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 sloting to dynamic networks as envisioned in the Internet of Things. Commonly, these applications do not have pre-defined 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

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., 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 chal-
This 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 particular, Orchestra achieves this strong reliability while keeping energy and latency close to the state of the art.

**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 [33] developed in the IETF 6TiSCH working group [33], which employs a slotframe-based scheduling 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 = 0.48%.

In practice, this baseline cost can be pushed even lower. In our testbeds (see §6.1) we measured an average drift between two nodes of 0.5 ppm among two nodes. Assuming a guard period of 12 ms, we measure a 0.088% cost. In practice, this baseline cost can be pushed even lower. In our testbeds (see §6.1) we measured an average drift between two 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.

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 amount 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 Network Bootstrap such as TSCH beacons, RPL signaling, or application data. With slotframes and slots assigned to different traffic planes, information from the RPL topology, and following a set of scheduling (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 3e, #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 assignment.
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.

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.

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.

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

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 wake 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.
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]:

\[ p(\text{cont}_{\text{slotted ALOHA}}) = 1 - e^{-T} \]

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:

\[ p(\text{cont}_{\text{CS}}) = 1 - e^{-TL} \]

(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:

\[ p(\text{cont}_{\text{RBS}}) = p(\text{cont}_{\text{SBS}}) = \begin{cases} 1 - e^{-T/L} & \text{if } L \geq N \\ 1 - e^{-T} & \text{otherwise} \end{cases} \]

(3)

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

\[ p(\text{cont}_{\text{SBD}}) = 0 \]

Figure 4 shows \( p(\text{cont}_{\text{CS}}) \), \( p(\text{cont}_{\text{RBS}}) \) and \( p(\text{cont}_{\text{SBS}}) \) for a 20-node network, with an overall traffic load of one packet per 500 ms and 10 ms slots (\( T = \frac{1}{2} \)). 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 \( B_{\text{slots}} \) be the number of slots in slotframe \( B \) of length \( B_{\text{len}} \). Let \( \text{coll}_B \) denote the event of a given slot colliding with any slot in \( B \). The probability for any slot to collide with \( B \) is:

\[ p(\text{coll}_B) = \frac{1}{B_{\text{len}}/B_{\text{slots}}} \]

(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:

\[ p(\text{skip}_A) = 1 - \prod_{B \in SF, B_h < A_h} \left( 1 - p(\text{coll}_{A,B}) \right) \]

(6)

4.3.3 Duty Cycle Bounds

Let \( r_x \text{Min} \text{De} \) be the radio duty cycle of a \( R_x \) slot when no communication occurs, defined as \( r_x \text{Min} \text{De} = \frac{r_x \text{GuardTime}}{\text{slotLength}} \) where \( r_x \text{GuardTime} \) denotes the TSCH \( R_x \) guard time and \( \text{slotLength} \) the TSCH slot duration. \( A_{dc,RxBase} \) is the listening cost of slotframe \( A \), in the absence of communication:

\[ A_{dc,RxBase} = (1 - p(\text{skip}_A)) \times \frac{A_{rx}\text{Slots} \times r_x \text{Min} \text{De}}{A_{len}} \]

(7)

Here, \( A_{rx}\text{Slots} \) is the number of slots with \( R_x \) 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 \( A_{dc,RxBase} = A_{dc,RxBase} \).

Because of reception guard times, \( R_x \) slots result in a maximum duty cycle (denoted \( r_x \text{Max} \text{De} \)) higher than \( T_x \) slots (denoted \( T_x \text{Max} \text{De} \)). The upper bound duty cycle is reached when a full-sized packet is received (resp. sent) at every \( R_x \) slot (resp. \( T_x \) only slot):

\[ A_{dc,Upper} = (1 - p(\text{skip}_A)) \times \frac{A_{rx}\text{Slots} \times r_x \text{Max} \text{De} + A_{tx}\text{Only}\text{Slots} \times T_x \text{Max} \text{De}}{A_{len}} \]

(8)

Where \( A_{tx}\text{Only}\text{Slots} \) is the number of slots in slotframe \( A \) with \( T_x \) flag but not \( R_x \) flag.

The system-wide lower and upper bound duty cycle are denoted respectively \( dc\text{Lower} = \sum_{A \in SF} A_{dc,Lower} \) and \( dc\text{Upper} = \sum_{A \in SF} A_{dc,Upper} \).

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.
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:

\[
p(\text{skip}_{S1}) = 1 - (1 - p(\text{coll}_{S0})) = \frac{1}{S0_{\text{len}}/S0_{\text{slots}}} \approx 0.005 \tag{9}
\]

**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 \( \text{hash}(\text{MAC})\%Z \) 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:

\[
p(\text{skip}_{S2}) = 1 - (1 - p(\text{coll}_{S0})) \times (1 - p(\text{coll}_{S1})) \approx 0.037 \tag{10}
\]

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.
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 Coojia, Contiki’s network simulator. Cooja emulates TelosB nodes running compiled MSP430 firmware. Coojia allows us to have full control over network conditions and emulate varying connectivity in a repeatable manner.

**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 in the network at random and transmits a request to it. The protocol answers immediately by sending a response.

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 towards the sender of this DIO.

### Table 1: Testbed experiments summary. Orchestra consistently achieve the lowest loss rates, outperforming the low-power alternative ContikiMAC by two orders of magnitude.

<table>
<thead>
<tr>
<th>Testbed</th>
<th>Traffic</th>
<th>Protocol</th>
<th>PDR [%] (loss rate)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Indriya</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nodes: TelosB</td>
<td>98 nodes</td>
<td>Upwards</td>
<td>Always-on-50s interval</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ContikiMAC-64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.91 (1 / 1139)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ContikiMAC-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.9 (1 / 98)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TSCH-min-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.87 (1 / 779)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TSCH-min-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.2 (1 / 127)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TSCH-RB-7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.96 (1 / 27444)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TSCH-SB-7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.996 (1 / 25450)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TSCH-SB-47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.997 (1 / 35709)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.991 (1 / 11160)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.86 (1 / 715)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.85 (1 / 674)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.98 (1 / 5607)</td>
</tr>
</tbody>
</table>

**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 Coojia, Contiki’s network simulator. Cooja emulates TelosB nodes running compiled MSP430 firmware. Coojia allows us to have full control over network conditions and emulate varying connectivity in a repeatable manner.

**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.

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 towards the sender of this DIO.
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.

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 60s 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. ContikiMAC, 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 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
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 maximum 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. TSR-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.

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.

<table>
<thead>
<tr>
<th></th>
<th>Duty Cycle (%)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Experiments</td>
<td>Theory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>ContikiMAC@8Hz</td>
<td>0.66</td>
<td>1.89</td>
<td>0.6</td>
</tr>
<tr>
<td>TSCH-min-7</td>
<td>1.30</td>
<td>2.56</td>
<td>1.14</td>
</tr>
<tr>
<td>TSCH-RB-29</td>
<td>0.68</td>
<td>1.90</td>
<td>0.54</td>
</tr>
<tr>
<td>TSCH-SB-29</td>
<td>0.38</td>
<td>2.75</td>
<td>0.28</td>
</tr>
</tbody>
</table>

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.
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
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

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


