Enabling Centralized Scheduling Using Software Defined Networking in Industrial Wireless Sensor Networks

Farzad Veisi Goshtasb, Julien Montavont, Fabrice Theoleyre

Enabling Centralized Scheduling Using Software Defined Networking in Industrial Wireless Sensor Networks

Farzad Veisi, Julien Montavont and Fabrice Theoleyre ICube, CNRS / University of Strasbourg
Pole API, 67412 Illkirch, France
Emails: veisigoshtasb@unistra.fr, montavont@unistra.fr and fabrice.theoleyre@cnrs.fr

***Abstract*—Industrial Wireless Sensor Networks (IWSN) play a key role in the Industry 4.0 revolution. The network infrastructure is critical to interconnect sensors and actuators and needs to respect Key Performance Indicators. IEEE 802.15.4-TSCH is a candidate technology for IWSN since it relies on scheduled transmissions and frequency hopping to make the network more reliable. However, distributed scheduling solutions fail to provide high reliability and low end-to-end latency. Software Defined Network (SDN) tends now to emerge in wireless networks as well, where a controller is in charge of the whole network configuration. But scheduled wireless networks require to go beyond usual SDN forwarding rules by including the radio resource allocation (dedicated time-frequency blocks). Moreover, radio links are known to be unreliable, and we need to adapt the control and data planes to make the network efficient. We propose SDN-TSCH to orchestrate a scheduled network adapting the SDN paradigm. More specifically, the controller is in charge of i) selecting the time source, ii) maintaining a tree structure for the control plane, with scheduled resources dedicated to the control plane, and iii) installing a new data flow while guaranteeing flow isolation. We also propose a very efficient link quality estimation technique tailored for scheduled TSCH networks. Our simulations highlight that the SDN controller can allocate in SDN-TSCH just-enough resources to respect both latency and reliability constraints.**

***Keywords*—Industrial Internet of Things; Software Defined Networking; scheduling; resource allocation; flow isolation**

I. Introduction

Industry 4.0 relies on Wireless Sensor Networks (WSNs) to automate industrial applications [1]. Typically, each device hosts a critical application that generates data packets requiring high Quality of Service (QoS) in terms of reliability and latency. Since wireless links are known to be lossy, the communication network has to provide mechanisms to respect these Service Level Agreements (SLAs).

IEEE 802.15.4-TSCH [2] is a good candidate for industrial networks. It relies on a Time Division Multiple Access (TDMA)-based Medium Access Control (MAC) layer with a frequency hopping mechanism to provide reliable communications.

To respect the SLAs defined by each application, each transmitter has to carefully select the resources to use in the Frequency-Time Division Multiple Access (FTDMA) matrix. Distributed algorithms such as the Minimal Scheduling Function (MSF) [3] are popular but cannot guarantee a collision-free schedule, as demonstrated in Section V-B. Indeed, each pair of nodes has to negotiate the resources to use, which may create collisions with already established interfering flows. Alternatively, centralized scheduling algorithms can provide strict guarantees by exploiting graph-based and coloring approaches. However, most of them have been evaluated with numerical analyses, assuming that all the inputs are known accurately. More specifically, the scheduler is preconfigured with the network topology and the traffic matrix.

Software Defined Networking (SDN) represents a promising solution: devices execute simple forwarding rules that are installed by a controller. However, in wireless networks, the controller has a broader role and needs to i) maintain the network synchronized to exploit the FTDMA matrix without collision, ii) support unreliable links in the control and data planes and avoid collisions, iii) admit data flows and install forwarding rules. SDNWISE-TSCH [4] represents a pioneering piece of work to support a centralized controller. However, it still suffers from collisions, as demonstrated in Section V-A, resulting in packets that exceed QoS limits which are unacceptable for most industrial applications.

In lossy networks, the controller can allocate additional radio resources for retransmissions to compensate for packet loss. However, the controller may inaccurately estimate the link qualities. Under-estimation leads to over-provisioning (energy wastage), and over-estimation leads to under-provisioning (unreliability). Active techniques rely on probes (aka control packets) transmitted regularly to estimate the link quality [5]. However, a node must dedicate resources for each neighbor, which represents an unacceptable amount of energy and bandwidth. Passive techniques are less expensive but may inaccurately estimate the link quality if packets collide [6].

We already proposed a solution to implement a SDN architecture in IWSN [7]. The controller assigns radio resources to maintain the control plane such that any device has a collision-free path from and to the controller. In addition, the controller can reserve radio resources for critical flows in the data plane to respect end-to-end reliability and latency constraints per flow. Packet forwarding uses a label-switching approach such that a node automatically selects adequate radio resources to forward a specific packet using a simple forwarding table. We also defined the format of the report and the configuration packets for the SDN network.

We extend here this solution in the following way:

1) we propose a new discovery process to improve the link quality estimation. In particular, we introduce a new slotframe structure to allow the controller to assign separated resources for Enhanced Beacons that are used in the passive computation of link quality. By this means, we prevent collisions from impacting the accuracy of the estimation. In addition, we bound the discovery time such that the controller has a consistent view before making a decision;
2) we improve the time synchronization of the network by delegating the TSCH time source selection for each new device to the controller;
3) we compare our solution through a thorough simulation campaign using Contiki-OS and Cooja with a state-of-the-art distributed solution (MSF [3]) to highlight the advantage of SDN, and with a state-of-the-art SDN solution for IWSN (SDNWISETSCH [4]) to demonstrate the relevance of maintaining a reliable control plane with flow isolation.

It is important to note that our primary focus is not on the scheduling algorithm itself, and we currently use a greedy approach. However, our architecture allows for the integration of any centralized algorithm if desired.

II. Background & Related Works

We detail here background notions and related works on IEEE 802.15.4-TSCH networks and SDN with a focus on industrial WSN.

A. IEEE 802.15.4-TSCH Background

IEEE 802.15.4-TSCH is an operational mode designed for low-power industrial applications. IEEE 802.15.4-TSCH combines Time Division Multiple Access (TDMA) with a frequency hopping mechanism to provide reliable and energy-efficient communications. A scheduling mechanism determines when a transmitter must start its transmission (timeslot) and which frequency to use (channel offset). The transmissions are organized in a slotframe, *i.e.*, a matrix of cells as pairs of timeslots and channel offsets. Each node has in its schedule a list of TX (respectively RX) cells during which it has to stay awake to transmit (respectively receive) packets.

IEEE 802.15.4-TSCH exploits two types of cells in the slotframe:

a dedicated cell is allocated to a single transmitter and one or several receivers. Thus, the transmitter can engage its transmission without contention;

a shared cell is allocated to a collection of transmitters, resulting in potential collisions. If `ack` is expected but not received, the transmitter waits for a random number of shared cells to retransmit the packet. Collisions may be frequent even with a low traffic intensity [8].

In IEEE 802.15.4-TSCH, nodes synchronize using periodic Enhanced Beacons (EB). The packet piggybacks Information Elements (IE), which is an extensible mechanism to exchange information at the MAC sublayer. In particular, the Absolute Sequence Number (ASN) serves as a global clock in the network. Since the timing is fixed (no backoff) in a timeslot, the receiver can adjust its clock after any packet reception. Figure 1 illustrates a simple TSCH schedule composed of shared and dedicated cells. In this example, each node has a dedicated cell to its parent in the routing tree and can use the shared cell for broadcasting. The frequency hopping mechanism ensures that the same cell uses a different frequency in consecutive slotframes.

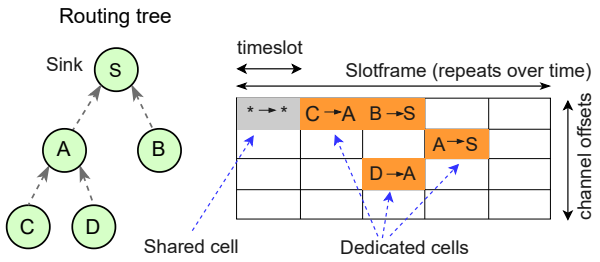

Fig. 1: Simple TSCH schedule with shared and dedicated cells

B. Scheduling algorithms for industrial WSN

One of the main challenges of TSCH is how to schedule the transmissions into the FTDMA matrix. Distributed algorithms can be found in the literature, such as the 6TiSCH Minimal Scheduling Function (MSF) [3]. MSF is the IETF standard to allocate cells in the TSCH schedule in a distributed way. MSF operation relies on three types of cells:

- *minimal cells* are used for broadcast control traffic (Enhanced Beacons, and routing packets);
- *autonomous cells* are derived pseudo-randomly from the ID of the transmitter (TX-cell) or the receiver (RX-cell). Typically, a node uses a neighbor RX-cell to dynamically negotiate additional cells with that neighbor with the 6P [9] protocol;
- *negotiated cells* are dedicated cells negotiated by a pair of nodes to exchange traffic.

MSF generally relies on RPL to build the network topology and derive autonomous cells. Upon RPL convergence, every node requests its parent for one negotiated cell using an autonomous cell. Then, a node can request additional negotiated cells if its current schedule cannot handle its current traffic load. Similarly, a node can remove negotiated cells in case of unused cells in its schedule. However, MSF may converge slowly because one single cell only can be allocated/removed per time window.

DeTAS [10] is another distributed scheduler for TSCH. The network is organized as a tree in which every node knows the amount of traffic it will generate and receive from its children. This information is transmitted hop-by-hop to the root tree. Then, the schedule is initiated by the root and built at every hop: each node assigns an alternate sequence of Tx and Rx slots to every child node. Those sequences follow an even (Tx slots are placed on even time slots) or odd (Tx slots are placed on odd time slots) schedule regarding the relative position of

the node in the tree. Finally, channel offsets prevent collisions from occurring. The main drawback of DeTAS lies in its static schedule, which lacks the ability to be updated in response to changing traffic conditions.

LDSF [11] proposes additional cells for retransmissions, while still allocating cells back-to-back in the path to minimize the end-to-end latency.

By contrast, centralized solutions allow the scheduling algorithm to be executed on a centralized entity. AMUS [12] proposes a greedy scheduling approach that allocates more cells to nodes closer to the sink, assuming those nodes have more traffic to forward. More advanced solutions rely on a complete view of the network topology, as well as the traffic requirements of nodes. TASA [13] represents a pioneering piece of work in this field. It constructs a schedule using graph coloring and a greedy allocation. Additional cells can be allocated for retransmissions to make the network reliable [14]. Various fog computing solutions have been proposed to enhance network processing in close proximity to the network edge. For instance, SPSRP [15] introduces task-based fog computing with an efficient resource allocation and task offloading algorithm. However, collecting network information and redistributing the computed schedule on a multi-hop wireless network can be a highly intricate and complex process. SDN can serve as a practical solution for this task.

C. SDN for industrial WSN

Software Defined Networking (SDN) [16] promises to make the network more efficient and flexible by centralizing the intelligence in a controller. Since the SDN controller has a complete view of the network, it makes optimal decisions on paths. The controller exploits a so-called southbound API (*e.g.*, OpenFlow) to collect information on the network and to push forwarding rules on the devices.

SDN-WISE [17] proposes a pioneering piece of work to adapt SDN paradigms to wireless networks. It focuses on the aggregation problem when installing forwarding rules. TinySDN [18] supports multiple controllers to reduce the bottleneck problem, but it supports only basic actions, such as forward and drop. IT-SDN [19] defines additional interfaces for neighbor discovery and controller discovery. However, all these solutions do not support scheduled networks, where the controller needs to allocate radio resources. SD6WSN [20] proposes a SDN architecture for 6LoWPAN networks to reduce latency by leveraging the controller's knowledge. It deploys a SDN-based agent in each node, utilizing RPL topology for control plane communication.

uSDN [21] implements the SDN concept in TSCH networks. It relies on a distributed routing protocol, RPL [22], to maintain the control plane. In that way, each device has a route to join the controller, that allocates resources when a new flow is admitted. However, the distributed nature of RPL may result in routing instabilities, inconsistencies, and loops, making the control plane unstable. On the contrary, Whisper [23] exploits a centralized controller that controls the behavior of RPL and 6P protocols. The controller artificially manipulates the RPL rank of a node to force parent changes, setting up new paths. The controller also injects fake 6P commands to (de)allocate time-frequency blocks between two nodes. The resulting communication system is very complex, requiring the nodes to fully implement RPL and 6P, while SDN promises to simplify the network elements.

SDNWISE-TSCH [4] adapts SDN-WISE for scheduled networks. This solution focuses mainly on the scheduling process, computing the path and the number of resources to allocate to each flow. However, the mechanisms proposed to set up the control and data planes lack the necessary features to guarantee reliability and efficiency. We are convinced that securing the communication channel with the controller and providing accurate network information to the controller are the first mandatory steps toward an operational SDN solution. Regardless of the SDN mechanisms in place, any scheduling algorithm should be integrated into the controller.

III. Problem Statement

Industrial networks require bounded end-to-end latency and high reliability. Thus, we need to define an efficient end-to-end schedule for each critical flow to respect these constraints. We propose to rely on a SDN controller that computes the resource allocation and installs the forwarding rules. Each device discovers already associated network nodes and then engages in a procedure with the SDN controller to join the network. More precisely, the controller deploys rules in the control plane to establish the association of the new device with the network, thereby configuring the data plane accordingly. Table I presents the summary of the solutions existing in the literature and their properties.

A. Topology Discovery

The controller needs to discover already associated nodes to join the network. SDN-WISE [17] and SDNWISE-TSCH [4] exploit extra beacon packets for topology discovery. A broadcast packet is sufficient to be detected possibly by all the neighbors, making this solution energy efficient. In TinySDN [18], each node sends a probing packet to its neighbors. However, sending unicast probes is expensive in scheduled networks since dedicated bandwidth has to be reserved a priori for each neighbor. Also, the existing traffic of the network can be used to discover the neighbors. Several solutions [20, 21, 23] exploit the RPL neighbor list instead of performing topology discovery. However, it is proved that RPL has inconsistencies in its routing table [24]. It is worth noting that we need a continuous discovery process to detect new neighbors.

B. Link quality estimation

The controller allocates radio resources based on the link qualities it has collected. If the link quality is over-estimated, the controller will not allocate enough bandwidth, leading to SLAs violations. Over-provisioning is not a solution since it would waste energy and bandwidth.

Link quality estimation has been broadly investigated in the literature [25] with active versus passive techniques. Active

TABLE I: Summary of related works with supported features

		Topology discovery		Link quality estimation		Reliability of the control/data planes		
		mode	mechanism	mode	metric	data/control separation	collision-free control plane	data flow isolation
No scheduling	SD6WSN [20]	passive	RPL DIO	passive	ETX	no	no	no
	SDN-WISE [17]	active	extra broadcast probing	active	RSSI	no	no	no
	TinySDN [18]	active	extra broadcast probing	active	ETX	no	no	no
	IT-SDN [19]	active	extra broadcast probing	active	ETX	no	no	no
Scheduling	SDNWISE-TSCH [4]	active	extra broadcast probing	active	RSSI	no	no	no
	uSDN [21]	passive	RPL DIO	passive	ETX	yes	no	no
	Whisper [23]	passive	RPL DIO	passive	ETX	no	no	no

techniques rely on the transmission of additional unicast or broadcast packets. However, radio resources have to be dedicated to probes in scheduled networks, wasting energy. Passive techniques rely on existing control or data packets to estimate the link quality. However, unbalanced traffic and collisions may increase inaccuracy.

In [17, 4], the controller exploits the RSSI value of the last beacon reported by each device. However, a high RSSI does not directly lead to a high packet reception ratio [26]. TinySDN [18] uses rather probing packets. Each node transmits a broadcast probing packet and waits for the unicast reply of its neighbors. In this way, a node calculates the Expected Transmission Count (ETX) toward each of its neighbors. The solution presented in [20] relies on the routing protocol (RPL) and more specifically on the ETX metric maintained by RPL. Each node periodically sends unicast probing packets to neighbors and calculates the corresponding ETX.

C. Reliable control plane

Making the control plane reliable is of uppermost importance for wireless networks. Accurate reports are essential for the controller to make decisions, while reliable transmission of commands is necessary to ensure globally consistent forwarding rules. Unfortunately, all the existing schemes do not focus on this unreliability problem. Several solutions rely on a min-hop routing for the control plane, which tends to favor longest and thus most unreliable links [18]. Similarly, relying on RPL is particularly risky since this routing protocol has been proven to exhibit oscillations [27]. In scheduled networks, it means reallocating resources through the new RPL tree. To save energy, using the same resources for the control and data planes [23] is particularly detrimental to reliability. Only Baddeley [21] propose to separate the resources for the control and data planes. However, they employ a distributed scheduling mechanism for the control plane, which lacks collision-free properties.

Maintaining a consistent SDN control plane in constrained IWSNs imposes an additional energy overhead: the continuous exchange of control messages introduces extra data transmissions and processing tasks for the network devices. Therefore, minimizing this energy cost becomes paramount to ensure the efficient operation and longevity of battery-powered devices within the network.

D. Reliable data plane

Critical applications require high reliability (*e.g.*, end-to-end PDR = 99%) while wireless networks are unreliable. The SDN controller has to compensate with extra resources for the weak links. A node that fails to send a packet will use backup cells for retransmissions. Moreover, flow isolation is a key enabler for the reliability of critical low-power applications [8]. To the best of our knowledge, no solution constructs a reliable data plane with flow isolation.

IV. SDN-TSCH

We propose SDN-TSCH for IWSN to respect Service Level Agreements. It relies on the SDN paradigm and TSCH to benefit from a centralized scheduler. SDN-TSCH addresses all the requirements previously presented in Section III.

A. Tree structure to maintain the control plane

To avoid collisions, we allocate dedicated radio resources for the control plane, such that a device can send `reports` to the controller and receive `commands` from the controller. To save energy, we need to minimize the number of resources dedicated to the control plane. Thus, we propose to use a tree structure: each device maintains a next hop (its parent) toward the SDN controller. The controller selects the parent of each joining device to keep in the tree structure only the best links: good links mean less retransmissions.

B. Slotframe and schedule organization

We isolate the radio resources for the control and data planes. More precisely, the slotframe (*e.g.*, Fig. 2) regroups:

shared cells are used for broadcast traffic. More precisely, these shared cells are divided into:

- **EB cells** (*e.g.*, cells 3, 9, and 15) allocated by the controller to one single transmitter at a time so that it can transmit its EBs.
- **non-EB cells** (*e.g.*, cells 0 or 6) allocated for the traffic of the nodes that want to join the network (*i.e.*, the unassociated nodes). Typically, a joining node needs to contact the SDN controller through one non-EB cell (no cell has been already reserved for it).

dedicated control cells are used for the control plane: one cell (in green) toward the parent to reach the controller

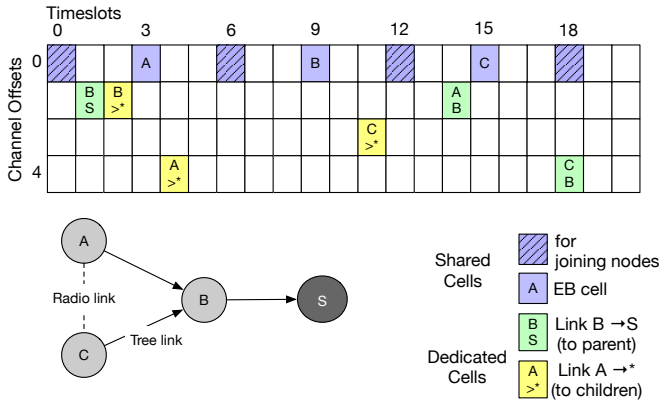

Fig. 2: Organization of the slotframe

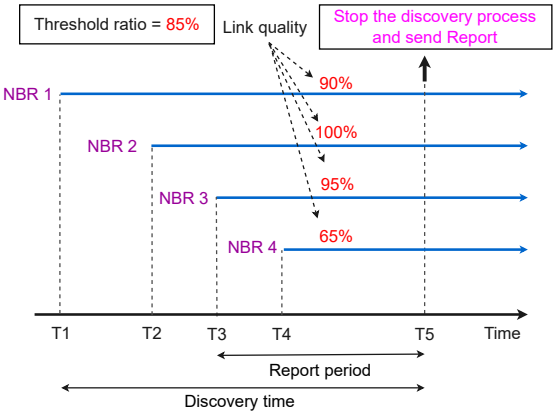

Fig. 4: Discovery time of a joining node

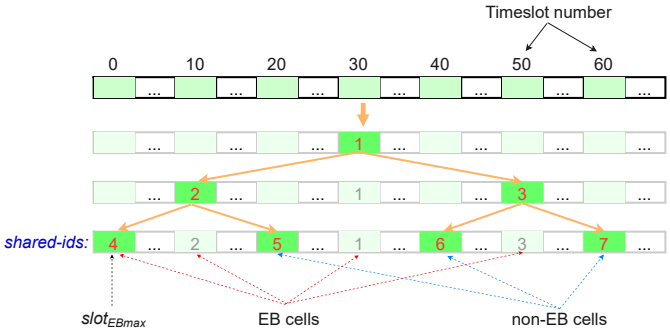

Fig. 3: Shared cell ID allocation in slotframe for the control plane

(`report` packets), and one cell (in yellow) toward the children (`config` packets). It is worth noting that no collision can arise: one single transmitter is active at a time;

dedicated data cells are assigned by the SDN controller for each data flow. Forwarding rules guarantee flow isolation. For the sake of conciseness, the data cells were not depicted in Fig. 2.

We must make a clear distinction between EB and non-EB shared cells. EBs cells are reserved by the controller and cannot be used for the nodes that want to join the network. Else, they could create collisions, leading to instabilities in the control plane. For this purpose, the controller updates the number of nodes that have joined the network, denoted as the $slot_{EBmax}$ value. This value is transmitted to the root tree, then piggybacked in all EBs for dissemination throughout the network. In that way, all the nodes, including those joining the network, can distinguish EB shared cells from non-EB shared cells: if the slot ID is smaller than $slot_{EBmax}$, the shared cell is reserved for EBs, else, the cell can be used by any unassociated node.

To minimize the collision probability, the EB cells should be distributed in the slotframe. More precisely, we assign to each shared cell a unique ID (designated as *shared-id*). Then, we use a recursive algorithm (Fig. 3) to assign *shared-id*s regularly in the slotframe. Recursively, the controller assigns the *shared-id*s to split the remaining available space into two equal parts. For instance, in the first step, the *shared-id*1 is located in the middle of the slotframe (between 0 and the slotframe length). As illustrated in Fig.3, the *shared-id*1 corresponds to the timeslot 30. Then, the subsequent *shared-id*s are positioned in the middle of their respective halves. Each node can apply the same recursive algorithm to identify EB and non-EB cells using $slot_{EBmax}$.

C. Link quality estimation & discovery process

Link quality estimation is achieved passively by listening to EBs from neighbors during EB shared cells. Since we exploit a collision-free schedule of EBs, counting the number of EBs received from each neighbor is sufficient to estimate the PDR according to Eq. 1:

$$\widehat{PDR}(n) = \frac{counter(n) * T_{EB}}{T_{report}} \quad (1)$$

with $counter(n)$ the EB counter of node n, T_{EB} the period of EBs, $\widehat{PDR}(n)$ the measured Packet Delivery Ratio (PDR) for n, and T_{report} the period of report.

Let us focus on the discovery process: a collection of nodes has already joined the SDN-TSCH network, and a new node turns its radio on to join the network. When it receives an EB, it adjusts its clock to be synchronized with the network, using the source of the EB as a time synchronization source. The joining node also extracts the IEs from the EBs to get the TSCH parameters (e.g., the slotframe length, the $slot_{EBmax}$ value described previously) and installs the shared cells in its slotframe. The discovery process stops when all good neighbors (neighbors for which the link quality is above a predefined threshold) have been seen for at least one T_{report} period. More precisely, the process stops if Eq. 2 is satisfied:

$$\forall n \in \mathcal{N} | \widehat{PDR}(n) \geq PDR_{min}, t_0(n) + T_{report} \geq t \quad (2)$$

with $t_0(n)$ the receiving time of the first EB from n, $\mathcal{N}$ the set of neighbors, and t the current time.

As illustrated in Fig. 4, the joining node stops the discovery process only when the link quality of all good neighbors

($NBR1$, $NBR2$, and $NBR3$) is measured for at least one report period. This ensures a reliable margin in the measured link quality. It is worth noting that the joining node does not consider the discovery time of $NBR4$ as it proposes a PDR lower than the threshold ratio.

The SDN controller needs real-time updates on the evolution of link qualities, and hence, computed PDRs are continuously transmitted to the controller every T_{report} once the discovery process is completed.

D. Label switching

We use label switching for forwarding. Each time a node receives a packet, it extracts the associated flow-id from the header and adds the packet to the corresponding queue. When the controller installs cells in the schedule, it flags the corresponding cells with a flow-id. At the beginning of a TX-cell flagged with flow-id X, the node transmits the first packet of the corresponding queue. All the routing decisions rely on the controller, and we predefine two flow-ids for the control plane:

"to-controller" to handle the upward control packets generated by nodes;

"from-controller" to handle the downward packets generated by the controller.

E. Joining processes

When a joining node has performed the discovery process, it needs to join the SDN network. For his purpose, it creates a `report` packet including the list of neighbors and their associated PDR. This first `report` packet is sent in unicast to the neighbor with the highest link quality to maximize the transmission success probability. Since the joining node has no configured control plane, it must use a non-EB shared cell to transmit its `report` packet. That's not the case for the next-hop nodes: the `report` packet traverses hop-by-hop the network to reach the controller in a reliable manner using the *"to-controller"* flow-id.

When receiving a report packet, the controller verifies if the report packet comes from a joining node. In that case, the controller registers the node in the joined list and selects the neighbor with the best link quality as the parent to maximize the reliability of the control plane. Since the parent is also used as the time source in TSCH, it minimizes the probability of desynchronization. Then, the controller selects randomly two dedicated cells in the slotframe for the control plane: one for sending upward control packets to the parent node and one for receiving downward control packets from the parent node. Obviously, allocated cells must correspond to an unused timeslot for the parent (half-duplex condition) and cannot be allocated to another interfering link (collision-free condition).

Finally, the controller constructs two `config` packets piggybacking these two cells to configure the control plane of the joining node. To reach the joining node, the controller can use the flow-id *from-controller* already configured in the rest of the network (except the last hop). However, we need to implement routing: several children may exist at each hop,

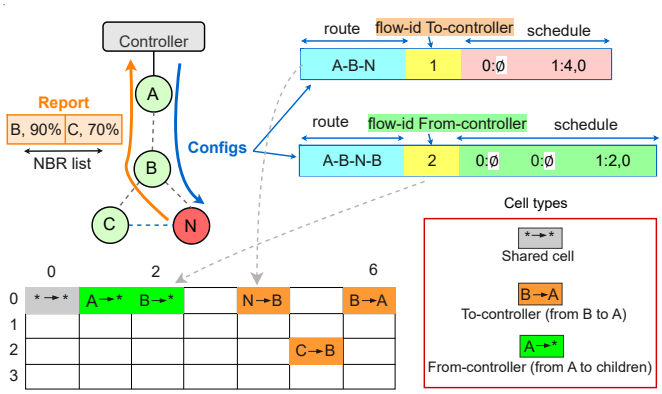

Fig. 5: Dedicated control plane in SDN-TSCH

and a `config` packet should be routed in the correct subtree, toward the joining node. Hopefully, the controller knows the complete topology and can compute a path to the joining node. Thus, we implement source-routing: the whole path is piggybacked in each `config` packet. Each node that receives a config packet finds its position in the source routing list and extracts the next address in the list.

When a node n receives the config packet, it exploits the route and the schedule to update its configuration:

1) it extracts the flow-id and looks for its position in the route. n searches for a pair of nodes in the route that corresponds to the address of the transmitter of the `config` packet, and its own address. We denote by i the position of its address in the route ($i \in [0, k[$, k being the number of nodes in the source routing path);
2) if $i > 0$, it installs the $(i-1)^{th}$ element of the schedule as the TX cells in its scheduling table;
3) if $i < k-1$, it installs the i^{th} element of the schedule as RX cells in its scheduling table;
4) it forwards the `config` packet to the next hop following the source routing path. For configuring the *"to-controller"* flow-id, nodes use the *"from-controller"* flow-id if $i < k-2$, or the next available shared cell if $i = k-2$. For configuring the *"from-controller"* flow-id, nodes use the *"from-controller"* flow-id if $i < k-3$, or the next available shared cell if $i = k-3$, or the *"to-controller"* flow-id if $i = k-2$. Indeed, the *"to-controller"* flow-id is installed first.

Figure 5 illustrates the control plane configuration. As a joining node, node N sends a report packet to the controller. Upon reception, the controller selects node B as the parent of node N since it is the neighbor with the highest link quality. It creates two `config` packets including the addresses for source routing and the schedule. More specifically, we encode the schedule of each hop as <`number_of_cells:list_of_cells`>. Notably, the first forwarding nodes do not have any cell to install (`number_of_cells` $= 0$). It is worth noting that we always configure a flow from the destination toward the source. This approach allows for a generic scheme that enables the configuration of data flows in a reliable manner (see Section IV-F).

Thus, the configuration process starts from node N to B for the flow ID *"from-controller"*.

F. Resource allocation for the data plane

Any critical application opens a socket connection, describing its QoS requirements (*i.e.*, end-to-end minimum reliability and maximum latency). The node engages a call admission by sending a `flow-request` to the controller through the *"to-controller"* flow-id. When receiving a `flow-request`, the controller constructs a schedule that respects the QoS requirements:

1) the controller computes a path from the source to the destination using the tree topology (through the node/parent links);
2) the controller allocates backup dedicated cells for retransmissions for weakest links. Additional backup cells are assigned until the minimum end-to-end reliability is respected [28];
3) the controller schedules cells back to back (or as close as possible) to minimize the end-to-end delay.

If the schedule computation is impossible, the request is rejected, and the controller sends a negative (empty) `config` packet to the source node. If the computation is successful, the controller defines a new flow-id for the flow and constructs a `config` packet including the new schedule and the corresponding flow-id. The controller uses a single `config` packet to configure the whole path, which is composed of:

1) subpath from the root to the destination: the schedule of this part is empty, and the node only forwards the `config` packet with the *from-controller* cells;
2) subpath from destination to source: the configuration starts from the destination node toward the source node. Each node in this part extracts and installs the flow-id and the corresponding cells. When the source node receives the `config` packet, the whole path is configured, and it starts sending packets without delay.

It is worth noting that the technique to identify the position of a forwarding node (cf. subsection IV-E) is still valid when a node is present in both subpaths. Indeed, we individually identify each **link** in the whole path when installing the schedule.

For the second part of the path, each node has to select either the *"to-controller"* or *"from-controller"* flow-ids to forward the `config` packet. To determine the direction, each node checks if the address of the next hop is also present in the list of nodes preceding its position in the source routing. If this is the case, the next hop is an upper-hand node, and the `config` packet uses the *"to-controller"* flow-id. Otherwise, the node uses the *"from-controller"* flow-id to forward the `config` packet.

Figure 6 illustrates a scenario where S is the source node. The `flow-request` packet describes the flow requirements (PDR = 90%, end-to-end delay = 70 ms) and the destination D. In return, the `config` packet is forwarded by source routing, first from A to D and then from D to S. The controller allocates more cells to weak links (2 cells for S→E and 2 cells for E→B) to meet the end-to-end PDR

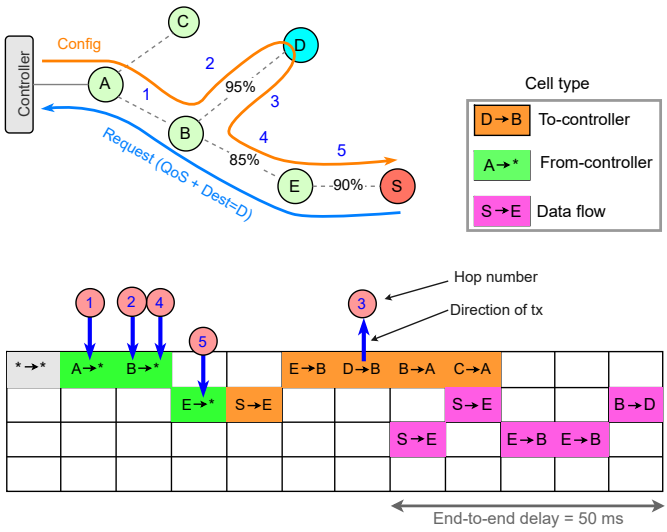

Fig. 6: Data flow Configuration in SDN-TSCH

of flow (90%). The controller allocates cells sequentially to consider the end-to-end delay of flow (70 ms). The route $A - B - D - B - E - S$ is piggybacked in the `config` packet. Thus, D knows B is an upper-hand node since B is both present after and before D: it has to use the *"to-controller"* flow-id to forward the `config` packet. Inversely, S is a downward node for E since S is only present after E: it has to use the *"from-controller"* flow-id.

V. Performance Evaluation

We implement SDN-TSCH in Contiki-ng to assess its performance. We compared SDN-TSCH with two state-of-the-art approaches:

MSF [3] is the IETF standard for distributed scheduling[1], combining autonomous and negotiated cells to avoid collisions;

SDNWISE-TSCH [4] is a variant of SDNWISE to cope with TSCH networks. SDNWISE-TSCH enables the SDN architecture for Industrial Wireless Sensor Networks. It computes a schedule taking into account the deadline of each flow.

We simulate network sizes of 10, 20, 30, 40, and 50 nodes with 10 random topologies per network size. Each node hosts a critical application toward the sink node. Each application needs an end-to-end Packet Delivery Ratio higher than 99% and a deadline of 2 s. We use the default values for the parameters of MSF [3]. Table II represents our different parameters.

A. Comparison of SDN-TSCH and SDNWISE-TSCH

We compare SDN-TSCH with SDNWISE-TSCH to compare two different SDN-based approaches. More precisely, we i) evaluate the reliability and energy efficiency of the control

[1] https://github.com/alexrayne/contiki-ng.git

TABLE II: Simulation parameters

Simulation environment	OS: Contiki-ng (version 4.7) Simulator: Cooja https://github.com/Farzadv/Contiki-ng-SDN-TSCH.git Simulation time: 2.2h Propagation model: Unit Disk Graph Medium Tx range = 100 m Interference range = 150 m Rx success = proportional to distance (100% - 0%) Initial energy = 2400 mAH (AAA battery) Energy consumption model = Energest tool in Contiki-ng (tx_curr = 17.7mA, rx_curr = 20mA)
Topology	Network sizes: 10, 20, 30, 40, and 50 nodes
Application	Number of data flow: 1 flow per node Traffic pattern: Convergecast, Constant 1 packet every 5s Required QoS: PDR >= 99%, Deadline <= 2s
SDN-TSCH	TSCH EB period: 15s SDN report period: 5 min `flow-request` timeout: 50s

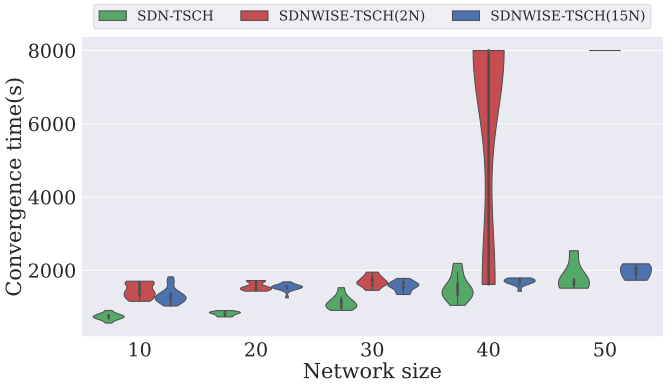

Fig. 7: Network convergence time

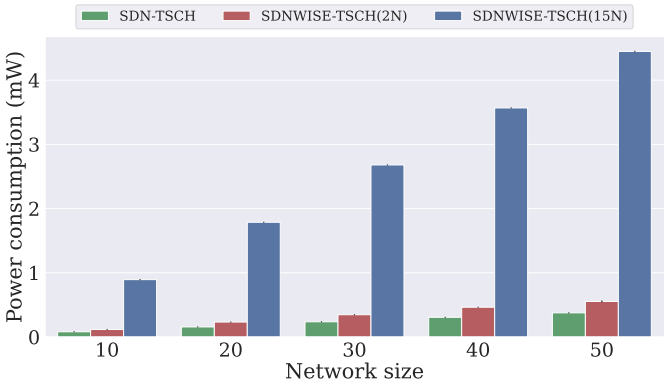

Fig. 8: Power consumption of nodes in joining period

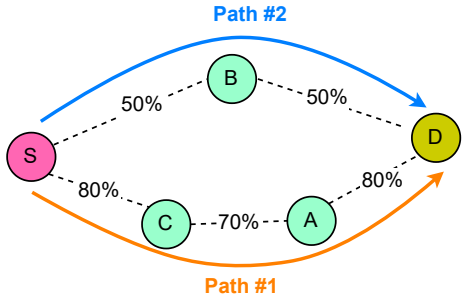

Fig. 9: Data flow handling in SDN-TSCH and SDNWISE-TSCH

plane, and ii) assess the performance of the flow guarantees in the data plane.

SDNWISE-TSCH employs shared cells for both EBs and SDN control packets. According to [4], a network of 10 nodes and slotframe size of 19 uses 2 shared cells. To accommodate different network sizes without impacting the reliability of the control plane of SDNWISE-TSCH, we propose to maintain constant the time per node between two shared cells. Thus, we define the default number of shared cells as follows:

$$N_{shared-cells} = \frac{2 * SF_{length}}{19} * \frac{N}{10} \approx N * SF_{length} * 0.01 \quad (3)$$

with $N_{shared-cells}$ the number of shared cells, N the number of nodes, and SF_{length} the slotframe length.

1) Reliability and energy efficiency of the control plane: Figure 7 shows the convergence time of each approach for different network sizes. We define convergence as the time when the last node in the network is admitted by the SDN controller. We test two ratios of shared cells in the shared control plane: i) $15N$ (eq. 3) and ii) $2N$, which may create more collisions for larger networks. With 2N shared cells, the number of collisions becomes very high for medium-sized networks. In the worst conditions, the network never converges. With 15 shared cells per slotframe, SDNWISE-TSCH succeeds to converge. However, the convergence time is still larger for SDNWISE-TSCH compared with SDN-TSCH. Indeed, shared cells receive a peak of control traffic, and the loss of control packets significantly impacts the convergence.

Using shared cells also impacts energy consumption, as illustrated in Figure 8. We focus here on the power consumption of the network during the convergence period. Using only 15N shared cells consumes much energy: all the nodes wake up during these slots. On the contrary, using only 2N shared cells is much more energy efficient (but with an impact on reliability). Only SDN-TSCH is able to converge fast while providing a very reasonable energy consumption: using dedicated cells is much more efficient, even for control traffic. Indeed, the nodes have to wake up less frequently, and the transmissions are more reliable since we cannot create collisions.

2) Data flow handling: We focus now on the scenario illustrated in Figure 9 to assess the performance of the solution to identify good paths. Nodes A, B, C, and S host critical flows to the destination node D, and node S has two possible paths to reach it. Path 1 has an end-to-end link quality of 45% ($0.8 \times 0.7 \times 0.8 = 0.45$), while path 2 has an end-to-end link quality of 25% ($0.5 \times 0.5 = 0.25$).

To configure the flow of node S, the SDNWISE-TSCH controller selects path 2 based on the shortest path criteria. It only allocates one cell for each hop and schedules cells back to back to respect the flow deadline. However, most of the transmissions fail because of low link quality. Besides,

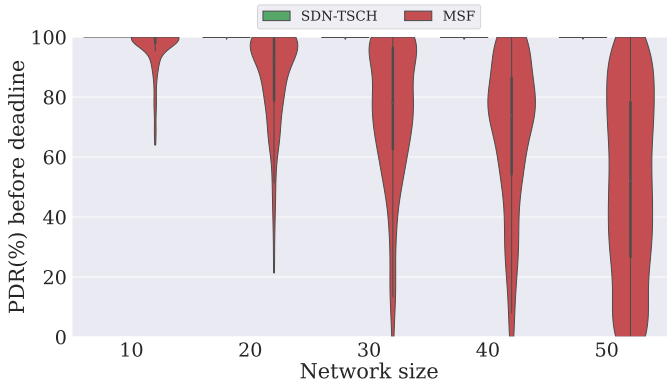

Fig. 10: PDR of flows befor deadline

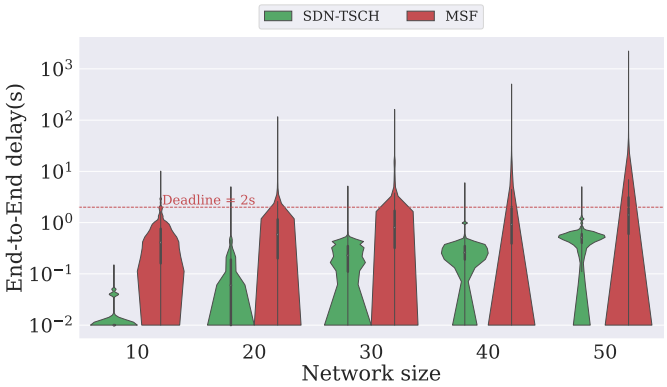

Fig. 11: End-to-end delay

the buffering delay increases and the packets are received after the deadline. Moreover, due to the lack of flow isolation, node B uses any cell scheduled toward node D to send its packets. While it allows using cells unused by the other flow, it also impacts the reliability of the competing flows (from S). Finally, 78% of packets are lost along the path, and 22% of packets are received at the destination after the deadline.

Let us focus on the behavior of SDN-TSCH. The controller selects path 1 when it admits the flow of S: the path uses links with better quality, and fewer retransmissions would be required to reach the same level of reliability. Besides, the controller allocates a sufficient number of cells (including the backup cells): 100% of the packets of node S are received by the destination before the deadline. SDN-TSCH selects efficient data paths and allocates enough resources while guaranteeing flow isolation.

B. Comparison of SDN-TSCH and MSF

We compare SDN-TSCH with MSF to assess the differences between a centralized solution and a distributed solution in terms of both reliability and energy efficiency. Figure 10 illustrates the end-to-end PDR of each flow. Clearly, MSF provides very low reliability. With 20 nodes, the average e2e PDR is larger than 95%, but some flows exhibit a PDR of only 50%. The reliability is even worse with 50 nodes: more flows are forwarded, and the region around the sink becomes a bottleneck. Thus, MSF fails to provision enough backup cells to provide high reliability. On the contrary, SDN-TSCH achieves perfect end-to-end reliability, whatever the conditions. Even better: the reliability is equal to 100% even in the worst case, which is an expected property for industrial networks.

Figure 11 illustrates the end-to-end delay. We plot the 2s deadline (horizontal line) to see its impact. MSF tends to deliver packets very close to the deadline,but too many packets are received after the deadline, particularly for large networks. On the contrary, the SDN controller provisions enough resources in SDN-TSCH. While the latency increases because packets are forwarded through longer paths, the deadline is still respected. The packet losses correspond mostly to statistical outliers.

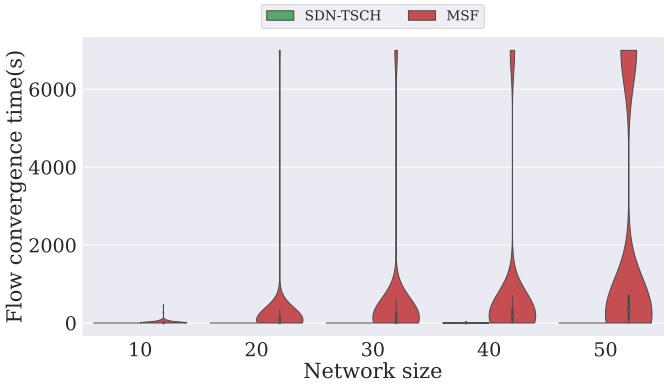

Fig. 12: Data flow convergence time

We compare the convergence of MSF and SDN-TSCH in Figure 12. We define the convergence time for a flow as the duration it takes for the flow's Packet Delivery Ratio (PDR) to reach the desired PDR. In MSF, increasing the network size leads to longer convergence times: each hop of the path needs to negotiate cells reactively. In addition, the arrival of new flows impacts the performance of the ongoing ones because MSF does not provide flow isolation. This observation is clearer on Fig. 13, which illustrates the instantaneous PDR of a given flow while MSF converges. As we can see, the PDR value fluctuates significantly between 40% and 100% because a new flow is admitted in the network and competes for the same resources, or when a collision arises temporarily and the corresponding cell has to be relocated. As a result, relying on distributed algorithms, such as MSF, impacts convergence and PDR. In SDN-TSCH, the flow convergence time is effectively reduced to zero because the controller pre-configures the entire path for a flow. Thus, we have an efficient call admission scheme: the flow starts only when and if enough resources can be scheduled all along the path.

Figure 14 focuses on efficiency. We measured the ratio of the cells that are reserved for each link. To measure the efficiency, we normalize this amount of cells by the required number of cells computed directly from the parameters of the PHY layer. We can note that MSF makes under-provisioning: the ratio is smaller than 1.0, which means that the number

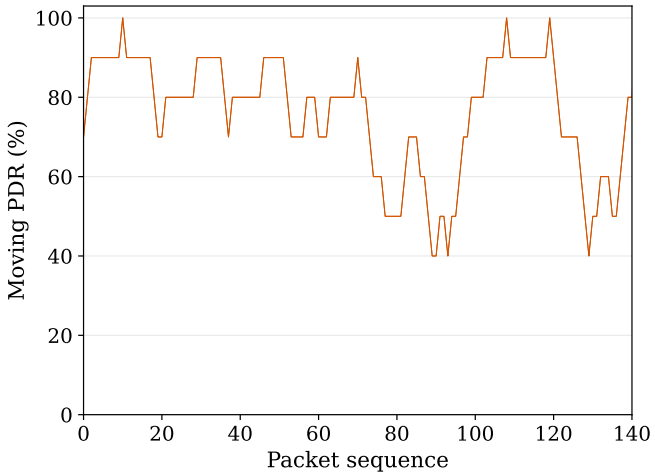

Fig. 13: Instantaneous PDR of a given flow during a time interval using MSF

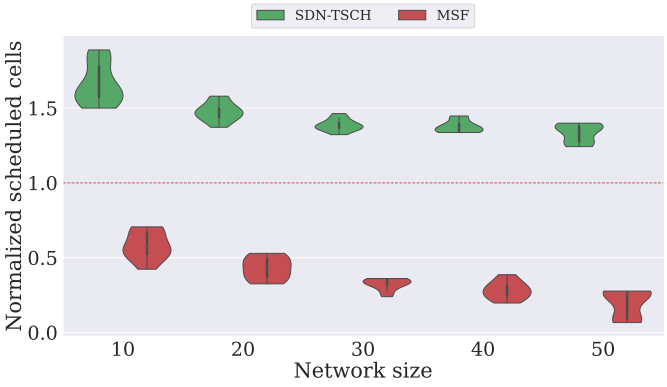

Fig. 14: Ratio of scheduled cells

of cells is insufficient to provide the expected end-to-end reliability. In contrast, SDN-TSCH is based on an accurate link quality estimation and allocates cells according to this estimation. SDN-TSCH makes over-provisioning, allocating more cells than the strict minimum. However, this safety margin compensates for the possible over-estimation, and SDN-TSCH finally provides the expected end-to-end reliability.

We finally measured the network lifetime (Figure 15), defined as the time until the first node dies due to depleted energy. MSF provides the longest network lifetime: since fewer cells are reserved, nodes have to wake up less frequently. Thus, they maximize their sleeping time, but at the cost of poor reliability. SDN-TSCH achieves a shorter but comparable lifetime and respects all the reliability and latency constraints. Obviously, ensuring strong Service Level Agreements have an impact on energy consumption. It corresponds to the price to pay for a communication infrastructure dedicated to critical applications.

VI. Conclusion & Perspectives

We have presented SDN-TSCH, a SDN solution for scheduled wireless networks with traffic isolation. The SDN controller establishes a collision-free control plane, allowing devices to reliably connect to the controller through lossy wireless links. To construct a precise network topology, we propose an accurate link quality estimation approach without generating extra control packets. The scheduler allocates cells for EBs to avoid collisions for an accurate link quality estimation. The SDN controller exploits this information to compute a compact schedule that ensures end-to-end constraints. Our simulation results demonstrated that SDN-TSCH satisfies end-to-end flow constraints while converging in a timely manner for a new node or flow admission. In addition, our solution only selects the best links to construct the forwarding paths, thanks to our link quality estimation. By contrast, SDNWISE-TSCH relies on the shortest path criterion, which includes links with low PDR, impacting the end-to-end PDR and delay. Also, our solution outperforms MSF by converging faster upon new flow admission, maintaining flow constraints, and offering improved stability in providing a stable PDR to each flow. However, it does come at the cost of higher energy consumption.

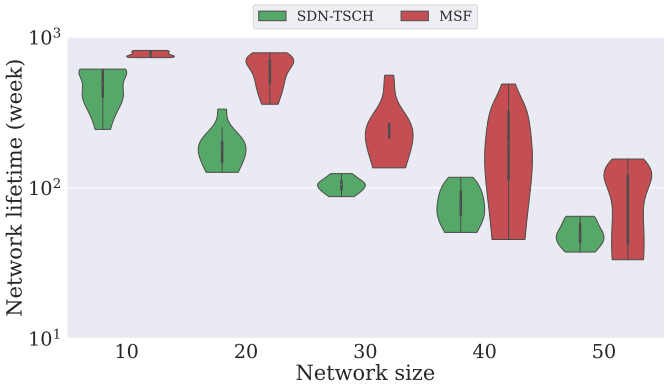

Fig. 15: Network lifetime

In our future work, we aim to enhance the controller's capabilities by enabling continuous optimization. Indeed, link qualities may change because of *e.g.*, interference. In that situation, the SDN controller has to detect changes and relocate flows and resources to accommodate these changes. We also need to propose additional mechanisms to be fault-tolerant. Multipath may help to save energy while providing high reliability. However, we need to provide statistical multiplexing to reduce the cost of energy consumption for independent backup paths.

References

[1] X. Li *et al.*, "A review of industrial wireless networks in the context of industry 4.0," *Wireless networks*, vol. 23, pp. 23–41, 2017.

[2] "IEEE Standard for Low-Rate Wireless Networks," IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015), 2020.

[3] T. Chang *et al.*, "6TiSCH Minimal Scheduling Function (MSF) ," IETF, https://tools.ietf.org/id/draft-chang-6tisch-msf-00.html, draft 00, 2017.

[4] F. Orozco-Santos *et al.*, "Enhancing sdn wise with slicing over tsch," *Sensors*, vol. 21, no. 4, 2021.
[5] E. Ancillotti, C. Vallati, R. Bruno, and E. Mingozzi, "A reinforcement learning-based link quality estimation strategy for rpl and its impact on topology management," *Computer Communications*, vol. 112, pp. 1–13, 2017.
[6] R. T. Hermeto *et al.*, "Experimental in-depth study of the dynamics of an indoor industrial low power lossy network," *Ad Hoc Networks*, vol. 93, p. 101914, 2019.
[7] F. Veisi *et al.*, "Sdn-tsch: Enabling software defined networking for scheduled wireless networks with traffic isolation," in *IEEE ISCC*, 2022.
[8] F. Theoleyre *et al.*, "Experimental validation of a distributed self-configured 6tisch with traffic isolation in low power lossy networks," in *ACM MSWiM*, 2016.
[9] Q. Wang *et al.*, "6top protocol (6p)," IETF, draft, October 2018, draft-ietf-6tisch-6top-protocol-11.
[10] N. Accettura *et al.*, "Decentralized traffic aware scheduling for multi-hop low power lossy networks in the internet of things," in *IEEE WoWMoM*, 2013.
[11] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "Ldsf: Low-latency distributed scheduling function for industrial internet of things," *IEEE internet of things journal*, vol. 7, no. 9, pp. 8688–8699, 2020.
[12] Y. Jin *et al.*, "A centralized scheduling algorithm for ieee 802.15. 4e tsch based industrial low power wireless networks," in *2016 IEEE WCNC*, 2016.
[13] M. R. Palattella *et al.*, "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.
[14] G. Gaillard *et al.*, "High-reliability scheduling in deterministic wireless multi-hop networks," in *IEEE PIMRC*, 2016.
[15] G. Li, J. Wu, J. Li, K. Wang, and T. Ye, "Service popularity-based smart resources partitioning for fog computing-enabled industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4702–4711, 2018.
[16] Y. Jarraya *et al.*, "A survey and a layered taxonomy of software-defined networking," *IEEE communications surveys & tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.
[17] L. Galluccio *et al.*, "Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *INFOCOM*, 2015.
[18] de Oliveira *et al.*, "Tinysdn: Enabling tinyos to software-defined wireless sensor networks," *XXXIV Simpósio Brasileiro de Redes de Computadores. Bahia*, pp. 1229–1237, 2016.
[19] R. C. Alves *et al.*, "It-sdn: Improved architecture for sdwsn," in *SBRC*, 2017.
[20] M. L. Miguel *et al.*, "Sdn architecture for 6lowpan wireless sensor networks," *Sensors*, vol. 18, no. 11, p. 3738, 2018.
[21] M. Baddeley *et al.*, "Isolating sdn control traffic with layer-2 slicing in 6tisch industrial iot networks," in *IEEE NFV-SDN*, 2017.
[22] T. Winter, "Routing protocol for low-power and lossy networks," IETF, rfc 6550,6551,6552, 2012.
[23] E. Municio *et al.*, "Whisper: Programmable and flexible control on industrial iot networks," *Sensors*, vol. 18, no. 11, p. 4048, 2018.
[24] H.-S. Kim *et al.*, "Challenging the ipv6 routing protocol for low-power and lossy networks (rpl): A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2502–2525, 2017.
[25] N. Baccour *et al.*, "Radio link quality estimation in wireless sensor networks: A survey," *In ACM TOSN*, vol. 8, no. 4, pp. 1–33, 2012.
[26] M. Rondinone *et al.*, "Designing a reliable and stable link quality metric for wireless sensor networks," in *Proceedings of the workshop on Real-world wireless sensor networks*, 2008, pp. 6–10.
[27] O. Iova *et al.*, "Stability and efficiency of rpl under realistic conditions in wireless sensor networks," in *IEEE PIMRC*, 2013.
[28] G. Gaillard *et al.*, "Kausa: KPI-aware Scheduling Algorithm for Multi-flow in Multi-hop IoT Networks," in *ADHOC-NOW*, 2016.

Farzad Veisi received the M.Sc. degree in telecommunications engineering from the Isfahan University of Technology, Isfahan, Iran, in 2019. He is currently pursuing his Ph.D. at the University of Strasbourg, France since 2020 and is a member of the Network research group at ICube lab. His research interests mainly focus on wireless sensor networks, end-to-end quality of services, and software-defined networks.

Julien Montavont is a senior associate prof. at the University of Strasbourg (France) since 2007 and a member of the Network research group at ICube lab. He received his Ph.D. in computer science from University of Strasbourg (France) in 2006, and was an invited researcher at Ecole Polytechnique de Montréal (Canada) in 2008. He is the co-author of more than 30 international journal and conference articles and regularly serves as TPC member or external reviewer for international journals and conferences.

Fabrice Théoleyre is a senior 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 held several visiting positions at the University of Waterloo (Canada), Inje University (South Korea), Jiaotong University (China). His research interests mainly concern distributed algorithms and experimental design for the Internet of Things.