbor to wake up before sending.

6.3 Contention Control and Scalability

One of the main goals of Orchestra is to reduce contention through scheduling, in order to increase link success rate and overall reliability. A limitation, however, is that Orchestra achieves this by having slotframes with a fixed size, and



(a) Contention Rate (b) Unicast PRR

Figure 7: With TSCH-SB, longer slotframes translate to more available timeslots and less contention. With 47+53 slots, the network is contention-free. Lower values result in a contention rate below 3%. TSCH-RB suffers from longer slotframes, due to increased pressure on the nodes' *Rx* slots. Note that the y-axis of the PRR plot does not begin a zero.

spreading out nodes across all slots in the slotframe. This might cause network capacity and scalability issues. We investigate this by varying the unicast slotframe length in TSCH-RB and TSCH-SB. We argue that varying the slotframe for a fixed number of nodes (Indriya's 98 nodes) produces an effect similar to increasing traffic load or network size for a fixed slotframe; this allows us to get insights on Orchestra's network capacity and scalability.

Figure 7 shows our results with TSCH-RB-3 to TSCH-RB-47 and TSCH-SB-3 to TSCH-SB-47 (receiver-based and sender-based shared slots). We also introduce an extra case, TSCH-RB-47+53 and TSCH-SB-47+53, where two unicast slotframes are used, with size 47 and 53. This increases the number of different unicast slots to 100, more than the network size. In this particular case, we use the nodes' unique ID to allocate a unique slot to each. TSCH-SB-47+53 is therefore guaranteed to be fully contention-free (sender-based dedicated slots).

Figure 7a shows that, as predicted by our analytical model in §4.3.1, the contention rate for TSCH-RB increases at larger slotframes. TSCH-SB shows much lower contention rates, and with the opposite trend: it performs at its best with longer slotframes to eventually reach no contention at

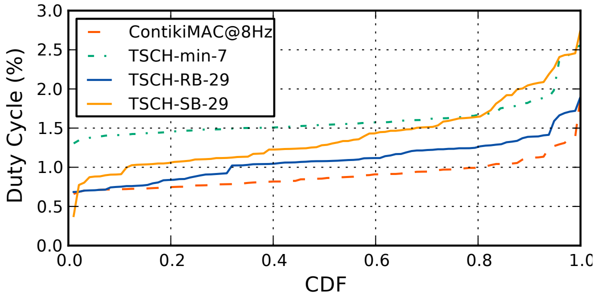


Figure 8: Orchestra's sender-based schedules (TSCH-SB) have the drawback of producing less evenly distributed duty cycles among nodes, compared to ContikiMAC, TSCH-min, or TSCH-RB.

all in the 47+53 case. With slotframes of length 3 to 29, contentions remain roughly constant, in the 2–3% range. Such low contention rates result in high link PRR, above 95% in most cases (Figure 7b). We attribute TSCH-SB's superiority in here in handling contention to its sender-based nature. Receiver-based slots are more likely to cause contention, where all transmissions to a given node take place in the same slot. This is not captured by our analytical model from §4.3.1, which considers clique networks only (in a clique, transmissions from/to any node may collide).

6.4 Energy Distribution and Bounds

Figure 8 shows how different Orchestra configurations affect the distribution of radio duty cycle among nodes. For each configuration, we pick parameters leading to comparable duty cycles, *i.e.*, with all nodes between 0.3% and 3%. A first observation is that TSCH-SB results in less evenly-balanced radio duty cycles than TSCH-RB. This can be attributed to the varying number of unicast *Rx* slots in this *upwards* traffic scenario. TSCH-SB requires one such slot for every child, whereas in TSCH-RB nodes have a single unicast *Rx* slot.

Table 2 shows the minimum and maxium per-node radio duty cycle, as measured during all experiments, against the theoretical bounds defined in §4.3. For all configurations, including ContikiMAC, we find the theoretical lower bounds to be accurate. Deriving accurate upper bounds proves more difficult. For ContikiMAC, which is asynchronous, the upper bound is the case where a node transmits broadcasts continuously, which can lead to a duty cycle as high as 88%. With TSCH-SB-29 we also get a very high maximum value of 31%, which corresponds to the case where a node uses all of its 29 unicast slots for listening to its children. TSCH-min-7 and TSCH-RB-29, due to their more predictable schedules, allow us to derive more realistic and usable upper bounds, both under 1.75× the maximum measured.

To summarize, in scenarios where energy consumption must be bounded, Orchestra can produce schedules that guarantee no node will have its radio turned on more than *e.g.*, 3% of the time.

6.5 Orchestra in IoT Scenarios

We now move to the JN-IoT testbed, and run both *upwards* and *down-up* traffic experiments. Because ContikiMAC is not ported to the JN5168 platform, we only run *Always-on* and TSCH. We use the TSCH-min-5 and TSCH-SB-29 configurations. In the latter, as we have a unicast slotframe of size greater than the network size, we use the node's unique ID to derive contention-free slots. The results are shown in Table 1.

In general, the PDRs in JN-IoT are lower than in Indriya. This is due to differences in physical topologies and deployment sites. In particular, we noticed significant fluctuations in link quality during workdays when compared to nights, to an extent much greater than in Indriya.

In both *upwards* and *down-up* experiments, TSCH-SB-29 achieves the highest PDRs: about 4× fewer losses than with the *Always-on* MAC and 10× fewer than TSCH-min-5. For all three configurations (*Always-on*, TSCH-min-5, TSCH-SB-29), the results when involving downwards routing are worse than in the *upwards* scenario. We attribute this to inherent properties of RPL rather than TSCH or Orchestra.

In RPL, the topology is optimized towards the root, and downward traffic is merely enabled by reusing links in the reverse direction, with no guarantee on link quality. Figure 9 shows, for every node, the average link PRR when routing up (from the node) or down (towards the node). Although links are mostly symmetric, there are a few notable exceptions, such as node 20 with an average PRR of 86% up, 71% down. In the TSCH-SB-29 experiment, out of 258k packets sent in 72h, 46 were lost, 35 of them while going downwards, 11 while going upwards. Half of the downwards losses are attributed to link losses (in part due to link asymmetry), others are due to temporary RPL inconsistencies (loops or outdated routes following topology updates).

Overall, this series of experiment in the JN-IoT testbed demonstrates that Orchestra can run reliably on different hardware platforms and networks, as well as with more challenging traffic patterns.

	Duty Cycle (%)			
	Experiments		Theoretical	
	Min	Max	Min	Max
Contikmac@8Hz	0.66	1.89	0.6	88
TSCH-min-7	1.30	2.56	1.14	4.48
TSCH-RB-29	0.68	1.90	0.54	3.00
TSCH-SB-29	0.38	2.75	0.28	31.19

Table 2: Measured and theoretical min and max duty cycle. Lower bounds are predicted accurately in all cases, but upper bounds with ContikiMAC or TSCH-SB are very pessimistic.

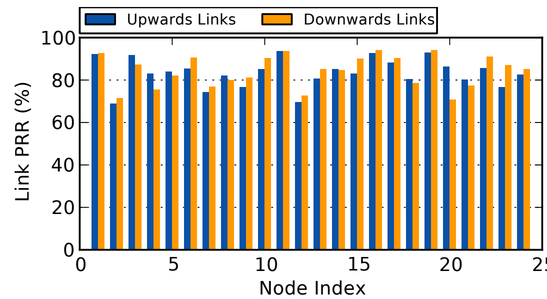


Figure 9: *Down-up* experiment in the JN-IoT testbed: per-node link PRR. All links are usable both ways, albeit not perfectly symmetric.

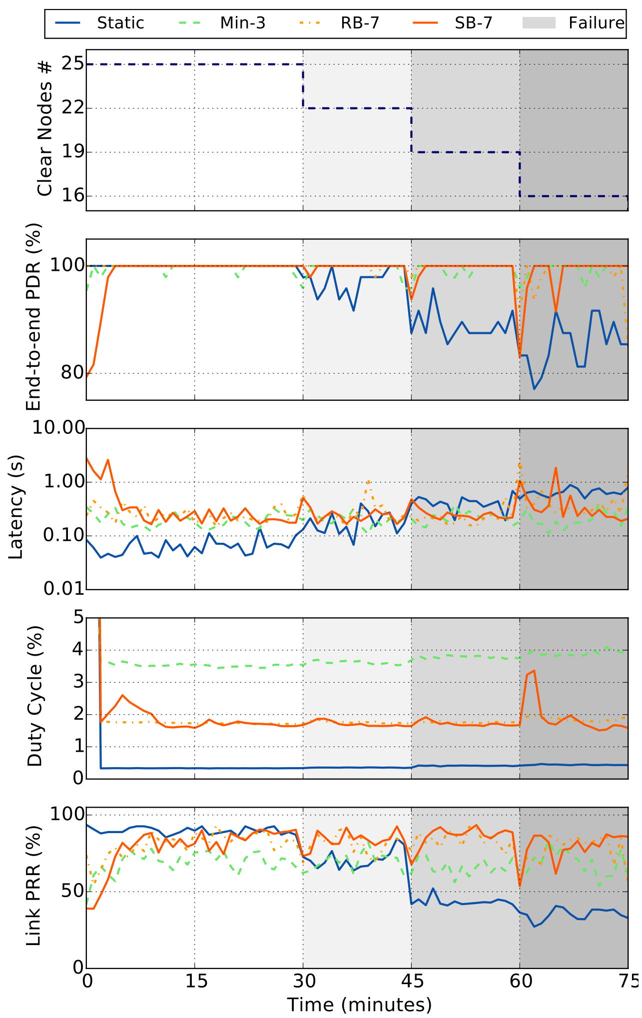


Figure 10: Orchestra compared to a simple static schedule. From minute 30 onward, we inject link failures on three random nodes every 15 minutes. Orchestra quickly adapts to these changes while the static schedule suffers despite retransmission slots. Note that the y-axis of the node count and the PDR plots does not begin a zero.

6.6 Comparison with Static Scheduling

The goal of this section is to evaluate the overhead of using Orchestra with RPL when compared to a centralized, static scheduler, and at the same time show the flexibility of Orchestra with RPL in reacting to network dynamics.

As a benchmark, we implement a simple offline scheduler inspired by the work of Pöttner et al. [28]. The scheduler takes PRR measurements for every link as input, computes their routing metric – it uses squared ETX, as in Orchestra, to favor good links – and uses Dijkstra to compute the shortest path from each node to the sink. Routes are built to support data-collection traffic only, at a pre-defined interval (30 s in our experiments). For each route, we compute a static TSCH schedule of transmissions throughout the network. Routes are scheduled redundantly to allow for retransmissions. Note that this is a simple scheduler, with no runtime re-configuration nor multi-path transmissions.

For this particular evaluation, we utilize Cooja/MSPSim simulations in order to fully control the conditions of the experiments. We configure the wireless links in the model to reflect the links quality we measured in the JN-IoT testbed. It should be noted, however, that this wireless link model in Cooja exhibits uniformly distributed losses which are not realistic in bursty low-power networks. Nonetheless, we argue that it is sufficient for this particular experiment.

In order to simulate network dynamics, we tune down the reception probability of three randomly chosen nodes to 0.1× their initial PRR *i.e.*, the affected nodes can still transmit as before, but are 10× more likely to drop incoming data packets. We repeat this every 15 minutes, choosing another three nodes without recovering the previously attenuated nodes. This is done exactly in the same order for the different experimental setups. We run the experiments for 75 minutes in Cooja, and start introducing failures only after 30 min to demonstrate a baseline of stable conditions.

We compare the static schedule (denoted as *Static*) to the following three configurations of Orchestra: TSCH-min-3, TSCH-RB-7, TSCH-SB-7. Figure 10 shows the results for each setup. First of all, when the network is stable (until minute 30), we notice the extremely low cost of *Static*. Compared to Orchestra, *Static* yields a 4 to 8× lower duty cycle (TSCH-SB-7 and TSCH-Min-3) and 10× lower latency. We attribute these to the schedules and routes in *Static*, which are (1) dimensioned to the traffic load (2) optimized to minimize latency along the network shortest path (3) and built offline involving no neighbor discovery and routing protocol at runtime. In terms of end-to-end PDR, both solutions perform exceptionally well, with the exception that Orchestra needs a little bit of startup time (about 5 minutes in this particular setup) to find good links and stabilize, while *Static* comes pre-configured with the best links and schedules.

Second, we examine the results after injecting network reception failures (minute 30 and onward): *Static* directly decreases in terms of performance, because it does not adjust to topology changes. With Orchestra, however, the PDR drops temporarily each 15 minutes and then RPL quickly finds new routes and recovers.

Overall, Orchestra reacts quickly to changes in network connectivity, but compared to a schedule built centrally for a specific topology and traffic pattern, it has a significant overhead in both latency and energy.

7. RELATED WORK

We review related work in three categories: (1) scheduled MAC and routing, (2) asynchronous, low-power routing and (3) synchronous transmissions.

Scheduled MAC and Routing. The idea of synchronizing nodes and channel hopping to combat multi-path fading and external interference is established in many technologies, including Bluetooth and cellular systems. It was brought to low-power wireless networking through a proprietary protocol called Time Synchronized Mesh Protocol (TSMP). Early promising results [8] pushed the core technology of TSMP to be standardized as WirelessHART [14], ISA100.11a [17] and IEEE802.15.4e [1].

In a WirelessHART network, a central entity computes the communication schedule based on application requirements and on information it gathers about network connectivity. The schedule is injected into the network and continuously updated throughout the lifetime of the network [18]. In static networks with predictable traffic patterns, commercial TSCH-like networks such as WirelessHART or SmartMesh IP [36] offer very high reliability (published results include 99.999% on a 49 node industrial deployment [8], 99.95% on a 15-node testbed [28]), and a decade of battery lifetime [39]. Note that the centralized scheduling algorithms are not part of the standards and much attention has been given to scheduling theory in the context of TSCH networks [27, 30, 31, 44].

In scenarios where the network topology and traffic patterns are not fixed, decentralized scheduling is applicable. Tinka et al. [34] present, to the best of our knowledge, the first paper to propose and demonstrate a decentralized scheduling solution for TSCH networks. A common shared slot is used for neighbor discovery and for negotiating the addition of dedicated slots. This work motivated further approaches [26, 43, 22]. For example, Morell et al. [22] take schedule negotiation one step further and propose multi-hop reservation through label switching. Focusing on very low data-rates and not limiting itself to TSCH, Dozer [2] employs distributing scheduling for energy-efficient routing. Its goal of high energy efficiency leads to significantly higher latency when compared to Orchestra. For example, experiments report an average latency of 30 s and a PDR of 98.5% at a radio duty-cycle of 0.2% for a testbed of 90 nodes [12]. To the best of our knowledge, Orchestra is the first distributed solution which does not require negotiation between nodes.

In October 2013, the IETF created the 6TiSCH working group, which standardizes how to use an IPv6-enabled upper stack on top of IEEE802.15.4e TSCH. 6TiSCH specifies a number of mechanisms to manage the TSCH schedule [33]. It defines a CoAP-based management protocol which can be used for central scheduling, and a protocol for neighbor nodes to negotiate distributed scheduling. 6TiSCH, however, leaves it to the implementer to decide which scheduling approaches fit best. Orchestra could be considered a third scheduling option for 6TiSCH networks.

Asynchronous, Low-power Routing. At the other end of the spectrum, asynchronous, low-power routing enables dynamic applications and transparently allows nodes to join and leave. However, its packet loss can be up to several percent: For example, CTP reports delivery ratios between 94% and 99.9% in data collection depending on the network size and topology [15]. Recent approaches such as ORW [21], ORPL [11] and BFC [29] improve in terms of energy efficiency and – in part – latency over CTP but still have an average PDR of roughly 99%. Moreover, these do not employ scheduling and channel hopping; thus, they cannot avoid external interference, multi-path fading, or contention as efficiently as Orchestra. EM-MAC [32] and MiCMAC [24] integrate channel hopping into asynchronous, low-power routing. As a result, they increase reliability, especially in the presence of external interference, but channel hopping with the loosely synchronized operation leads to an extra overhead in terms of latency and radio duty-cycle [24].

Synchronous Transmissions. A recent direction in low-power wireless networks is synchronous transmissions: Glossy [13] provides fast and efficient network flooding by precisely timing transmissions: it ensures that a receiver successfully receives even in the presence of multiple, concurrent transmissions – of the same packet – by exploiting constructive interference and capture effect. LWB [12] provides data collection and dissemination primitives on top of Glossy. In LWB, a central scheduler dynamically injects collection and dissemination schedules based on the traffic requests of the individual nodes. Chaos [20] combines synchronous transmission and in-network processing to provide fast, reliable, and energy efficient data collection and processing. Others show fast data dissemination [7, 5] and point-to-point routing [3, 42] based on synchronous transmissions. Approaches based on synchronous transmissions report very high reliability, low latency, and energy efficiency. For example, Glossy reports a reliability of more than 99.99%. However, synchronous transmissions always need one central entity to control and initiate the synchronous transmissions. Moreover, they – inherently – have a high channel utilization and are often limited in terms of generality such as being restricted to periodic traffic. Orchestra, in contrast, provides general purpose routing with RPL and IPv6.

Summary. Orchestra differs from previous TSCH scheduling approaches in that nodes compute their own schedule locally and autonomously, based on routing-layer information. Thus, it does not require any central entity nor communication among nodes to reserve schedules and paths. This makes it as flexible and application-independent as non-scheduled asynchronous solutions. Orchestra is significantly more reliable – by two orders of magnitude in our experiments – than state-of-the-art asynchronous low-power routing while achieving a similar latency-energy balance. Compared to synchronous transmissions, Orchestra has the advantage to support random-access traffic, which makes it suitable for non-deterministic low-power IPv6 applications.

8. CONCLUSION

This paper introduces Orchestra, a solution for autonomous scheduling of TSCH in RPL networks. Orchestra runs without any central scheduling entity nor negotiation, and supports low-power random-access traffic. The key idea is to provision a set of slots for different traffic planes, and to define the slots in such a way that they can be automatically installed/removed as the RPL topology evolves.

We implement Orchestra in Contiki and conduct an extensive evaluation in simulation and on two different testbeds. We demonstrate the practicality of Orchestra and its ability to consistently achieve the highest delivery ratio, while striking an interesting latency-energy balance.

As part of future work, we plan to investigate how to optimize time synchronization, reduce energy consumption further, and explore management solutions for 6TiSCH to enable runtime reconfiguration of Orchestra.

Acknowledgments

We would like to thank the CIR Lab in Singapore for providing the Indriya testbed, and Amy L. Murphy, our shepherd, for her insightful comments. This work was partly supported by the distributed environment Ecare@Home funded by the Swedish Knowledge Foundation 2015-2019, the EIT Digital RICH Activity, and VINNOVA (Sweden's Innovation Agency).

9. REFERENCES

[1] 802.15.4e Task Group. 802.15.4e-2012: IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks

(LR-WPANs) Amendment 1: MAC sublayer, 16 April 2012.

[2] N. Burri, P. V. Rickenbach, and R. Wattenhofer. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, 2007.

[3] D. Carlson, M. Chang, A. Terzis, Y. Chen, and O. Gnawali. Forwarder selection in multi-transmitter networks. In *Proceedings of the Conference Distributed Computing in Sensor Systems (DCOSS)*, 2013.

[4] T. Chang, T. Watteyne, K. Pister, and Q. Wang. Adaptive synchronization in multi-hop tsch networks. *Comput. Netw.*, 76(C):165–176, Jan. 2015.

[5] M. Doddavenkatappa and M. C. Chan. P3: A practical packet pipeline using synchronous transmissions for wireless sensor networks. In *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, 2014.

[6] M. Doddavenkatappa, M. C. Chan, and A. Ananda. Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed. In *Proceedings of the Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, 2011.

[7] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *Proceedings of the Symposium on Networked Systems Design & Implementation (USENIX NSDI)*, 2013.

[8] L. Doherty, W. Lindsay, and J. Simon. Channel-Specific Wireless Sensor Network Path Data. In *Proceedings of the 16th International Conference on Computer Communications and Networks (ICCCN)*, pages 89–94, Turtle Bay Resort, Honolulu, Hawaii, USA, 13-16 August 2007. IEEE.

[9] A. Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical Report T2011:13, Swedish Institute of Computer Science, 2011.

[10] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the Conference on Local Computer Networks (IEEE LCN)*, 2004.

[11] S. Duquennoy, O. Landsiedel, and T. Voigt. Let the Tree Bloom: Scalable Opportunistic Routing with ORPL. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys 2013)*, Rome, Italy, Nov. 2013.

[12] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-Power Wireless Bus. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2012.

[13] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient Network Flooding and Time Synchronization with Glossy. In *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, 2011.

[14] H. C. Foundation. WirelessHART Specification 75: TDMA Data-Link Layer, 2008. HCF_SPEC-75.

[15] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2009.

[16] J. W. Hui and D. E. Culler. Ip is dead, long live ip for wireless sensor networks. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2008.

[17] ISA. ISA-100.11a-2011 – Wireless Systems for Industrial Automation: Process Control and Related Applications, 2011.

[18] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle. When HART goes wireless: Understanding and implementing the WirelessHART standard. In *ETFA*, pages 899–907. IEEE, 2008.

[19] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler. Industry: Beyond Interoperability: Pushing the Performance of Sensor Network IP Stacks. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 1–11, New York, NY, USA, 2011. ACM.

[20] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and Efficient All-to-All Data Sharing and In-Network Processing at Scale. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys)*, 2013.

[21] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson. Low Power, Low Delay: Opportunistic Routing meets Duty Cycling. In *Proceedings of the Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, 2012.

[22] A. Morell, X. Vilajosana, J. L. Vicario, and T. Watteyne. Label switching over IEEE802.15.4e networks. *Transactions on Emerging Telecommunications Technologies*, 24(5):458–475, Aug. 2013.

[23] D. Moss and P. Levis. BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Technical Report SING-08-00, Stanford, 2008.

[24] B. A. Nahas, S. Duquennoy, V. Iyer, and T. Voigt. Low-Power Listening Goes Multi-Channel. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (IEEE DCOSS 2014)*, Marina Del Rey, CA, USA, May 2014.

[25] NXP Laboratories UK Ltd. *Data Sheet: JN516x IEEE802.15.4 Wireless Microcontroller*, 2013.

[26] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia. Traffic Aware Scheduling Algorithm for reliable low-power multi-hop IEEE 802.15.4e networks. In *PIMRC*, pages 327–332. IEEE, 2012.

[27] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel. On Optimal Scheduling in Duty-Cycled Industrial IoT Applications using IEEE802.15.4e TSCH. *IEEE Sensors Journal*, 13(10):3655 – 3666, Oct. 2013. doi:10.1109/JSEN.2013.2266417.

[28] W.-B. Pöttner, H. Seidel, J. Brown, U. Roedig, and L. Wolf. Constructing Schedules for Time-Critical Data Delivery in Wireless Sensor Networks. *ACM Trans. Sen. Netw.*, 10(3):44:1–44:31, May 2014.

[29] D. Puccinelli, S. Giordano, M. Zuniga, and P. J. Marrón. Broadcast-free collection protocol. In *Proceedings of the 10th ACM Conference on Embedded*

Network Sensor Systems, SenSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[30] A. Saifullah, P. B. Tiwari, B. Li, C. Lu, and Y. Chen. Accounting for Failures in Delay Analysis for WirelessHART Networks, 2012.

[31] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Real-Time Scheduling for WirelessHART Networks. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 150–159, Nov 2010.

[32] L. Tang, Y. Sun, O. Gurewitz, and D. B. Johnson. Em-mac: A dynamic multichannel energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '11, pages 23:1–23:11, New York, NY, USA, 2011. ACM.

[33] X. Thubert (Ed.), T. Watteyne, R. Struik, and M. Richardson. An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4e - draft-ietf-6tisch-architecture-06, Mar. 2015. IETF Draft.

[34] A. Tinka, T. Watteyne, K. S. J. Pister, and A. M. Bayen. A Decentralized Scheduling Algorithm for Time Synchronized Channel Hopping. *EAI Endorsed Transactions on Mobile Communications and Applications*, 11(1), 9 2011.

[35] X. Vilajosana (Ed.) and K. Pister. Minimal 6TiSCH Configuration - draft-ietf-6tisch-minimal-06, Mar. 2015. IETF Draft.

[36] T. Watteyne, L. Doherty, J. Simon, and K. Pister. Technical Overview of SmartMesh IP. In *Proceedings of the 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, IMIS '13, pages 547–551, Washington, DC, USA, 2013. IEEE Computer Society.

[37] T. Watteyne, S. Lanzisera, A. Mehta, and K. Pister. Mitigating Multipath Fading through Channel Hopping in Wireless Sensor Networks. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5, May 2010.

[38] T. Watteyne, A. Mehta, and K. Pister. Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense. In *Proceedings of the 6th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*, PE-WASUN '09, pages 116–123, New York, NY, USA, 2009. ACM.

[39] T. Watteyne, J. Weiss, L. Doherty, and J. Simon. Industrial IEEE802.15.4e Networks: Performance and Trade-offs. In *Proceedings of IEEE International Conference on Communications (ICC), Internet of Things Symposium*, London, UK, June 2015.

[40] T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 Routing Protocol for Low power and Lossy Networks, Mar. 2012. RFC 6550.

[41] Y.-C.Jenq. On the stability of slotted aloha systems. *Communications, IEEE Transactions on*, 28(11):1936–1939, Nov 1980.

[42] D. Yuan, M. Riecker, and M. Hollick. Making 'Glossy' Networks Sparkle: Exploiting Concurrent Transmissions for Energy Efficient, Reliable, Ultra-Low Latency Communication in Wireless Control Networks. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, 2014.

[43] P. Zand, A. Dilo, and P. Havinga. D-MSR: A Distributed Network Management Scheme for Real-Time Monitoring and Process Control Applications in Wireless Industrial Automation. *Sensors*, 13(7):8239–8284, 2013.

[44] H. Zhang, P. Soldati, and M. Johansson. Optimal Link Scheduling and Channel Assignment for Convergecast in Linear wirelessHART Networks. In *Proceedings of the 7th International Conference on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, WiOPT'09, pages 82–89, Piscataway, NJ, USA, 2009. IEEE Press.