TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation

Simon Duquennoy*†, Atis Elsts‡, Beshr Al Nahas§ and George Oikonomou‡
*Inria Lille - Nord Europe, France †RISE SICS, Sweden
‡University of Bristol, UK §Chalmers University, Sweden

***Abstract*—Synchronized communication has recently emerged as a prime option for low-power critical applications. Solutions such as Glossy or Time Slotted Channel Hopping (TSCH) have demonstrated end-to-end reliability upwards of 99.99%. In this context, the IETF Working Group 6TiSCH is currently standardizing the mechanisms to use TSCH in low-power IPv6 scenarios. This paper identifies a number of challenges when it comes to implementing the 6TiSCH stack. It shows how these challenges can be addressed with practical solutions for locking, queuing, scheduling and other aspects.**

With this implementation as an enabler, we present an experimental validation and comparison with state-of-the-art MAC protocols. We conduct fine-grained energy profiling, showing the impact of link-layer security on packet transmission. We evaluate distributed time synchronization in a 340-node testbed, and demonstrate that tight synchronization (hundreds of microseconds) can be achieved at very low cost (0.3% duty cycle, 0.008% channel utilization). We finally compare TSCH against traditional MAC layers: low-power listening (LPL) and CSMA, in terms of reliability, latency and energy. We show that with proper scheduling, TSCH achieves by far the highest reliability, and outperforms LPL in both energy and latency.

I. Introduction

As research in low-power networking matures, synchronized communication is emerging as a prime option for critical applications. Synchronized network flooding with *e.g.*, Glossy [12], or mesh networks based on IEEE 802.15.4-2015 [14] Time Slotted Channel Hopping (TSCH) [27], have demonstrated end-to-end reliability upwards of 99.99 %, in real deployments.

At the same time, the industry has been pushing for standardized solutions with the IETF's low-power IPv6 stack: 6LoWPAN, RPL, CoAP, etc. This promises (1) interoperability; (2) integration with existing management tools and networks; and (3) industrial-grade security levels. In this context, the IETF Working Group 6TiSCH [23] is currently standardizing the mechanisms to run IPv6 on top of TSCH.

TSCH is essentially a MAC layer that offers a globally-synchronized network of sleepy nodes. Each node's activity is dictated by a time-slotted schedule. In this schedule, communication uses channel hopping. 6TiSCH defines the sublayer for the management of TSCH nodes and schedules.

This paper first identifies five distinct challenges we must address for a flexible and efficient 6TiSCH implementation. We show how to address all five challenges in a real setting, through our implementation for Contiki [6], a leading operating system for the IoT. In particular, we discuss software architecture, locking, queuing and many other aspects. In total, our implementation is already ported to 9 different hardware platforms, and was successfully tested for interoperability.

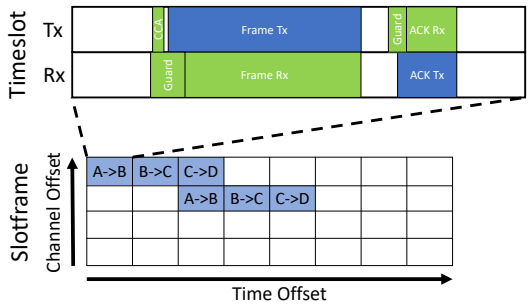


Figure 1. Diagram of a standard TSCH timeslot and example slotframe. A timeslot is typically 10 ms long. Blank time before/between/after frames are for processing: queuing and cryptography. Slotframes can be of any finite length; they repeat over time.

We then run testbed experiments (up to 340 nodes) to answer the following questions:

- *How accurate can synchronization get in large networks, and what is the energy baseline?*
- *How does TSCH compare to the traditional MAC layers CSMA and Low-power listening?*
- *How is energy spent, at both sending and receiving sides, both with and without security?*

Next, §II provides necessary background, and §III introduces the five challenges we focus on. §IV presents our solutions and implementation, followed by an evaluation in §V. We discuss related work in §VI and conclude in §VII.

II. Background

A. IEEE 802.15.4-2015 TSCH

Time Slotted Channel Hopping (TSCH) is a MAC layer specified in IEEE 802.15.4-2015 [14], with a design inherited from WirelessHART and ISA100.11a.

TSCH builds a globally synchronized mesh network. Nodes may join the network after hearing a beacon from another node. Time synchronization trickles from the PAN coordinator down to leaf nodes along a Directed Acyclic Graph (DAG) structure. Nodes update their synchronization relative to their *time source parent* every time they receive a data or ACK frame from it.

Time is cut into timeslots; timeslots are grouped into one or more slotframes, which repeat over time (as illustrated in

Fig. 1). A timeslot, typically 10 ms long, is long enough to accommodate the transmission of a frame and its acknowledgement, including encryption/decryption times. A schedule dictates whether to transmit, receive or sleep within each timeslot. A timeslot in a slotframe is identified by its time offset (when in the slotframe it occurs) and its channel offset (denoting the frequency to communicate on). 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 [25].

How the communication schedule in the TSCH network is built and maintained is out of the scope of the established standards. 6TiSCH, detailed next, proposes the options to schedule either (1) centrally like WirelessHART and ISA100.11a which computes routes and schedules from link statistics and traffic information gathered in the network or (2) in a distributed fashion.

B. IETF 6TiSCH

The IETF Working Group 6TiSCH [23] was chartered in 2013 to enable IPv6 on top of TSCH. It covers architecture, interface, scheduling and security aspects. The Working Group is still active at the time of writing, with one RFC and a number of active Internet-Drafts. 6TiSCH enables node and schedule management via a CoAP interface, building on COMI (CoAP Management Interface). It also defines 6top, a sub-layer that enables neighbor-to-neighbor slot installation/removal. 6top can run one or several *scheduling functions*, which defines rules about when and how to add and remove slots at each node. The architecture documents also define a secure join process.

6TiSCH also defines a so-called *minimal configuration*, which must be supported by all implementations in order to achieve basic interoperability. This configuration includes a simple static schedule and security architecture, based on two keys K1 and K2. K1 is used for authentication-only of beacons, while K2 is used to encrypt and authenticate data and ACK traffic. In addition, the minimal configuration defines how TSCH interacts with upper layers and the RPL routing protocol. In particular, it ensures a consistent mapping between the RPL routing topology and the TSCH time-source graph.

III. Challenges

For an implementation of TSCH and 6TiSCH to be standards-compliant, versatile and efficient, we need to address a number of key challenges (some of which were discussed in [27]). We identify the following five challenges: standards-compliance, timing precision, portability, efficiency and interface with upper layers.

C1: Standards-compliance The first challenge in our stack is to ensure compliance with the standards: TSCH and 6TiSCH. We support only a subset of the standards, but we do so in a fully compliant way. Standards-compliance includes timeslot and slotframe operation as well as security.

C2: Timing Precision The second challenge is to achieve precise timings. Operations within a TSCH timeslot are defined at a micro-second precision. Nodes are required to stick to the timeslot as closely as possible, and to maintain synchronization with their time source(s) as tightly as possible.

C3: Portability The third challenge is to ensure portability across the many platforms supported by the Contiki OS, including both 16 and 32-bit MCUs, and a variety of radio chips. Portability is a pragmatic but important constraint, as it precludes relying on features that only few platforms offer. In our design, we carefully select relevant radio interfaces offered by Contiki, and propose extensions where needed.

C4: Efficiency The next challenge is efficiency, in MCU time and energy. Our aim is to be as energy-efficient as possible, to the extent that does not compromise portability or timing precision.

C5: Interface with Upper Layers Finally, we need means for upper layers to interact with the timing-sensitive TSCH MAC layer. This is non-trivial because the time-sensitivity of TSCH requires it to be interrupt-driven. Our design includes solutions for race-condition free operation of Contiki's IPv6 stack on top of TSCH.

IV. Design

This section discusses how we address the five technical challenges (**C1** to **C5**), so as to enable a flexible and efficient 6TiSCH stack for constrained micro-controllers.

A. Slot Operation

In order to support standard-compliant slots (**C1**) and address the timing precision challenge (**C2**), our design is fully driven by timer interrupts. Nodes wakeup at the beginning of every slot that is active (listen and/or transmit) in their schedule. For transmit slots, they prepare and encrypt the packet to be sent, and go back to sleep until the planned transmission time. They then transmit, go back to sleep until the expected arrival time for the ACK. After receiving the ACK, they look up their schedule for the next active slot, set a timer for the next wakeup and go back to sleep. Receive slots operate conversely, and nodes also sleep between the different events within timeslots.

All operations within a timeslot take place in interrupt handlers, to ensure tight timing. We implement the transmit and receive procedures as protothreads [7]. This result is

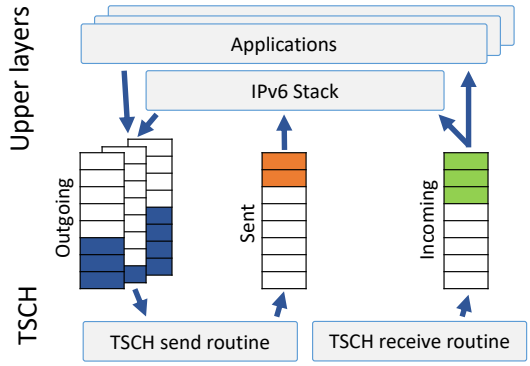


Figure 2. TSCH and the upper layers access different packet queues in a lock-free manner, respectively from and outside of interrupt context. Note that there is one distinct transmit queue per (active) neighbor.

more readable (sequential) code that does not sacrifice the timing requirements (runs on an event kernel).

Note that in our current implementation, nodes stay busy waiting within an interrupt context whenever expecting an incoming frame. Alternatively, they could turn the radio on, go back to sleep and wait for a radio interrupt to process the frame and schedule the next wakeup (*e.g.*, for ACK transmission). The latter approach, albeit more efficient, is made difficult due to limitations of Contiki's current radio driver interface and the different triggers radio chips employ to signal interrupts; *i.e.*, start or end of frame. Enabling such a busy wait-free operation in a way that does not compromise portability (**C3**) is on our roadmap.

B. Locking and Queuing

An important challenge in TSCH is to safely interface the synchronous interrupt-driven engine with the asynchronous system's upper layers (**C5**). To this end, we rely on two tools: ring buffers for outgoing, sent, and incoming packet queues, and a global lock for core TSCH operations.

We use a fixed-size lock-free ring buffer implementation that is safe and lightweight to realize the packet queues between TSCH and the upper layers. We have a distinct transmit queue for each neighbor, filled by upper layers and consumed by TSCH, within interrupts. In addition, there is a queue for packets that need post-transmission processing; *i.e.*, upper layer callbacks. This queue is filled by TSCH inside an interrupt, and consumed by upper layers within a process context. Finally, TSCH has an input queue, where it puts all valid incoming data frames, processed later by an upper layer process. Fig. 2 illustrates which modules read and write to the different queues, in a lock-free manner.

Some other operations, however, need to write to sensitive TSCH state and require a global lock. Examples of such are schedule and neighbor table maintenance where new neighbor queues are created. In our design, all such operations must be performed outside interrupt contexts. Whenever attempting such maintenance while an active TSCH slot is going on, the process will busy wait until the end of the slot, take the lock, proceed, and release the lock. Slots starting with a busy lock will be skipped; the node goes back to sleep until the next scheduled active slot.

C. 5-byte Modular Arithmetic

TSCH relies on the Absolute Slot Number (ASN), which counts slots from the beginning of the network's life. In order to never virtually loop, the ASN is a 5-byte unsigned integer. With 10 ms slots, it would take over 348 years for the ASN wrap. The ASN is used (1) to compute the current channel, given a slot's channel offset; (2) to find the current time offset in any given slotframe; (3) as cryptographic nonce for link-layer security. During slot operation (within an interrupt context), nodes need to compute the modulo of the ASN to some integer value v (*e.g.*, slotframe length, hopping sequence length), in order to obtain the current time offset or current physical channel respectively.

For fast and memory-efficient 5-byte modulo arithmetic (**C4**), our design proposes a data structure dedicated to ASN arithmetic. Integers that will be used as divisor of modulo operations are stored in this structure. In addition to the integer actual value v, the structure stores the result of $2^{32} \bmod v$, denoted r. Whenever $ASN \bmod v$ is needed, it is then simply computed as $(ASN_{lsb} \bmod v + r \times ASN_{msb}) \bmod v$, where ASN_{lsb} are the four least significant bytes of ASN, and ASN_{msb} is the ASN's most significant byte. Doing so, TSCH can operate free of any 64-bit arithmetic, rendering it efficient both on 16-bit and 32-bit platforms.

D. Distributed Time Synchronization

Crystal oscillators are never perfectly accurate, and exhibit a clock drift typically in the range of 10 ppm (parts-per-million). As TSCH requires global network synchronization, it is essential that nodes correct the drift to their time source regularly. At 10 ppm each, two nodes have a relative drift up to 20 ppm, which leads to a desynchronization of 20 μs every second. Assuming guard times of $\pm 1100\,\mu s$ (the default), independent clocks run out of synchronization after 55 s.

In TSCH, nodes re-synchronize their clock with their time source(s) on two distinct occasions (1) upon receiving a frame from a time source or (2) upon receiving an ACK from a time source. In the former case, the node simply compares the reception time against the expected time, and adjusts its clock internally. In the latter case, the time source does the time delta computation upon receiving a unicast, and then includes the delta as part of the ACK (as an Information Element). When receiving the ACK, the node adjusts its internal clock. Note that the clock may be adjusted backwards with no adverse effect on slot operation as the time deltas are bounded by one TSCH guard time.

We build on adaptive time synchronization [22], whereby each node learns the relative drift to its time source for tighter synchronization. The drift is basically learnt from past synchronization. Whenever scheduling a wakeup, the

node will internally compensate for the anticipated drift and set its timer accordingly. The mechanism enables re-synchronization intervals in the order of a few minutes [2].

In addition, we offer dynamic synchronization traffic, both beacons and keep-alives. For TSCH beacons, we map the transmission period to RPL's beaconing interval (Trickle [15]). For keep-alive messages (send on-demand by nodes that need a clock update), nodes use a short period, *e.g.*, 30 s, upon attaching to a new time source. As they learn the relative drift to their time source, they start using a longer period, such as 120 s. These two mechanisms result in reduced synchronization overhead as the topology stabilizes, fulfilling both **C2** (timing) and **C4** (efficiency).

E. Timers and Timestamping

For enabling time synchronization, nodes must be able to produce accurate packet timestamps (**C2**). We offer two solutions, where the timestamping is either done (1) by the radio chip or (2) by the MCU. The former case is the common case, where we rely on the radio to provide accurate reception timestamps. The latter case is useful on platforms such as the jn516x where the MCU offers a higher-frequency oscillator than the radio.

As for the internal clock, our design can support both low- and high-frequency crystal oscillators, each with their own tradeoff between energy and precision. Our port to the Texas Instruments CC2650 System-on-Chip even supports the joint use of two crystals, for low-power and fine-grained synchronization (sub-microsecond accurate assuming frequent re-synchronization [9]).

F. Security

Our implementation performs TSCH and 6TiSCH security as per the standard documents, with distinct security keys for beacons and data traffic (**C1**). For portability, we rely on Contiki's generic AES-CCM interface, which can perform frame encryption and decryption by using the cryptographic hardware acceleration capability available on a number of chips supported by Contiki (**C3**). For timing accuracy, we encrypt the full frame as early as possible during a timeslot, go back to sleep and wakeup a few microseconds prior to actual transmission (**C2**). At the receive end, however, the incoming frame must be decrypted and the outgoing ACK encrypted, right in the critical path. The standard provisions $1000\,\mu s$ for this operation, and this duration proved sufficient for most hardware platforms we experimented with. Platforms without a cryptographic coprocessor have to run in networks where looser timeslot timings are accepted.

G. Interaction with RPL

We offer a configurable interface for the interaction between TSCH and upper layers, RPL in particular (**C5**). The interface is basically a set of callbacks that notify RPL that TSCH has joined or left a network, and that RPL uses to notify TSCH of a parent switch or new beaconing interval. This enables to implement the 6TiSCH architecture, which, for instance, defines how to map the RPL *rank* (distance to the root) and *preferred parent* to the TSCH *join priority* and *time source*. It also makes it possible to flush the RPL state whenever leaving a TSCH network, for a clean-slate re-join.

H. Scheduling

Scheduling is out of the scope of the TSCH standard, and is not yet fully covered by 6TiSCH either. Our implementation provides a software API for scheduling, supporting slotframe and timeslot addition, removal and update. By default, we run the so-called 6TiSCH minimal configuration (**C1**), where all communication takes place on a single timeslot, within a single slotframe, based on contention. This results in a behavior equivalent to slotted ALOHA [20]. Alternatively, we provide Orchestra [8], which is an autonomous scheduler, where nodes compute their schedule locally, based on their routing state. With Orchestra, timeslot coordinates are simply based on a hash of the node's and neighbor's MAC addresses. The standardization of this mechanism in 6TiSCH is currently under discussion.

I. Logging

Logging through UART communication is made difficult by the timing constraints of TSCH (**C2**). Our implementation provides a logging module that enables delayed logging from timing-critical interrupts. The logs are simply written to a queue, and converted to human-readable format and dumped over UART later, from a process outside of interrupt context.

J. Portability

One of our main design goals is portability (**C3**). Our implementation is done as a MAC-layer module of Contiki, *i.e.*, it follows well-defined interfaces to the upper layers (6LoWPAN) and lower layers (radio drivers). In addition to the existing Contiki requirements, a hardware platform must provide the following features in order to run TSCH:

- Radio poll mode: Disables radio interrupts – instead, the MAC layer will poll the radio for new frames;
- Timestamping: Radios must provide access to packet timestamps (required at join time, used my most platforms for subsequent synchronization);
- Raw Tx/Rx modes: To run TSCH, the radio must support disabling automatic channel check assessment (CCA), automatic transmission of ACK frames and address filtering;
- Radio delays: Drivers must provide numbers on the H/W delay involved when turning the radio on or transmitting.

Because of our portability goals, we chose not to exploit hardware-specific features of certain platforms. For instance, the jn516x SoC offers advanced radio management features such as delayed transmissions (the MCU sleeps during the delay), or time-bounded listening (listen until a packet is received or some deadline is reached). Both these features

Platform	MCU	Timer(s)	Radio	Timeslot (ms)	Largest experiment to date
Tmote Sky	msp430 (16 bits)	32.768 kHz	cc2420 (2.4 GHz)	15	98 nodes (Indriya [3])
Zolertia Z1	msp430 (16 bits)	32.768 kHz	cc2420 (2.4 GHz)	15	A few nodes
NXP jn516x	OpenRISC (32 bits)	32.768 kHz or 16 MHz	jn516x (2.4 GHz)	10	25 nodes (private testbed)
cc2538dk	Cortex-M3 (32 bits)	32.768 kHz	cc2538 (2.4 GHz)	10	A few nodes
OpenMote-cc2538	Cortex-M3 (32 bits)	32.768 kHz	cc2538 (2.4 GHz)	10	A few nodes
Zolertia Zoul	Cortex-M3 (32 bits)	32.768 kHz	cc2538 (2.4 GHz)	10	25 nodes (private testbed)
			cc1200 (868 MHz)	32	
cc2650dk	Cortex-M3 (32 bits)	65.536 kHz and 4 MHz	cc2650 (2.4 GHz)	10	A few nodes
SPES-2 [11]	Cortex-M3 (32 bits)	65.536 kHz and 4 MHz	cc2650 (2.4 GHz)	10	14 nodes (private testbed)
IoT-LAB M3 node	Cortex-M3 (32 bits)	32.768 kHz	AVR rf2xx (2.4 GHz)	10	348 nodes (IoT-LAB Grenoble [1])

Table I. OVERVIEW OF THE PLATFORMS CURRENTLY SUPPORTED BY OUR TSCH/6TISCH IMPLEMENTATION.

would make for a more energy-efficient implementation on that specific platform, but would increase complexity in the TSCH codebase and make it less portable. Another optimization we choose not to implement is to overlap packet loading with transmission and packet reading with reception. It is possible as well to inline packet encryption/decryption with Tx/Rx on many platforms by utilizing special features on their cryptographic coprocessor, instead of encrypting first and transmitting later. The effect of these is shorter slot times, but it would make the code less portable.

K. Interoperability Tests and Supported Platforms

Contiki versions of TSCH were demonstrated to be interoperable (**C1**) in the ETSI plug-tests in Prague (2015) and in Paris (2016). In these events, it was tested against a number of independent implementations, including the *de facto* reference OpenWSN [26].

A year after its release, our implementation has already been ported to 9 hardware platforms (**C3**), involving 6 different radio chips. Tab. I provides a summary of the currently supported platforms and their peculiarities. The current platforms include 16- and 32-bit architectures, with low- and high-frequency crystal oscillators, and over both 2.4 GHz and 868 MHz. In the 2.4 GHz case, the slot length varies between 15 ms (legacy platforms, with slower interfaces) and 10 ms (the default in IEEE 802.15.4-2015 and IETF 6TiSCH). In the 868 MHz case, where the cc1200 radio hardware runs in 802.15.4g mode at 50 kbps, the slot is made as large as 32 ms assuming a maximum frame length of 127-byte frames (note that the standard allows up to 2047 bytes).

V. PERFORMANCE EVALUATION

This section analyzes the performance of TSCH and compares it to state-of-the-art MAC layers.

A. Synchronization

We evaluate our system's ability to keep global synchronization in a large-scale network. We use the FIT IoT-LAB testbed in Grenoble [1] with 340 M3 nodes at the time of the experiments. The nodes run raw TSCH, without 6TiSCH. Nodes basically listen for a beacon and attach to the first node they hear (average hop count: 5.2). Please note that on a few occasions (we recorded eight in the course of this experiment), nodes attach to a time source they have a very weak link to, to only later lose synchronization and rejoin. In real networks, with 6TiSCH running on top, TSCH would use the links selected by RPL at the routing layer, with guaranteed connectivity.

The setup is as follows: nodes transmit a beacon every 20 s. If a node does not hear from its parent for 30 s, it initiates a keep-alive. After learning the relative drift to its parent, the keep-alive timeout extends to 120 s. Doing so, we base synchronization primarily on beacons, with keep-alive as a fallback. Such a setup is particularly suited to dense networks, where a single beacon is enough to re-synchronize many nodes.

We use Orchestra [8] for scheduling, with two slotframes. The first slotframe is for beacons, with length 397 and an Orchestra sender-based slot. The second slotframe is for keep-alives, with length 101, and an Orchestra receiver-based slot. Fig. 3 shows the results from a 60 min experiment.

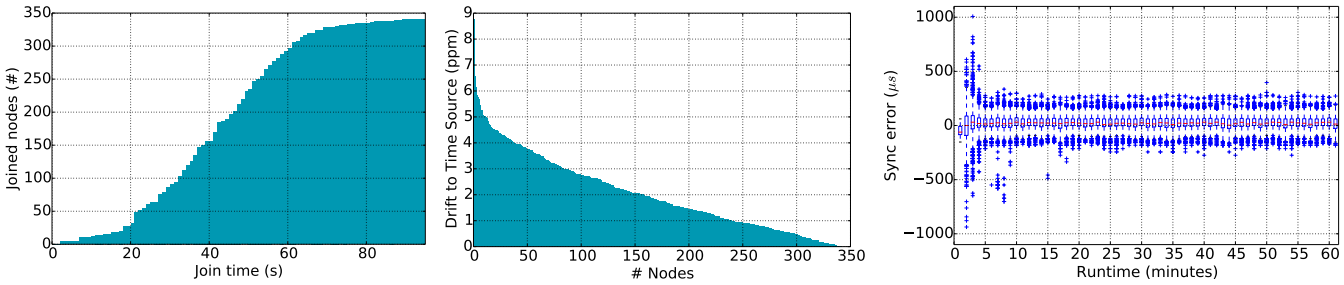
It takes less than 90 s for all nodes to join, as shown in Fig. 3a. Fig. 3b shows the nodes' relative drift to their time source, as computed by our internal drift compensation mechanism. The largest recorded drift was 9 ppm.

Fig. 3c shows the time corrections applied by the nodes during the run. Here, nodes use the TSCH standard guard time, of $\pm 1100\,\mu s$. After a few minutes, nodes adjust to their time source's drift, and get more tightly synchronized. Quite interestingly, over 97 % of the packets recorded were received with an error below $160\,\mu s$. This level of synchronization is sufficient to enable capture effect; a phenomenon that TSCH benefits from in contention-based slots.

Note that after convergence (the few first minutes), all nodes reach a duty cycle of about 0.3 %. TSCH results in particularly low channel utilization, here only 0.008 % at each node, spread over 16 channels. Most of the energy is spent listening, and more fine-tuning of guard times could further reduce the baseline cost.

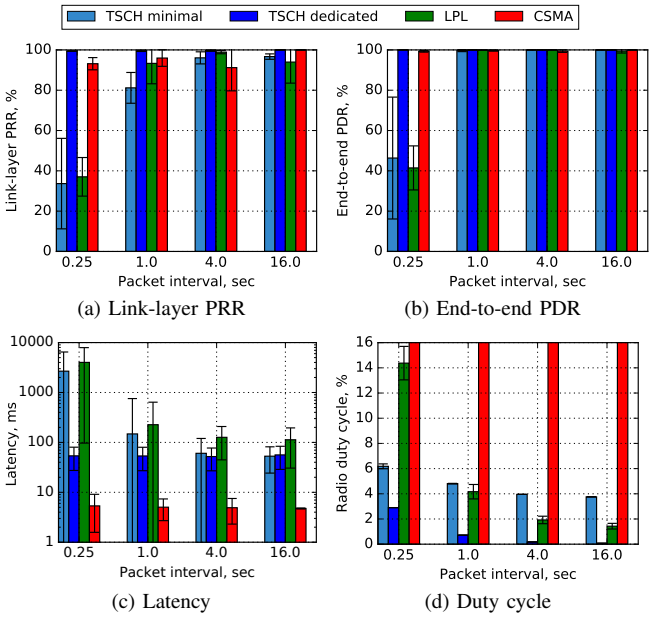
B. Comparison against other MAC Protocols

We compare the performance of TSCH with traditional IoT MAC layers – low-power listening (LPL) and always-on CSMA – in terms of reliability, latency and energy. To this end, we set up a small controllable testbed with five CC2650 SoC nodes [11]. The nodes are connected in a star network. As the LPL protocol we use ContikiMAC [5]. For always-on CSMA, we enable the `nullrdc` layer of Contiki.



(a) Join time (b) Drift (c) Sync. Error

Figure 3. **Time synchronization experiment, 340 nodes.** (a) Shows the number of joined nodes at any given time after bootstrap. (b) Shows the distribution of the nodes' drift to their time source. (c) Shows how synchronization errors evolve with time. The duty cycle converges to 0.3 %.



(a) Link-layer PRR (b) End-to-end PDR

(c) Latency (d) Duty cycle

Figure 4. **Comparison of TSCH with Low-Power Listening (LPL) and CSMA.** For TSCH, the 6TiSCH minimal schedule and a collision-free schedule are compared. Note the log y-scale used for latency. Note that the duty radio cycle for CSMA is 100 %, not fully shown in the graph.

We compare two schedules of TSCH: the 6TiSCH minimal one, with one single shared Tx/Rx slot, and a custom dedicated one, with one Tx slot assigned to each leaf node. The slotframes have a length of 9 slots. The back-off exponent for shared slots is set to 4, the TSCH guard time to 800 μsec. For LPL, the channel sampling frequency is set to 8 Hz; phase lock is enabled, with the max Tx time towards known neighbors set to 30.5 ms. The number of MAC-layer retries is 8 for all protocols. Each of the leaf nodes generates one IPv6 UDP packet at a random point during a period ranging from 0.25 s to 16 s to emulate high and low rate traffic. LPL and CSMA run on channel 25, while TSCH runs on 25 and 26.

Fig. 4 summarizes the results, discussed next.

Link-layer Packet Reception Rate (PRR) The PRR is the link success rate, before MAC retries. In this topology the vast majority of lost packets are due to contention. For TSCH, this leads to an important difference between the minimal (contention-based) and dedicated schedules, especially with high-rate packet generation. For the 0.25 second packet generation interval, TSCH *minimal* shows 33.66 % PRR while TSCH *dedicated* shows 99.46 % PRR.

LPL (37.2 % PRR in the 0.25 s experiment) outperforms TSCH minimal as in LPL, transmissions are more randomly distributed, while TSCH minimal offers only one slot for all nodes to contend. CSMA (93.15 % PRR in the 0.25 s experiment) outperforms LPL, as packet transmissions without LPL are shorter and less likely to collide.

End-to-end Packet Delivery Rate The PDR is the end-to-end success rate, after MAC retries. The results show that PDR in the 0.25 s experiment for TSCH *minimal* and LPL are below 50 %. Both the number for active slots per second and the maximal number of retransmissions are too small to accommodate for the extensive link-layer packet loss. For all other experiments, all protocols show >99 % PDR. However, only TSCH *dedicated* achieves consistent 100 % packet delivery rate in all scenarios.

Latency For the high-rate experiments, TSCH *dedicated* shows one to two orders of magnitude lower latency than TSCH *minimal* and LPL. Here, the main factor affecting the latency is the collision rate and the resulting MAC-layer back-off. For the low-rate experiments, the latency is similar between the three protocols. It is mostly determined by the slotframe size (for TSCH) and the channel check interval (for LPL). CSMA shows by far the best latency of all – it sends packets immediately as they are generated, leading to nearly no collision nor back-off.

Duty Cycle The radio duty cycle of the sink node is excluded from this statistic, as sink nodes usually are mains powered. LPL is less efficient than TSCH *minimal* with these specific configuration settings when the datarate is high (0.25 s period). This is explained by the fact that most of the energy in low-rate TSCH networks is wasted in idle listening [16]. In contrast, the dedicated schedule has no idle listening and is by far the most energy-efficient (down to 0.073 % duty cycle in the 16 s experiment).

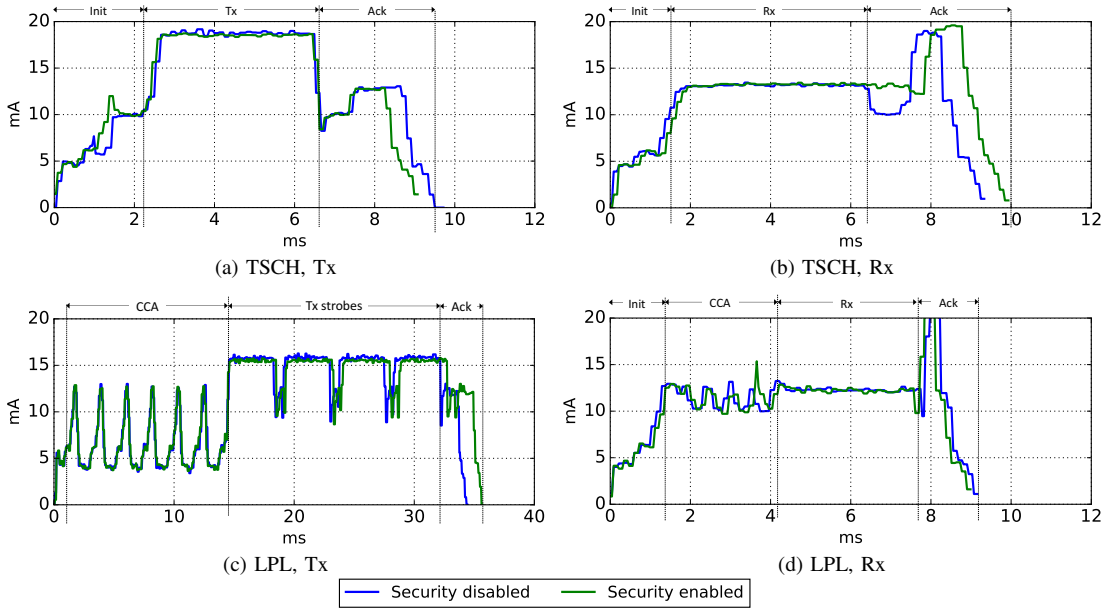


Figure 5. **Energy usage profiles for Tx and Rx of a single packet.** 98 byte packet size, excluding MAC headers. Note the x-axis scale for "LPL, Tx".

C. Energy consumption

We compare in Fig. 5 the energy consumption profile for transmission and reception of a single packet in TSCH and in LPL (ContikiMAC) and with security on/off.

In these experiments we use the maximal packet size that was supported with all four configuration settings, 98 bytes, excluding MAC header. For experiments with security enabled, the IEEE 802.15.4 packet security level is set to 0x5: data encrypted and authenticated with 32-bit MIC block [14]. Encryption and authentication is done using hardware-accelerated AES-128 algorithm.

While TSCH and LPL have roughly similar packet Rx profiles, Tx is much more efficient for TSCH because of its time-synchronized nature (notice the different x-axis scales – the result is a factor 3 in cumulative energy). LPL repeats the packet transmissions several times to wake the receiver up, while TSCH transmits it only once. Another major difference is that TSCH does not perform CCA (unless configured to do so), while LPL shows multiple CCA peaks before the transmissions start (needed by to resolve contention in a unslotted context).

All recordings start with approximately 1 ms long period of approximately 5 mA current level. This is the time required for the CC2650 radio to become fully operational. LPL Tx activity proceeds by doing the CCA checks. Subsequently, for Tx activities, the packet is loaded in the FIFO ($<$1 ms), and then transmitted (approximately 4 ms). In Rx mode, after receiving the packet, it is decrypted if required and an ACK is created and transmitted.

Enabling security increases packet size just by a few bytes (depending on the protocol), hence has no large impact on Tx/Rx duration. However, for TSCH Rx the packet must be decrypted before an ACK is generated. This is visible in the graphs as increased energy consumption between packet and ACK transmission. The additional time required to encrypt or decrypt the packet on CC2650 is 340 μs ($\pm$30 μs) according to our measurements. Also note that with LPL, reception starts earlier with security than without: this is only incidental, caused by the randomness in the time between the node's wakeup and the start of actual packet reception.

Note that on CC2650 the radio is not turned off between the packet and the ACK; this behavior is different on other Contiki platforms, for example, CC2240 and CC2538.

VI. Related Work

This section reviews work related to TSCH and 6TiSCH.

Synchronization A key to achieving tight and low-power time synchronization in TSCH is for nodes to learn the relative drift to their time source(s). Stanislowski [22] showed it was possible to consume one order of magnitude less energy by adopting adaptive drift compensation. The concept was later applied to multi-hop [2], with an additional mechanism to cascade time synchronization from coordinator down to leaves. As a result, nodes increase their synchronization period, reaching as high as a 3 minutes.

Our implementation builds on the same principles. When running at a high synchronization frequency (period of 4 s), we have demonstrated accuracy below 1μs [9]. Further, we also evaluated in details the impact of guard times on energy consumption [16].

Channel Hopping Watteyne *et al.* [25] proposed an in-depth evaluation of channel hopping in the TSCH context. Trace-driven simulation results show how channel hopping mitigates multi-path fading and interference. By default,

TSCH uses all available 16 channels of the IEEE 802.15.4 PHY layer. Hopping blindly over all channels enables reliable operation even when a subset of the channels has poor quality. Adaptive channel blacklisting was nonetheless proposed [10] that pushes reliability and throughput further.

By standard, a coordinator can advertise its own hopping sequence, that joining nodes will learn when receiving beacons. Our implementation supports this feature, enabling an administrator to perform (re-)deployment-time blacklisting.

Scheduling Scheduling in TSCH and 6TiSCH has received significant attention from both a theoretical and practical standpoints. Early solutions where focusing on static networks with pre-defined traffic patterns [4], [18]. With a centralized scheduler, these solutions demonstrated reliability over 99.999%. Distributed solutions were later proposed, *e.g.*, by Tinka et al. [24], where a rendez-vous slot is used for discovery and slot installation. Another interesting approach is to perform multi-hop reservation and label switching [17]. On-the-fly resource reservation in a 6TiSCH context was demonstrated by Palattella *et al.* [19], demonstrating reliable operation in a 50 node network (simulation).

Our scheduler Orchestra [8] is complementary, as it provides a basic schedule that nodes can maintain autonomously, simply based on the local knowledge of their neighborhood. Orchestra can potentially be used jointly with other schedulers.

Other MAC-layer Aspects We presented fine-grained energy measurements of the energy overhead of link-layer security in TSCH. On a very related topic, Sciancalepore *et al.* [21] dissected the impact of various security levels on timeslot length on a wide range of platforms. Their measurements show that the minimal timeslot length varies greatly among platforms, ranging from 9 to 88 ms.

Finally, Deguglielmo *et al.* [13] proposed an analytical model of the back-off mechanism in TSCH, taking into account the occurrence of capture effect in shared slots.

VII. Conclusion

We discussed and addressed the main challenges that lie in providing a flexible and efficient TSCH and 6TiSCH implementation. Through our Contiki implementation and a series of testbed experiments, we find that: (1) Synchronization in large (340 nodes) networks is possible at high accuracy (97% of the time under 160 μs) for a low cost (duty cycle of 0.3%); (2) TSCH, when running dedicated slots, outperforms LPL in all key metrics: reliability, latency, duty cycle; (3) At a micro-level, a TSCH and LPL spend about the same amount of energy for receptions, but TSCH has an edge (factor 3) on transmissions. Link-layer security comes at a low overhead. We believe our implementation to be an enabler for more further research, in particular as it brings the 6TiSCH stack to the Contiki ecosystem, where a variety of upper-layer protocols and applications already exist.

References

[1] C. Adjih et al. FIT IoT-LAB: A large scale open experimental IoT testbed. In *WF-IoT*, 2015.

[2] T. Chang et al. Adaptive synchronization in multi-hop tsch networks. *Comput. Netw.*, 2015.

[3] M. Doddavenkatappa et al. Indriya: A low-cost, 3D wireless sensor network testbed. In *ICST TridentCom*, 2011.

[4] L. Doherty et al. Channel-Specific Wireless Sensor Network Path Data. In *ICCCN*, 2007.

[5] A. Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical Report T2011:13, SICS, 2011.

[6] A. Dunkels et al. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. 2004.

[7] A. Dunkels et al. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. 2006.

[8] S. Duquennoy et al. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. 2015.

[9] A. Elsts et al. Microsecond-Accuracy Time Synchronization Using the IEEE 802.15.4 TSCH Protocol. In *SenseApp*, 2016.

[10] A. Elsts et al. Adaptive Channel Selection in IEEE 802.15.4 TSCH Networks. In *Global Internet of Things Summit*, 2017.

[11] X. Fafoutis et al. Demo: SPES-2–A Sensing Platform for Maintenance-Free Residential Monitoring. In *EWSN*, 2017.

[12] F. Ferrari et al. Efficient Network Flooding and Time Synchronization with Glossy. 2011.

[13] D. D. Guglielmo et al. Analysis and experimental evaluation of IEEE 802.15.4e TSCH CSMA-CA Algorithm. In *IEEE TVT*, 2016.

[14] IEEE. Standard for Low-Rate Wireless Networks. Std 802.15.4-2015.

[15] P. Levis et al. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *NSDI*, 2004.

[16] A. Mavromatis et al. Impact of Guard Time Length on IEEE 802.15.4e TSCH Energy Consumption. In *SECON*, 2016.

[17] A. Morell et al. Label switching over IEEE802.15.4e networks. *Trans. on Emerging Telecom. Technologies*, 2013.

[18] M. R. Palattella et al. Traffic aware scheduling algorithm for reliable low-power multi-hop ieee 802.15.4e networks. In *PIMRC*, 2012.

[19] M. R. Palattella et al. On-the-fly bandwidth reservation for 6tisch wireless industrial networks. *IEEE Sensors*, 2016.

[20] L. G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 1975.

[21] S. Sciancalepore et al. Link-layer security in tsch networks: effect on slot duration. *Trans. on Emerging Telecom. Technologies*, 2016.

[22] D. Stanislowski et al. Adaptive synchronization in IEEE802.15.4e networks. *IEEE Trans. on Industrial Informatics*, 2014.

[23] X. Thubert (Ed.) et al. An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4 - draft-ietf-6tisch-architecture-10, June 2016. IETF Draft.

[24] A. Tinka et al. A Decentralized Scheduling Algorithm for Time Synchronized Channel Hopping. *EAI Trans. on Mobile Comm. and Applications*, 2011.

[25] T. Watteyne et al. Mitigating Multipath Fading through Channel Hopping in Wireless Sensor Networks. May 2010.

[26] T. Watteyne et al. OpenWSN: a standards-based low-power wireless development environment. *Trans. on Emerging Telecom. Technologies*, 2012.

[27] T. Watteyne et al. Industrial Wireless IP-based Cyber Physical Systems. In *PIEEE, special issue on CPS*, 2016.