Link Quality Estimation in Wireless Software Defined Network with a Reliable Control Plane

Farzad Veisi Goshtasb, Julien Montavont, Fabrice Theoleyre

Link Quality Estimation in Wireless Software Defined Network with a Reliable Control Plane

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*—The Industrial Internet of Things (IIoT) has now emerged for many industrial applications. However, many critical applications require high reliability and bounded end-to-end latency. Fortunately, scheduled wireless networks such as IEEE 802.15.4-TSCH try to orchestrate finely the transmissions. In particular, the Software-Defined Networking (SDN) paradigm is promising to concentrate the intelligence of the network in a controller and to simplify the functions achieved by the motes. SDN controller relies on link quality information of wireless devices to construct the routing topology of the network and to provision adequate radio resources for each link. We propose here an accurate link quality estimation scheme based on the SDN-TSCH, as an SDN solution for IEEE 802.15.4-TSCH network. We exploit the centralized view of the SDN controller to transmit packets of link quality estimation in a collision-free manner. Moreover, we investigate the performance of the control plane of SDN-TSCH through different shared, dedicated, and hybrid (mix of shared and dedicated cells) approaches. We figured out that the dedicated control plane provides high reliability and fast convergence for wireless SDN networks.**

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

I. Introduction

Industrial Internet of Things (IIoT) is a base enabler for Industry 4.0 to automate the industrial processes [1]. IIoT relies on wireless communication, which provides a high level of flexibility and scalability for industrial applications. A large number of devices construct a network and transmit their measurements to a gateway through multi-hop wireless links.

Channel conditions and interference can affect the reliability of wireless links, and consequently the Quality of Service (QoS) of applications. Wireless links are known to be lossy and time-variant: by changing the channel conditions, a link may not provide good quality anymore and the receiver does not receive all the packets correctly. Moreover, interference happens between concurrent transmissions of the same radio frequency and fails the transmissions.

Deterministic Medium Access Control (MAC) protocols have been proposed to cope with these constraints and to provide reliability. IEEE 802.15.4e-Time Slotted Channel Hopping (TSCH) [2] proposes a TDMA-based frequency hopping mechanism to avoid the collision. A scheduling matrix of timeslots and channel offset determines the transmission opportunity of each link. By staying in the same timeslot, the frequency channel changes for the next transmission of a link. IEEE 802.15.4-TSCH supports both centralized and distributed scheduling algorithms.

Software Defined Network (SDN) paradigm proposes a high potential to manage industrial networks [3]. By centralizing the intelligence part of the network and providing a global view, the SDN controller can perfectly define flow rules and manage the network efficiency. Particularly in scheduled industrial wireless networks, exploiting SDN concept helps to fulfill the flow guarantee of critical applications. SDN-TSCH [4] leverages the SDN concept for IEEE 802.15.4-TSCH network to: i) construct a dedicated control plane, coping with collision and unreliability of wireless links. ii) meet the QoS requirements of critical applications by flow isolating and per-flow resource provisioning.

SDN controller relies on local link quality measurements of nodes to build the network topology and allocate sufficient radio resources for weak links. Accuracy of link estimation has a severe impact on bandwidth and energy of the network.

Also, devices are energy-constrained in IIoT networks, and maintaining a fully dedicated control plane proposes a high cost of energy.

The contributions of this article are as follows:

1) we propose an approach to enhance the accuracy of link quality estimation in the SDN-TSCH. In particular, we delegate the control of Enhanced Beacon (EB) transmissions to an SDN controller. A controller schedules EB packets to provide a collision-free link estimation for nodes.
2) we separate the shared cells of EB transmissions from the shared cells of the rest of the broadcast traffic to avoid impacting each other.
3) we evaluate the performance of shared and dedicated control planes in SDN-TSCH network to quantify the impact of realistic conditions (*i.e.*, lossy links) on the SDN network, which (to the best of our knowledge) has never been addressed in scheduled wireless networks. We illustrate how a fully dedicated control plane provides high-reliability to maintain the network consistent.

II. Related Works

Software Defined Networking (SDN) [5] is a promising paradigm to make the network more agile. SDN decouples the intelligence part (control plane) of network devices from the forwarding part (data plane). It centralizes the network

intelligence in the SDN controller, which has a complete view of the network. The SDN controller uses a so-called southbound API (*e.g.*, OpenFlow) to communicate with the data plane and install rules on forwarding devices. The node has just to apply rules in its flow table to forward any packet. Many propositions have been done to adapt the SDN for Wireless Sensor Networks (WSNs), and delegate processing tasks from constrained devices to the SDN controller.

The SDN controller relies on the local information gathered from nodes to construct a representation of the network (as a graph) and to make decisions. As a result, link quality estimation is of crucial importance to allow a controller to make relevant decisions. Link quality estimation is widely investigated in the literature [6]. Link quality can mainly be estimated based on [7]:

- physical layer metrics including Link Quality Indicator (LQI), Signal to Noise Ratio (SNR), and Received Signal Strength Indicator (RSSI);
- link-layer information such as Packet Reception Rate (PRR).

In SDN-WISE [8], the RSSI of the last beacon packet is used as the link quality metric. The RSSI value is later used as edge weight in the topology graph of the network. However, RSSI is an inadequate metric in estimating the quality of links, and a high RSSI does not automatically imply a high packet reception ratio [9].

TinySDN [10] relies on probing packets to estimate the link quality. Each node broadcast probing packets and waits for the responses from neighbors to calculate the Expected Transmission Count (ETX) for each link. ETX takes into account link asymmetry and provides high-throughput routes. However, sending probing packet increase the communication overhead. Also, in overloaded networks, many nodes fail to compute the ETX because they cannot receive packets.

SDN-TSCH [4], without defining a new probing packet, leverages Enhanced Beacons (EBs) of the TSCH network to estimate the link quality [11]. Each node counts the number of received EBs from each neighbor in a given time window and provides this information to a controller. A controller computes the Packet Receiving Rate (PRR) of EBs for each neighbor as a link quality metric. The periodically sending nature of EBs lets nodes continuously perform link estimation. However, EB packets are handled on shared cells, so they may experience collisions resulting in a potential underestimation of the link quality.

III. Background

We first detail background notions on IEEE 802.15.4-TSCH, a scheduled protocol for low-power networks. Then, we present SDN-TSCH which is an SDN scheme for IEEE 802.15.4-TSCH network.

A. IEEE 802.15.4-TSCH

IEEE 802.15.4-TSCH [2] is a mode of the IEEE 802.15.4 standard which targets industrial applications. The approach exploits a frequency-time division multiple access (FTDMA) MAC layer, with a channel hopping mechanism. Transmissions rely on an FTDMA matrix, that defines when a transmitter can start a transmission, and which channel it has to use. FTDMA helps to combat *internal* interference: when two transmitters may interfere, the network schedules their transmissions at different instants or frequencies. Frequency hopping mitigates multipath fading and *external* interference. Indeed, other networks may be colocated, using the same unlicensed band, adding noise to some frequencies. Frequency hopping helps to reduce the probability of repetitive packet losses: link-layer retransmissions will use a different frequency.

TSCH relies on a slotframe that is repeated over time. A slotframe can be represented as a scheduling matrix composed of cells (pairs of timeslots and channel offsets). The standard defines two types of cells in the scheduling matrix:

a dedicated cell is allocated to one transmitter. The transmitter waits for a fixed offset from the beginning of the cell to accommodate clock drifts, and then sends its frame. If the same dedicated cell on the same channel offset is assigned to interfering transmitters, collisions will be repetitive.

a shared cell can be allocated to more than one transmitter, so collision may happen between two concurrent transmissions. Thus, a contention resolution mechanism is applied for acknowledged packets. The transmitter starts the TX in the next shared cell and waits for an `ack`. If no `ack` is received, it waits for a random number of shared cells to retransmit the same packet. Thus, collisions may be quite frequent in shared cells [12].

The standard lets the scheduling mechanism unspecified. The scheduling algorithm may be distributed as well as centralized [13]. In distributed scheduling, each node executes an algorithm to dynamically modify its local schedule. Versus, in centralized scheduling, an algorithm is executed by a centralized controller such as a Path Computation Element (PCE) [14] that has complete knowledge of the network topology, as well as the traffic needs.

While many centralized scheduling algorithms have been proposed, they have been mostly evaluated with Monte-Carlo simulations, assuming that all the inputs are known accurately. However, collecting inputs and pushing scheduling decisions are challenging in wireless low-power networks: transmissions are unreliable, and the multi-hop nature of the network complexifies the control plane deployment.

B. Overview of SDN-TSCH

In our previous work [4], we proposed SDN-TSCH, an SDN architecture for industrial wireless sensor networks. The SDN controller performs the flow isolation in the network and allocates sufficient resources for the flows with respecting their QoS requirements.

1) Label switching for SDN: the SDN controller defines a flow-id for each flow and populates the flow tables accordingly. More precisely, each TX cell is labeled by a flow-id, and at the beginning of the TX cell, a node picks the first

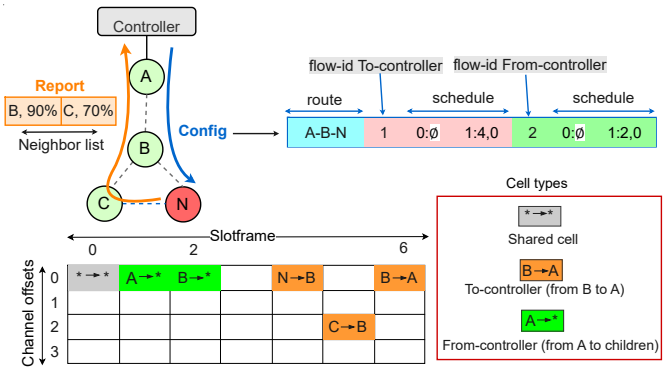

Fig. 1: Dedicated control plane in SDN-TSCH

packet corresponding to the flow-id to send. This mechanism provides flow isolation for data traffic because two flows can not compete for the same cell if they are assigned to different flow-ids. The flow-id is piggybacked in the header of each packet: when receiving a packet, a device extracts the flow-id from the header and looks up the flow table to find the matching rule to forward the packet. Particularly, two flow-ids are defined for the control plane:

"to-controller" for upward control packets generated by nodes;

"from-controller" for downward packets generated by a controller;

2) Configuration of control plane: a novel node joins the network, iIt needs first to join the TSCH network. In addition to TSCH synchronization Information Elements (IEs), the EB also includes the number of shared cells. The novel node extracts the number of shared cells and installs shared cells in its slotframe uniformly. The novel node listens to the medium for a period of time (report period) and tries to discover its neighbors by receiving EB packets from them.

When the novel node has performed the discovery process, it creates a `report` packet including the list of neighbors and the number of EBs received from each neighbor. It sends the `report` packet through a shared cell to any neighbor that has already joined the network. The node that receives the `report` packet already has a configured control plane to send the `report` packet through a dedicated control path toward a controller.

When a controller receives the report packet, it admits the novel node by adding it to the list of configured nodes and selects the neighbor with the highest EB number (extracted from `report`) as the parent of the novel node. A controller chooses two free cells in the schedule to configure the control plane of the novel node for upward and downward directions. Then, a controller prepares two `config` packets to include flow-ids and the cell IDs in which.

SDN-TSCH exploits the source routing technique to handle the `config` packets. Each `config` packet includes the list of addresses from the sink node to the novel node, through which the `config` packet will be forwarded. Each node that receives a `config` packet finds its position in the source routing list and extracts the next address in the list as the node to which it has to forward the `config` packet. The two last hops (*i.e.*, the novel node and its parent) install the schedule as specified in the `config` packet.

In Figure 1, a novel node sends a list of its neighbors and their EB number to a controller. In return, it receives a `config` packet including the schedule for the "to" and "from controller" flow-ids. Indeed, the schedule is encoded for each hop as `<number_of_cells:list_of_cells>` that each cell corresponds to a TX cell. All the cells of the control plane are dedicated to avoid collisions.

3) Admission of a critical data flow: A node that needs to admit a novel flow sends a `flow-request` packet through the control plane toward a controller. The `flow-request` contains the source and the destination addresses, as well as the required QoS (end-to-end reliability and deadline) for the application. A controller when receiving the `flow-request` allocates resources accordingly to respect these QoS requirements. For this purpose, the scheduler:

1) allocates backup retransmission cells for weak links along the path to respect the end-to-end reliability;
2) makes effort to schedule cells back to back before the deadline. The first forwarding cell for a node is placed right after the last retransmission cell of the previous hop.

Finally, a controller prepares a `config` packet to push the novel configuration. The `config` packet is forwarded through the control plane to configure the path from the destination toward the source. When the source node receives the `config` packet, the whole path is configured, and the node starts sending data packets.

Figure 2 illustrates a simple scenario where S sends a `flow-request` (including QoS requirements and address of D as destination) to a controller through the *"to-controller"* flow-id. In return, the `config` packet is forwarded in source routing, like the initial join process. It uses the *"to-controller"* or *"from-controller"* depending on if the next hop in the path is up or downward. A and B relay the `config` packet to reach D with no modification on their own schedules. Then, D receives the `config` packet and installs the first cells of the flow (starting from the destination).

To find the direction of handling `config` packet, each node verifies whether the address of the next hop also exists in the list of nodes before its position. Indeed, a node is present twice in the `config` packet for the subpath between the destination and the common ancestor (*e.g.*, subpath *D, B* in Fig. 2). If this is the case, the next hop is an upper-hand node, and the config packet must be handled in the upward direction by *"to-controller"* flow-id. Otherwise, the `config` packet is handled in the downward direction through *"from-controller"* flow-id. In Figure 2, the `config` packet includes the path A, B, D, B, E, S so D knows that B is reachable through the *"to-controller"* flow-id and B knows that E is reachable through the *"from-controller"* flow-id.

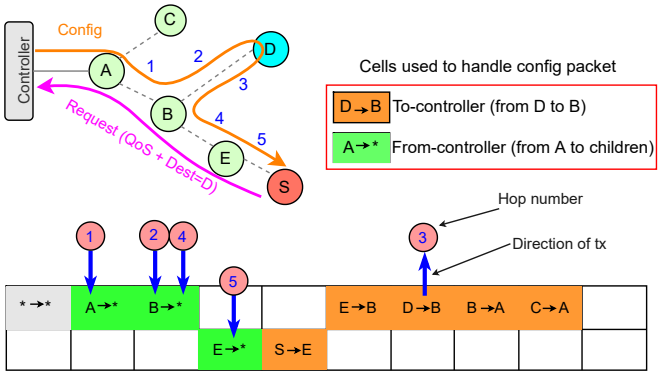

Fig. 2: Data flow Configuration in SDN-TSCH

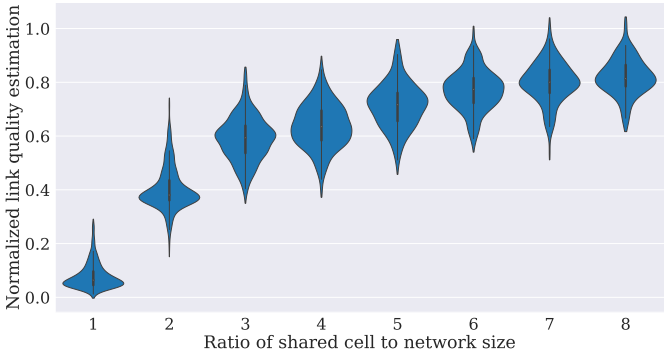

Fig. 3: Accuracy of link quality estimation in SDN-TSCH

IV. Link Quality Estimation

We need to estimate accurately the link quality to make the wireless SDN network efficient. The SDN controller exploits the link quality information to compute the best route and to allocate sufficient radio resources for each link. Basically, two different approaches may be applied to estimate the link quality in wireless networks [6]:

active monitoring: a node sends probing packets to neighbors to estimate the quality of their links. Probing packets can be sent in broadcast or unicast. However, unicast-based link monitoring can provide higher accuracy. By contrast, broadcast-based link monitoring is easy to implement and has lower overhead compared with the unicast-based approach.

passive monitoring: exploits the existing traffic of the network. Common packets or their acknowledgments can be leveraged for link quality estimation. Passive link monitoring is the widely used method in low-power wireless networks because of its energy efficiency. However, the regularity and rate of traffic can impact the accuracy of the measurements.

A. Requirements

We need to consider the following challenges that are specific to scheduled wireless networks:

all-neighbors discovery: to build a complete routing topology, the SDN controller should be aware of all links and their qualities in the network. Unfortunately, in scheduled networks, a node can only estimate the link quality of the neighbor with which it exchanges packets.

accurate estimation: link quality misestimation has a severe impact on the network capacity and on reliability. In the case of underestimation, a controller will allocate more bandwidth for the link and waste both the network capacity and the energy. By contrast, if the link quality is overestimated, not enough bandwidth is allocated, and some packets will be dropped or received after the deadline.

continuous estimation: due to the dynamic nature of wireless networks, link qualities may change. Thus, the link estimation system should continuously inform a controller of the last changes in the network.

energy efficiency: exploiting active monitoring in the scheduled wireless network is costly. Allocating one dedicated cell to each neighbor for probing consumes high bandwidth and energy. To have an energy-efficient link estimation system, Hermeto *et al.* [11] proved that accurate link quality estimation can be achieved by continuously sending broadcast packets.

B. Limitations of SDN-TSCH

SDN-TSCH proposes a passive link monitoring scheme. It leverages the EB packets of the TSCH network to estimate the quality of the links. Each node counts the number of received EBs from each neighbor in a given period of time. Because the EB packets are sent periodically, a node knows how many EBs it should receive from a neighbor during a given amount of time. A controller calculates the PDR of EB packets and uses it as a link quality metric.

SDN-TSCH relies on shared cells to send EB packets. To minimize the collision rate of EB packets, shared cells are uniformly located in the slotframe. Each node generates EB packets periodically and tries to send them in the first coming shared cell.

Figure 3 illustrates the inefficiency of the link quality estimation in SDN-TSCH. We will detail the simulation setup in Section VI. We measure the accuracy of the link quality estimation when varying the ratio of the number of cells and the number of nodes (network size). More precisely, we measure the normalized link quality, as the ratio of the Packet Delivery Ratio (PDR) estimated by the algorithm and the actual PDR of the link (directly extracted from the PHY model). As shown, EBs still collide even when allocating a large number of shared cells, resulting in an inaccurate link quality estimation. SDN-TSCH keeps on underestimating the link quality, biased by the collisions of EBs.

We propose to solve here this link quality misestimation by scheduling more appropriately the EBs. We keep on exploiting shared cells to optimize the discovery process and the continuous link quality re-estimation, while removing collisions.

C. Organization of the shared cells in the control plane

We need to send control packets in the control plane (joining, configuration, etc). More specifically, `report` packets for the novel nodes cannot use dedicated cells since no resource has been reserved. Thus, the flow-ids *"to-controller"* and *"from controller"* have not yet been configured. Similarly, keep-alive packets are required at the beginning to maintain the synchronization with the synchronization parent if no data packet is forwarded. We propose to use shared cells to send those control packets.

EBs use also shared cells for the network discovery. Thus, we propose that the SDN controller determines in which shared cell a node must send its EB packets. A controller allocates a given shared cell to a single node for the EBs transmissions. Thus, we cannot have anymore EB collisions. Since we use shared cells, all the neighbors receive EBs, contrary to dedicated cells. More importantly, the link quality may be estimated and updated for unused or even undiscovered links.

At the beginning of a shared cell, a node makes the following verifications:

1) if the shared cell is the EB cell dedicated to the node, it engages the transmission of its EB;
2) if the shared cell is the EB cell for another neighbor node, the node keeps awake to receive possibly its EBs;
3) if the shared cell is not an EB cell, the node transmits the first control packet in its queue. If none is present, it stays awake to receive possibly solicitations from neighbors.

In any case, a node in listening mode can turn off its radio after a fixed offset if no preamble is detected on the medium.

D. Schedule of EBs shared cells

Each node uses a given shared cell in slotframe to transmit its EBs. Indeed, each shared cell has a unique ID in the slotframe, and in joining the network, a controller specifies which shared ID must be used by a novel node. A controller allocates shared cell IDs sequentially, in the order of the arrival of the nodes. The last joined node has the maximum shared cell id (MAX-ID). In that way, we can make a distinction between the EB and non-EB parts of the slotframe. Since the MAX-ID value is announced in the EBs, the novel value is pushed hop by hop in the network, and after a given amount of time, all the nodes are aware of the novel cells reserved for the EBs.

We allocate some extra shared cells for non-EB traffic, and also separate the shared cells of EBs from non-EBs. To avoid collision with non-EB packets, nodes should only use the shared cells that are not used for any EB. Since EB cells are assigned sequentially, then a node can use any shared cells that have IDs larger than MAX-ID.

Meanwhile, the distribution of non-EB shared cells can impact the amount of collision in non-EB traffic. If the shared cells for non-EB traffic are grouped contiguously in the slotframe, the nodes experience a high collision rate [15]. Indeed, the nodes that generate their non-EB packets at the beginning of slotframe have to wait until the end of slotframe to send their packets. Then, all the nodes try to use the first coming non-EB shared cell which leads to a cumulative collision.

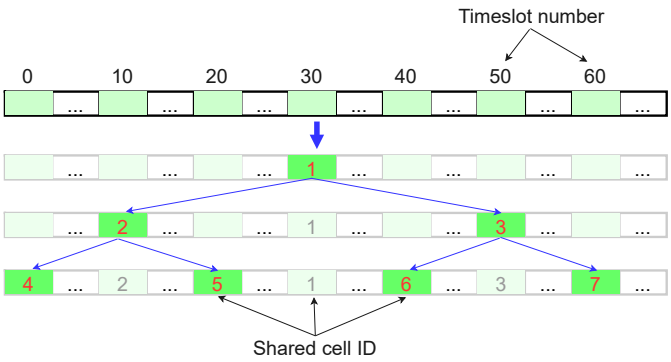

Fig. 4: Shared cell ID allocation in slotframe

We exploit a recursive division algorithm to distribute uniformly the shared IDs in the slotframe, and thus to reduce collisions. The recursive division algorithm allocates cell IDs as far as possible from each other and helps to not have a burst of traffic after the EB cells. Basically, a controller allocates the next EB shared cell in the middle of the largest remaining space without EB shared cells.

Figure 4 also shows an illustration of our algorithm in a slotframe with 7 shared cells. Recursively, in each step, the algorithm divides the number of shared cells into two and allocates the next ID to the middle cell.

E. Estimation of shared cell for non-EB traffic

We use the slotted Aloha model to estimate the number of shared cells in a slotframe for both EB and non-EB traffic. We assume that for joining the SDN network, nodes wake up at any time and generate a join request when they received a sufficient amount of EBs from already attached neighbors. We assume the control traffic in shared cells (keep-alive and `report` packets) follows a Poisson distribution with a mean of λ. We calculate the collision probability through:

$$\begin{aligned} P(collision) &= 1 - P(idle) - P(success) \\ &= 1 - e^{-\lambda} - \lambda e^{-\lambda} \end{aligned} \tag{1}$$

$P(idle)$ denotes the probability of zero transmission, and $P(success)$ is the probability of one unique transmission in the timeslot. By fixing the maximum acceptable value of $P(collision)$, we obtain the λ value. With the number of generated packets in the slotframe, we find the number of required shared cells:

$$Non\_EB\_cell = \lambda \cdot Num\_packet \tag{2}$$

Num_packet is the sum of all non-EB packets for all the nodes in the slotframe. Indeed, to provide a safety margin, we assume the worst-case situation in which all the nodes are in the transmission range of each other, and a shared cell may be used by any node in the network. Moreover, for EB

transmissions, each node needs one shared cell. Hence, the total number of shared cells is finally:

$$Num\_shared\_cell = Non\_EB\_cell + Node\_number \quad (3)$$

V. Resource allocation and configuration of control plane

We may allocate a large number of dedicated cells and flow-ids to create a reliable control plane. However, the more cells we allocate, the more we increase energy consumption. Indeed, unused cells consume a fixed amount of energy [16]: a receiver has to stay awake during a fixed offset from the beginning of the timeslot to accommodate clock drifts.

We investigate here three different methods to implement the control plane:

shared: all the control packets use the shared cells, that are mutualized in the network to reduce the amount of resource allocated to the control plane;

dedicated: each node maintains one dedicated cell to send control packets to its parent, and another one to its children. While no collision can arise among control packets, more resources are allocated to the control plane;

hybrid: each node maintains one dedicated cell to send control packets to its parent and sends other control packets through shared cells. Downward transmissions may be adjusted by a controller, and we may expect a low volume of collision in downlink.

The cost of these different architectures for the control plane will be evaluated in the following section.

A. Dedicated control plane

We propose in this variant to use only dedicated cells when the node has joined the network. Thus, control packets are protected against collisions, and we can upper-bound the delay to reconfigure a network (that depends on the size of each subtree rooted at the sink). When admitting a novel node, a controller allocates two dedicated cells:

dedicated-up: is reserved for the control traffic sent to the parent (for the flow-id *"to-controller"*);

dedicated-down: is reserved for the control traffic sent to any child (for the flow-id *"from-controller"*). Practically, all the children have to stay awake for this dedicated cell. Only the link-layer destination will acknowledge the control packet.

However, exploiting dedicated cells has a cost: each node has to stay awake during 4 dedicated cells (*i.e.*, to and from children, to and from parent) per slotframe, even if no control traffic is transmitted. In particular, the cells corresponding to the flow-id *"from controller"* are uniquely unused for `config` packets and are thus unused after the network has converged.

B. Shared control plane

It may be energy efficient to exploit only the shared cells to handle all the control traffic [17]. We face to a burst of control packets when the network bootstraps, and all the shared cells are mutualized to send all the control traffic. We could expect statistical multiplexing to make the number of collisions reasonable. However, using shared cells impacts the convergence time of the network: collisions may delay or even prevent nodes to exchange admission request/response packets.

Moreover, a shared control plane may affect the recovery time of critical applications. In particular, an event may occur locally that triggers flow reconfiguration. For instance, the decrease in the quality of a critical link may force a controller to reconfigure all the flows that pass through this faulty link, changing the paths, or rescheduling the whole flow to respect deadline constraints. A shared control plane may jeopardize the re-convergence of the network. Thus, a shared control plane is likely to be not agile enough to deliver the reconfiguration request in a timely manner.

C. Hybrid control plane

We propose a hybrid variant to combine dedicated cells in uplink and shared cells in downlink. Indeed, `report` packets are only transmitted uplink, through the *"to controller"* flow-id. The rest of the control packets that may compete correspond to i) `config` packets from a controller, ii) keepalive packets to maintain the synchronization, iii) `report` packets from novel nodes. It is worth noting that the EBs cannot collide with other control packets, since shared cells allocated by a controller to EBs are *protected*.

Keepalive packets are only used when the network bootstraps. Indeed, nodes use any packet (including the data ones) for re-synchronization. Thus, when the network has converged and when each node forwards or generates data packets, the network is maintained synchronized. Keepalive packets are only required to maintain the synchronization before the arrival of the `config` packet from a controller.

In conclusion, we would expect a lower amount of collisions compared with the full share control plane, but the convergence delay has to be finely measured to quantify the cost of using shared cells in the downlink.

VI. Performance Evaluation

We first assess the accuracy of our link quality estimation solution and compare it with SDN-TSCH. Then, we evaluate the performance of our three different control plane architectures. We consider here a single controller that allocates all the resources.

A. Evaluation setup

We use the Contiki-ng operating system and the Cooja simulator for our implementation. We simulate networks with sizes of 10, 20, 30, 40, and 50 nodes. Each node has a critical data flow to transmit to the sink node (*i.e.*, convergecast traffic). We consider critical applications that require an end-to-end PDR larger than 99% and with a deadline of 2 seconds. A controller allocates a novel flow-id and a set of cells to each critical flow to meet the requirements. Table I represents our different parameters.

TABLE I: 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 [18] 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 Traffic rate: Constant Bitrate, 1 packet every 5s
Applications	Requested PDR: 99% Requested deadline: 2s
SDN-TSCH	TSCH EB period: 15s SDN report period: 5 min `flow-request` timeout: 50s

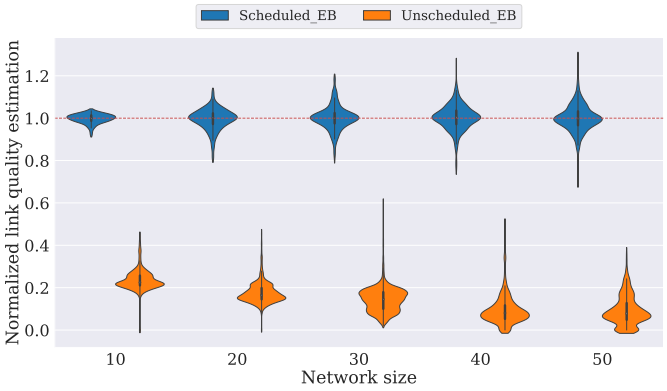

Fig. 5: Link quality estimation

B. Accuracy of the link quality estimation

We measure the accuracy of our link quality estimation for our scheduled EB approach and compare it with the original approach described in [4]. We measured in particular the normalized link quality, which is the ratio between the PDR measured by the approach, and the actual PDR as modeled by the simulator (Figure 3). An ideal estimation would always return quality of 1. Values below 1 mean that the PDR is under-estimated, and values above 1 mean that the PDR is over-estimated.

We use the same number of shared cells for both scheduled EB and unscheduled EB solutions. As shown in the scheduled EB approach, link qualities are estimated with high accuracy regardless of the network size. The error seems following a normal distribution centered on the real value. The quality is highly under-estimated when EBs are not scheduled: an EB may not be received because of a low SNR value, or because of a collision. Even worse, the error increases for larger network sizes, with a normalized quality tending toward zero for the largest network sizes. This misestimation leads to high energy consumption: a controller allocates too much resources for all the flows, wasting bandwidth and energy.

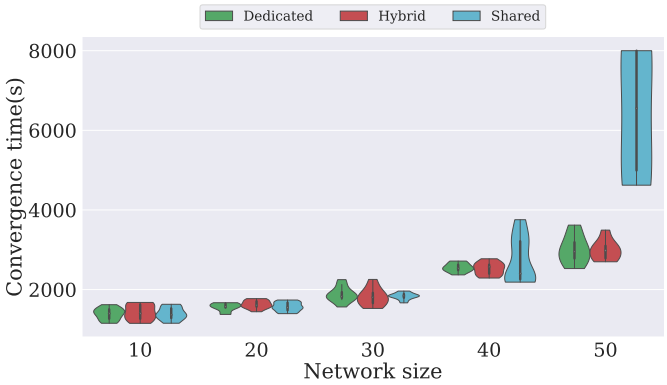

Fig. 6: Network convergence time

Our scheduled-based solution is very efficient since we can estimate very accurately the PDR of each link, that is reported in `report` packets. Thus, we consider only the schedule-based approach in the rest of the performance evaluation.

C. Efficiency of dedicated/shared/hybrid control planes

We compare the performance of different control planes: shared, dedicated, and hybrid. In light of our previous results, we use the scheduled EB transmission on separated shared cells for all approaches.

We first focus on the convergence (Figure 6). We define the convergence time as the time required from the bootstrap of the network until the last node to join the control plane (*i.e.*, receives the `config` from a controller). With the shared control plane, the convergence time increases exponentially with the network size. The collision rate can explain this observation: increasing the network size increases the volume of control packets transmitted over shared cells, resulting in numerous collisions. These collisions create a stream of retransmissions, leading to other collisions with a domino effect. With a network size of 50, some nodes are unable to join the SDN network before the end of the simulation time (8,000s). On the contrary, the dedicated and hybrid control planes maintain a reasonable convergence time, with appears to be linear with the number of nodes. Indeed, the density is kept constant, and a larger network size means a larger mean hop distance from the sink. Thus, the nodes need to wait longer that the previous hops synchronize and join the network.

Then, we measure in Figure 7 the time required to configure a novel data flow in the network – the time between the transmission of the `flow-request` and the reception of the corresponding `config` packet. We compare here only the hybrid and dedicated control plane since the shared one does not converge in all the cases, because of collisions.

The dedicated solutions provide the lowest configuration time: resources are dedicated to send the `flow-request` and the `config` packets without collisions. On average, less than 5 seconds are required for the call-admission and the configuration of the whole path, even with 50 nodes. The hybrid control plane provides also a very reasonable

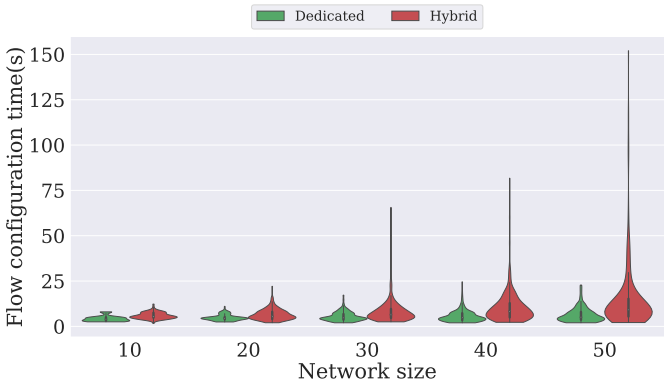

Fig. 7: Data flow configuration time

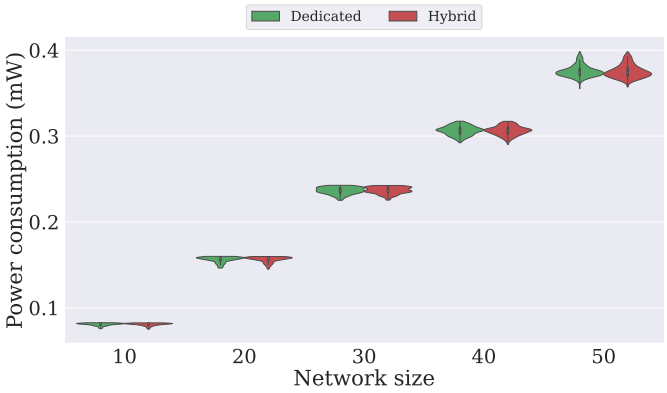

Fig. 9: Power consumption of nodes in joining period

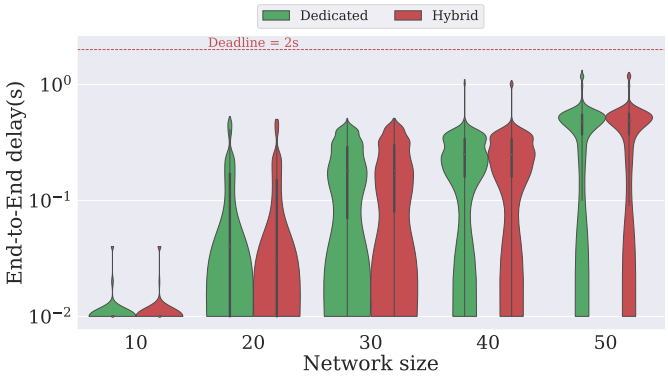

Fig. 8: End-to-end delay

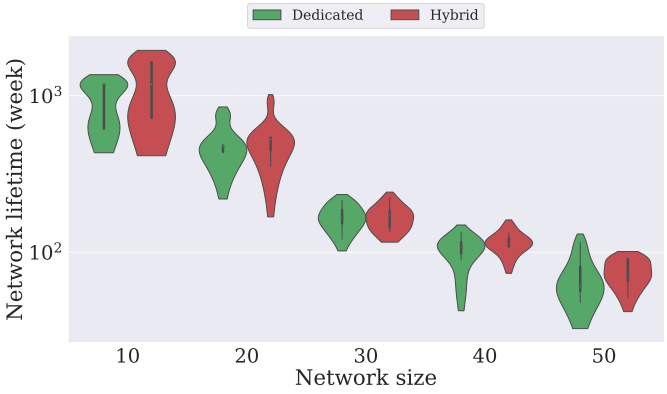

Fig. 10: Network lifetime

configuration time for any network size. However, the worst-cases may exhibit a large configuration duration. When many devices send their `flow-request` in a short time window, the sink is unable to transmit the corresponding `config` packets in a timely fashion, due to the collision happening between the `config` packets themselves or between the `config` and keep-alive packets. Some nodes still have not been configured yet, and send keep-alive packets to keep synchronized with their parents. Also, in the burst of collision, the sink node sometimes cannot dequeue all of its queued `config` packets and drops some of them due to the queue size limitation. This forces the source nodes to retransmit another `flow-request` after the timeout (50s in the simulations).

It is worth noting that we verified that the controller configures accurately the network. In particular, **all the flow requirements are respected** and in our simulations, 100% of packets are received at the destination before the deadline. Figure 8 illustrates the per-flow end-to-end delay. Both dedicated and hybrid solutions have the same end-to-end delay because they have the same scheme in link quality estimation. Also, by increasing the hop numbers in large networks, the end-to-end delay slightly increases accordingly.

Then, we measure the power consumption of devices, focusing specifically on the period before the network has converged (Figure 9). Thus, we report the power consumption for this period – the energy consumed divided by the time elapsed before receiving the first `config` packet. Dedicated and hybrid control planes have the same level of power consumption. The power consumption linearly increases with the network size. Indeed, based on the proposed model for the estimation of shared cells, we increase the number of shared cells by increasing the network size so that makes nodes wake up in more timeslots and consume more power.

Finally, we measure the network lifetime (Figure 10). We assume the lifetime is given by the first death. To extrapolate our simulations, the lifetime is estimated as the initial energy divided by the power consumption of the most constrained node. Since the configuration is done once, we consider the power consumption after the node has joined the network. Thanks to the use of shared cells for downward control traffic, the hybrid control plane provides a higher lifetime rather than a dedicated control plane. However, the difference is not a substantial value. For instance, in the network size of 50, the hybrid control plane improves the lifetime by 7% on average.

While the hybrid control plane impacts the time of flow admission, it also does not improve the lifetime impressively. Moreover, we may later need to extend the control plane to support more functionalities. Thus, the controller has more control packets to send. From our point of view, the dedicated

control plane can provide a higher reliability to handle control packets in a timely fashion, with a small increase in energy consumption compared with the hybrid control plane.

VII. Conclusion & Perspectives

Link quality estimation is of crucial importance when considering the SDN paradigm for industrial wireless sensor networks. In this article, we extended SDN-TSCH, an efficient implementation of an SDN architecture in a scheduled, industrial wireless sensor network. To improve the accuracy of the link quality estimation, we proposed here a scheduled EB approach that reserves some of the shared cells for collision-free EB transmission. The results show that such a strategy allows an accurate link estimation while keeping the general concept of shared cells. We compared an hybrid, dedicated and shared control plane, and highlighted with our simulations the strength of a dedicated control plan to provide high-reliability.

In our future work, we plan to enhance the capabilities of the controller to enable continuous optimization in response to changing network characteristics, such as varying traffic volumes and link quality. A controller will be developed to detect faulty links and automatically reconfigure the network to utilize the most reliable routes. Additionally, a controller will consistently update the link schedule to ensure compliance with Service Level Agreements (SLAs). We described here a proprietary protocol (with `report` and `config` packets) for the Southbound API. Thus, we plan to explore how these features can be integrated in an enhanced OpenFlow version. Finally, we need to address the security aspects of our solution: how devices can trust a controller (and vice-versa), and how we can guarantee the integrity of the data exchanges.

References

[1] Emiliano Sisinni et al. Industrial internet of things: Challenges, opportunities, and directions. *IEEE transactions on industrial informatics*, 14(11):4724–4734, 2018.

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

[3] Dominik Henneke et al. Analysis of realizing a future industrial network by means of software-defined networking (sdn). In *IEEE WFCS*, 2016.

[4] Farzad Veisi et al. Sdn-tsch: Enabling software defined networking for scheduled wireless networks with traffic isolation. In *IEEE ISCC*, 2022.

[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] Nouha Baccour et al. Radio link quality estimation in wireless sensor networks: A survey. *In ACM TOSN*, 8(4):1–33, 2012.

[7] Henry-Joseph Audéoud et al. Single reception estimation of wireless link quality. In *IEEE PIMRC*, 2020.

[8] Laura Galluccio et al. Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks. In *INFOCOM*, 2015.

[9] Michele 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*, pages 6–10, 2008.

[10] de Oliveira et al. Tinysdn: Enabling tinyos to software-defined wireless sensor networks. *XXXIV Simpósio Brasileiro de Redes de Computadores. Bahia*, pages 1229–1237, 2016.

[11] Rodrigo Teles Hermeto, Antoine Gallais, Kristof Van Laerhoven, and Fabrice Theoleyre. Passive link quality estimation for accurate and stable parent selection in dense 6tisch networks. In *EWSN*, pages 114–125, USA, 2018. Junction Publishing.

[12] Fabrice Theoleyre et al. Experimental validation of a distributed self-configured 6tisch with traffic isolation in low power lossy networks. In *ACM MSWiM*, 2016.

[13] Rodrigo Teles Hermeto et al. 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.

[14] Adrian Farrel et al. A path computation element (pce)-based architecture. RFC 4655, IETF, 2006.

[15] Fabrice Theoleyre et al. Experimental validation of a distributed self-configured 6tisch with traffic isolation in low power lossy networks. In *ACM MSWiM*, 2016.

[16] Xavier Vilajosana, Qin Wang, Fabien Chraim, Thomas Watteyne, Tengfei Chang, and Kristofer S. J. Pister. A realistic energy consumption model for tsch networks. *IEEE Sensors Journal*, 14(2):482–489, 2014.

[17] Baver Özceylan, Berk Ünlü, and Buyurman Baykal. An energy efficient optimum shared cell scheduling for tsch networks. In *2017 IEEE WiMob*, pages 1–8. IEEE, 2017.

[18] UDGM: a Cooja Radio Medium. https://github.com/contiki-ng/cooja/blob/06863708772e33504b3a397c225db9466082fcaf/java/org/contikios/cooja/radiomediums/UDGM.java, accessed: 2023-01-13.