LDSF: Low-latency Distributed Scheduling Function for Industrial Internet of Things

Vasileios Kotsiou, Georgios Papadopoulos, Periklis Chatzimisios, Fabrice Theoleyre

LDSF: Low-latency Distributed Scheduling Function for Industrial Internet of Things

Vasileios Kotsiou, Georgios Z. Papadopoulos, *Member, IEEE*, Periklis Chatzimisios, *Senior Member, IEEE*, Fabrice Theoleyre, *Senior Member, IEEE*

***Abstract*—The Industrial Internet of Things (IIoT) is expected to be a key enabler for the Industry 4.0. However, networked control automation often requires high reliability and a bounded latency to react properly. Thus, modern wireless protocols for industrial networks, such as IEEE 802.15.4-2015 Time Slotted Channel Hopping (TSCH), rely on a strict schedule of the transmissions to avoid collisions and to make the end-to-end traffic deterministic. Unfortunately, guaranteeing a bounded end-to-end latency is particularly challenging since transmissions have to be temporally chained. Even worse, potential degradation of the link quality may result in reconstructing the whole TSCH schedule along the path. In this article, we propose the Low-latency Distributed Scheduling Function (LDSF) that relies on the organization of the slotframe in smaller parts, called blocks. Each transmitter selects the right set of blocks, depending on its hop distance from the border router, so that retransmission opportunities are automatically scheduled. To save energy, a node can still turn off its radio as soon as its packet is correctly acknowledged. Our mathematical analysis as well as our simulation evaluation show the efficiency of the proposed LDSF algorithm compared to three state-of-the-art scheduling functions, the Minimal Scheduling Function (MSF), Low Latency Scheduling Function (LLSF) and Stratum.**

***Index Terms*—Industrial Internet of Things; Distributed Scheduling; End-To-End Delay; High Reliability**

I. Introduction

The Industrial Internet of Things (IIoT) consists of a large collection of wireless sensors and actuators for various industrial applications. Typically, networked control systems comprise of sensors, actuators and a controller [1]. The sensors send their measurements regularly to a controller. According to this feedback, the controller may trigger a reaction by activating some actuators. This control loop has to react in real-time.

Industry 4.0 is expected to rely heavily on the IIoT to make the manufacturing process reconfigurable [2]. For instance, industrial robots tend to exploit wireless communications since cables are prone to breakage after a few thousands of flexions [3]. The network topology can also be changed easily to reconfigure the production lines [2]. Most of the industrial applications require a highly reliable communication network, with a bounded end-to-end latency to react appropriately.

This work was supported by the French National Research Agency (ANR) project Nano-Net under contract ANR-18-CE25-0003.

V. Kotsiou and F. Theoleyre are with the ICube Laboratory, CNRS / University of Strasbourg, 67412 Illkirch, France, (e-mail: lastname@unistra.fr).

G. Z. Papadopoulos is with the IRISA, IMT Atlantique, UBL, 35510 Cesson-Sévigné, France, (e-mail: georgios.papadopoulos@imt-atlantique.fr).

P. Chatzimisios is with Department of Science and Technology, International Hellenic University (IHU), 57001, Thessaloniki, Greece (e-mail: pchatzimisios@ihu.gr).

Typically, big data requires to collect a large amount of measurements to create an intelligent manufacturing system [4]. For instance, predictive maintenance helps to replace hardware *before* a failure/breakdown targeting to decrease the shutdown time [5]. We here focus on the end-devices of the Industrial Internet of Things, that use wireless transmissions to send/receive their packets. To reduce the deployment cost, some of the devices may forward the packets from others in order to reach a gateway with a wired connection. The devices are battery-operated and need to turn off regularly their radio chipset to save energy. Thus, a transmitter must know when the receiver will schedule its next wake-up. In the rest of this article, we will use the term *node* to designate all these devices, that generate and forward data packets.

To cope with these constraints, deterministic Medium Access Control (MAC) protocols have been proposed in the literature. These proposals rely on a strict schedule of the transmissions: only non-interfering transmissions are triggered at the same time to alleviate collisions. Moreover, slow channel hopping strategies help to increase network capacity and to combat external interference.

IEEE 802.15.4-TSCH typically adopts these strategies [6]. A scheduling matrix is composed of *cells*, defined by a timeslot (when to transmit the packet) and channel offset (which physical channel to use). Thus, the scheduling algorithm has to allocate a certain number of cells to each transmitter. Since a cell is designed to contain at most one packet transmission and its acknowledgment, the number of cells represents directly a number of transmission opportunities. IEEE 802.15.4-TSCH supports both centralized and distributed scheduling algorithms [7], denoted as *Scheduling Functions*.

Since we rely on a wireless infrastructure, the receiver may not always be able to decode a packet due to potential external interference or multi-path fading. Thus, the transmitter has to retransmit the corresponding packet through another cell. Consequently, the scheduling function needs to allocate several cells for each hop of the path, inversely proportional to its link reliability metric [8].

Guaranteeing a maximum end-to-end delay requires to chain the timeslots along the path: a node has to receive the packet before being able to forward it. However, this sequential scheduling has a negative impact on the end-to-end delay: a packet can be forwarded only after all the retransmission cells. In addition to this, a local change of the link quality may imply a whole schedule reconfiguration along the path.

In this article, we propose a Low-latency Distributed Scheduling Function (LDSF) tailored to minimize the latency, while providing high reliability. We address the problem of designing a scheduling algorithm that provides low end-to-end

delay even in the presence of retransmissions. LDSF is able to schedule a large number of retransmissions to handle the worst-case situation and to minimize the buffering delay where the over-provisioned cells are chained. The contributions of this article are as follows:

1) We design a novel organization of the slotframes, divided into repetitive short blocks. Chaining the blocks reduces drastically the overall end-to-end delay. Moreover, a transmission opportunity is automatically reserved in consecutive blocks to deal with retransmissions;
2) We define the concepts of primary and ghost cells to save energy. While the primary cell corresponds to the earliest expected reception time, ghost cells are automatically reserved to deal with retransmissions, while limiting the impact on the delay;
3) We provide a mathematical analysis of the average end-to-end latency of the state-of-the-art scheduling algorithms defined to minimize latency, including our LDSF algorithm;
4) Simulations help us to investigate more complex scenarios and highlight the practical interest of our proposed scheduling function.

The rest of the article is organized as follows. Section II provides the technical background and overviews the related work on slow channel hopping and specifically scheduling. Section III details the impact of the reliability on the delay when scheduling the transmissions. Section IV describes the proposed Low-latency Distributed Scheduling Function to organize properly the transmissions targeting to reduce the end-to-end delay. Section V provides a mathematical analysis of the delay provided by different state-of-the-art solutions, while Section VI presents a performance evaluation based on simulations to assess the performance. Finally, Section VII concludes this article and provides future perspectives.

II. Technical Background and Related Work

In this section, we will detail the default operations of the IEEE 802.15.4-TSCH and 6TiSCH standards, and present the existing scheduling algorithms that focus on the end-to-end delay minimization.

A. Industrial Stack

1) IEEE 802.15.4-TSCH: Relies on channel hopping to combat external interference and signal fading [6]. The protocol relies on a classical Automatic Repeat reQuest (ARQ) approach, where a packet is retransmitted if no acknowledgement is received.

The *slotframe* contains a fixed number of timeslots, during which a frame and its acknowledgment are transmitted. Each timeslot is labelled with an Absolute Sequence Number (ASN) which serves as a global clock, counting the number of timeslots since the network bootstrapped. Based on a schedule, a node can decide its role at the beginning of each timeslot. Typically, a node can transmit if the cell is in *Transmitting mode* in the schedule, and stays awake to possibly receive a packet if the cell is in *Receiving mode*. If the cell is allocated to the node, the physical frequency to use for transmission/reception is derived from the ASN of the timeslot and the channel offset, following a pseudo-random approach.

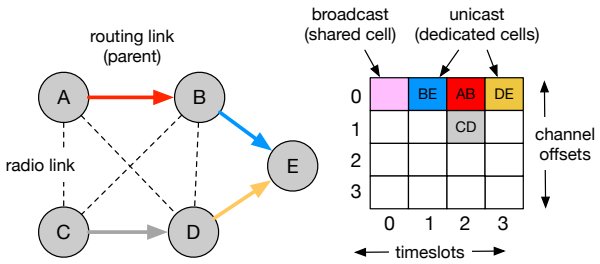


Fig. 1: TSCH schedule for a 5 nodes topology.

The standard supports two medium access techniques:

- **Shared cells** are contention-based cells that can be used by multiple possibly interfering transmitters. A node dequeues a packet and transmits it immediately in the next shared cell. If an acknowledgement was required and was not received, the transmitter assumes a collision occurred and selects a random backoff, skipping the corresponding number of shared cells;
- **Dedicated cells** are contention-free cells and are allocated carefully to avoid collisions. Thus, a transmitter does not implement any random access during these cells. However, a packet can be retransmitted if no acknowledgement is received, due to a bad radio link quality or external interference.

Let us consider the topology illustrated in Figure 1. One shared cell is placed at the beginning of the slotframe for best-effort control traffic and broadcast packets. Then, each node has a dedicated cell to send packets to its neighbors.

2) 6TiSCH: The IETF Working Group has defined a stack of protocols to operate IPv6 on top of the IEEE 802.15.4-TSCH standard. In particular, the standard makes a distinction between how to negotiate the cells, with the 6P protocol [9], and the *Scheduling Function* (SF) that represents the scheduling algorithm. The Scheduling Function defines a method to compute the number of cells, and which ones to pick in the scheduling matrix for each link. Then, a 6P transaction is engaged: a two-way handshake takes place over the shared cells. More specifically, the sender transmits a list of available cells to the receiver with a 6P request in unicast. Consequently, the receiver verifies if all (or part of) these cells are available in its schedule and sends a 6P reply accordingly.

B. Scheduling Approaches

1) Centralized scheduling algorithms: Determine the cells to be allocated to each link in order to respect end-to-end delay and reliability constraints. TASA [10] applies graph theoretical tools to derive a compact schedule, and thus to increase the network capacity. Additional cells can be reserved in the schedule to deal with packet losses due to unreliable links [11]. SchedEx reinforces the schedule by allocating more transmission opportunities to the different flows while still respecting the end-to-end delay constraints [12]. However,

centralized algorithms need a precise view of the network conditions, and generate a large overhead when the schedule has to be updated.

2) Distributed scheduling algorithms: These algorithms are rather traffic adaptive. Typically, a hysteresis function decides how many cells to reserve, (de)allocating some of them to maintain the number of cells between two bounds [13]. Colliding cells are detected autonomously and moved in the slotframe. To deal with unreliable links, additional cells have to be provisioned for retransmissions: the number of cells should be at least equal to the inverse of the packet delivery ratio of the link, multiplied by the number of packets in the queue [8]. PID based scheduling uses the well-known proportional, integral, and derivative algorithm to decide how many cells to reserve when a burst arrives [14]. Inversely, cells are maintained in the schedule to possibly retransmit the packets if required in the future. While these solutions decide how many cells to use, the scheduling algorithm has also to select the right ones to reduce the end-to-end delay.

Minimizing the end-to-end delay is also of primary importance since many industrial applications rely on deadline constraints. Stratum scheduling [15] divides the network into stratums, regrouping the nodes by their hop distance from the border router. By picking the cells in the corresponding stratum, the system guarantees that the packet is delivered before the end of the slotframe, even if the packet is retransmitted. Hosni *et al.* [16] derive the right size of each stratum to minimize the collision probability. However, the packet is guaranteed to be delivered at the end of the slotframe, which may correspond to a very long delay. Furthermore, stratum scheduling does not allow frequency re-use.

ReSF [17] targets recurrent traffic with strict latency requirements. Some cells are pre-reserved in the schedule for these sporadic flows, allocating several cells per hop to cope with retransmissions. The cells are chained along the path to minimize the end-to-end delay. The Low Latency Scheduling Function (LLSF) [18] attempts to move the transmitting and receiving cells closer in the schedule to reduce the buffering delay. However, a change in the link quality may result in changing the whole schedule in the path.

Anycast helps to allocate the same cell to several receivers: a transmission is successful if *any* of the receivers is able to decode it. If we select the right number of parents, we can limit the impact on the energy consumption [19]. Sending several copies of the same packet through different paths may help to improve the reliability while reducing the jitter [20]. These techniques do not focus on the end-to-end delay optimization.

Handling unreliable links is of prime importance for low-power and lossy networks. We need to propose a scheduling algorithm able to provide a low delay, even if a packet has to be retransmitted along the path to the border router. We will detail hereafter the challenges to tackle to reach this objective.

III. Problem Formulation and Objectives

Centralized scheduling algorithms require the whole radio and interference topology knowledge, while they are not adaptive to changes. Thus, in this article, we rather propose a distributed Scheduling Function to optimize both the following performance metrics:

End-to-end delay: The packet has to be delivered to the border router before a certain deadline constraint;

Reliability: Even if links are unreliable, the system has to schedule retransmissions to guarantee a minimum end-to-end Packet Delivery Ratio (PDR).

TABLE I: Notation.

$P_{ll}(A,B)$	Probability that A receives the acknowledgement from B (packet delivery ratio at the link layer)
$P_{net}(A,B,k)$	Probability that A receives at least one ack from B after k retransmissions (packet delivery ratio at the network layer)
$MaxRetries$	Maximum number of retransmissions
$nbretx$	Current number of retransmissions for a given packet
$Cells(N_i, N_{i+1})$	number of cells from N_i to N_{i+1}
$\mathcal{SF}length$	Slotframe length (in timeslots)
$\mathcal{B}Length$	Block length (in timeslots)
$p = \{A..B\}$	path from A to B
hop_{delay}	hop delay, considering the link-layer retransmissions
$E2E_{delay}$	end-to-end delay
$Hops$	the hop distance between the forwarding node and the source of the packet
B_{id}	block id (number of blocks from the beginning of the slotframe)
$RxSlotId$	receiving timeslot (in the 6P request if received by the radio chipset, or given by the application layer if the packet is generated locally)
$NbGhostCells$	the number of ghost cells to add in the schedule for this flow
$\mathcal{NL}$	Network lifetime
Lf_i	Node's lifetime for the node i

Guaranteeing end-to-end delay is actually very challenging, since we have to also consider queuing delays. Let us consider Figure 1. Typically the links CD and DE are scheduled in two consecutive cells, and the delay is minimal. On the contrary, the links BE and AB are scheduled inversely and the packet has to be enqueued in B until the next slotframe. The slotframe may be very long since we can handle very infrequent transmissions. The slotframe duration should typically be equal to the inter-packet time. With heterogeneous periods, the slotframe length should be equal to the least common multiplier of all the packet's periods.

A. Providing High-Reliability

Unreliable links require retransmissions. However, most of the algorithms provision cells for the *average* case. Typically, most scheduling algorithms allocate 2 cells if the PDR is equal to 50%. If a packet needs more retransmissions, they will be scheduled in the next slotframe, which may be very long.

Let us assume that a link provides a PDR of $P_{ll}(link)$, i.e., ratio between the number of packets acknowledged by the receiver with respect to the total amount of sent packets. We assume here that k different cells are allocated, and thus, the transmitter can transmit at most k times this frame. The packet will be delivered from the transmitter A and correctly acknowledged by the receiver B within the current slotframe with the following probability:

$$P_{net}(A,B,k) = \sum_{i \in [1,k]} (1 - P_{ll}(A,B))^{i-1} * P_{ll}(A,B) \quad (1)$$

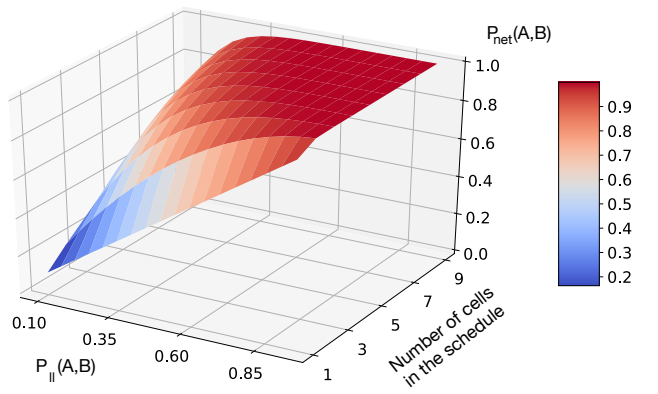


Fig. 2: Probability to receive correctly the packet ($P_{net}()$) depending on the link reliability of the link ($P_{ll}()$) and the number of cells allocated in the schedule.

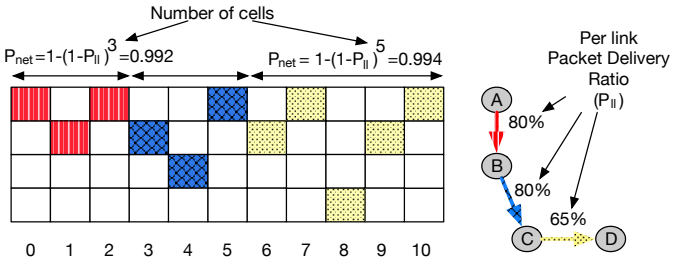


Fig. 3: Scheduling consecutive ranges of cells to limit the end-to-end delay.

We assume here that all cells provide the same Packet Error Rate (PER) for a given link. Indeed, the frequency hopping scheme helps to mitigate external interference, so that the PER is the same for *any* cell [13].

Figure 2 illustrates a numerical analysis of the network reliability ($P_{net}()$) achieved, depending on the link quality of the link ($P_{ll}(A,B)$) and the number of cells provisioned in the schedule for the (re)transmissions. Increasing the number of cells allows the transmitter to retransmit the packet. However, we need a very large number of cells to achieve a very high reliability. For a Packet Delivery Ratio of the link ($P_{ll}()$) of 50%, we need 7 cells to achieve a 99% delivery to the next hop, after the retransmissions.

B. Delay Constraint with Dynamic Scheduling

Some scheduling algorithms reserve a *range* of consecutive cells for retransmissions to optimize the end-to-end delay [21]. The number of cells in this range has to be sufficient to handle the worst case situation, with possibly a very large number of cells (Eq. 1). Thus, the delay is proportional to the worst case, since it corresponds to the sum of the ranges along the path. Furthermore, negotiating cells is expensive since 6P control packets use shared cells, which are prone to collisions and, thus, increasing the convergence delay [8].

Even worse, such approach is inaccurate for dynamic network conditions. Indeed, a node may detect a change in the PDR, and has in that case to increase the number of cells with its parent. Unfortunately, the cells are chained along the path. Thus, inserting a novel cell requires to re-schedule all the cells until the border router. This renegotiation is time-consuming, and implies packet drops before the re-convergence.

Let us now consider the schedule illustrated in Figure 3. The first two hops provide a reliability of 80% and 3 cells have to be provisioned to provide a per link reliability of 99%. However, inserting a novel cell for the link (AB) is expensive: we have to move all the subsequent cells. Otherwise, we have to place the retransmission cell after the 10^{th} cell, requiring to reserve novel cells for the rest of the path.

Stratum [15] tackles this reconfiguration problem by dividing the whole slotframe into stratums (one stratum per hop). However, this organization reduces the network capacity, and the end-to-end delay can only be the slotframe length.

C. Objectives of LDSF

Our proposed Low Latency Distributed Scheduling Function (LDSF) relies on the following features:

- Slotframe organization (in blocks): We reduce the end-to-end delay by chaining appropriately the blocks. All nodes that are at an even number of hops (respectively odd) from the border router are scheduled during the even blocks (respectively odd). This way, the packet progresses on average by one hop at the end of each block.
- Cell allocation in sequence: A node identifies automatically the expected earliest arrival of a packet to schedule cells to forward the packet. Then, the node reserves a cell in the next block, to minimize the buffering delay. Additionally, it also reserves the same cell every two blocks to cope with retransmissions. This way, a node can proceed to a large number of retransmissions, to handle the worst case.
- Energy Saving: The receiver wakes-up only when a packet may be received (earliest time of arrival). We also keep a deterministic behavior, without any false negative.

IV. Low-latency Distributed Scheduling Function

We here consider very long slotframes, where packets are generated infrequently but in a periodic manner. We propose to organize the long slotframe to reduce the end-to-end delay, by using smaller parts, that we call *blocks* (Fig. 4).

A. Slotframe organization

We here consider sporadic traffic, where each sensor reports periodically its measurements to a border router. Thus, the slotframe length has to be equal to the least common multiplier of all the traffic periods. Consequently, in each long slotframe, shared cells are reserved for control traffic, such as 6P packets. One dedicated cell is also reserved for each node to send its unicast control packets corresponding to routing or synchronization. All data packets are transmitted through dedicated cells, that each pair of nodes has to reserve.

We propose to organize the long slotframe into small blocks that repeat over time. To reduce the end-to-end delay, we have to limit the buffering delay when a packet is retransmitted. By reserving consecutive blocks for retransmissions, the buffering

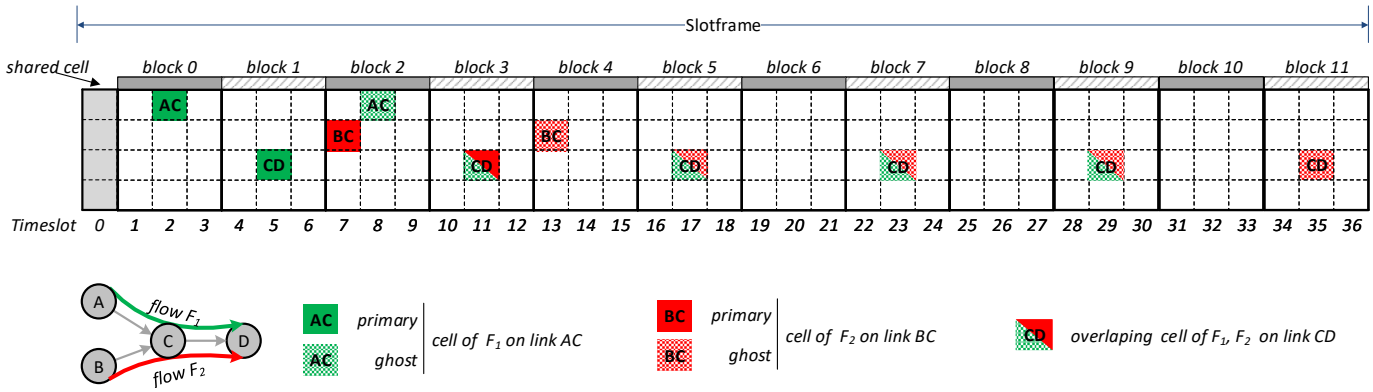


Fig. 4: Slotframe organization in blocks, where $MaxRetries = 1$

.

delay is proportional to the block size. Thus, we divide here the slotframe into small blocks with a few timeslots.

A packet is typically received during a block, and forwarded during the next one. Thus, a transmitter selects its block according to its hop count from the border router. More precisely, each block has a block id, that counts the number of blocks since the beginning of the slotframe. We have consequently even blocks (with an even block id), and odd blocks (with an odd block id). A transmitter has to select a block, so that the remainder of the Euclidean divisions of the hop count and of the block id by 2 are equal. More formally, a transmitter can select any cell in the blocks which respect the following property:

$$HC \pmod 2 = B_{id} \pmod 2 \quad (2)$$

where HC denotes the hop count from the transmitter to the border router, and B_{id} the block id.

Let us consider the topology and the LDSF schedule illustrated in Figure 4. The node A is two hops away from the border router, and the packet is assumed to be generated in the timeslot 0. It must select a block with an even block id ($2 \pmod 2 = 0$). In our example, it selects the timeslot 2. The node C is 1 hop away from the border router, and considers only the blocks with an odd block id. It selects the block 1 (the consecutive block), and reserves one cell (here, the timeslot 5) to forward the packets from A. The blocks are daisy-chained, the packet received during the block i being forwarded in the block $i + 1$.

We make a distinction between the following types of cells:

- **Shared cell** for control packets in broadcast (Enhanced Beacons, routing advertisements), and control packets in unicast when no dedicated cell has been reserved (i.e., 6P requests/replies);
- **Primary** (dedicated) **cell** corresponds to the earliest expected reception time of the data packet from the previous hop (or from the application layer);
- **Ghost** (dedicated) **cells** correspond to the retransmission opportunities, that are automatically used by the transmitter if it did not receive an acknowledgment for its previous transmission. The ghost cells are scheduled in the same timeslot and channel offset as for the corresponding primary cell, but in the subsequent blocks.

When a node reserves a primary cell in a block, a fixed number of ghost cells is automatically reserved every two blocks. Thus, we can daisy chain the transmission opportunities along the path: a node is able to receive a packet during a block, and forward it during the subsequent blocks. This way, we maintain a low end-to-end delay.

A link quality degradation means more retransmissions: in classical scheduling algorithms, we would need to reserve additional cells. Here, we pre-reserve a large number of cells, at the very beginning, to cope with the worst link qualities. Thus, the number of ghost cells (for retransmissions) is fixed, whatever the link quality.

Besides, the impact of the retransmissions on the end-to-end delay is limited since the blocks comprise a small number of timeslots. We use the Automatic Repeat reQuest (ARQ) feature of IEEE 802.15.4-TSCH: the transmitter schedules a retransmission in the next ghost cell only if it does not receive an acknowledgement for its previous transmission.

B. Number of Ghost Cells

We have now to compute the number of ghost cells to provision for the retransmissions.

1) Standard case: The delay induced by the retransmissions is cumulative along the path. Thus, we have to cope with the worst case: a packet may be retransmitted at most $MaxRetries$ times by each transmitter in the path. The latest time of arrival corresponds to the last RX ghost cell (e.g., C receives the packet from A at the latest during the timeslot 8 in Fig. 4).

We make here a distinction between the *source* of the flow that generates a data packet, and a *transmitter* that forwards this packet. The first transmitter in the path corresponds trivially to the source. We can note that the number of ghost cells is proportional to the hop distance from the source. More precisely, a transmitter has to provision $(MaxRetries * (Hops + 1))$ ghost cells for the retransmissions, where $Hops$ denotes the hop distance from the source to the transmitter.

In Figure 4, A is the source ($Hops = 0$) and provisions one primary cell (timeslot 2) and one ghost cell (timeslot 8). For

the node C, it is one hop away from the source ($Hops = 1$). Thus, C allocates for the flow F_1 one primary cell (timeslot 5) and 2 ghost cells ($MaxRetries * (Hops + 1)$). We can note that the node C can receive the packet through the primary or the ghost cells. Thus, even if it receives a packet during the last ghost cell (timeslot 8), it has still two transmission opportunities (timeslots 11 and 17) for one transmission, and one retransmission.

2) Overlapping case: Some flows may overlap, i.e., one relaying node uses the same ghost cells for two different flows. For instance, flows F_1 and F_2 are both forwarded by the node C, where some ghost cells are in common for both flows. A node can easily detect an overlap when receiving a 6P request: the primary cell corresponds to a ghost cell already reserved for another flow.

Even with this overlap, we must be sure to have enough ghost cells to handle the worst case. Let us consider the two following cases:

Case 1) the node receives a packet from the novel flow (F_2) while a packet from the previous flow (F_1) is already in the queue. By construction, the first flow F_1 has still enough ghost cells to handle $MaxRetries$ retransmissions. At the latest, the packet for the flow F_2 is received while only $MaxRetries + 1$ ghost cells remain in its schedule (primary transmission + retransmissions). Thus, we need to provision $MaxRetries+1$ ghost cells for the novel flow F_2, after the ghost cells that would have been allocated to the flow F_2.

Case 2) the node receives a packet from the novel flow (F_2) while the packet from the other flow (F_1) was not yet received. For the same reason, the node has enough ghost cells for F_2 for $MaxRetries$ retransmissions. Thus, we have also to insert in that case $MaxRetries + 1$ additional ghost cells at the end of the range, but they will be used to forward the packet for the flow F_1.

In conclusion, it is sufficient to provision $MaxRetries + 1$ additional ghost cells when an overlap is detected, whatever the hop distance from the source.

C. Scheduling process

We now detail how the cells are reserved by each pair of nodes. Shared cells are only used for signaling, i.e., sending/receiving the 6P packets to negotiate which dedicated cells to use. A 6P request typically piggybacks a list of possible (dedicated) primary cells. The (dedicated) ghost cells are automatically derived from a primary cell. The receiver sends a 6P reply to the transmitter to validate the reservation. Since no dedicated cell is present in the schedule, the 6P packets use the shared cell.

A novel allocation is triggered when a node receives either a 6P request from the previous hop or directly the packet from the application layer. Thus, we propose the following procedure (see Algorithm. 1):

1) First of all, we need to allocate the receiving cells if we receive a 6P request. We reserve in Receiving mode all the timeslots every 2 blocks (lines 1-7), located after the timeslot specified in the 6P request.
   We allocate one primary cell, and ($MaxRetries*Hops$) ghost cells for the retransmission. The value $Hops$ comes from the fact that the receiver for the 6P request is one hop farther from the source than the transmitter. Please note that the pseudo-random function `pseudoRandom()` is executed with the same argument by the receiver and the transmitter to derive the same channel offset (lines 3, and 16).
2) We identify the block which is directly located after the block of the first receiving timeslot (line 8). We will schedule the TX cells after this block;
3) We have to allocate the primary cell, and ($MaxRetries * Hops$) ghost cells (line 10);
4) We have then to make a distinction between the two possible cases:

Algorithm 1: Cell allocation process

Input:
`Schedule`: the current schedule
`Hops`: the hop distance between the node and the source of the packet
`RxSlotId`: receiving timeslot (in the 6P request if received by the radio chipset, or given by the application layer if the packet is generated locally)
$\mathcal{SF}length$: slotframe length
$\mathcal{B}Length$: block length
Output:
Schedule: the schedule updated with the novel cells

```
   // Allocation of the slots in Receiving mode,
      only if the packet is received from the
      radio chipset
1  if packetNotReceivedFromApplication() then
2      for nbretx ∈ [0, MaxRetries * Hops] do
3          ChOff ← pseudoRandom(0, NbChannels − 1)
           // select the timeslots at regular
              intervals (every 2 blocks) after the
              timeslot present in the 6P request
4          TsOffset ←
            (RxSlotId + nbretx * 2 * BLength) % SFlength
5          Schedule.AddCell(TsOffset, ChOff, 'RX')
6      end
7  end
   // Identify the block which is located just
      after the first receiving timeslot
      (allocated in the previous loop)
8  TxSlotId ←
    (RxSlotId + (RxSlotId % BLength) ) % SFlength
   // Allocation of TxSlot (transmitter side)
9  if node ≠ BorderRouter then
10     NbGhostCells ← MaxRetries * (Hops + 1)
       // If we have overlapping flows, pick-up the
          TX cell already allocated in the block
11     if GetBusyTXSlotnBlock(TxSlotId) ≠ ∅ then
12         {TxSlotId, ChOff} ←
            GetBusyTXSlotnBlock(TxSlotId)
           // ghost cells to handle queuing delays
13         NbGhostCells ← NbGhostCells + MaxRetries + 1
14     else
           // Select randomly one timeslot in the
              block
15         TxSlotId ← TxSlotId + random(BLength)
16         ChOff ← pseudoRandom(0, NbChannels − 1)
17     end
       // Allocates the corresponding slots in the
          schedule (every 2 blocks)
18     for nbretx ∈ [0, NbGhostCells] do
19         TsOffset ←
            (TxSlotId + nbretx * 2 * BLength)% SFlength
20         Schedule.AddCell(TsOffset, ChOff, 'TX')
21     end
22 end
23 return Schedule
```

- **Overlapping** flows: A cell in this block is already assigned (line 11). Thus, the allocation will re-use the same timeslot and channel offset (line 12). We have also to allocate $(MaxRetries + 1)$ additional ghost cells to handle the worst queuing delay with the overlapping flow (line 13).
- **No-overlap:** We select randomly the timeslot and channel offset (line 15-16).

5) When we have determined the number of ghost cells, and the first timeslot to allocate, we can proceed to the allocation (lines 18-21). It is worth noting that with overlapping flows, some TX cells may be reserved by several flows (e.g., timeslot 11 in Fig. 4).

Let us illustrate this scheduling algorithm with Figure 4. For the sake of better representation, we assume that the maximum number of retransmissions is equal to one ($MaxRetries = 1$):

- As explained previously, A selects a cell in the block 0. It also reserves automatically one associated ghost cell ($MaxRetries * 1\ hop$) in block 2. After the 6P reservation, the primary and ghost cells are reserved for both the transmitter and the receiver.
- The earliest time of arrival for the node C corresponds to the timeslot 2. Thus, it reserves a cell in the next block (1). It also reserves two ghost cells ($MaxRetries* 2hops$), in the blocks 3 and 5;
- We can note that C is also forwarding the flow F_2 (from B). Its primary cell for the flow F_2 is already reserved as ghost cell for the flow F_1: C detects an overlap. Thus, it first reserves one primary cell (timeslot 11) and 2 ghost cells (timeslots 17 and 23) for the flow F_2. Because, of the overlap, it has also to allocate additional ghost cells to consider the latest time of arrival (cf. section IV-B2). Thus, it allocates two additional ghost cells (one primary cell + $MaxRetries$) after its last ghost cells. In conclusion, it selects the timeslots 29 and 35 as ghost cells.

Since we rely on a distributed random scheduling algorithm, two interfering links may select the same cell. A collision may occur if both transmitters select the same primary cells, or if the ghost and primary cells overlap. For instance, the links AB and CD in Figure 1 may reserve the same timeslot and channel offset. Then, if both nodes A and C send the packet over the same cell, there will be a collision. Since, in our scenarios, the packet generation period is sporadic, and we because consider long slotframes, the probability two or more transmitters to transmit over the same cell is very low. In case of collision, a relocation mechanism [13] will be applied.

D. Energy Savings using Ghost Cells

Reserving ghost cells allows the network to improve the reliability: LDSF can efficiently handle the fast link quality changes since ghost cells are a priori over-provisioned. Concerning the energy efficiency, the transmitter can safely sleep when it has no data packet to transmit. For the receiver side, we have to limit idle listening [22], forcing the node to wake-up at the beginning of the timeslot because it is not aware that the transmitter has nothing to transmit.

Under the LDSF algorithm, we configured a fixed number of ghost cells, based on the hop distance from the source, and a constant, whatever the link quality. A receiving node must wake-up at the beginning of each primary cell, to possibly receive a packet. Then, it must also wake-up for all the subsequent ghost cells until a packet has been received and correctly acknowledged. Once, a packet has been received or the last ghost cell is encountered, the receiver can safely save energy until the next primary cell. The receiver has then to forward the packet, and becomes a transmitter. It selects the corresponding cell in the next blocks, and starts to transmit the packet to the next hop.

Let us consider the scenario illustrated in Fig. 4. Let us assume that the node C has been able to decode the packet from the node A in the timeslot 2. It can stop listening to the ghost cell in timeslot 8. However, it will wake-up during the next block (timeslot 5) to forward the packet to the node D.

The primary cell corresponds to the earliest time arrival to optimize the end-to-end delay. Thus, we do not have any false negatives: the receiver is always awake when the transmission takes place, we thus keep the deterministic behavior of IEEE 802.15.4-TSCH.

V. Mathematical Analysis

Let us analyze next the end-to-end delay, i.e., average time required to deliver a data packet from a node A to the border router S via a path $p = \{N_i\}_{i\in[1,||p||]}$, where $||p||$ denotes the number of nodes in the path p.

A. Model

To obtain a fair evaluation, we use the same mathematical model to compare each scheduling approach (cf. notation in Table I). We have made the following considerations:

- We consider the worst case delay, when the data packet is enqueued at the beginning of the slotframe;
- We assume that the inter packet time is sufficiently large, and the queue for the nodes is empty at the beginning of the slotframe.

We apply here the same methodology as in [15] to compute the average time before a packet is delivered to the border router (i.e., the end-to-end delay).

1) MSF: Since cells are randomly scheduled, we consider a uniform distribution of the cells in the slotframe.

To be generic, we compute the normalized delay in number of timeslots. We have to multiply the normalized delay by the timeslot duration T_{slot} to compute the actual delay. Compared with [15], we have the following normalized delay:

$$hop_{delay}(N_i, N_{i+1}) = \frac{\mathcal{SF}length}{2 * Cells(N_i, N_{i+1})} \tag{3}$$

where $\mathcal{SF}length$ denotes the slotframe length, i.e., number of timeslots.

If the radio link between N_i and N_{i+1} is unreliable, the transmitter needs $\frac{1}{P_{ll}(N_i,N_{i+1})}$ transmissions before the packet is successfully decoded and acknowledged. So using eq. 3:

$$hop_{delay}(N_i, N_{i+1}) = \frac{\mathcal{SF}length * \frac{1}{P_{ll}(N_i,N_{i+1})}}{2 * Cells(N_i, N_{i+1})} \tag{4}$$

Finally, the average end-to-end delay to deliver a packet from a source node to the border router along the routing path $p = \{N_i\}_{i\in[1,||p||]}$ is:

$$E2E_{delay}(p) = \mathcal{SF}length * \sum_{i=1}^{||p||-1} \frac{1/P_{ll}(N_i, N_{i+1})}{2 * Cells(N_i, N_{i+1})} \quad (5)$$

2) Stratum: If we consider the length of each stratum is sufficient to handle retransmissions, the maximum end-to-end delay corresponds to the slotframe length ($\mathcal{SF}length$). In this case, the packet is delivered successfully before the end of the stratum:

$$E2E_{delay}(p) = \mathcal{SF}length \quad (6)$$

If the number of cells is insufficient, the retransmissions may be scheduled during the subsequent slotframe, increasing the end-to-end delay by one slotframe length. We may use the distribution of the packet delivery success according to eq 1 to derive the distribution of this additional delay. It depends also on the length of each stratum, since it upper bounds the number of cells we can allocate. While eq. 6 corresponds to an optimistic case, we consider that the stratum length is correctly sized to handle the worst case situation.

3) LDSF: The cells are pseudo-randomly allocated in a block. Besides, the node N_{i+1} will forward a packet it received from N_i in the just immediately consecutive block (by construction).

Thus, the average buffering time is equal to the block length:

$$hop_{delay}(N_i, N_{i+1}) = \mathcal{B}Length \quad (7)$$

If the radio link between N_i and N_{i+1} is unreliable, they require $1/P_{ll}(N_i, N_{i+1})-1$ retransmissions. Since consecutive primary/ghost cells are interspaced by $2 * \mathcal{B}Length$ timeslots, we have:

$$hop_{delay}(N_i, N_{i+1}) = \mathcal{B}Length + \mathcal{B}Length \left(\frac{2}{P_{ll}(N_i, N_{i+1})} - 1 \right) \quad (8)$$

Finally, the average end-to-end delay is:

$$E2E_{delay}(p) = \mathcal{B}Length \sum_{i=1}^{||p||-1} \left(\frac{2}{P_{ll}((N_i, N_{i+1})} - 1 \right) \quad (9)$$

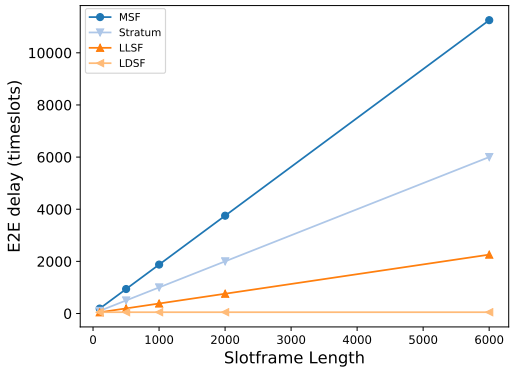
4) LLSF: the approach daisy-chains the cells along the path, relying on relocation when they are not contiguous because of retransmission cells. Thus, we will here focus on the steady-state, after the convergence, i.e. we neglect the effect of suboptimal relocations.

The first hop delay is obtained similarly to MSF. Since the nodes select the transmission cells randomly on the first hop, the delay of the first hop is:

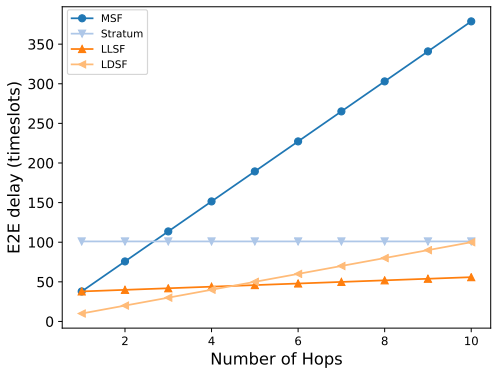
$$hop_{delay}(N_1, N_2) = \frac{\mathcal{SF}length * \frac{1}{P_{ll}(N_1, N_2)}}{2 * Cells(N_1, N_2)} \quad (10)$$

Then, the end-to-end delay is similar to LDSF. Indeed, two cells for the retransmissions are now consecutive instead of being separated by $2 * \mathcal{B}Length$ timeslots. The average end-to-end delay is thus:

$$E2E_{delay}(p) = hop_{delay}(N_1, N_2) + \sum_{i=2}^{||p||-2} \left(\frac{2}{P_{ll}(N_i, N_{i+1})} - 1 \right) \quad (11)$$



(a) Flow through a route of 5 hops.



(b) Slotframe with 101 timeslots.

Fig. 5: Impact of the slotframe length and number of hops on the end to end delay with a PDR per link of 66%.

B. Numerical results

We consider here a scenario, with unreliable radio links (i.e., $P_{ll} = 66\%$). Thus, two cells are allocated for each link ($Cells = 2$) for possible retransmissions, and the block length is fixed to 5 timeslots ($\mathcal{B}Length = 5$).

Figure 5a illustrates the impact of the slotframe length. The end-to-end delay of both MSF and Stratum is linear with the slotframe length. Indeed, Stratum can only guarantee a delay equal to the slotframe length. As can be observed, the performance of LLSF is affected by the slotframe length due to the selection of the transmission cells randomly at the first hop. On the contrary, the delay of LDSF is independent of the slotframe length: the cells are chained along the path.

Similarly, we can see that the delay increases with the number of hops for MSF (Figure 5b): the TX and RX cells are uniformly distributed, and a node has to buffer the packet for a long time before being able to forward it. On the

contrary, Stratum provides a delay independent of the hop length. However, this hop distance cannot exceeds the number of stratums, which is 10 in this case. LDSF achieves to provide a much smaller delay, increasing only slightly for long routes. More precisely, the delay is increased by one block length per hop, i.e., 5 timeslots. Finally, the LLSF achieves low delay too especially when the number of hops increases.

VI. Performance Evaluation

We now assess the performance of our distributed scheduling function in a more realistic environment, where packets can be dropped because of collisions and low link qualities.

A. Simulation Setup

We use here the 6TiSCH Simulator [23], a discrete-event simulator written in Python. We consider the following scenarios:

1) Constant Bit Rate (CBR) flows
2) Topologies with a variable number of nodes (by default 39 nodes and one border router) to assess the scalability of our Scheduling Function

We generate random topologies, where each node is randomly located in an area of $2000X2000\ m^2$. Thus, each node has at least 3 neighbors. The propagation model of 6TiSCH Simulator is based on the Pister-Hack model [24]. Table II regroups all the values of the different parameters.

We extended the 6TiSCH Simulator with three additional scheduling functions (LDSF, LLSF [18] and Stratum [15]). For the LDSF and Stratum algorithms, each node has to know its hop distance from the border router. We implement in Stratum the opportunistic de-allocation of cells described in [8], where an unused cell is removed after a long timeout to avoid schedule's inconsistencies.

The node's lifetime is $Lf_i = \frac{Battery_{Capacity} \times SF_{duration}}{Q_{slotframe} * 3600 * 24 * 365}$. Where $SF_{duration}$ is the duration of the slotframe (in seconds), $Battery_{Capacity}$ is the node's battery capacity (in μC), $Q_{slotframe}$ is the average energy drawn during a slotframe (in μC), and $(3600 * 24 * 365)$ allows to convert seconds into years. To compute the lifetime of each node, we rely on the energy model described in [22] that relies on real measurements. The model provides the power of the node in each state:

$TxDataRxAck$**:** the transmitter sends a packet and waits for an acknowledgement;

$TxData$**:** the transmitter sends a packet without waiting for an ACK (e.g., a broadcast packet);

$RxDataTxAck$**:** the receiver decodes a packet and sends an ACK;

$RxData$**:** the receiver decodes the packet, and switches directly to sleeping mode, without sending an ACK;

$Idle$**:** the receivers listens to the medium but does not sense anything;

$Sleep$**:** the nodes turns off its radio chipset.

Thus, the simulator [23] has just to count the time spent in each state, combined with the values measured in [22] and reported also in Table II to compute the lifetime metric.

TABLE II: Simulation setup.

Topology	Parameter	Value
	# of nodes	39 + 1 border router
	# of Experiments	20 per algorithm
Simulation		
	Duration	60 min
	Payload size	90 bytes
Protocol		
CoAP	CBR (*Unicast*)	$1\ pkt/[5, 10, 20..120] sec$
RPL	DAO period	60 s
	DIO period	8.5 s
TSCH	NShared cells	1
	Timeslot duration	10 ms
	Maximum retries	5
MSF, Stratum & LLSF	Slotframe length	101 timeslots
LDSF	Slotframe length	$CBR * 101$ timeslots
	Block length	5 timeslots
Queues	Timeout	10 s
	Queue size	10 packets
Energy		
	Energy Model	[22]
	$Battery_{Capacity}$	$10157.4 \times 10^6\ \mu C$
	$Idle$	$6.4\ \mu C$
	$Sleep$	$0\ \mu C$
	$TxDataRxAck$	$54.5\ \mu C$
	$TxData$	$49.5\ \mu C$
	$RxDataTxAck$	$32.6\ \mu C$
	$RxData$	$22.6\ \mu C$

We measure here the network lifetime as the time until the first node dies because it runs out of energy (n-of-n lifetime in [25]). We assume that the system reaches a steady state, and we just have to identify the node with the largest energy consumption. Finally, the network's lifetime is:

$$\mathcal{NL} = min\{Lf_i\}_{i \in [1, ||Nodes|| - 1]} \tag{12}$$

B. Scheduling Algorithms

We compare the following approaches:

MSF: [26] is the default scheduling function of 6TiSCH, where autonomous (pseudo-random) cells are used for control traffic and dedicated cells are used for data packets;

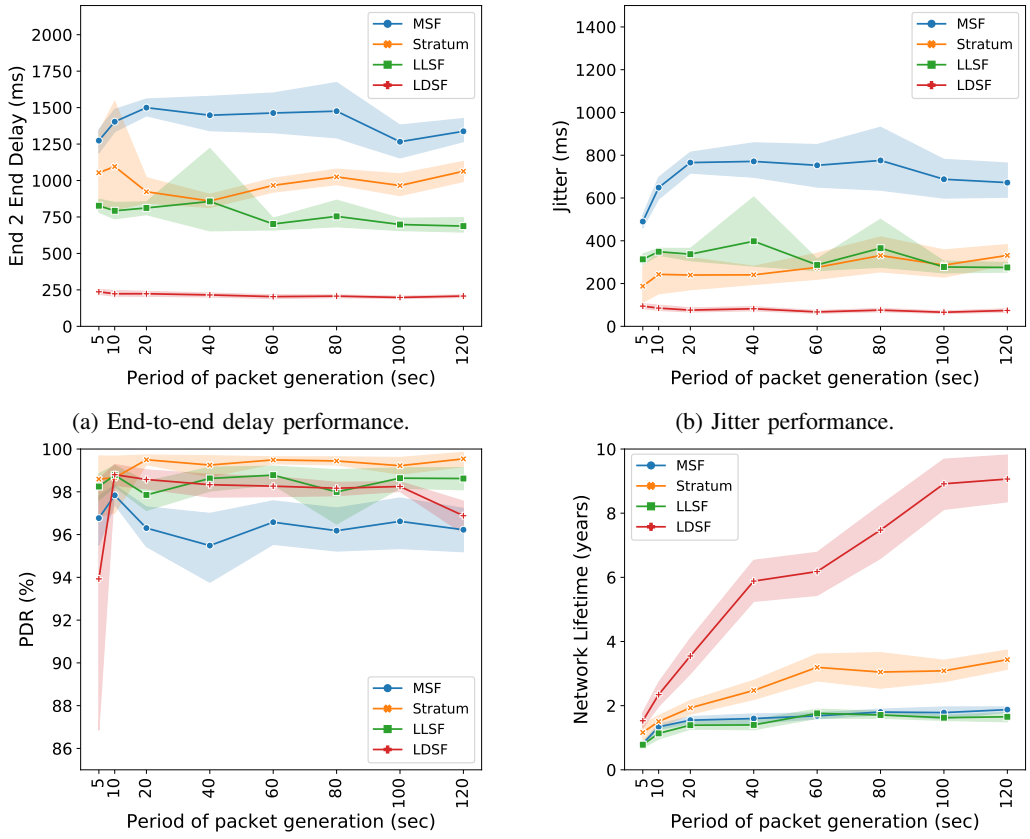
Stratum: [15] divides the slotframe in blocks (i.e., stratums). Each node selects a block according to its hop distance, so that a packet is delivered in the current slotframe;

LLSF: [18] aims to reduce the end-to-end latency by allocating receiving and transmitting cells as close as possible in the schedule.

LDSF: Our scheduling function described in Section IV.

Stratum uses a slotframe length of 101 timeslots, to be able to provide an end-to-end delay equal to 1010 ms (=101 * 10ms). MSF and LLSF also uses the default slotframe length (101). LDSF uses rather a slotframe length proportional to the maximum flow rate, since it was designed for this purpose. The same cell is used every two blocks for transmissions and retransmissions. Since each block comprises 5 timeslots in LDSF, the transmitter has to wait on average 10ms * 5 timeslots * 2blocks = 100ms.

Our implementation (simulation code, scripts, and raw data) is freely available



(a) End-to-end delay performance.

(b) Jitter performance.

(c) End-to-end packet delivery ratio.

(d) Network lifetime.

Fig. 6: Impact of the traffic rate (i.e., inter packet time).

(https://github.com/vkotsiou/Scheduling for the implementation, and https://doi.org/10.5281/zenodo.3748712 for our dataset).

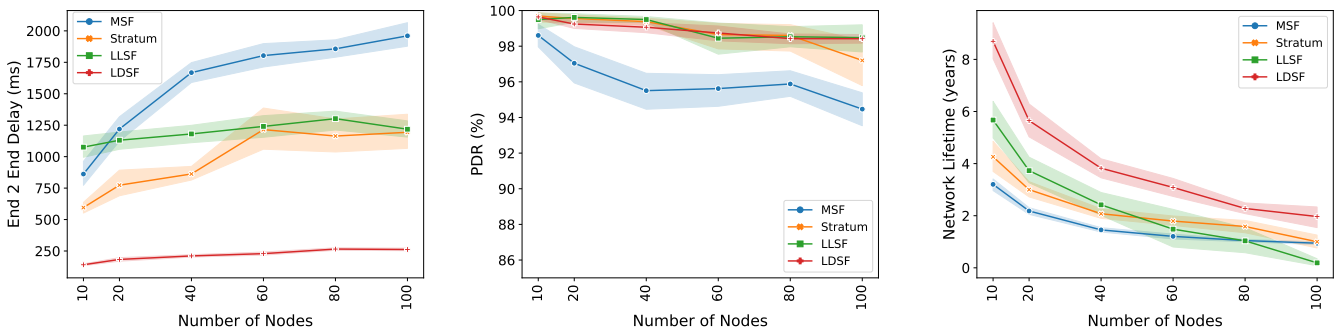
C. Traffic Rate

We first measure the average end-to-end delay (Fig. 6a). LDSF is very robust to large traffic rates: it keeps on providing a very low delay. Stratum presents a very stable end-to-end delay: packets are delivered at the end of the slotframe exactly (101 timeslots * 10ms). MSF presents the highest end-to-end delay because it does not have any cell allocation strategy to minimize the delay: it picks randomly the cells. LLSF provides also a very stable delay. While the cells are reserved consecutively along the path, the first cell is picked randomly and generates a large buffering delay (half of the slotframe = 505ms).

We can make the same remarks concerning the jitter (Fig. 6b). LDSF provides the lowest jitter performance which is less than 150 ms, even for very high traffic rates. Collisions are accurately handled, and the packets are retransmitted efficiently in the subsequent blocks to minimize the buffering delay. LLSF achieves a larger jitter: the schedule is modified as soon as some retransmission cells have to be inserted. However, this optimization has a cost since the whole schedule has to be modified along the path Stratum provides jitter performance similar to LLSF, corresponding to the length of the last *stratum* (i.e., block). Indeed, a packet is randomly scheduled in the last *stratum*to be received by the border router. Since this *stratum* is typically much larger than the LDSF's block, the jitter is mechanically increased. Finally, MSF provides the worst jitter since retransmission cells can create a cumulative effect along the path, since they can be allocated after the cells of the next hop.

Figure 6c focuses on the reliability. The three schemes are able to guarantee end-to-end reliability above 96% in most cases. Stratum achieves the highest reliability for low traffic rates since the blocks are large to avoid collisions. However, the number of collisions starts to increase for high traffic rates (inter packet time $<$ 10s). LDSF is able to provide an end-to-end packet delivery ratio higher than 98%. LLSF provides also a good reliability, except for high-traffic rates: many collisions arise and are particularly challenging to resolve since the cells are contiguous. Moreover, the scheduling process needs to solve the collisions for each cell, while LDSF is more robust since the same cell is pre-reserved also for the retransmissions.

Fig. 6d illustrates the network lifetime. We extrapolate the average energy consumption for the most loaded node to derive the network lifetime. MSF generates a large number of control packets with many (de)allocations, which impact negatively the lifetime. Stratum increases slightly the lifetime, by reducing the renegotiation of cells. LLSF achieves the same characteristics since it uses a short slotframe (101 timeslots) and that shared cells consume energy. Finally, LDSF is very efficient to handle unreliability: ghost cells are automatically



(a) End-to-end delay (b) End-to-end packet delivery ratio (c) Network lifetime.

Fig. 7: Impact of the number of nodes.

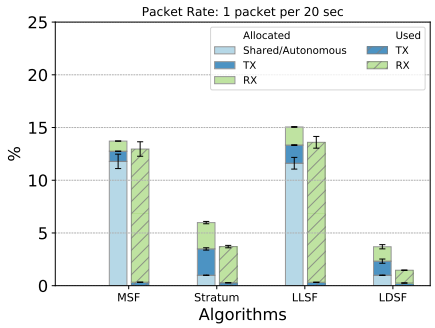


Fig. 8: Percentage of the slotframe occupied (i.e., cells allocated), with one packet generated by each node every 20 seconds.

reserved after a 6P transactions, minimizing the amount of control traffic. Thus, the lifetime increases for very low traffic conditions.

D. Scalability

We then measure the scalability of these distributed scheduling functions by increasing the number of nodes up to 100. LDSF is very scalable; even a large number of nodes does not increase significantly the collisions. The delay remains below 200 ms. The delay of MSF and Stratum is much higher, while the delay of LLSF scales smoothly with respect to the number of nodes. MSF provides the lowest end-to-end reliability (Fig. 7b): some packets are dropped because of an excessive number of retransmissions, or because of a buffer overflow. Stratum, LLSF and LDSF achieve to still provide a very high reliability even with 100 nodes generating one packet every 20 seconds. More than 98,5% of the packets are delivered to the border router.

LDSF achieves the higher network lifetime whatever the number of nodes (Figure 7c). The ghost cells are pre-reserved, but the transmitter and the receiver do not have to wake-up in every ghost cells. More precisely, they will wake-up only if no acknowledgement was received correctly. This way, LDSF provides the larger lifetime compared against MSF, LLSF and Stratum approaches.

E. Slotframe occupation

Finally, Figure 8 illustrates the percentage of the slotframe occupied, i.e., the cells are allocated to at least one transmitter. We can note that MSF and LLSF use short slotframes, thus the ratio of shared cells is higher compared to long slotframes. Thus, idle listening will consume a large quantity of energy. Stratum may generate more collisions since interfering links have often to select their cell in the same stratum. Thus, the number of allocated cells is reduced, but keeps larger than LDSF. LDSF organizes the cells appropriately and can exploit long slotframes (as long as the CBR period). While LDSF reserves ghost cells for the retransmissions, they are used only if the transmission has failed. If the packet was acknowledged, neither the transmitter nor the receiver will wake-up during the next ghost cells (cf. amount of unused cells in Figure 8).

VII. Conclusion and Future Work

Minimizing the end-to-end delay requires to chain the cells along a path, so that the buffering delay is minimized. However, achieving high reliability in the context of IEEE 802.15.4-TSCH is particularly challenging since increasing the number of retransmissions may imply to reschedule all the cells in the path toward the destination. We propose here LDSF which divides the long slotframe into small blocks that repeat over time. Each node selects the right block corresponding to its hop distance from the border router to minimize the delay. In order to still provide high reliability, ghost cells are automatically reserved in the slotframe and the delay is still minimized even if the packet has to be retransmitted. Our simulation results demonstrate that LDSF achieves low latency and jitter with high reliability, even for multihop topologies.

In this article, we have assumed that a cell can be used to send a packet from any flow. Considering flow isolation, each flow can use only a specific set of cells. In other words, ghost cells have to be reserved per flow, impacting negatively the network capacity. Inversely, using the same ghost cells for all flows may impact both the end-to-end delay and the reliability.

There are several techniques in the literature that improve the reliability, such as multipath routing [27], whitelisting [28], or even at the PHY layer [29]. For future work we aim at investigating how these techniques can be combined with LDSF in order to cope with time-variable delivery rates.

REFERENCES

[1] S. Longo, T. Su, G. Herrmann, and P. Barber, *Optimal and robust scheduling for networked control systems*. CRC press, 2017.

[2] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-Defined Industrial Internet of Things in the Context of Industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, Oct 2016.

[3] G. Z. Papadopoulos, P. Thubert, F. Theoleyre, and J. Bernardos, "Raw use cases," IETF, draft-bernardos-raw-use-cases 03, March 2020.

[4] X. Xu and Q. Hua, "Industrial Big Data Analysis in Smart Factory: Current Status and Research Strategies," *IEEE Access*, pp. 17 543–17 551, 2017.

[5] W. Yu, T. S. Dillon, F. Mostafa, W. Rahayu, and Y. Liu, "A Global Manufacturing Big Data Ecosystem for Fault Detection in Predictive Maintenance," *IEEE Transactions on Industrial Informatics*, 2019.

[6] IEEE Standard for Low-Rate Wireless Networks, "IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)," April 2016.

[7] R. T. Hermeto, A. Gallais, and F. Theoleyre, "Scheduling for IEEE802.15.4-TSCH and Slow Channel Hopping MAC in Low Power Industrial Wireless Networks," *Comput. Commun.*, pp. 84–105, Dec. 2017.

[8] F. Theoleyre and G. Z. Papadopoulos, "Experimental Validation of a Distributed Self-Configured 6TiSCH with Traffic Isolation in Low Power Lossy Networks," in *MSWiM*. ACM, 2016.

[9] Wang, Qin and Vilajosana, Xavier and Watteyne, Thomas, "6TiSCH Operation Sublayer (6top) Protocol (6P)," RFC 8480, November 2018.

[10] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel, "On Optimal Scheduling in Duty-Cycled Industrial IoT Applications Using IEEE802.15.4e TSCH," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3655–3666, Oct 2013.

[11] G. Gaillard, D. Barthel, F. Theoleyre, and F. Valois, "High-reliability scheduling in deterministic wireless multi-hop networks," in *PIMRC*. IEEE, Sep. 2016, pp. 1–6.

[12] F. Dobslaw, T. Zhang, and M. Gidlund, "End-to-End Reliability-Aware Scheduling for Wireless Sensor Networks," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 758–767, April 2016.

[13] T. Chang, T. Watteyne, X. Vilajosana, and Q. Wang, "CCR: Cost-aware cell relocation in 6TiSCH networks," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 1, p. e3211, 2018.

[14] M. Domingo-Prieto, T. Chang, X. Vilajosana, and T. Watteyne, "Distributed PID-Based Scheduling for 6TiSCH Networks," *IEEE Communications Letters*, vol. 20, no. 5, pp. 1006–1009, May 2016.

[15] I. Hosni, F. Theoleyre, and N. Hamdi, "Localized scheduling for end-to-end delay constrained Low Power Lossy networks with 6TiSCH," in *ISCC*, June 2016, pp. 507–512.

[16] I. Hosni and F. Theoleyre, "Self-healing distributed scheduling for end-to-end delay optimization in multihop wireless networks with 6TiSCh," *Computer Communications*, vol. 110, pp. 103 – 119, 2017.

[17] G. Daneels, B. Spinnewyn, S. Latré, and J. Famaey, "ReSF: Recurrent Low-Latency Scheduling in IEEE 802.15.4e TSCH networks," *Ad Hoc Networks*, vol. 69, pp. 100 – 114, 2018.

[18] T. Chang, T. Watteyne, Q. Wang, and X. Vilajosana, "LLSF: Low Latency Scheduling Function for 6TiSCH Networks," in *DCOSS*, May 2016.

[19] I. Hosni and F. Theoleyre, "Adaptive k-cast Scheduling for High-Reliability and Low-Latency in IEEE802.15.4-TSCH," in *Ad-hoc, Mobile, and Wireless Networks (ADHOCNOW)*. Springer, 2018, pp. 3–14.

[20] R.-A. Koutsiamanis, G. Z. Papadopoulos, X. Fafoutis, J. M. Del Fiore, P. Thubert, and N. Montavont, "From Best Effort to Deterministic Packet Delivery for Wireless Industrial IoT Networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4468–4480, Oct 2018.

[21] G. Gaillard, D. Barthel, F. Theoleyre, and F. Valois, "Kausa: KPI-aware Scheduling Algorithm for Multi-flow in Multi-hop IoT Networks," in *ADHOC-NOW*, 2016, pp. 47–61.

[22] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. J. Pister, "A Realistic Energy Consumption Model for TSCH Networks," *IEEE Sensors Journal*, vol. 14, no. 2, pp. 482–489, Feb 2014.

[23] E. Municio, G. Daneels, M. Vučinić, S. Latré, J. Famaey, Y. Tanaka, K. Brun, K. Muraoka, X. Vilajosana, and T. Watteyne, "Simulating 6TiSCH networks," *Transactions on Emerging Telecommunications Technologies*, vol. 30, no. 3, 2019.

[24] H.-P. Le, M. John, and K. Pister, "Energy-aware routing in wireless sensor networks with adaptive energy-slope control," *EE290Q-2 Spring*, 2009.

[25] I. Dietrich and F. Dressler, "On the Lifetime of Wireless Sensor Networks," *ACM Trans. Sen. Netw.*, vol. 5, no. 1, Feb. 2009.

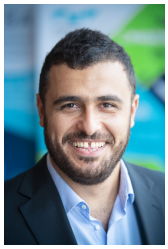
[26] T. Chang, M. Vucinic, X. Vilajosana, S. Duquennoy, and D. Dujovne, "6TiSCH Minimal Scheduling Function (MSF)," IETF, draft 3, April 2019, https://tools.ietf.org/html/draft-ietf-6tisch-msf-03.

[27] E. M. Ahrar, M. Nassiri, and F. Theoleyre, "Multipath aware scheduling for high reliability and fault tolerance in low power industrial networks," *Journal of Network and Computer Applications*, pp. 25 – 36, 2019.

[28] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "Whitelisting without Collisions for Centralized Scheduling in Wireless Industrial Networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5713–5721, June 2019.

[29] N. Saxena, A. Roy, B. J. R. Sahu, and H. Kim, "Efficient IoT Gateway over 5G Wireless: A New Design with Prototype and Implementation Results," *IEEE Communications Magazine*, pp. 97–105, Feb. 2017.

Vasileios Kotsiou is a Ph.D. student at the ICube laboratory, University of Strasbourg (France). Previously, he received his M.Sc. in Engineering of Pervasive Computing Systems from Hellenic Open University in 2014 and his B.Sc. in Computer Science from University of Crete (Heraklion) in 1996. He has already co-authored several peer-reviewed papers in international conferences and journals. His research interests span primarily in the area of Wireless Sensor Networks (WSNs), Internet of Things (IoT), Ubiquitous and Mobile Computing.



Georgios Z. Papadopoulos [S'10, M'15] serves as an Associate Professor at the IMT Atlantique in Rennes, France. He received his Ph.D. from University of Strasbourg (France), in 2015 with honors. Dr. Papadopoulos has been involved in the organization committee of many international events (IEEE ISCC'20, AdHoc-Now'18, IEEE CSCN'18, GIIS'18). Moreover, he has been serving as Editor for Wireless Networks journal and Internet Technology Letters. He is author of more than 50 peer-reviewed publications in the area of networking.

Periklis Chatzimisios [S'02, M'05, SM12'] received his B.Sc. from Alexander TEI of Thessaloniki (ATEITHE) (Greece) in 2000 and his Ph.D. from Bournemouth University (U.K.) in 2005. He serves as a Professor in the Department of Science and Technology at the International Hellenic University. He is involved in several standardization and IEEE activities serving as a member of the Standards Program Development and the Education Services Boards for the IEEE Communication Society (ComSoc) as well as the Chair of the IEEE ComSoc Technical Committee on Information Infrastructure and Networking (TCIIN).

Fabrice Théoleyre [S'05, M'09, SM'16] is a researcher at the CNRS. After having spent 2 years in the Grenoble Informatics Laboratory (France), he is part of the ICube lab (Strasbourg, France) since 2009. He received his Ph.D. in computer science from INSA, Lyon (France) in 2006. He was a visiting scholar at the University of Waterloo (Canada) in 2006, and a visiting researcher at INRIA Sophia Antipolis (France) in 2005 and at Inje University (Korea) in 2014. He's involved in about ten TPC per year, and is area editor for Ad Hoc Networks since 2018. He has been associate editor for IEEE Communications Letters and guest editor for Computer Communications and Eurasip JWCN.