SDN-TSCH: Enabling Software Defined Networking for Scheduled Wireless Networks with Traffic Isolation

Farzad Veisi, Julien Montavont and Fabrice Theoleyre
ICube, CNRS / University of Strasbourg, France
{veisigoshtasb,montavont,theoleyre}@unistra.fr

***Abstract*—The Industrial Internet of Things (IIoT) applications need to rely on a wireless infrastructure able to provide low end-to-end latency, and high reliability. Software-Defined Networking (SDN) is promising to make the network more agile, pushing the decision process to a controller. However, radio links are unstable while the controller needs to construct an accurate view of the network to schedule the transmissions efficiently. We propose here SDN-TSCH to separate the data and control planes for a scheduled network. We construct a reliable control plane, maintaining a collision-free path to and from the controller. Besides, SDN-TSCH guarantees flow isolation: each flow can reserve dedicated resources so that end-to-end reliability and latency constraints can be respected per flow. Finally, we also dedicate resources for best-effort traffic, to accommodate various applications. Our Cooja simulations highlight the flow isolation characteristics of SDN-TSCH: we provide very high reliability even in presence of best-effort traffic.**

***Keywords*—Industrial Internet of Things; Software-Defined Networking; scheduling; dedicated control plane; flow isolation**

I. Introduction

Industry 4.0 aims to modify the industrial processes to make them more flexible, through reconfigurable assembly lines. Cyber Physical Systems collect measurements in real-time and take proper decisions to optimize the system: inserting novel devices expands the capabilities of the system [1]. For this purpose, Industry 4.0 relies extensively on wireless transmissions to create the Industrial Internet of Things (IIoT) [2]. A large collection of sensors and actuators (aka motes) are disseminated in the environment to control, continuously retrieving measurements to take *smart* decisions. Because the devices are battery powered, the community has redesigned energy efficient protocols to use scarcely the resources, turning off the motes most of the time.

However, wireless networks are known to be lossy: depending on channel conditions, a packet may or may not be decoded by the receiver. The link quality is even time-variant: external interference may arise, which negatively impacts the reliability if the Signal To Noise Ratio (SNR) is too low. In these conditions, providing high reliability and low delays, as required by most industrial applications, is particularly challenging.

To cope with these constraints, deterministic Medium Access Control (MAC) protocols have been proposed in the literature. Typically, IEEE 802.15.4-TSCH [3] relies on a strict schedule of the transmissions to avoid collisions. The scheduling matrix defines, for each timeslot, if a device has to stay awake, and on which channel (frequency) to listen/transmit a frame. IEEE 802.15.4-TSCH supports both centralized and distributed scheduling algorithms [4].

In recent years a novel paradigm emerged based on Software-Defined Networking (SDN) [5]. The approach clearly separates the control and data planes so that all decisions can be centralized in a controller. To our mind, such centralization has two major assets: i) the network devices are simpler, and just enforce rules provided by the controller, ii) with a complete view of the network conditions, the controller makes optimal decisions.

OpenRadio [6] already advocates for a more agile wireless stack, with a processing and a decision plane for cellular networks. SDN-WISE [7] introduces a pioneering piece of work to adapt the SDN paradigm to the Internet of Things. To our mind, an SDN solution should focus on connectivity, and must support scheduled networks. Data processing and in-network aggregation may be achieved rather by Network Function Virtualization (NFV) [8] to maintain layer independence. We have also to address the unreliability problem in the control plane: a command or report packet may be lost between the controller and network devices. Such unreliability should not impact the convergence of the network.

The contributions of this article are as follows:

1) we propose mechanisms to support a control plane in scheduled networks. In particular, we provide a join process so that a device can discover neighbors and contact the controller. In return, the controller reconfigures the control plane for the novel device;
2) we provide a compact resource allocation for the control plane, which is still collision-free while limiting the number of cells to allocate so that a device has both upward and downward routes to reach the controller;
3) we detail how to reserve resources in the data plane for critical flows, while still guaranteeing flow isolation;
4) we evaluate the performance of our SDN-TSCH solution in Cooja, and compare it against Orchestra [9], and highlight how it can respect per flow guarantees (reliability and latency).

II. Background & Related Work

We detail here background notions and related work on scheduled low power networks, and SDN.

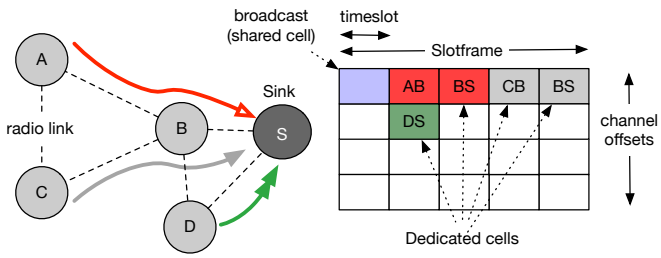

Fig. 1: Simple TSCH schedule with shared and dedicated cells

A. IEEE 802.15.4-TSCH

IEEE 802.15.4 TSCH [3] targets industrial wireless networks. Slow channel hopping combats external interference: if a packet cannot be decoded, it is retransmitted through another channel. Besides, IEEE 802.15.4-TSCH relies on scheduled access: a scheduling matrix (a slotframe) is composed of cells (pairs of timeslots and channel offsets). Since the slotframe repeats over time, a transmitter that is assigned to a cell, has a reserved bandwidth for its transmissions. The Absolute Sequence Number (ASN), defines a global clock in the network and corresponds to the number of timeslots since the sink has bootstrapped. More precisely, the standard supports two types of cells in the scheduling matrix:

shared cells implement a contention resolution for acknowledged packets. More precisely, if the transmitter does not receive an `ack`, it skips a random number of shared cells (corresponding to its backoff value) before retransmitting the same packet. Hopefully, different transmitters will choose different backoff values. However, the backoff is among and not inside the cells, and is thus expensive in bandwidth because collisions are frequent for medium densities [10]. Typically, the shared cell in Fig. 1 may be used for non critical broadcast traffic such as Enhanced Beacons (EBs);

dedicated cells have no contention resolution. Thus, a dedicated cell should be allocated to non interfering transmissions to avoid collisions. In particular, different links may use different channel offsets in the same timeslot to multiplex the transmissions across different channels, increasing the network capacity.

TSCH supports both centralized and distributed scheduling algorithms [4]. A centralized scheduler may construct non-colliding schedules since it has a complete knowledge of all the transmissions. Distributed approaches need to detect and correct collisions. For flow isolation, the scheduler must allocate different cells for the different flows. In Figure 1, the flows from A (in red) and C (in gray) use different cells when they are transmitted through the link $B \rightarrow S$.

Orchestra [9] is a very efficient solution to construct the IEEE 802.15.4-TSCH schedule in a distribued manner. Orchestra relies on RPL [11] to construct a route toward the sink. Once RPL has converged, a device derives the cells used by each of its neighbors, using their ID. More precisely, a pseudo-random function derives a timeslot and channel offset from the ID, for each slotframe. Cells can be shared, receiver-based, or sender-based. The two first categories require contention resolution as multiple transmitters can use those timeslots. Orchestra relies on three different slotframes, to handle Enhanced Beacons, RPL packets, or data packets. However, the number of slotframes, the size of each slotframe and slot types should be hardcoded in devices before running the network.

B. SDN paradigm

Software-Defined Networking (SDN) [5] has been very popular to make the network more agile. SDN removes the network intelligence from the network devices by separating the forwarding process of network packets (data plane) from the routing process (control plane). One or multiple controllers centralize the network intelligence and form the control plane while the network devices form the data plane. The controller is provided with a complete view of the network and can therefore make optimal decisions on traffic forwarding. Typically, the network devices start with no knowledge and ask the controller how to handle the received packets. The controller then enforces forwarding rules on network devices that match specific packets of flows, *e.g.*, drop the packet or forward it through a specified output queue if the flow-id is equal to i.

Wireless networks are unreliable: the control plane needs mechanisms to measure the link quality, to provision enough retransmission opportunities, etc. Besides, many nodes are battery-powered, and the network must minimize the amount of control (and data) packets to forward. SDN-WISE focuses on in-network aggregation [7]. Stateful tables in each node enforce rules to apply for each received packet, based on the reading of the packets and their content. A node may be configured for instance to forward a packet from node B, if the last temperature value forwarded from the node C exceeds a threshold value. While SDN-WISE pushes the computation as close as possible to the data producers, it needs to read the packets and their payload, with fixed size headers.

Industrial wireless sensor networks rely on scheduling to avoid collisions and to respect end-to-end guarantees. Orozco-Santos *et al.* [12] enhanced SDN-WISE to support scheduled networks. The objective is to construct a schedule, such that a deadline can be respected, specific to each flow, whenever a packet is generated. The controller computes both the routes and the schedule. It applies a Dijkstra strategy to compute shortest routes, based on a *pressure* metric. This pressure is typically proportional to the amount of traffic that each node has to forward. Then, the controller schedules first the flows with most stringent deadlines. They extend then this scheme to support multicast traffic [13]. However, the control plane is still unreliable since it relies on shared cells, that are used for Enhanced Beacons, report packets, and commands from/to the controller. Shared cells are known to be very lossy, particularly for bursty traffic [10], *e.g.*, when a network reconfiguration is required.

uSDN [14] implements the SDN concept in the 6TiSCH stack. Relying on RPL and a distributed scheduling function, each node reserves a dedicated track toward the controller to maintain the control plane. The Control plane maintenance

does not rely on the controller. However, we are convinced that the controller should allocate resources for the control plane: which route(s) to use, and how many cells to reserve.

III. SDN-TSCH

We propose here a novel SDN scheme tailored for industrial wireless sensors networks. SDN-TSCH constructs a wireless infrastructure able to:

- respect per flow guarantees (minimum end-to-end Packet Delivery Ratio, and maximum end-to-end latency) for critical flows;
- operate on top of a scheduled MAC layer (here IEEE 802.15.4-TSCH), separating clearly the control and the data planes, with dedicated radio resources;
- set-up a reliable control plane: we exploit dedicated (non colliding) cells, with a collision free path to and from the controller for each network device. We propose an efficient way to exploit a single cell from one node to all its children, to make the schedule more compact and to save energy;
- exploit label switching, so that a packet is forwarded transparently to the destination. The controller keeps the full control, and can establish a specific path for a given flow, with dedicated resources to guarantee flow isolation;
- configure both the control and the data planes when a novel device has to be admitted.

We detail now the operations of SDN-TSCH to enable SDN in an industrial wireless sensor network.

A. Label switching

To simplify the behavior of all the nodes, we implement a label switching approach, that relies on a flow-id piggybacked in the SDN headers of any packet. Each enqueued packet has a flow-id, that defines the path(s) to the destination. Indeed, a flow-id is defined by the controller, that maintains also a flow-id table per node. Thus, at the beginning of a TX cell, the transmitter extracts the corresponding flow-id from the flow-id table, and dequeues the first packet in the queue that corresponds.

A specific flow-id exists for the control traffic destined to the controller (*"to controller"*). Here, the radio resources are reserved by the controller for all the convergecast paths. Symmetrically, another flow-id exists for the packets generated by the controller, to any node in the network (*"from controller"*). We will explain in section III-D how such packet reaches its unicast destination.

B. Discovery process

Let us assume here that the network is configured (control and data planes):

1) the node is synchronized, and has an installed schedule;
2) the control plane is configured, and each device has dedicated cells to and from the controller. Thus, hop-by-hop, a path to and from the controller exists, with dedicated resources.

A novel node has to trigger a discovery process to identify an already attached neighbor. We reuse here the classical Enhanced Beacons (EB), sent periodically by each node. These EBs have two purpose:

1) synchronization for the unattached nodes, that can adjust their clock to follow the schedule of the network;
2) configuration which is piggybacked in an Information Element (IE). This IE allows the receiver to know the Absolute Sequence Number (ASN), the slotframe length, the number of shared cells, etc.

We keep also the default behavior of IEEE 802.15.4-TSCH where EB are transmitted through shared cells, to which all the nodes have to listen to. To reduce the latency and the collisions, we distribute uniformly the shared cells in the slotframe [15].

A node maintains a list of neighbors, and counts the number of received EB from each neighbor as a link quality indicator. Just a counter is maintained: since the transmissions are periodic, the associated Packet Delivery Ratio can be derived safely later by the controller from the counter.

C. Join process

A novel node has to announce its presence to the controller. In return, the controller will configure the control plane to admit the novel node.

An unattached node collects continuously all the EBs received during the discovery process. In particular, a node should not stop the discovery as soon as an EB is received: another neighbor may constitute a better choice. While the controller will detect it later, a reconfiguration is expensive and useless.

When a sufficient time has elapsed, the novel node constructs a `report` packet in which it includes the list of its neighbors, as well as the number of EB received since the beginning for each of the neighbors (*i.e.*, the EB counter). The novel node adopts a hot potato strategy: it can use any shared cell to send the `report` to any of its neighbors. The transmission is in unicast, and the novel node waits for an `ack`. It is worth noting that the controller may typically choose for the novel node a routing parent different from this neighbor (cf. Figure 2). Thus, we do not constrain the routes with this pragmatic strategy.

An attached node that receives a `report` has to forward it to the controller. Fortunately, its control plane has already been configured, and a path exists with dedicated cells to the controller. Hop-by-hop, the `report` packet will reach the sink, that will forward the SDN packet to the controller. Since the path is collision free, we expect a reliable control plane, through the link layer retransmissions. Our simulations validate this assumption.

A node continuously monitors the reception of EB and updates the EB counter, even after having joined the network. Besides, it keeps on sending periodical `report` packets to the controller, so that the controller can detect link quality changes.

Figure 2 illustrates a simple scenario in which node A is a novel node while the rest of the topology is already

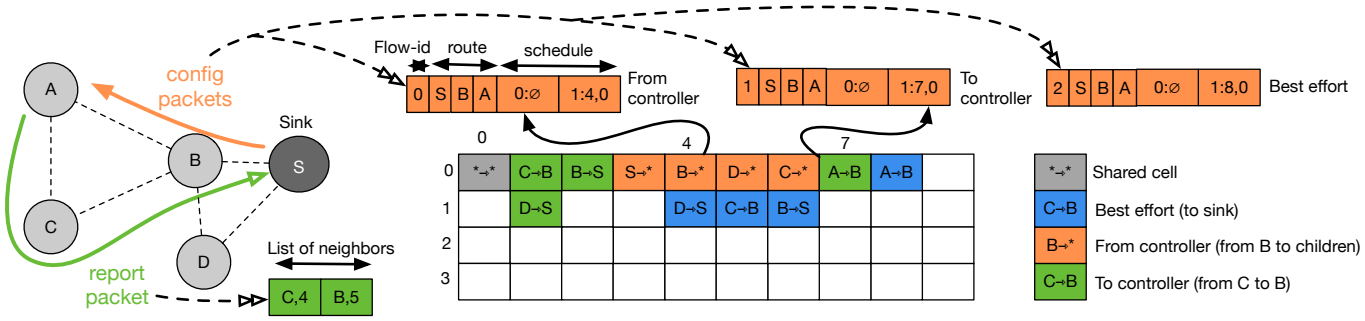

Fig. 2: Join process for a novel node

configured. Just after the discovery process, node A sends a `report` packet to C through one shared cell. C receives the `report`, and needs to forward it. C is already configured, and has already a dedicated cell (for the flow-id *to controller*). Hop-by-hop, the report packet is received by S.

D. Admission and configuration of a novel node

When the controller receives a `report` packet, it searches the source in the list of already configured nodes. If it is not present in this list, it corresponds to a novel node, that needs to be configured. The controller extracts first the list of neighbors and their EB number from the `report` packet. Then, the controller identifies the best neighbor to serve as parent. To maximize the reliability of the control plane, the controller selects the neighbor with the highest counter of EB. This neighbor presents the highest reliability and maximizes the probability to receive later the `report` packets from the node.

The controller selects also an available dedicated cell in the scheduling matrix from and to the controller for this novel node. A timeslot is candidate if the timeslot is unused by the best neighbor (half duplex condition). In that case, a random channel offset is selected, while verifying the same cell is not given to an interfering transmitter. The controller has not to verify that the cell is also unused by the novel node: its schedule is by definition empty (except the shared cells).

Then, the controller prepares a `config` packet, that is composed of:

seqnum specifies the sequence number of config packet. It helps to handle timeouts and retransmissions of `config` packets;

client-req-id identifies the application that asked for the admission of a novel flow. Thus, a single node may support several critical flows;

Cycle indicates the period of the schedule in the `config` packet. More precisely, the given timeslot is reserved every `cycle` timeslots;

flow-id identifies the flow in each intermediary hop for label switching. It is unique in the network;

SFid identifies the slotframe ID;

route (L_{route})**:** the list of addresses of each node in the path toward the novel node, that the `config` packet must follow. We implement source routing to give the full control to the controller. Each node in the path that receives the `config` packet extracts its position in the route and finds the next-hop address to which the `config` packet has to be forwarded;

schedule ($L_{schedule}$)**:** the list of timeslots and channel offsets that correspond to TX cells. For a novel node, only the two last hops of the route (*i.e.*, the last link) have to modify their schedule: the rest of the nodes just forward the `config` packet. However, the format is sufficiently generic to accommodate more complex situations (cf. section III-F).
More precisely, the schedule is encoded into a list of `<number_of_cells,list_of_cells>`. It is worth noting that $L_{route} - 1 = L_{schedule}$. Indeed, the route comprises the destination, that has no TX cell to install.

A node that receives a `config` packet will read the headers. It processes the configuration part and installs the cells and flow-id in the scheduling and flow-id tables respectively. Let us consider that the node identifies that it is the i^{th} hop in the route. It executes the following actions:

1) it then extracts the cells for the i^{th} element in the schedule, and install them as TX cells, with the flow-id contained in the header of the `config` packet;
2) it extracts the TX cells for the $i + 1^{th}$ element in the schedule (if it exists). The cells are installed as RX cells;
3) it extracts the $i + 1^{th}$ id in the route, and replaces the link layer destination address in the frame.

Such `config` packet can then be used unchanged to configure the data plane, as explained below in section III-F.

Let us continue our example illustrated in Figure 2. The controller that received the `report` packet from A needs to configure the network for the novel node. The controller selects B as the parent of node A (to reach the controller in the control plane). It generates two config packets:

from controller: the route to follow is (S,B,A). Besides, the controller piggybacks the schedule in the download direction (to children): the timeslot 4 (channel offset 0) has to be installed in TX mode by the node B, and in RX mode by the node A. The flow-id corresponds here to 0 (=*"from controller"*);

to controller: the `config` packet follows the same route, but piggybacks the schedule for upload, with the timeslot 7, and channel offset 0. The flow-id corresponds here to 1 (=*"to controller"*).

As one can note, we use a single cell for one node to all its children, to make our schedule more compact, and to save energy. Indeed, all the children are in RX mode, and will receive the `config` packet. However, only the child with the correct link-layer destination address will process it, other children will drop the packet. Over-listening is counter balanced by the more compact schedule for the control plane brought by this children mutualization.

Since the novel node is not yet configured, the last hop is handled as a specific case. The node which is the before last id in the route will use one shared cell to forward the `config` packet to the novel node. Indeed, no dedicated cell exists yet to/from the novel node. However, the load is very low (only for novel nodes), and the next updates will use the novel dedicated control plane cells installed by the novel node.

In Figure 2, node B receives the `config` packet *from controller*. It extracts the item in the schedule that corresponds to its TX cells (*i.e.*, the first one). Here, the timeslot 4 is already present in its schedule (it corresponds to the *to children / from controller case*), and B skips it. It then extracts the TX cells corresponding to the next hop: none exists, and the packet is then forwarded to the next hop (A in the route).

After the two `config` packets have been received by the node A, the control plane is configured, and a collision-free path exists from and to the controller. A can start sending Enhanced Beacons for unattached neighbors.

E. Best Effort Traffic

We propose a best-effort strategy in which right after joining the control plane, each device receives cells for its best-effort traffic toward the sink. More precisely, the controller generates a novel `config` packet to reserve BF_TS cells per slotframe from the novel node to its parent. No collision can occur since one single transmitter is active during this cell. However, all the best-effort flows use the same cells, and data packets may be dropped because of a buffer overflow.

In Figure 2, the controller would send a novel `config` packet to announce the best-effort cell (timeslot 8, channel offset 0) for the link $A \rightarrow B$.

F. Admission of a novel critical flow

SDN-TSCH supports also flow isolation. Each critical application has specific Quality of Service (QoS) requirements, and defines its minimum end-to-end reliability and maximum end-to-end latency. When the application opens an UDP connection, the transport layer asks the controller for a novel reservation, with the application profile. The SDN layer of the source creates a `flow-request` packet and sends it to the controller through the control plane, using the *to controller* flow-id.

The `flow-request` contains i) the socket address, ii) the traffic profile. Thus, the controller knows exactly the requirements, and can compute a schedule to reserve enough bandwidth for the novel flow:

- it provisions additional cells along the path to respect the end-to-end reliability;
- it tries to schedule the cells back-to-back (the forwarding cell after the last retransmission cell of the previous hop) to minimize the end-to-end latency.

In our solution, one `config` packet is enough to configure a new flow-id and the schedule for the whole path. We use source routing, such that the route contains two parts:

1) from the sink to the destination: the schedule of this part is empty (no cell has to be reserved);
2) from the destination to the source: each hop has a certain number of cells in the schedule part of the `config` packet. Since the packet is then routed from the destination to the source, the source can be sure that everything has been configured when it receives the `config` packet. It can safely start transmitting the data packets corresponding to the novel application.

TABLE I: Simulation parameters

Platform	OS: Contiki-ng Simulator: Cooja
Common parameters	Network sizes: 5 or 10 or 15 Number of critical nodes: 3 Traffic pattern: Convergecast Critical traffic: Constant Bitrate, 1 packet every 5s Best-effort traffic: Poisson, on average 1 packet every 5s TSCH EB period: 10s
SDN-TSCH	Slotframe length: 509 Number of best-effort cells (BF_TS): 1 or 5
Orchestra	EB Slotframe len: 397 RPL Slotframe len: 31 Data Slotframe len: 17, 53, 151 RPL: storing mode

IV. Performance Evaluation

To assess the performance of SDN-TSCH, we simulated SDN-TSCH and Orchestra [9], which represents a state-of-the-art distributed scheduler for IEEE 802.15.4-TSCH.

A. Evaluation setup

We use the Contiki-ng operating system and the Cooja simulator to implement SDN-TSCH. We compare Orchestra [9] and SDN-TSCH. We simulate networks with 5, 10, or 15 nodes, and a convergecast traffic pattern. We simulate two types of flows:

critical: the controller reserves a flow-id and a set of cells for a single critical flow. Three nodes are selected randomly in the topology to generate one critical flow. Critical applications generate a data packet every 5 s and need a 99% end-to-end Packet Delivery Ratio;

best-effort applications generate Poisson traffic with an average of one packet every 5 seconds. The best-effort traffic mimics for instance event-triggered flows. All the nodes not selected for the critical flows generate one best-effort flow.

Orchestra is run with different application slotframe sizes to quantify the cost of high PDR against the energy consumption.

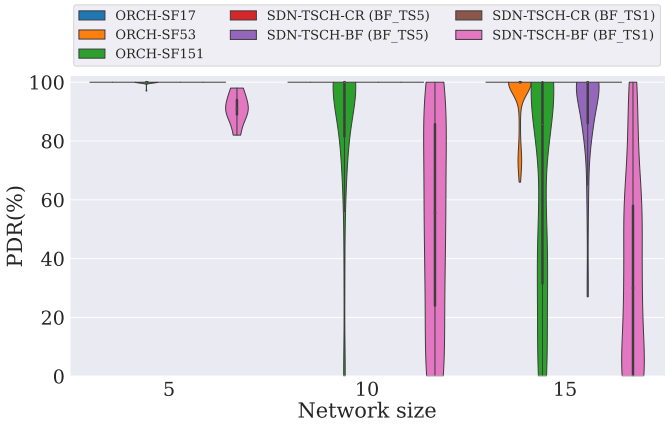

Fig. 3: End-to-end Packet Delivery Ratio per Flow

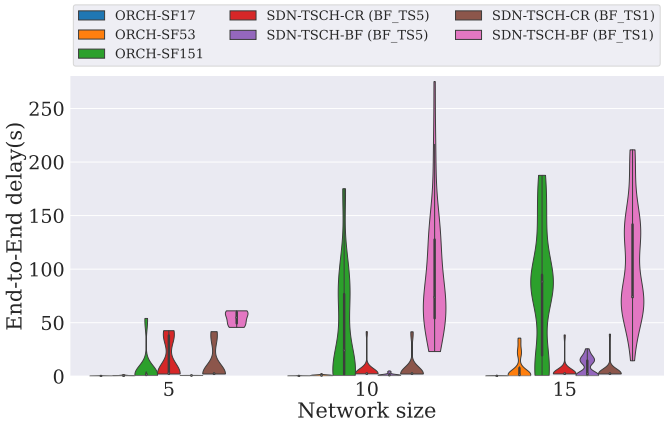

Fig. 4: End-to-end latency per flow

Similarly, we consider in SDN-TSCH two values for BF_TS (number of best effort-cells from a node toward the sink in each slotframe): 1 and 5. We keep the default slotframe sizes for the RPL and EB slotframes in Orchestra, as defined in [9]. Orchestra does not support flow isolation, and the same resources are used for best-effort and critical flows. Table I regroups the different values of our parameters.

We use a simple greedy scheduling algorithm, to focus on the SDN-TSCH architecture, and not on the scheduling algorithm itself. Indeed, many centralized algorithms exist in the literature and may be adapted to SDN-TSCH. To meet the required end-to-end PDR, the scheduler

1) allocates greedily more cells (for retransmissions) to the weakest link along the flow path. It stops the process when the end-to-end PDR constraint is respected: it obtains a number of cells for each hop [16].
2) reserves greedily cells from the scheduling matrix for each hop, respecting the number of cells per hop. To minimize the end-to-end delay, the scheduler allocates the first timeslots available after the last timeslot of the previous hop.

B. Results and comments

We first measure the end-to-end Packet Delivery Ratio (PDR). More precisely, we measure the end-to-end PDR for each flow at the end of each simulation. Then, Figure 3 represents the distribution of the PDR for all the flows, for different network sizes. For SDN-TSCH, we report separately the per-flow average PDR for critical flows and for best-effort flows. SDN-TSCH respects the constraints for the critical flows: their PDR is equal to 100%, whatever the conditions. On the contrary, best-effort flows may be a bottleneck when an insufficient number of best-effort cells is reserved. However, we keep on providing an average PDR of 100% with 15 nodes, while a few flows may exhibit a lower PDR.

Orchestra is run with the same traffic profile. However, It is not able to respect per flow guarantees, and thus the PDR of critical and best effort flows is the same. Besides, the slotframe size should be sufficiently small to give enough transmission

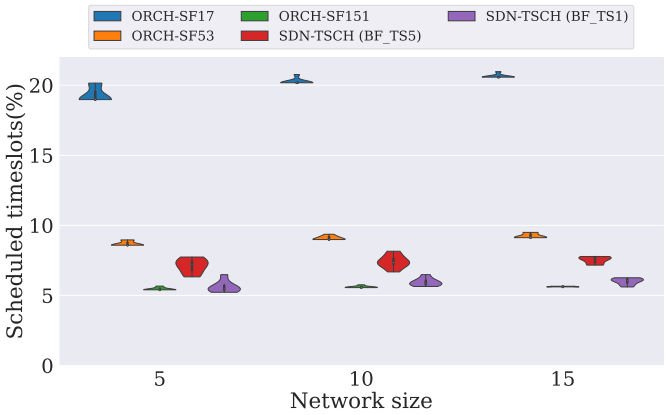

Fig. 5: Number of cells scheduled per node

opportunities. When Orchestra operates with a similar number of cells per slotframe as SDN-TSCH (slotframe length of 151 for orchestra, and 5 best effort cells in a slotframe length of 509 in SDN-TSCH), Orchestra provides a lower PDR than SDN-TSCH for larger networks. For 15 nodes, the PDR is equal on average to 87% for Orchestra vs. 100% for SDN-TSCH. Orchestra should operate with a smaller slotframe length, which consumes obviously more energy.

Then , Figure 4 illustrates the distribution of the end-to-end latency. The latency remains ultra-small whatever the network size with SDN-TSCH. A centralized scheduling algorithm, combined with a reliable control plane is particularly efficient. The latency tends to increase for Orchestra. It is worth noting that for 15 nodes, the latency is significantly higher for Orchestra than for best-effort flows in SDN-TSCH, even when a single best-effort cell (BF_TS) is reserved per slotframe with SDN-TSCH. The funneling effect [17] is very detrimental to Orchestra with much more collisions, and longer queuing delays.

Finally, we measure the number of scheduled cells per node (Figure 5). Indeed, a node which is active in a larger number of cells has to wake-up more frequently, and will consume more energy. Besides, the network capacity is also

reduced if a node has on average more active cells. Because it is not traffic aware, Orchestra has a fixed number of cells per node, whatever the conditions. Thus, to provide high reliability, Orchestra needs to operate with short slotframes, which has a negative impact on the energy consumption. SDN-TSCH is much more flexible and is able to adapt to the conditions: the energy is reduced for only best-effort traffic, while maintaining a reasonable energy consumption to support critical applications, with flow isolation.

V. Conclusion & Perspectives

We have presented here SDN-TSCH, able to efficiently implement a SDN architecture in a scheduled, industrial wireless sensor network. Each device is admitted in the network, and the controller configures the control plane to maintain a collision-free path from and to the controller. The controller also collects the report packets to maintain a consistent view of the link qualities, and to construct a schedule which respects the end-to-end guarantees. The controller has a full control on the network, and can reserve dedicated resources for critical vs. best-effort flows. Our performance evaluation demonstrates the ability of SDN-TSCH to support flow isolation, with a small number of active cells, and thus, a lower energy consumption.

In a future work, we plan to extend the controller to enable continuous optimization, to save energy when the network characteristics change (*e.g.*, volume of traffic, link quality). In particular, we expect to evaluate the performance of different routes and schedules on the energy consumption as well as the fault tolerance. Our architecture also naturally supports multipath, since a given flow-id may be associated with multiple TX cells, to different neighbors. Thus, we expect to investigate the fault-tolerance properties of our solution to provide very high reliability even in presence of faults.

Acknowledgment

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

References

[1] Xiaomin Li, Di Li, Jiafu Wan, Athanasios V. Vasilakos, Chin-Feng Lai, and Shiyong Wang. A review of industrial wireless networks in the context of industry 4.0. *Wireless Networks*, 23(1):23–41, Jan 2017.

[2] M. Wollschlaeger, T. Sauter, and J. Jasperneite. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27, March 2017.

[3] Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pages 1–800, 2020.

[4] Rodrigo Teles Hermeto, Antoine Gallais, and Fabrice Theoleyre. Scheduling for IEEE802.15.4-TSCH and Slow Channel Hopping MAC in Low Power Industrial Wireless Networks. *Comput. Commun.*, 114(C):84–105, December 2017.

[5] Murat Karakus and Arjan Durresi. A survey: Control plane scalability issues and approaches in software-defined networking (sdn). *Computer Networks*, 112:279–293, 2017.

[6] Manu Bansal, Jeffrey Mehlman, Sachin Katti, and Philip Levis. Openradio: A programmable wireless dataplane. In *HotSDN*, pages 109–114, 2012.

[7] Laura Galluccio, Sebastiano Milardo, Giacomo Morabito, and Sergio Palazzo. Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks. In *INFOCOM*, 2015.

[8] Mike Ojo, Davide Adami, and Stefano Giordano. A sdn-iot architecture with nfv implementation. In *IEEE Globecom Workshops*, pages 1–6, 2016.

[9] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In *SenSys*, pages 337–350. ACM, 2015.

[10] Fabrice Theoleyre and Georgios Z. Papadopoulos. Experimental validation of a distributed self-configured 6tisch with traffic isolation in low power lossy networks. In *MSWiM*, pages 102–110. ACM, 2016.

[11] T. Winter. Routing protocol for low-power and lossy networks. rfc 6550,6551,6552, IETF, 2012.

[12] Federico Orozco-Santos, Víctor Sempere-Payá, Teresa Albero-Albero, and Javier Silvestre-Blanes. Enhancing sdn wise with slicing over tsch. *Sensors*, 21(4), 2021.

[13] Federico Orozco-Santos, Víctor Sempere-Payá, Javier Silvestre-Blanes, and Teresa Albero-Albero. Multicast scheduling in sdn wise to support mobile nodes in industrial wireless sensor networks. *IEEE Access*, 9:141651–141666, 2021.

[14] Michael Baddeley, Reza Nejabati, George Oikonomou, Sedat Gormus, Mahesh Sooriyabandara, and Dimitra Simeonidou. Isolating sdn control traffic with layer-2 slicing in 6tisch industrial iot networks. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 247–251. IEEE, 2017.

[15] Rodrigo Teles Hermeto, Antoine Gallais, and Fabrice Theoleyre. Experimental in-depth study of the dynamics of an indoor industrial low power lossy network. *Ad Hoc Networks*, 93:101914, 2019.

[16] Guillaume Gaillard, Dominique Barthel, Fabrice Theoleyre, and Fabrice Valois. Kausa: KPI-aware Scheduling Algorithm for Multi-flow in Multi-hop IoT Networks. In *ADHOC-NOW*, pages 47–61, 2016.

[17] Hongyan Xin and Xuxun Liu. Energy-balanced transmission with accurate distances for strip-based wireless sensor networks. *IEEE Access*, 5:16193–16204, 2017.