ASAP: a decentralized slot reservation policy for dynamic 6TiSCH Networks in Industrial IoT

Gianfranco Micoli$^{(1)}$, Pietro Boccadoro$^{(1,3)}$, Giovanni Valecce$^{(1,3)}$, Antonio Petitti$^{(2)}$, Roberto Colella$^{(2)}$, Annalisa Milella$^{(2)}$, Luigi Alfredo Grieco$^{(1,3)}$
$^{(1)}$Dep. of Electrical and Information Engineering (DEI), Politecnico di Bari, Bari, Italy,
Email:name.surname@poliba.it
$^{(2)}$Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing,
National Research Council, Bari, Italy, Email: name.surname@stiima.cnr.it
$^{(3)}$ CNIT, Consorzio Nazionale Interuniversitario per le Telecomunicazioni, Politecnico di Bari, Bari, Italy.

***Abstract*—In Industrial Internet of Things (IIoT) networks, end-nodes may disconnect or become no longer reachable, due to manifold causes, such as battery discharge, network interference and/or packet collisions. Overall, this could lead to low Quality of Service (QoS) levels. To satisfy application requirements, i.e., reliability, it is mandatory to continuously monitor connectivity and properly managing disconnections, without causing signaling overhead. The optimization of the exchange of messages can be pursued thanks to dedicated communication slots, eventually rescheduled in cases of deletion, replacement or dynamic reassignments. This solution lowers energy consumption while accounting for a number of improvements in network operative conditions. In this work, network formation optimization mechanisms for IPv6 over the TSCH mode of IEEE 802.15.4 (6TiSCH) networks are analyzed in real use cases in which the number of nodes in the network varies over time. Once the network was fully connected, a performance analysis has been carried out in order to measure typical networks parameters, such as latencies and Packet Loss Ratio (PLR). The study has proven the effectiveness of dedicated slots dynamic assignment. In particular, it has been measured a sensible reduction of latencies and losses of information. The obtained results pave the way towards robotics-aided set up of Internet of Things (IoT) networks in industrial contexts.**

***Index Terms*—Internet of Things, Medium Access Control, scheduling, Quality of Service**

I. Introduction

Nowadays, IoT technologies are considered as the angular stone of the industrial digital innovation process that will lead to the so called Industry 4.0 [1]. In this context, two key challenges to face are cooperation among dynamic systems and resource sharing. The automated deployment of industrial IIoT networks may represent an enabling perspective, especially in harsh and/or large scale environments. Communications reliability is one of the mandatory requirement to ensure the employability of IoT networks in industrial applications, i.e., control, telemetry, and prognostics. Typical reference scenarios may be related to (i) monitoring vines in a vineyard or (ii) trees in a reforestation project, (iii) studying traffic and/or pollution levels on city streets with real-time traffic, (iv) measuring humidity and temperature at regular intervals on library shelves, (v) performing acoustic testing at each of the seats in a theater [2]. In wide scenarios, deployment/maintenance/replacement/(re)configuration of multiple IoT nodes (known as motes) should be automated as much as possible. To reach this goal, the IIoT network deserves to be optimized, connected and has to guarantee reliable connections among nodes, minimizing disconnections and data losses.

Upon deployment, each mote should be fully configured in order to exchange data with the rest of the network. Once IoT devices are released, the scheduling function will arrange and orchestrate all the timeslots in a slot frame as to avoid overlapping and/or collision phenomena [3][4].

In this work, a slot reservation policy, called ASAP, operates in order to link creation requests for (a) new parents, (b) new children, together with (c) multicast message. The resulting slotframe demonstrates a correspondence between the dedicated links and the reserved timeslots. As a proof, the function has been experimentally evaluated on a real 6TiSCH network, increasingly populated by IoT devices dynamically joining the network. Once deployed, the IIoT network has been tested with the aim of verifying typical QoS parameters, such as: (i) Destination Advertisement Object (DAO) reception time, (ii) PLR, (iii) Round Trip Time (RTT), (iv) average slot usage, and (v) queue filling. Results demonstrated a messages exchange improvement in a measure of the 92%, and a reduction of packet queues by the 50%. The most thrilling outcome is that the road ahead towards robotics-aided automated deployment of IoT networks in industrial context [5] has been proven feasible.

This work is organized as follows: Section II is dedicated to 6TiSCH technology and scheduling functions. Section III presents both the network switching function and the decentralized policy. The experimental campaign and the obtained results are discussed in Section IV. Section V highlights achievements and proposes future works.

II. IoT background

The Internet Engineering Task Force (IETF) 6TiSCH technology [3][4] is a well known protocol stack specifically aimed at enabling short range communications over constrained IoT devices in IIoT applications [6][7]. The 6TiSCH technology does not standardize any Medium Access Control (MAC) layer scheduling function or policy for time-slot management. Generally speaking, a schedule can be designed to better serve network topologies, especially in multi-hop scenarios. Scheduling functions can optimize schedule dissemination, thus, facilitating dynamic slot allocation and network joining procedures, optimizing communication efficiency, latencies, duty cycling, and energy consumption [8][9]. A scheduling solution can either be centralized or decentralized [8]. In the former approach, all scheduling tasks are carried out by the network coordinator. In the latter approach, instead, nodes can negotiate links amongst themselves, communicating directly to the neighboring nodes. The gap between the two can be filled by an hybrid solution, the hierarchical approach, in which a routing tree is formed before scheduling links between node.

In [8] a thorough survey of such solutions is proposed. Among them, it is important to mention Scheduling Function Zero (SF0) [10] and Low Latency Scheduling Function (LLSF) [11], both specifically designed for 6TiSCH Networks. SF0, which is currently under development, dynamically allocates dedicated Transmission timeslot (TX) for direct communications between nodes. A device in a 6TiSCH network executing SF0, mainly performs two operations: (i) Cell Estimation Algorithm (CEA) to estimate how many timeslots are required for communication towards neighbor and (ii) Allocation Policy (AP) to adapt the number of required timeslots to the number of currently number of allocated timeslots. LLSF, instead, is a scheduling function that builds upon SF0 and improves it as to reduce latency during packet exchange. It optimizes allocation in the slotframe to reduce the number of timeslots thus minimizing transmissions/receptions events. CEA calculates the number of timeslots to allocate for a specific neighbor by summing the number of currently used cells with an *over provision* value, therefore taking into account traffic variability and potentially reducing it. Afterward, the resulting number of required cells is used by the AP to allocate or deallocate timeslots. This routine is able to compare the number of required cells to the number of timeslots that are currently allocated. LLSF, instead, is a scheduling function that builds upon SF0 and improves it in order to reduce packet exchange latency along network paths. It optimizes allocation tasks in the slotframe trying to minimize transmission events reducing timeslots in number. Moreover, it aims at scheduling child-parent slots keeping the TX ones in close proximity to the respective Reception timeslots (RXs). This can reduce transmission latency as transmissions may get executed in a single iteration of the slotframe.

Unfortunately, both these solutions may still result inefficient in operating scenarios. In fact, frequent topological changes may cause non-negligible signaling overhead. Moreover, none of the surveyed contributions address the network switching functionalities, even if this is a crucial feature in highly mutable configurations.

III. The proposal

This work proposes a network switching mechanism and a slot reservation policy, called ASAP. The former allows for a dynamic increase in the number of nodes in the network. Such routine is able to properly handle the situation in which a new node wants to join the IoT network, already deployed and at work in a certain area. The latter, instead, can be successfully employed in the context of a 6TiSCH network that gradually increases in size.

A. Network Switching Function

The need for a network switching function is motivated by the fact that the upcoming IIoT operating scenario will increasingly employ robotic units to automate processes. Among them, the deployment of an IoT network has to be properly triggered and handled in a standard-compliant way. To bridge this gap, an event-driven release call scenario is assumed. The proposal foresees several routines: (i) the *mote_sower*, (ii) the *manual sower*, (iii) the *sower_server* (acting as a supervisor), (iv) *arduino_release*, and (v) *queue_switcher*. The *mote_sower* has been created to handle the release of each mote (*manual sower*). The *manual sower*, instead, handles each mote-release

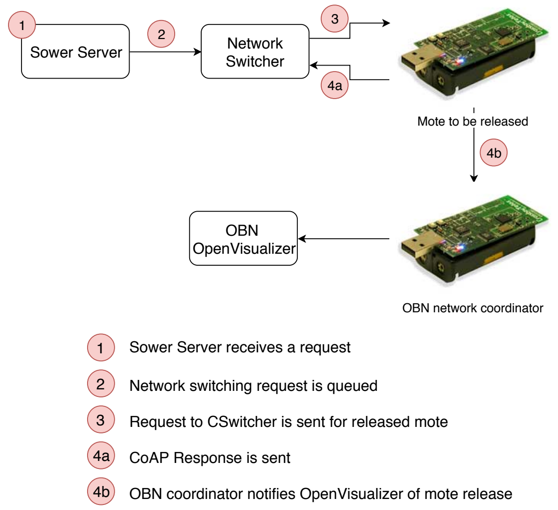
1
Sower Server
2
Network Switcher
3
4a
Mote to be released
4b
OBN OpenVisualizer
OBN network coordinator
1 Sower Server receives a request
2 Network switching request is queued
3 Request to CSwitcher is sent for released mote
4a CoAP Response is sent
4b OBN coordinator notifies OpenVisualizer of mote release

Fig. 1: Network switching request elaboration steps.

allowing a detachment from a network and the connection to a new one. The *sower_server* routine is used as an interface for the whole release mechanism (see Figure 1). Specifically, it exposes a service called *full_release* that accepts two arguments: the last two bytes of the mote's Internet Protocol version 6 (IPv6) address, and a boolean value indicating if the mote to be released, has to be set as the coordinator of the network. When the request is received (Algorithm 1), the topological release is triggered via a service named *queue_switcher*. The *queue_switcher* service lets the Network Switcher handle a queue of network switching requests in order to avoid conflicts deriving from concurrent requests. Each network switch is triggered by a dedicated (firmware) application, called CSwitcher. It answers to a single Constrained Application Protocol (CoAP) request at a time, so that the mote will disconnect from its current network and connect to a new one by changing its PAN IDentifier (PANID). Functionally speaking, the mote that has switched the network should remove all the references to the old network (e.g. neighbors, network topology, ASAP links) while it synchronizes with the new one. Even if the most efficient

Algorithm 1: On Network Switcher request received

```
1: function SWITCH_REQUEST(req)
2: open CoAP client;
       construct mote IPv6 address;
       CoAP PUT on CSwitcher for mote to be released;
       if CoAP Response Received then
           remove mote from switching queue;
           close CoAP client;
3: end function
```

way to accomplish this task would be a complete reboot of the device, this could still lead to a connection to the old coordinator, instead of joining the new one. Therefore, a *soft reboot* procedure has been created to call bootstrap functions to reset the internal state while the PANID is set to the new one. The algorithm on which CSwitcher is based can be represented as a Finite State Machine (FSM) (see Figure 2). The three states are:

- IDLE, when no requests have been received, yet;
- REQ, when a network switch request is received;
- SWITCH, when the mote is switching to the new network;

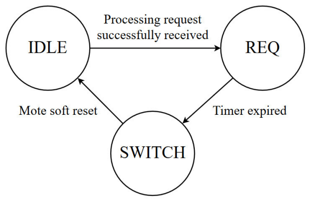

Fig. 2: State diagram of the soft reboot procedure (CSwitcher).

The mote is still in the IDLE state. Once a CoAP request is received, CSwitcher goes in the REQ state. The request contains an identifier of the network to switch to and a flag indicating if the mote has to become root of the second network. Then, the IoT device is in the SWITCH state. The actual network switch function is called after a period of time useful to let the mote dispatch the message without problems. This last function activates the *soft_reboot*, in order to let the mote connect to the second network. This also automatically resets the CSwitcher application to the default IDLE state, so the mote can eventually process another request. It is worth noting that the FSM does not foresees any exception/error. This is so as, once the mote has left the old network, the mote will certainly either join the new network or loose connection. The latter case is related to battery discharge events or accidental hard reboot. Upon receiving a request, the service will answer to the client node, for instance, the *sower_server*. Then, the node answers the caller in order to proceed with the execution of his tasks, while multiple requests can be processed in background. The CoAP client used here actually has a synchronous Application Programming Interface (API), so it would block the caller if the requests is not processed in threads. Both the node and the *network_switcher* have to complete post-release actions, for instance:

- the node has to remove all the references to the detached mote, with a trigger coming from the multicast sent by CSwitcher;
- *network_switcher* has to close the CoAP client used for the request and terminate the thread, with a trigger coming from the CoAP response.

B. Decentralized Scheduling Function

ASAP is able to identify new nodes willing to joint the network and dynamically modify the slotframe allocating time slots for neighbor nodes. In the start-up phase, the only working node is the coordinator, whose radio interface is always active, and continuously listening for any node willing to join. When another node is in proximity, the synchronization procedures at MAC level start. The network coordinator modifies the current schedule structure and inserts two new timeslots, the first for transmissions activities and the second for receptions. To properly address network formation and energy consumption reduction criteria, ASAP operates in a distributed manner, mirroring the routing graph and properly managing faulty links, as it falls-back to shared slots if a communication task cannot be executed in a dedicated one. Moreover, ASAP has been designed to avoid deallocating dedicated parent-child links in case IoT devices are temporarily disconnected. The result is that ASAP lowers multiple link negotiations. Since the schedule is reduced and optimized, ASAP also results to have a lower memory footprint, when compared to other solutions. Instead of estimating bandwidth like SF0 does, ASAP immediately creates timeslots with preferred parents. Its objective is to enable duplex communications over dedicated timeslots between mote couples, so that shared timeslots could be used as fallbacks. ASAP is based on the *link* concept, for instance a bidirectional communication channel composed of two adjacent timeslots (Figure 3). Links are created following a *child-preferred parent* criteria: each mote allocates timeslots for communication with its preferred parent. This enables optimization of real-world packet transfers (e.g. requests for data packets). Optimizing communications with the preferred parent also adapt links to the network topology, handled at the network layer. Each link (Figure 4), whether it is allocated or not, will be represented by an element inside an array. The index in the array (i.e., *position*), uniquely identifies the timeslots in the reserved area of the slotframe. In order to allocate links between mote couples, a lightweight negotiation algorithm is used. Each link is managed in the scheduling function algorithm; timeslots will be referred to only when they actually have to and link positions are used in the rest of the algorithm. With a minimum packet exchange, a mote couple can generate links, while deallocating those related to neighbors that are no longer reachable (Figure 5).

IV. Experimental scenario

In the reference scenario, the new network coordinator node is connected to a host PC, for instance an Acer Veriton N4620G, equipped with an Intel Core i3 processor with 4 GB of Random Access Memory (RAM). The IoT devices involved in the experimental setup are ten Telos rev B (namely

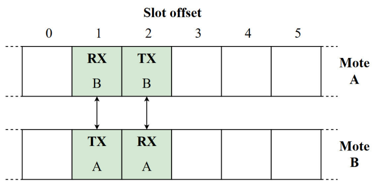

Fig. 3: ASAP Link Structure.

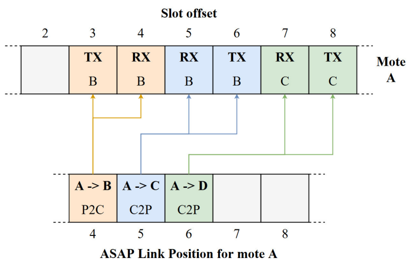

Fig. 4: Correspondence between ASAP positions and timeslots.

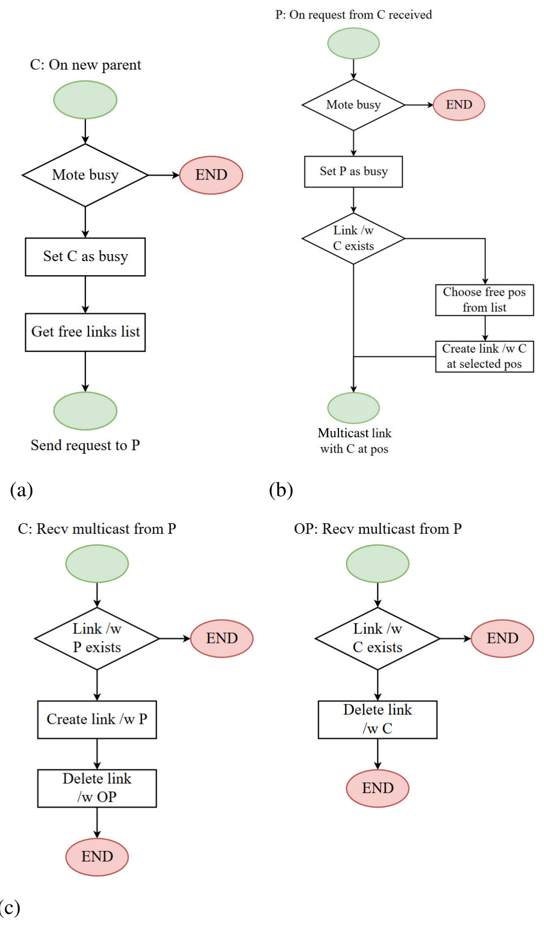

Fig. 5: ASAP link creation flowcharts for (a) new parent requests, (b) new child request, and (c) multicast message received.

TelosB[1]), a well known IoT platform that has been used in several researches over the last decade [6][7]. As for software, it has been employed OpenWSN[2] [12], an open source solution implementing the IETF 6TiSCH protocol stack. It provides two set of functionalities: (i) executing communication tasks through IoT devices (i.e., the firmware part) and (ii) real-time monitoring the IoT network activities while granting its connection to the whole Internet (i.e., the software part, namely OpenVisualizer). The software component, mainly acts as a border router for the IoT network, providing gateway functionalities to the Internet through the PAN Coordinator. Network traffic can be analyzed thanks to a visualization platform enabled to grant a direct interaction with the coordinator via a software TUN interface translating IPv6 packets to IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) and viceversa, while the Routing Protocol for Low-power and Lossy networks (RPL) component stores all the paths to the nodes in the network, read from DAOs.

A. *Performance evaluation*

The deployment algorithm has been functionally evaluated together with the slot reservation policy in a star IoT network with 1 coordinator and 9 IoT nodes (Figure 6). As soon as the release of a mote is completed, the joining procedure starts and the proposed scheduling function gets involved. To verify its effectiveness when applied to an automatically deployed IoT network, an experimental campaign has been carried out. The aim of the experiments was to compare the usage of dedicated slots with shared slots, while analyzing the effect on network performances of the automation provided by ASAP. In details, three different scheduling solutions have been tested:

1) ASAP, the proposed function;
2) the so-called ASAP Fixed, an hard-coded slotframe schedule generating links with specific motes at boot, using the same functions ASAP provides;
3) Shared, a slotframe schedule with only TX/RX shared timeslots.

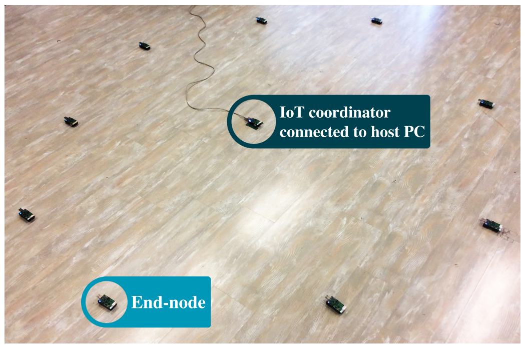

Fig. 6: The IoT network after the deployment has been completed.

With reference to the *Shared* scheduling solution, assuming to ignore the Carrier Sense Multiple Access with Collision Avoid-

[1]http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf
[2]https://openwsn.atlassian.net/wiki/spaces/OW

ance (CSMA/CA) backoff algorithm, communications would theoretically occur in a single slotframe if the number of slots at least equals the number of end-devices a certain node is connected to. Even though this condition could be theoretically sufficient, conflicts may verify in real operating conditions. Therefore, a little redundancy has been introduced by increasing the number of shared slot by one. In the ASAP-based scheduling solutions, the network coordinator creates a couple of link with each child in topology (see Section III-B). As a result, each child in the IoT network will rely on a scheduling solution that grants fair duty cycling conditions.

The experimental campaign was meant to communicate real data coming from sensors on-board of each mote. The IoT network has been tested with a CoAP application. Leveraging the request-response communication scheme, to each mote in the network 20 different requests were sent. Every time a request is done, it is sent to every mote in topology. The collected responses contained: (i) first DAO reception time, (ii) RTT and PLR for each request, (iii) timeslot usage during each test run (e.g., number of transmissions, receptions and Acknowledgements (ACKs)), and (iv) network coordinator packet buffer status. DAO reception time is an important parameter that can be used to evaluate the time period needed to complete network formation operations (Figure 7). This metric provides an indication of the time taken by the IoT devices to complete the Directed Acyclic Graph (DAG), which grants the both upward and downward paths are formed and packets are free to flow in both directions. Measured times do not directly depend on the scheduling function. Nevertheless, the layered structure of the protocol stack implies that all the operations defined at MAC layer must be completed before upper layers can be involved. Results in Figure 8 demonstrate the higher efficiency of dedicated links and timeslots. In particular, avoiding collisions in both transmission and reception, the PLR is lowered to 4.83%, which represents an interesting result when compared to other solutions [13][14]. In particular, the measured values prove that the usage of dedicated links lowers the PLR in a measure of the 91%. To prove the effectiveness of the proposed scheduling function, RTT has been tested and the average measured values are reported in Figure 9. Measured values show an improvement in the measure of the 50%. The failure causes for packet delivery can be vary but queuing time is among the major concerns. Hence, queues have been evaluated in terms of number of packets over time. In particular, the maximum and average numbers of packets in a queue are represented in Figures 10 and 11, respectively. For the sake of completeness, average slot usage measured over the tests is reported in Table I. The joint evaluation of all the measured parameters clearly demonstrates that the usage of dedicated slots better fits QoS criteria, in terms of reliability. The minimum RTT value for any test case does not fall below $1.2s$. For the Shared scheduling solution, this happens when the network traffic is still low, meaning that there are no conflicts and so the performance are similar to the best case in ASAP and ASAP Fixed solutions. If the traffic, or the number of motes in the IIoT network, is low, the usage of dedicated slots does not lead to

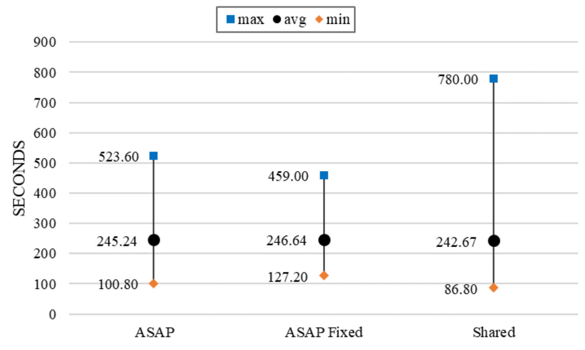

Fig. 7: Average first DAO reception time.

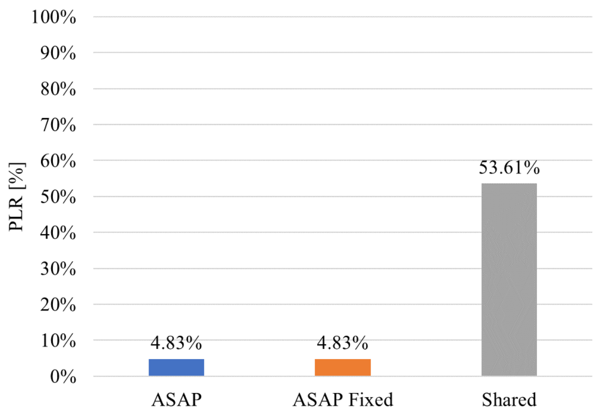

Fig. 8: Average CoAP PLR.

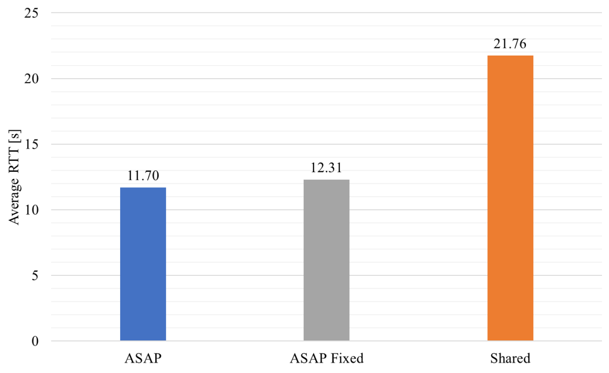

Fig. 9: CoAP - Average RTT.

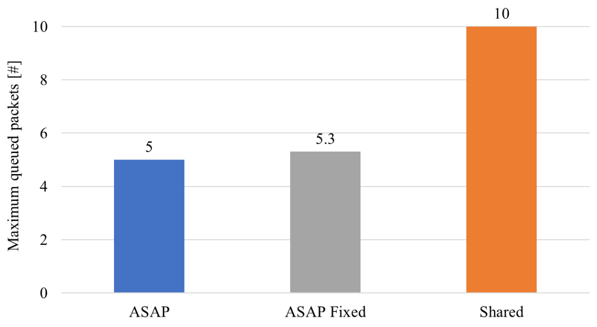

Fig. 10: CoAP - Maximum number of queued packets.

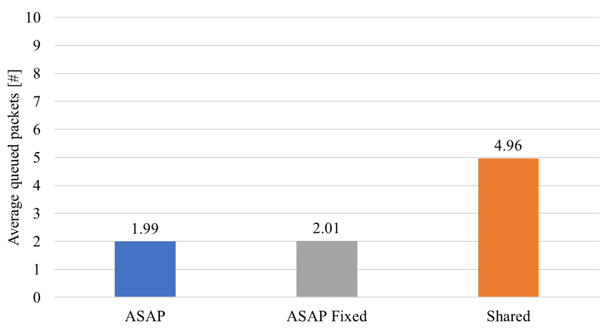

Fig. 11: CoAP - Average number of queued packets.

	Dedicated Slots			Shared Slots		
	TX	TX ACK	RX	TX	TX ACK	RX
ASAP	219	192	339	170	148	201
ASAP Fixed	341	319	363	214	196	186
Shared	-	-	-	764	580	726

TABLE I: CoAP - Average slot usage [No. of packets].

sensible advantages when compared to the shared slots solution, as CSMA/CA backoffs would be much less frequent. As shown in Figure 10, the Shared solution almost fill up the packet buffer in each test case. Here, conflicts cause noticeable delays in packet dispatch, which can impair network performances and communications, lowering QoS. This may result in worsening PLR values. Scheduling solutions with dedicated slots will result in a much less variable packet dispatch rate, which makes RTTs more predictable, also resulting in a lower probability of packets queue filling (Figure 11). Most of the requests are elaborated inside dedicated timeslots using ASAP-based solutions[3] (see Table I).

V. Conclusions and Future Research

This work presented a decentralized solution specifically conceived for continuously monitor connectivity, properly managing disconnections, without causing signaling overhead, in 6TiSCH-compliant IIoT networks. Among the main objectives, there was the automated assignment of dedicated timeslots. The experiments highlighted that, at MAC layer, a dedicated scheduling function can significantly lower the amount of packet losses, while improving communication latencies. Both network formation optimization mechanism and scheduling policy demonstrated a sensible reduction of queue filling, one the major stakeholders in latencies and losses. It was shown that the proposed ASAP algorithm is able to improve message exchange by the 92%, thus resulting in higher network efficiency, and has a remarkable impact on packet queues, diminished by the 50%.

Despite the results, the adopted decentralized approach could still be compared with centralized solutions. Moreover, ASAP link requests could be embedded within Information Element (IE) messages. Nodes in the IIoT network could also rely on multicast messages from parents to improve the setup phase. Finally, dynamic network redeployment capability could be introduced in multihop scenarios.

The most thrilling attainable perspective is represented by the employment of unmanned vehicles for automating the deployment of a self-organizing IoT network in industrial context.

[3]ASAP-based solutions require less packet transmissions before the end of the test, despite the lower measured values for the PLR. This means that the energy consumption would potentially be lower, while having a slightly higher duty cycle.

VI. Acknowledgments

This work was partially founded by Italian MIUR PON projects Pico&Pro (ARS01_01061) AGREED (ARS01_00254), FURTHER (ARS01_01283), RAFAEL (ARS01_00305) and by Apulia Region (Italy) Research Project E-SHELF (OSW3NO1).

References

[1] 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*, vol. 11, no. 1, pp. 17–27, Mar. 2017.

[2] S. R. Blackburn, T. Etzion, K. M. Martin, and M. B. Paterson, "Efficient key predistribution for grid-based wireless sensor networks," in *International conference on information theoretic security*. Springer, 2008, pp. 54–69.

[3] T. Watteyne, J. Weiss, L. Doherty and J. Simon, "Industrial IEEE802.15.4e Networks: Performance and Trade-offs," in *IEEE International Conference on Communications (IEEE ICC)*, vol. Internet of Things Symposium, London, UK, June 2015, pp. 8–12.

[4] T. Watteyne, M. R. Palattella and L. A. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement," *Internet Engineering Task Force (IETF) - Request for Comments: 7554*, 2015.

[5] C.-Y. Chang, Y.-T. Chin, C.-C. Chen, and C.-T. Chang, "Impasse-aware node placement mechanism for wireless sensor networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 8, pp. 1225–1237, Aug. 2018.

[6] V. Scilimati, A. Petitti, P. Boccadoro, R. Colella, D. Di Paola, A. Milella, and L. Grieco, "Industrial internet of things at work: a case study analysis in robotic-aided environmental monitoring," *IET Wireless Sensor Systems*, June 2017.

[7] H. Harb and A. Makhoul, "Energy-efficient sensor data collection approach for industrial process monitoring," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 661–672, Feb. 2018.

[8] R. Teles Hermeto, A. Gallais, and F. Theoleyre, "Scheduling for ieee802.15.4-tsch and slow channel hopping mac in low power industrial wireless networks," *Comput. Commun.*, vol. 114, no. C, pp. 84–105, Dec. 2017.

[9] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. J. Pister, "A realistic energy consumption model for tsch networks," *IEEE Sensors Journal*, vol. 14, pp. 482–489, 2014.

[10] D. Dujovne, L. A. Grieco, M. R. Palattella, and N. Accettura, "6TiSCH 6top Scheduling Function Zero / Experimental (SFX)," Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-6top-sfx-00, Sep. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-6top-sfx-00

[11] T. Chang, T. Watteyne, Q. Wang, and X. Vilajosana, "Llsf: Low latency scheduling function for 6tisch networks," in *2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, May 2016.

[12] Regents of the University of California, "OpenWSN," April 2016. [Online]. Available: https://openwsn.atlassian.net/wiki/pages/viewpage.action?pageId=688187

[13] P. Boccadoro, G. Piro, D. Striccoli, and L. A. Grieco, "Experimental comparison of industrial internet of things protocol stacks in time slotted channel hopping scenarios," in *Proc. of IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, May 2018.

[14] N. Accettura, E. Vogli, M. R. Palattella, L. A. Grieco, G. Boggia, and M. Dohler, "Decentralized traffic aware scheduling in 6tisch networks: Design and experimental evaluation," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 455–470, Dec. 2015.