# Decentralized traffic aware scheduling in 6TiSCH networks: design and experimental evaluation

# Decentralized Traffic Aware Scheduling in 6TiSCH Networks: Design and Experimental Evaluation

Nicola Accettura, *Member, IEEE,* Elvis Vogli, *Student Member, IEEE,* Maria Rita Palattella, *Member, IEEE,* Luigi Alfredo Grieco, *Senior Member, IEEE,* Gennaro Boggia, *Senior Member, IEEE,* Mischa Dohler, *Fellow, IEEE,*

***Abstract*—This paper capitalizes on two emerging trends, i.e. the growing use of wireless at the edge of industrial control networks and the growing interest to integrate IP into said networks. This is facilitated by recent design contributions from the IEEE and the IETF, where the former developed a highly efficient deterministic time-frequency scheduled medium access control protocol in form of IEEE 802.15.4e TSCH and the latter IPv6 networking paradigms in form of 6LoWPAN/ROLL, and scheduling approaches in form of 6TiSCH. The focus of the present work is on advancing the state of the art of deterministic 6TiSCH schedules towards more flexible but equally reliable distributed approaches. In addition, this paper aims to introduce the first implementation of 6TiSCH networks for factory automation environments: it outlines the challenges faced to overcome the scalability issues inherent to multi-hop dense low-power networks; the experimental results confirm that the naturally unreliable radio medium can support time-critical and reliable applications. These developments pave the way for wireless industry-grade monitoring approaches.**

***Index Terms*—6TiSCH, IEEE 802.15.4e, scheduling algorithms, Timeslotted Channel Hopping**

## I. Introduction

Industrial networks, often referred to as Operational Technology (OT), and computer networks, referred to as Information Technology (IT), emerged simultaneously some 40 years ago, each designed with a specific aim and different range of applications in mind. After being developed for years in parallel, IT and OT technologies commence to converge and be mutually integrated, enabling OT traffic to be transported over a shared IP-based IT infrastructure. Due to their different goals, OT and IT have evolved in a radically different way one from another. Therefore, a number of new challenges have to be overcome in order to make the IT/OT integration viable [1].

The IP version 6 (IPv6) protocol [2] is the de-facto IP standard version for the IT/OT convergence. While the IP protocol has been used since the beginning at the networking layer of IT systems, OT needs to adapt not only to the IP standard itself, but also to the whole IP protocols suite [3], including among others the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [4], and the Constrained Application Protocol (CoAP) [5]. At the same time it is necessary to adapt the IP suite to match the constraints and requirements of industrial networks (e.g., monitoring systems, motion detection, control loops, etc.), which are deterministic by design, and different from traditional IP QoS-based networks. To this aim, new protocols and an overall architecture tying the adapted protocol suite together should be defined by Standard Developing Organization (SDO) [1].

Following this trend, a new Working Group, namely 6TiSCH [6], [7], has been created at the IETF to enable IPv6 over the deterministic Time Slotted Channel Hopping (TSCH) mode of the IEEE802.15.4e standard [8]. In the 6TiSCH architecture, low-power wireless devices form a multi-hop Low power and Lossy Networks (LLN), that is plugged into Internet through one or more LLN Border Routers (LBRs) [9].

Inside the LLN, nodes communicate by following a common schedule, which is a matrix of cells, each of them assigned to a pair of neighbor nodes for communicating at a given time, on a specific channel. The performance of the LLN (e.g., throughput, average packet latency, node energy consumption, network lifetime) are strictly dependent by the way this schedule is built.

The IEEE802.15.4e standard [8] defines the mechanisms to execute a communication schedule. At the same time, it leaves as out of its scope defining how the schedule is built, updated and maintained, and designating the entity in charge of performing these tasks. Therefore, one of the goals of the 6TiSCH Working Group (WG) is to develop a standard approach to manage this schedule and build it according to the network requirements [10].

6TiSCH has been recently working on the definition of a so-called "*minimal*" schedule, a static one, which is either pre-configured, or learnt by a node when joining the network. Such schedule mode does not exploit the full benefits of IEEE802.15.4e TSCH, but it can be used during network bootstrap, as a fallback mode of operation when no dynamic scheduling solution is available or functioning, or during early interoperability testing and development [11].

At the same time, 6TiSCH aims to investigate and develop centralized and distributed scheduling solutions. In the *centralized* case, the schedule is built by a Path Computation Element (PCE), a specific schedule entity, located into the Internet that continuously collects information from the network (e.g., traffic requirements from the nodes), in order to adjust the

N. Accettura is with "Berkeley Sensor & Actuator Center", University of California Berkeley, USA (e-mail: nicola.accettura@eecs.berkeley.edu).

E. Vogli, L. A. Grieco, and G. Boggia are with the "Dip. di Elettrotecnica ed Elettronica", Politecnico di Bari, v. Orabona 4, 7015, Bari, Italy (e-mail: e.vogli@poliba.it; a.grieco@poliba.it; g.boggia@poliba.it).

M. R. Palattella is with the SnT, University of Luxembourg, 4 rue Alphonse Weicker, L-2721 Luxembourg (e-mail: maria-rita.palattella@uni.lu).

M. Dohler is with King's College London (KCL), London, UK (e-mail: mischa.dohler@kcl.ac.uk).

Manuscript received ...

TSCH schedule accordingly. In the *distributed* case, there is no central entity, but nodes in the LLN agree on the schedule to be used, by applying distributed multi-hop scheduling protocols and neighbor-to-neighbor scheduling negotiation [12].

After the publication of the IEEE802.15.4e standard [8], both centralized and distributed scheduling approaches have been investigated by the research community, even before the birth of the 6TiSCH WG.

The first pioneer work [13] proposed a centralized solution, i.e., a Traffic Aware Scheduling Algorithm (TASA) that, by exploiting matching and coloring procedures, allows to schedule cells to all the nodes across the entire network topology graph. Briefly, with TASA the TSCH schedule is settled based on the network topology and the traffic load. In detail, the PCE uses the information related to the paths, coming from the routing protocol (e.g., RPL), and those related to the traffic (e.g., average traffic load generated by each node) in order to assign cells within the schedule, and provide the required level of QoS (duty cycle, throughput, etc.) to each active flow [14]. However, such scheduling technique triggers the exchange of a huge amount of signaling overhead, since each node in the network is supposed to communicate end-to-end with the PCE for both (i) sending topology and traffic load related information and (ii) receiving its own portion of the collision-free schedule. In addition, an underneath assumption is that the network churn is very low. This is somehow unrealistic also in non-mobile network scenarios, since the radio medium reliability is unpredictable in nature: to adapt the network topology to the best reliability available (i.e., the Packet Delivery Ratio (PDR), of each link), the routing protocol will change the routing topology, thus triggering schedule re-computations and more signaling phases with additional signaling overhead in the end.

As a counterpart, in the context of the OpenWSN project [15], some distributed approaches have also been studied. For instance, uRes [16] proposes the use of a negotiation process between neighbor nodes to schedule cells. In detail, uRes allocates cells minimizing the number of collisions, based on the knowledge of their neighbors schedule. Since collisions can still occur, they are resolved by re-allocating the colliding cells [17].

Recently, the Decentralized Traffic Aware Scheduling (DeTAS) technique [18] was proposed to address the scheduling needs of deterministic networks and was conceived to comply with the following guidelines:

1) ensure the smallest end-to-end latency between data generation and its reception at the application sink (i.e., the root node in a RPL-organized network);
2) keep the queue utilization as small as possible, through a strict alternation of transmitting and receiving cells into the TSCH slotframe structure;
3) use neighbor-to-neighbor signaling for gathering minimal information about the network and traffic features and for distributing minimal information to compute a collision-free schedule, thus bounding the signaling overhead;
4) compute deterministic time schedules in a decentralized fashion to manage networks rooted to multiple coordinated sinks, while leaving the channel offset computation based on the RPL rank of each node in the network.

The guideline expressed by item 1) was also exploited in designing TASA, while the other ones are inherent to DeTAS. In [18], it has been shown that the guideline of 2) allows nodes' queues to be almost empty, as a natural consequence of a better schedule organization w.r.t. to that of TASA. To shed some light on these and other features, after having introduced in Sec. II the 6TiSCH architecture as the technical landscape behind DeTAS, we briefly recall its theoretical design in Sec. III.

With regard to item 3), in this paper a DeTAS neighbor-to-neighbor signaling is described in more details in Sec. IV. For the sake of completeness, we also mention here the work of Morell *et al.* [19], which defined a form of neighbor-to-neighbor signaling exchanges by exploiting the concept of label switching in TSCH networks and proposing the use of reservation to establish and manage tracks between nodes in the network. In that proposal, the TSCH schedule is built by collecting information along the track and installing it during the downstream reservation message, in a RSVP-like fashion [17]. It has to be noted that the neighbor-to-neighbor RSVP-like signaling computes the scheduled resources along a given path between a node in the network and the traffic sink. Instead, the neighbor-to-neighbor DeTAS signaling computes the scheduled resources for the routing sub-tree rooted at a given node in the network: that node spreads aggregated schedule information which are in turn hop-by-hop unbundled to compute the schedule of each node in the sub-tree. This feature is very important, because the signaling overhead is significantly small, since the schedule computation is distributed into the network with each DeTAS-enabled neighbor being able to decide how the schedule of its RPL-children has to be built.

It is worth to note that the original DeTAS proposal in [18] was only formulated as a high level algorithm design and many missing items have been contributed in this manuscript to let DeTAS become a real protocol ready to use in industrial plants and beyond. They include a lightweight signaling protocol, the implementation in the OpenWSN stack, and an experimental evaluation which actually proofs the theoretical findings of the original DeTAS formulation. Thanks to these contributions it is now possible to claim that DeTAS is a real protocol ready to be used and customized in different industrial settings. And thanks to the open freely available implementation in OpenWSN, it can be tested before deployment.

As a major contribution, in Sec. V we deeply show and analyze DeTAS performances in terms of end-to-end latency, reliability and duty-cycle by means of experimental results. Specifically, we only considered single-sink topologies and discovered that some assumptions on the channel offset reuse, made in [18] and cited in the guideline 4), are somehow unrealistic in practical scenarios. In this paper we show that a neighbor-to-neighbor signaling can be used also to overcome this issue. Finally, Sec. VI draws concluding remarks and pictures the envisaged future works.

## II. 6TiSCH Architecture

The IEEE802.15.4 PHY protocol [20] has been the de-facto standard with the longest-standing impact in low-power wireless mesh technology, and has been widely used by low-power battery-powered devices to build LLNs.

The need to interconnect IEEE802.15.4-based low power networks to the Internet triggered the birth of various WGs within the IETF, including 6LoWPAN [21] - now 6lo [22] -, ROLL (the group behind the RPL routing protocol [4]), and CORE (behind the CoAP web transfer protocol [5]) that have defined how to fit an IPv6 protocol stack on top of IEEE802.15.4.

The IEEE802.15.4e standard [8] was published in 2012 as an amendment of the IEEE802.15.4-2011 Medium Access Control (MAC) protocol only [20]. In other words, it did not amend the physical layer and therefore, it can still operate on any IEEE802.15.4-compliant hardware.

Despite that, the IEEE802.15.4e TSCH MAC mode – which is the main focus of the activities within the 6TiSCH group, and also of our work – is very different from the "legacy" IEEE802.15.4 MAC protocol. TSCH combines time synchronization with channel hopping, to achieve ultra low-power operation and high reliability, respectively. Moreover, unlike the industrial standards (i.e., WirelessHART [23] and ISA100.11a [24], [25]) from which it inherits, the IEEE802.15.4e TSCH focuses exclusively on the MAC layer. This clean layering lets TSCH fitting under an IPv6-enabled "upper" protocol stack.

Given the aforementioned appealing features of the IEEE802.15.4e TSCH, members of both academia and industry have created a new Working Group, 6TiSCH, at IETF to build IPv6-enabled LLNs, rooted in the IEEE802.15.4e TSCH MAC layer. The final aim of 6TiSCH consists in filling the gaps between IEEE lower layers of an industrial IoT protocol stack and the IETF higher layers, to enable an open standards-based protocol stack for deterministic wireless mesh networks.

As described in [26], when possible, the 6TiSCH architecture will reuse existing protocols such as IPv6 Neighbor Discovery (ND) [27], IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [21], and the RPL [4], with the minimum adaptation required to meet criteria for reliability and determinism within the mesh, and scalability over the backbone. 6TiSCH will fill the missing gaps within the architecture, so that IETF 6LoWPAN Header Compression and RPL, which enables respectively IPv6 encapsulation and routing, can optimally operate on top of the TSCH MAC layer.

### A. The MAC layer: IEEE802.15.4e TSCH

The IEEE802.15.4e TSCH is suitable for deterministic traffic, i.e., traffic flows with an emission rate and routing path patterns that are well-known in advance. In fact, it combines together Time Division Multiplexing (TDM), time synchronization and time formatted into slotframes, resulting in a *deterministic* wireless MAC standard.

All nodes in a TSCH multi-hop network are synchronized. Time is sliced up into timeslots which are grouped into one or more slotframes. A slotframe continuously repeats over time, and its duration can be fixed to meet the application requirements (e.g., available bandwidth, lower latency, power consumption). In a TSCH network, the bandwidth is pre-formatted in a TDM fashion. Thus, unlike the traditional CSMA/CA-based networks, there is no contention for gaining access to the channel (unless allowed explicitly in some specific timeslots, as detailed in the next paragraph).

Due to its scheduled nature, all nodes follows a common **schedule**. The latter is a matrix of scheduled cells, each of them identified by a `slotOffset`, and a `channelOffset` [28]. A cell represents an atomic unit of bandwidth that can be allocated by a centralized or distributed scheduling algorithm. A cell can be dedicated or shared: dedicated cells (those scheduled by DeTAS, as detailed in this paper) are assigned to the communication of a pair of neighbors; shared cells are used by more then two communicating neighbors in a CSMA/CA fashion (e.g., the static "*minimal*" configuration [11] deals only with this kind of cells).

Because of the channel hopping nature of TSCH, the scheduling algorithm does not care of the actual frequency the communication happens on, since it changes at each slotframe iteration. In fact, the `channelOffset` is translated into a frequency using a specific translation function which implies communicating neighbors to "hop" between the different available frequencies when exchaning data. Such channel hopping technique efficiently combats multi-path fading and external interference.

By following the schedule, each node knows when (i.e., at which time slot), and on which channel (based on the channel offset) it can exchange (either transmit or receive) data with its neighbor nodes.

### B. The routing protocol: RPL

The RPL Routing Protocol [4] plays a key role in the 6TiSCH architecture, in that it organizes the low power mesh in form of a Directed Acyclic Graph (DAG), rooted at a small set of LLN sinks. For each sink, a Destination Oriented DAG (DODAG) is created by accounting for link costs, node attributes/status information, and an Objective Function, which maps the optimization requirements of the target scenario. In this paper, we consider the simplest topology with a single sink, also referred as $DODAGroot$. Although RPL can manage several kinds of traffic flows (to and from the $DODAGroot$ or between any pair of nodes in the network), we have focused on the dominant multipoint-to-point traffic, i.e., that flowing from the nodes in the network towards the $DODAGroot$ and more related to monitoring applications in industrial environments.

RPL employs a gradient strategy, which introduces the concept of *rank* to define the individual position of a node with respect to its $DODAGroot$. A fundamental property of a RPL-organized network is that the rank should monotonically decrease along the DODAG and towards the destination, in accordance to the gradient-based approach. In general, the rank is computed based on path metrics, but it is used to let the routing topology being loop-free. In details, the rank is a 2-bytes value, whose most significant byte, called $DAGrank$,

is used to compare the position of nodes within the network. As an example, if the $DAGrank$ of a node A is lower than the $DAGrank$ of another node B, A could be safely a parent for B; if the $DAGrank$ of two neighbors tie, none of them can include the other as its own parent.

RPL can adopt several metrics for computing the rank and the $DAGrank$. In designing DeTAS, we have forced the $DAGrank$ to be computed according to the minimum hop distance metric. According to the standard [4], the $DODAGroot$ is assigned a $DAGrank$ equal to 1. Hence, the $DAGrank$ of given source node in the network is exactly equal to the minimum hop distance from the $DODAGroot$ augmented by 1. However, this approach can be easily extended to account for different metrics, e.g., the Expected Transmission Count (ETX).

## III. DeTAS Algorithm Description

The DeTAS algorithm has been designed for building optimum collision-free schedules in multi-hop IEEE802.15.4e TSCH networks. Using a tiny amount of information, locally exchanged among neighbor nodes, DeTAS allow to compute the schedule in a distributed manner. Whilst scheduling the traffic, DeTAS manages queue levels, avoiding traffic congestion and, thus, possible packet drops due to overflow of nodes' memory buffer. Finally, DeTAS can exploit the availability of the 16 IEEE802.15.4 frequencies [8] in order to: parallelize several transmissions at the same time, reduce the number of active slots per slotframe (i.e., the network duty cycle), and increase the reliability of wireless links.

In a 6TiSCH network running DeTAS, all devices are assumed to be synchronized with the same slotframe, having size equal to $S$ time slots. Moreover, all nodes follow a common TSCH schedule having width equal to $L$ timeslot, with $L \leq S$, and height equal to $W$ channel offset, with $W \leq 16$. Such schedule is set up minimizing its length, i.e., the number of active slots, $L$, needed for correctly delivering the network traffic (expressed as number of packets per slotframe) to the $DODAGroot$. More specifically, all transmissions are scheduled in consecutive $L$ slots, leaving the remaining $S-L$ slots within the slotframe available for packet transmissions related to other applications (e.g., other RPL instances). The length $L$ of the schedule is computed by the $DODAGroot$ using the formulation introduced in Sec. III-C. Instead, the width of the schedule, i.e., the number of available channel offsets, is at least $W = 3$, since frequencies can be reused every 3-hops, thus avoiding also collisions due to interference[1] [29],[30].

### A. Network Topology and Traffic

Being designed for 6TiSCH networks, hereafter we will use the 6TiSCH terminology [28], while describing the DeTAS technique. Fig. 1 shows an example network which is a destination-oriented tree graph coordinated by the $DODAGroot$. Given a network with $N$ source nodes, let $\{n_i\}$, with $i = 1, \ldots, N$, be the set of all source nodes, and $n_0$ the $DODAGroot$. Furthermore, for each node $n_i$ it is possible to identify the set $ch(n_i)$ of its children, and the sub-tree $ST_i$, composed by $n_i$ itself and all the nodes connected to it through multi-hop paths. Moreover, Table I summarizes the notation used throughout this paper.

[1]In our experiments (reported in Sec. III) we tested several values of $W$ parameters in the range $[3, 12]$.

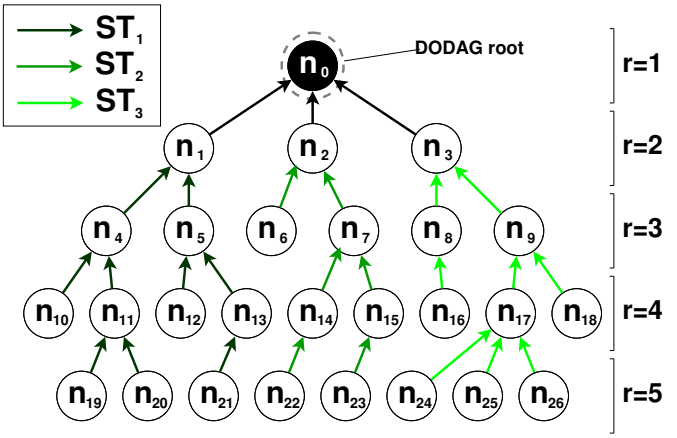



Fig. 1: Example of a LLN with a $DODAGroot$ acting as application sink and 26 source nodes ($r$ is the $DAGrank$).

TABLE I: List of used symbols.

| *Symbol* | Definition |
|---|---|
| $S$ | Number of slots within a slotframe |
| $L$ | Length of DeTAS schedule in slots |
| $W$ | Height of DeTAS schedule in number of channel offset |
| $N$ | Number of source nodes in the network |
| $n_0$ | LLN sink node |
| $n_i, i = 1, ..., N$ | Source nodes |
| $ch(n_i)$ | Children nodes of the node $n_i$ |
| $ST_i$ | Subtree rooted to the node $n_i$ |
| $q_i$ | Data traffic generated on the node $n_i$ (local packet number) |
| $Q_i$ | Data traffic generated from the subtree $ST_i$ (global packet number) |
| $p_i$ | Parent node of the node $n_i$ |
| $r$ | $DAGrank$ of nodes representing the distance in number of hops from the $DODAGroot$ augmented by 1 |
| $Q_M$ | Maximal global packet number among the children of the $DODAGroot$ |
| $n_M$ | Child of the $DODAGroot$ having the maximal global packet number $Q_i$ |
| $q_M$ | Local packet number of the node $n_M$ |
| $\alpha$ | Number of transmit slots scheduled consecutively in the node $n_M$ |
| $L_e$ | Length of the schedule of the even list |
| $L_o$ | Length of the schedule of the odd list |
| $Q_0$ | Data Traffic generated from all the source nodes |
| $Q_0^e$ | Data Traffic generated in the subtrees rooted to nodes in even list |
| $Q_0^o$ | Data Traffic generated in the subtrees rooted to nodes in odd list |
| $\beta$ | The parameter that balances even and odd data traffic ($Q_0^e, Q_0^o$) |
| $n_{cut}$ | Node which schedule is divided between the even and odd list |
| $ST_{cut}$ | Subtree rooted at the node $n_{cut}$ |

DeTAS is a traffic-aware algorithm that builds the schedule based on the traffic generated by each source node. We assume that the network supports a multipoint-to-point traffic. In

detail, every source node in the set $\{n_i\}$ generates a constant[2] integer number of packets, i.e., the *local packet number*, $q_i$, within a slotframe, destined to the $DODAGroot$. For each node $n_i$ belonging to the network with $i = 1, \ldots, N$, we define also the *global packet number*, $Q_i$, as the total amount of packets generated within a slotframe by the nodes belonging to the sub-tree $ST_i$.

### B. Decentralized scheduling

To setup the schedule in a distributed manner each node $n_i$ needs to know some traffic information: (a) the amount of traffic that it will receive from its children, and (b) the amount of traffic it will transmit to its parent node $p_i$. Therefore $n_i$ computes locally its *global packet number*, $Q_i$, as the sum of its *local packet number*, $q_i$, and the *global packet numbers* of its children, and then forward such information to its parent node, $p_i$. In a recursive way, thanks to the information exchanged at one hop distance, the $DODAGroot$, $n_0$, will be able to calculate the overall traffic of the subtree, $Q_0$ as well as the *local packet numbers* of its children. Starting from the aforementioned traffic information, exchanged at 1-hop distance, the schedule is built in a distributed fashion, where each node, $n_i$, allocates some slots within the schedule to its children. The built schedule is collision free for the whole network and keeps the queue utilization as low as possible. In what follows, we detail the rules adopted in designing DeTAS and the resulting scheduling technique.

*a) Scheduling the Slot-Offsets:* First of all, each DeTAS-enabled node in the network schedules the transmission slot in such a way to be synchronized with the reception slot of its parent. Afterwards it allocates on its own which cells it will use for receiving packets from its own children. In fact, a node $n_i$ does not perform any scheduling decision until it is informed by its parent $p_i$ about the $Q_i$ cells to be allocated as transmitting (or *tx*) slots. The allocation of *rx* slots to the children is performed in a recursive way, starting from the $DODAGroot$ and going downward towards the leaf nodes.

Once $n_i$ has been made aware of which are its *tx* cells, it can decide its $Q_i - q_i$ *rx* cells (i.e., those needed for receiving packets from its children) and it makes sure that these two sets of cells are not overlapped. In particular the *tx* and *rx* cells are alternated (i.e., as depicted in the example in Fig. 2), which means that if a type of cell is scheduled in an even slot the other type should be scheduled in the next odd slot. Then, $n_i$ splits its *rx* cells in subsets, with each subset being assigned to a child node. In order to fulfill the requirements each child, the corresponding subset is sized according to the supplied *global packet number*. In the end, each child is made aware by $n_i$ about the assigned subset of cells, and will configure such cells as *tx* ones. In the example of Fig. 2 the node $n4$ splits its subset of *rx* slots in two subsets, which in term should coincide with the set of their *tx* slots. *Such policy does not allow two nodes having a common parent to transmit using the same cell, thus it overcomes the hidden terminal problem.*

[2]Such an assumption is supported by the fact that traffic is typically different between sensors, but relatively constant over time in emerging heterogeneous embedded applications. Actually, with an efficient signaling, DeTAS could also support variable bit rate traffic flows.

To avoid any kind of collisions, DeTAS does not allow two nodes with the same $DAGrank$ to schedule transmissions in the same timeslot. With this feature DeTAS can build collision-free schedule, even being agnostic about the physical connectivity between nodes in the network. *Eventually, DeTAS overcomes the exposed terminal problem too.*

Interestingly, by alternating *tx* and *rx* cells, the queue of a node is emptied of 1 packet, as soon as it receives a new one, and viceversa. *Thus, buffer overflow is avoided and queue utilization is kept as low as possible.*

Obviously, to allow packets to be correctly sent and received, the *tx* cells of the children $\in ch(n_i)$ must be synchronized with the *rx* cells of $n_i$.

For the sake of clarity, and in order to add some formalization to the description, we introduce the following definitions.

*Definition 1:* A node $n_i$ is *even-scheduled*, if its *tx* cells are located in *even* positions within the scheduling interval, while its *rx* cells are located in *odd* positions.

*Definition 2:* A node $n_i$ is *odd-scheduled*, if its *tx* cells are located in *odd* positions within the scheduling interval, while its *rx* cells are located in *even* positions.

A sub-tree $ST_i$ rooted at a node $n_i$ can be *even-* or *odd-*scheduled, according to the following additional definitions.

*Definition 3:* A subtree $ST_i$ is *even-scheduled*, if all nodes $\in ST_i$ with *even* $DAGrank$ are *even-scheduled* and those with *odd* $DAGrank$ are *odd-scheduled*.

*Definition 4:* A subtree $ST_i$ is *odd-scheduled*, if all nodes $\in ST_i$ with *even* $DAGrank$ are *odd-scheduled* and those with *odd* $DAGrank$ are *even-scheduled*.

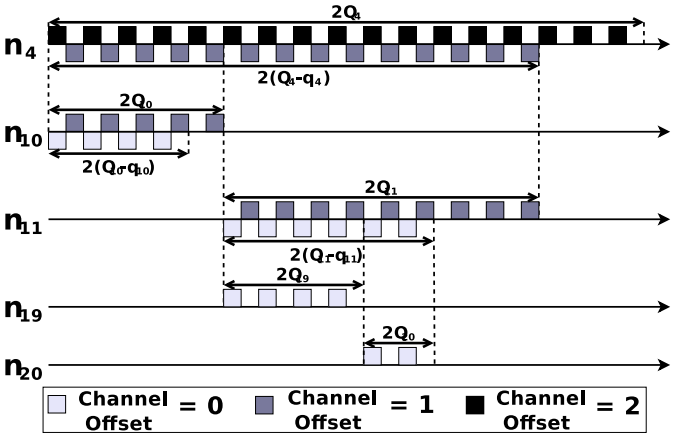



Fig. 2: *Odd*-schedule for nodes belonging to $ST_4$ of the destination oriented tree in Fig. 1 (note that $n_4$ has an even $DAGrank$).

Therefore Fig. 2 shows an example of an *odd schedule* followed by the sub-tree $ST_4$ of the destination oriented tree in Fig. 1, and, details the time slots reserved to the nodes $n_4$, $n_{10}$, $n_{11}$, $n_{19}$ and $n_{20}$. We can see that $n_4$, having *odd* $DAGrank$ is *even-scheduled* and spends the $Q_i$ *even* time slots in transmission, the first $Q_i - q_i$ *odd* time slots in reception, and the last $q_i$ *odd* time slots in idle (having already received all the packets from its children $n_{10}$ and $n_{11}$).

*b) Scheduling the Channel-Offset:* Besides the allocation of *tx* of *rx* cells, the channel offset is changed at each hop

and the same channel offset is reused only after $W$-hops. In particular, for each source node $n_i$, DeTAS allocates *tx* cells with a channel offset equal to $[(DAGrank-2) \mod W]$ (for transmitting packets to the correspondent parent $p_i$), and *rx* cells with channel offset equal to $[(DAGrank-1) \mod W]$ (for receiving packets from children $\in ch(n_i)$). If $W = 1$, there is a single channel offset used for the whole schedule, hence this setting does not exploit the bandwidth increase available with multiple channel offsets and collisions can occur when two or more motes are transmitting at the same time. If $W = 2$, packet collisions can happen when a node $n_i$ and its parent $p_i$ are respectively receiving (from a child) and transmitting (to the parent) at the same time. With $W \geq 3$, the channel offset is reused at least every 3 hops. This means that the absolute difference between the $DAGrank$s related to two nodes transmitting in the same cell (i.e., on the same timeslot and channel offset) can be 0 or $\geq 3$. In the latter case, there will be ideally no collision, due to the minimum hop count metric used for building the destination oriented tree. Instead, if the aforementioned difference is 0, some collision due to mutual interference can occur.

*c) Considerations:* It is worth noting that a node $n_i$ can accomodate all the schedule (i.e., *tx* or *rx* cells) in a *scheduling interval* of $2Q_i$ consecutive slots within the slotframe. In fact, within this interval, the $Q_i$ even (odd) slot offsets could be used for scheduling *tx* cells (i.e., the needed cells to deliver $Q_i$ to the parent $p_i$), while the first $Q_i - q_i$ odd (even) slot offsets could be used for scheduling *rx* cells (i.e., the cells needed by $n_i$ for receiving $Q_i - q_i$ packets from the other nodes $\in ST_i$). In the remaining $q_i$ odd (even) slot offsets, the node will be idle. Hence, the node $n_i$ must be informed only about the lowest boundary of such interval and about the policy for alternating *tx* and *rx* cells. *Such information can be carried with a small overhead, with the additional advantage of having an easy management of allocated resources into constrained devices.*

Moreover the nodes in a scheduled sub-tree allowed to transmit in the same timeslot have all an *even* (or an *odd*) $DAGrank$. Moreover, with such scheduling technique, it is not possible that two nodes with the same $DAGrank$ can transmit simultaneously. As a consequence, it is possible to schedule simultaneously two sub-trees rooted at the $DODAGroot$, with one being *even-scheduled* and the other one being *odd-scheduled*. The schedules will be perfectly interleaved, with at most a single node per $DAGrank$ being allowed to transmit packets to its parent.

### C. Schedule length L

Even though the schedule is built with a distributed approach, it is initialized by the $DODAGroot$, that computes the length $L$ of the schedule, and selects, among the sub-trees rooted at its children, the ones to be *even*-scheduled, and thus, those to be *odd*-scheduled.

Being DeTAS a traffic-aware scheduling algorithm, the length, $L$, of the schedule is a function of the network traffic. In fact, the $DODAGroot$ can receive at most one packet per time slot (i.e., $L \geq Q_0$). At the same time, the child of the $DODAGroot$, referred hereafter as $n_M$, having the maximum *global packet number*, i.e., $Q_M = \max_{n_i \in ch(n_0)} Q_i$, will need a schedule long enough for containing $Q_M$ transmit slots and $Q_M - q_M$ receive slots, i.e., $L \geq 2Q_M - q_M$. For every randomly scattered physical topology, the proposed DeTAS scheme is able to find the optimum schedule with the minimum length, given by:

$$L = \max \{2Q_M - q_M,\ Q_0\} . \tag{1}$$

To this aim, DeTAS simultaneously guarantees that a single sub-tree rooted at a sink's child is *even*-scheduled, while the sub-tree rooted at another sink's child is *odd*-scheduled, thus, they will not incur in any kind of collision, since the related schedules are perfectly interleaved. Therefore, a $DODAGroot$ running DeTAS must divide its children in two lists, i.e., an *even* list and an *odd* one. The sub-trees rooted at the children in the *even* list could be sequentially *even*-scheduled, in a time interval long $L_e$ time slots. At the same time, the sub-trees rooted at the children in the *odd* list could be sequentially *odd*-scheduled, for a time interval long $L_o$ time slots. Since the schedules associated with the two lists can be perfectly overlapped, the longest schedule between the two determines the length $L$ of the whole network schedule.

However, for allowing the coexistence of several applications sharing the same slotframe structure, the schedule length related to a given application running on a network must be bounded to the minimum possible given by eq. (1). In this case, the $DODAGroot$ can exactly calculate the length of the schedule based on the information about the *global* and *local packet numbers* provided by its own children.

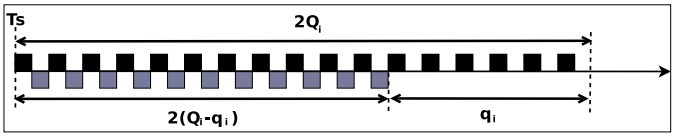
As a consequence, DeTAS has to let the even and odd lists be load balanced, in order to get their schedule lengths as close as possible. This load balancing problem falls into the class of *multiprocessor scheduling problems* [31]. The greedy heuristic employed in this paper is the same described in [32]: the $DODAGroot$'s children are ordered in a descending order, according to their *global packet number*. Then, they are appended subsequently to the list (*even* or *odd*) with the current smallest sum of *global packet numbers*.

DeTAS assumes a different behavior depending on the traffic loads of the children of the $DODAGroot$. Specifically, if $Q_M \geq Q_0/2$, the node $n_M$ will obviously be the only one in the *even*-list. With DeTAS, the sub-trees rooted at the nodes in the *odd*-list will be subsequently *odd*-scheduled. At the same time, the sub-tree rooted at $n_M$ will be *even*-scheduled for the first $2(Q_M - \alpha)$ slots, with

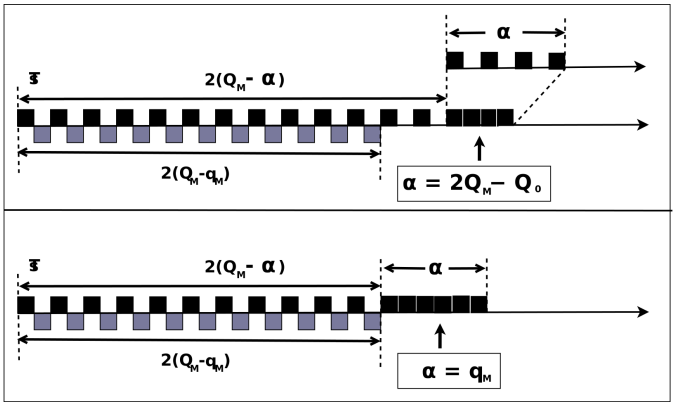
$$\alpha = \min\{2Q_M - Q_0, q_M\}; \tag{2}$$

subsequently, the schedule of the node $n_M$ will contain $\alpha$ additional consecutive slots for delivering $\alpha$ packets to the $DODAGroot$. In the schedule related to $n_M$ illustrated in Fig. 3(b) it can be noticed that the first $2(Q_M - \alpha)$ slots are even scheduled (i.e. alternated transmit and receive slots), whereas the remaining $\alpha$ slots are scheduled consecutively. Moreover in [18], it has been shown that, with this technique, eq. (1) is always fulfilled when $Q_M \geq Q_0/2$.

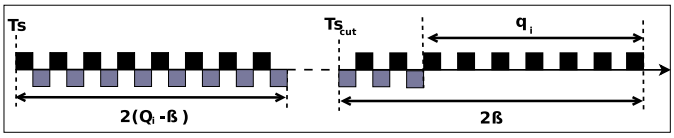
The previous strategy is applied in some bounded cases, i.e., when a child node of the $DODAGroot$ is the bottleneck




(a) Schedule pattern of a generic node of the network.




(b) Schedule pattern in the case $Q_M \geq Q_0/2$.




(c) Schedule pattern of a node in the case $Q_M < Q_0/2$ and the $Q_0^e \neq Q_0^o$, which is divided between the even and odd list.

Fig. 3: Schedule patterns of a node $n_i$.

for at least one half of the traffic offered by the network. As a matter of fact, the $DODAGroot$ will have more than two children in dense network deployments, and the routing protocol will load balance the traffic among the children. Hence, the most common network scenario entails the case $Q_M < Q_0/2$, i.e., the child of the $DODAGroot$ having the maximum *global packet number*, $n_M$, will manage less than one half of the traffic flowing towards the $DODAGroot$. In this case, many techniques could be found to load balance the *even* and *odd* list. We found that the easiest solution is to perform the following operations: (i) for each list calculate the sum of the *global packets numbers* related to the root nodes of the subtrees in the list itself; (ii) select the list with the biggest sum; (iii) within this list, select the subtree rooted at the node, $n_{cut}$, with the highest *global packets number*; (iv) split the schedule of $ST_{cut}$ in two parts, so that the first part will be placed at the beginning of the network schedule according to the same scheduling parity of the selected list, while the second part will be scheduled in the end of the network schedule according to the opposite scheduling parity. In this context, DeTAS decides the sizes of the two parts of the schedule related to $ST_{cut}$: since the target is to compute the schedule with the minimum length (i.e., $L = Q_0$), DeTAS has to make sure that the resulting lengths of the *even* and *odd* schedules are as close as possible.

In details, DeTAS reduces the discrepancy between the sum $Q_0^e$, computed over the *global packet numbers* related to the *even*-list, and the sum $Q_0^o$, computed over the *odd*-list, to 1 (if $Q_0$ is odd) or 0 (if $Q_0$ is even). It is worth to note that $Q_0^e + Q_0^o = Q_0$. The $DODAGroot$ computes the value $\beta$, which balances the two lists:

$$\beta = \left\lfloor \frac{Q_0^e - Q_0^o}{2} \right\rfloor . \quad (3)$$

If $\beta \geq 0$ ($\beta < 0$)[3], DeTAS will *even*-schedule (*odd*-schedule) firstly the sub-tree $ST_{cut}$ for $2(Q_{cut} - |\beta|)$ time slots and, then, all the sub-trees related to the other nodes appended in the *even*-list (*odd*-list). Simultaneously, it will *odd*-schedule (*even*-schedule) firstly all the sub-trees related to the nodes in the *odd*-list (*even*-list), then the sub-tree $ST_{cut}$ for exactly $2|\beta|$ time slots. The resulting schedule length is $L = Q_0$, as already shown in [18]. Note that in this traffic load conditions, $n_{cut}$ should be informed by the $DODAGroot$ about when to *even*- and when to *odd*-schedule $ST_{cut}$. This information must be accordingly updated and propagated along $ST_{cut}$. For the sake of clarity, Fig. 3(c) sketches an example schedule for $n_{cut}$.

## IV. DeTAS implementation

In order to evaluate the performance of DeTAS we have implemented it in the OpenWSN project [15]. OpenWSN is an implementation of a standards-based stack and, to the best of our knowledge, it is the first and unique open-source implementation of the IEEE802.15.4e TSCH standard. On top of IEEE802.15.4e TSCH, OpenWSN implements Internet of Things-related standards [3], namely 6LoWPAN, RPL and CoAP. In order to schedule some cells, each node must be able to exchange information with its parent and children. It should transmit the $Q_i$ and $q_i$ parameters to the parent node and receive back the necessary information. In a DeTAS enabled network, the portion of the network schedule related to a given node is shaped according to one of the three patterns described in Fig. 3 and discussed in the previous section. The amount and type of information needed from a node to build its own schedule will depend strictly on the schedule pattern of the node. However, despite some differences, the following parameters are used by all configuration patterns:

- $Ts$: slotoffset of the first cell allocated for that node and it indicates where to start scheduling cells for its communication;
- $EO$: parameter specifying if the sub-tree that a node belongs to is *even* or *odd*-scheduled;

In the following we describe in more details the different node patterns, by indicating the parameters needed by a DeTAS-enabled node to build them.

- PATTERN 1: it is the most common node schedule installed by DeTAS, and it is pictured in Fig. 3(a). For building such node schedule, a node needs to get from its parent only the data set $\{Ts, EO\}$. The schedule will be built by simply alternating $Q_i$ *tx* cells with $(Q_i - q_i)$ *rx* cells, starting from $Ts$. The parity of the schedule is computed based on the $EO$ parameter and on the $DAGrank$.

[3]In the rest of the paragraph, we indicate the alternative case and the related settings in brackets.

- PATTERN 2: this pattern is used when $Q_M \geq Q_0/2$. Fig. 3(b) shows a example of such pattern. As already detailed in the previous section, this pattern can be held by a single node in the network, i.e., the child of the $DODAGroot$ with the highest *global packet number*. Such node will firstly *even*-schedule ($Q_M - \alpha$) cells, than will schedule $\alpha$ consecutive *tx* cells, with $\alpha$ given by eq.2. In addition to the common data set, the $\alpha$ parameter is needed in order to schedule the final consecutive cells. Therefore the related data set is given by $\{Ts, EO, \alpha\}$.
- PATTERN 3: this schedule (pictured in Fig. 3(c)) can be present on a node when $Q_M < Q_0/2$ and $Q_0^e \neq Q_0^o$ are both satisfied. As described in section III, the $n_{cut}$ node will allocate its cells in two parts. The first part, composed by $Q_{cut} - \beta$ cells, will be *even* (*odd*) scheduled, while the second part, composed by $\beta$ cells, will be odd (even) scheduled, with $\beta$ being set according to eq.(3). In addition to the common data set, a node needs to be informed about the $\beta$ parameter, for calculating the length of the two parts of its schedule, and the starting slot offset $Ts_{cut}$ for the second part of the schedule. Therefore the complete data set in this case is $\{Ts, EO, \beta, Ts_{cut}\}$. It is worth noting that such kind of schedule can be present on a single node per $DAGrank$ in the network.

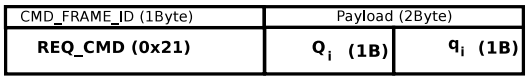
DeTAS must be able to recompute schedules when nodes join or leave the network. To this aim, a DeTAS Version Number (DVN), similarly to the DODAGversion number of RPL, is managed by the $DODAGroot$ to control the schedule version. Such parameter is initialized to 0 at the network bootstrap and than incremented each time the $DODAGroot$ triggers a new schedule distributed computation. Indeed, a new schedule computation is needed during network formation and whenever the topology and the traffic conditions change.

### A. DeTAS MAC command frames

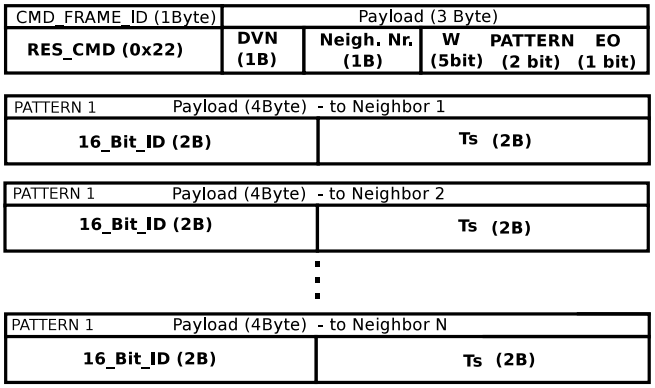
The DeTAS information exchange has required the definition of two "ad hoc" MAC command frames. Although the IEEE 802.15.4e amendment introduces the Enhanced Beacons (EBs), which could be used for exchanging minimal information about the schedule, command frames are more appropriate to DeTAS for the inherent possibility of quickly building the schedule. For the sake of completeness, we mention that the 6TiSCH working group is defining new Information Elements (IEs) to be used for cell negotiation and scheduling [33]. However, the schedule objects defined in such draft are still not suitable for the requirements of DeTAS. One of the aim of this paper is to highlight another form of schedule object, that eventually could be integrated in such working draft.

According to the signaling required by DeTAS, a node $n_i$ has to transmit to its preferred parent, $p_i$, some information about its $Q_i$ and/or $q_i$ parameters. For this purpose, the Request MAC command frame (REQ) has been defined and structured as shown in Fig. 4(a). This command frame is sent as unicast only to the preferred parent, either when a node joins the network or when $Q_i$ changes (i.e., in the case the node $n_i$ receives a new REQ from one of its children).

In turn, with a broadcast Response MAC command frame (RES), a parent can instruct its own children about the rules

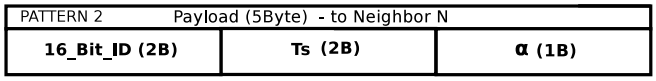



(a) REQ Command frame format.

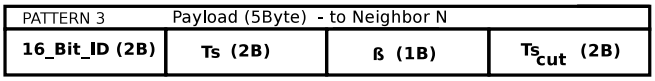



(b) RES Command frame format in the PATTERN 2 case.

Fig. 4: Frame formats for the REQ and RES commands.




(a) Format for the PATTERN 2 case.




(b) Format for the PATTERN 3 case.

Fig. 5: Format of the last neighbour (n-th) field for the RES command.

to be followed for building the DeTAS schedule. The RES payload contains detailed information for each of the children receiving the message, as shown in Fig. 4(b).

As introduced earlier, the amount of scheduling information to be sent by a node to a specific child depends on the pattern shape that the child must follow when building its schedule. The most common pattern to be communicated is PATTERN 1, which requires 4 bytes per child (2 bytes for the node ID and 2 bytes for the Ts time offset). It has to be noted that a node can have only a single child to be made aware about a different pattern: a single child of the $DODAGroot$ can be selected to schedule according to PATTERN 2 (which requires a 5-bytes long field); a single node per $DAGrank$ can be selected to schedule according to PATTERN 3 (which requires a 7-bytes long field), so that a single child of a given node can be selected to schedule according to PATTERN 3. In Fig. 4(b) it is shown the RES command format in the case where all the neighbors have PATTERN 1 schedule. If a neighbor has PATTERN 2 (or a PATTERN 3 ) schedule, the last field of the RES command frame (i.e., the data related to the N-th neighbor), is substituted with the field shown in Fig. 5(a) (or in Fig. 5(b)). Therefore the RES command payload contains:

(i) three bytes where are stored common information to all the neighbors; (ii) N-1 fields with 4 bytes each, required for the PATTERN 1 scheduled neighbors; (iii) The last field which can be 4, 5 or 7 bytes according to the particular PATTERN scheduled for the last neighbor.

As a consequence, when a RES command frame is built by a node, the common information useful to all neighbors is appended at the beginning of the payload: (i) the DVN, (ii) the number of neighbors that are notified in the command, (iii) the channel reuse factor $W$ (whose setting has been widely described and analyzed in Sec. V), (iv) the schedule PATTERN of the last neighbor, and (v), finally, the $EO$ flag (which is common to all children). The scheduling information related to each child is then appended, making sure that, if a child has to be scheduled according to PATTERN 2 or PATTERN 3, the correspondent field will be appended as last. This technique is used for a fast parsing of data on the receiving side. Note that the length of the RES command frame depends on the number of children and on the scheduling pattern related to the last field appended.

### B. DeTAS Signaling

The command frames described in the previous subsection are sent by a node according to some triggering events and trigger other events when received on a given neighbor. In the following list we describe the protocol used to exchange such command frames, thus making DeTAS a viable solution for scheduling in large networks; in details:

- **REQ transmission:** the REQ command is sent from a node $n_i$ to its preferred parent $p_i$. This event is triggered: (i) when the node has just joined the network and it has not yet a DeTAS schedule; (ii) when traffic load of the node has changed (i.e., $Q_i$ or $q_i$) and, therefore, it requests a new schedule; (iii) when the node has recognized that it is using a DVN which is smaller than the one advertised from the LLN sink. When $n_i$ sends a REQ, it waits for receiving a RES command from $p_i$. It may happen that one between the REQ and RES commands is lost. In order to cope with this issue, the node $n_i$ will transmit periodically REQ commands until it receives a RES command with the scheduling information required.
- **REQ reception:** a node $n_i$ receives this command from one of its children $\in ch(n_i)$ in two cases: (i) when the child is asking for a schedule; (ii) the child recognizes to have an outdated schedule and sends a REQ command to get an update. In the first case (i), if $n_i$ is not a $DODAGroot$, it will update its own $Q_i$ parameter and trigger a REQ transmission to its parent $p_i$. Hence, at each hop a new REQ will be created until the $DODAGroot$ is reached. When the $DODAGroot$ receives a new request, it calculates the new schedule with the updated data and triggers a RES command transmission. In case (ii), the node $n_i$, will send a RES command containing the information necessary to build the current schedule.
- **RES transmission:** the RES command is sent from a node to all its children as a broadcast frame. Four condition can trigger the transmission of a RES command: (i) the $DODAGroot$ has just (re)computed the schedule after receiving a REQ; (ii) a non-$DODAGroot$ node has just processed a RES command frame and has (re)computed the schedule for its children; (iii) an update request has been received; (iv) a node recognizes that one of its children has not received a schedule belonging to the current DVN, thus it sends an update with the right schedule.
- **RES reception:** When receiving a RES command a node $n_i$ firstly checks if it has a new DVN, than it checks the payload to find if there is any scheduling information to be consumed. If both conditions are verified the node builds its schedule according to the information received, and it calculates and sends the schedule to its children (if any).

Since a RES command is sent in broadcast, many children will receive such information at the same time. As they compute the schedule for their own children, they will start sending RES commands simultaneously, with possible collisions. In fact, RES commands are sent according to the 6TiSCH "minimal" schedule, which provides a set of shared cells known by all nodes in the network and used in Slotted Aloha mode. If the network is dense, collisions among RES commands could happen. To encompass this problem, the RES command transmission is delayed by a random time period[4].

| DeTAS IE | | | | Termination IE | | |
|---|---|---|---|---|---|---|
| Bit: 0-6 | 7-14 | 15 | Byte 1 | Bit: 0-6 | 7-14 | 15 |
| Length | IE - ID | Type | DeTAS V N | Length | IE - ID | Type |
| 1 | 0x19 | 0 | DVN | 0 | 0x7e | 0 |

Fig. 6: IE format that contains the DVN information.

Another important facet related to the DeTAS signaling is that related to the DVN. A DVN is communicated into a RES command and used by the receiving nodes to determine its validity: only RES commands which contain a DVN greater than the current one will be accepted. In addition, the DVN is inserted into the IEs of data and EB frames. Fig. 6 illustrates the IE that contains the DVN and the termination IE. A node can check in each received frame the DVN and recognize if there is any action to be taken. For example if the node $n_i$ receives a frame with a DVN bigger than the one it is using, then it recognizes it has to send a REQ. If it receives a frame from a child with DVN smaller than the one it is using, it recognizes it has to send a RES.

### C. Implementation in OpenWSN

Fig. 7 describes the software architecture of the OpenWSN project. Each protocol in the stack is depicted as a horizontal layer whereas the vertical module implements some common functions which are used from all the different layers of the stack.

In order to implement the algorithm the *"DeTAS"* component has been added to the OpenWSN stack. The relations

[4]In our implementation we have considered a delay which varies in the range from [1:5] slotframes with a uniform distribution of probability.

of the *"DeTAS"* component with the existing modules of OpenWSN are also described in Fig. 7.

In particular the new component is positioned inside the MAChigh module. The *"DeTAS"* component in its own is divided in three logical parts. (i) *"Signaling"* is the component that handles the reception and the transmission of REQ and RES command messages described in the previous subsection. (ii) *"Scheduling"* implements the scheduling functions. It has access to the existing *"SCHEDULE"* component in order to execute the schedule calculated according to DeTAS algorithm, and at the same time, it is accessed from the *"Signaling"* component to create the RES command payload. (iii) The last part, *"Attributes"* implements the data structures which are necessary to store all the DeTAS related information and the functions to handle them.

The relations of the *"DeTAS"* components with the existing OpenWSN components, as it can be verified from the Fig.7, are mainly unidirectional and the existing modules are accessed without any need to modify them. The only exception is the relation with the *"RES"* component. In this case the existing "RES" component which handles the forwarding of the packets from the lower layer to the upper ones and vice versa, is modified in order to handle the command packets defined for DeTAS. In addition it accesses the *"Signaling"* component for two reasons. First, when it needs to create the DVN IE in the data packets that are transmitted from the mote. And second, to notify the *"Signaling"* component in the case there is a mismatch between the DVN of the mote and the DVN of a received packet.

As the rest of the existing module, *DeTAS* will have access to the cross layer functions. Among other functions accessed, from the *"IDMANAGER"* it will retrieve the rank, which is obtained originally from the RPL protocol.

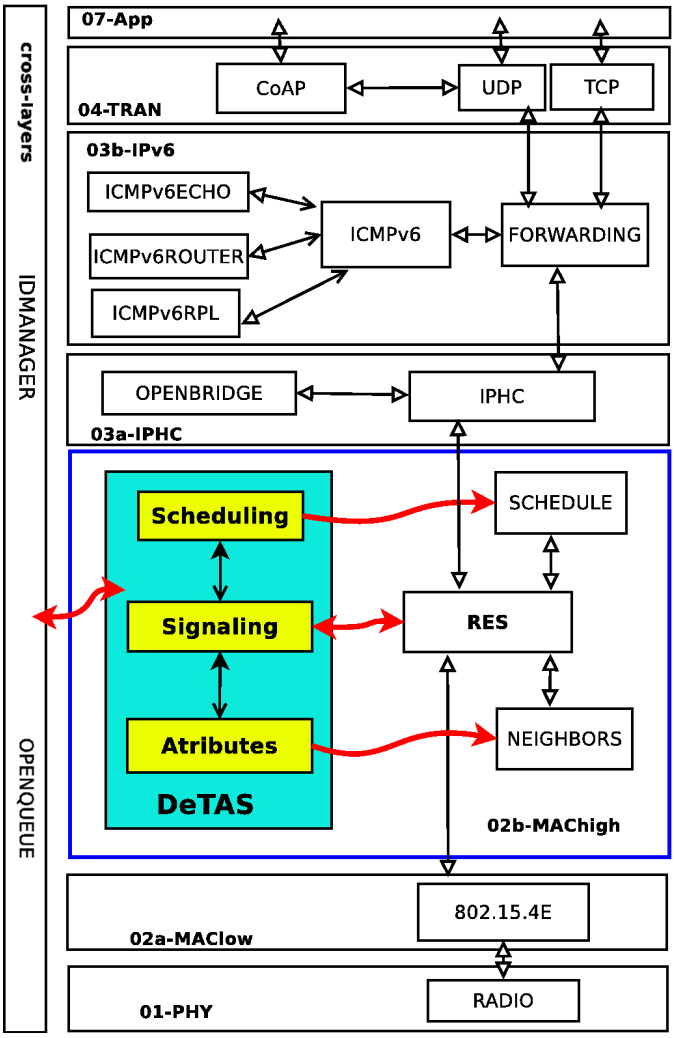



Fig. 7: DeTAS module inside OpenWSN stack.

## V. Experimental Evaluation

The performance of the DeTAS implementation in the OpenWSN protocol stack (described in Sec. IV) has been evaluated over some network topologies deployed with TelosB motes [34].

OpenWSN by default provides a very basic schedule for all nodes in a network. In details, it implements the "minimal" configuration schedule [11] over a slotframe structure long exactly 101 timeslots. Each timeslot can be configured as one among the following possibilities:

- advertisement (ADV): slot reserved for the transmission of EBs;
- transmission (TX) or receive (RX): slots respectively scheduled for data transmission or reception;
- TX/RX: shared slots used for both transmissions and receptions of all kinds of frames with Slotted Aloha contention access;
- serial receive (SERIALRX): slots reserved for the communication with the serial port.

At bootstrap, each node is preconfigured to run an initial schedule formed by an ADV slot followed by 5 shared TX/RX slots (with an associated channel offset equal to 0). Such slots are positioned at the very beginning of the slotframe structure.

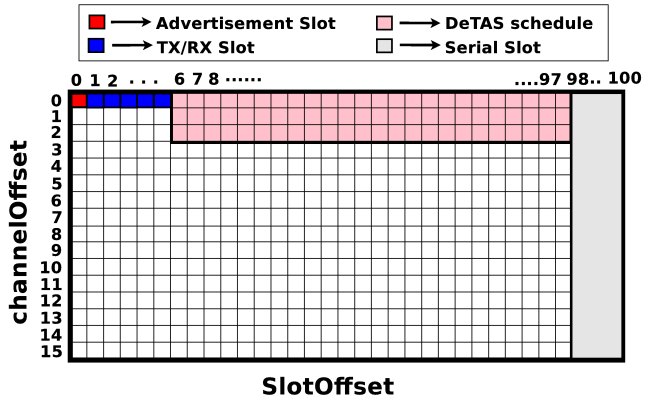



Fig. 8: Slotframe structure used in the experiments.

In addition, 3 SERIALRX slots positioned in the end of the slotframe are reserved for serial communication. Fig. 8 shows such basic schedule. This configuration could be exploited for maintenance operations and data exchange as well.

However, in very dense networks with huge traffic requirements, such setting is not sufficient in terms of bandwidth. Some dedicated cells (i.e., TX or RX) could be installed to

avoid collisions and increase the network bandwidth. Hence, the remaining 93 slots, as pictured in Fig. 8, are scheduled by DeTAS, which provides indeed a technique for configuring dedicated cells. Although the example shows a DeTAS schedule using 3 channel offsets (i.e., $W = 3$), we have explored the adoption of several values for $W$. The impact of this setting on the network performance has been thoroughly evaluated and will be discussed in the remaining part of this section.

In order to evaluate the efficiency of DeTAS in managing multipoint-to-point traffic flows, each source node runs an application to send dummy data toward the $DODAGroot$. In details, at the network bootstrap, whereas nodes join the network, DeTAS updates the network schedule through some signaling packets (those described in Sec. IV). When a node joins the network, signaling packets are conveyed in both "minimal" shared slots (i.e., for TSCH joining, RPL DIO exchange, and DeTAS RES command frames) and dedicated slots just allocated by DeTAS (i.e., for DeTAS REQ command frames). Once the network is formed, source nodes starts sending periodically data according to the aforementioned application.[5] Specifically, a node generates a packet every 2 slotframe cycles, i.e., 1 packet every 3.03 seconds.

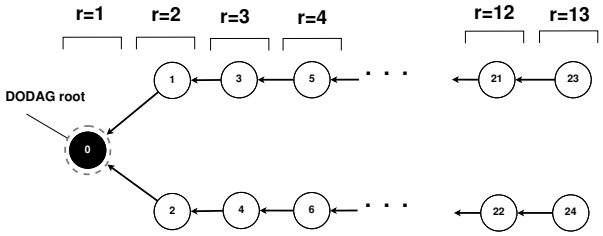
Since packets can be lost due to the mutual interference among nodes using the same cell, we have overprovisioned the number of cells installed by DeTAS in order to allow for retransmission. In details, the *local packet number* related to each node $n_i$ in the network has been configured as $q_i = 2$. In other words, a source node will install at least 2 cells per slotframe to deliver the data it has generated toward the $DODAGroot$. Since the traffic generation rate is equal to 0.5 packet per slotframe, the available cells for the transmission of a single packet are 4. In general, a data packet can be retransmitted several times until either it is acknowledged at the MAC layer (i.e., it has been correctly received by a neighbor) or the number of maximum retries has been reached [8]. In the experiments, we have varied also this parameter, in order to understand the tuning rule of thumb to be used in industrial deployments.

For testing the efficiency of DeTAS, the network topologies deployed are described in the following list:

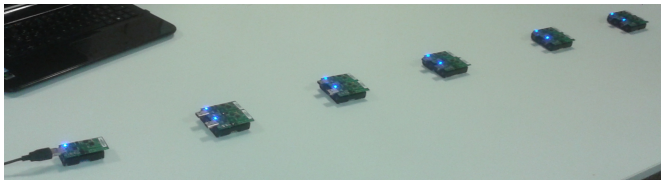
1) A double chain topology is able to characterize the network depth and complies with DeTAS. In fact, at each instant, only a single node per $DAGrank$ is allowed to transmit. Among these, nodes with an odd $DAGrank$ belong to the sub-tree related to a child of the $DODAGroot$, while nodes with an even $DAGrank$ belong to the sub-tree related to another child of the $DODAGroot$. Both sub-trees can be represented with a chain topology without loss of generality. Fig. 9(a) pictures an example of double-chain topology.
2) A binary tree topology is able to characterize the network width. Hence, this topology (sketched in Fig. 10(a)) permit us to assess the DeTAS performance when used in almost realistic dense networks.

With the double chain topology it is possible to investigate how

[5]We have used the DVN field to let nodes know that the network is completely formed.




(a) Topology.



(b) Testbed (the first five hops).

Fig. 9: Double-Chain (a) Topology and (b) Testbed.

the performance of DeTAS changes by increasing the diameter of the network. On the other side, the binary tree topology allows to investigate the sensitivity of DeTAS to the density of nodes. As a such, the findings of this experimental campaign can be also used to approximately characterize DeTAS also in different scenarios, based on the network diameter and node density. For the sake of clarity, Table II summarizes the list

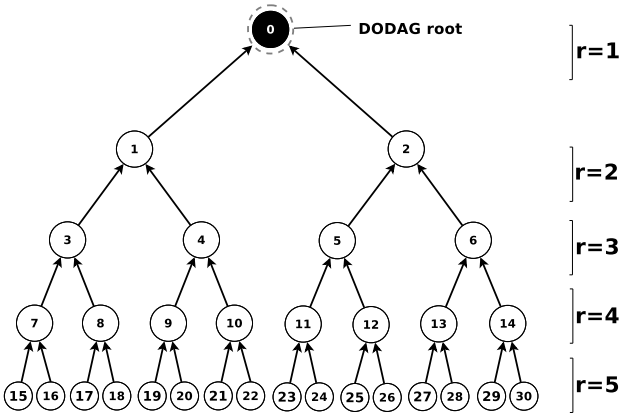
TABLE II: Set of parameters used for experiments

| | Binary Tree (%) | Double Chain (%) |
|---|---|---|
| Freq. Reuse (W) | 3, 4 | 3, 4, 6, 12 |
| Retransmissions | 1, 2, 4 | 1, 2, 3, 4 |
| Nr. Source Nodes | 30 | 24 |
| Max Rank | 4 | 12 |

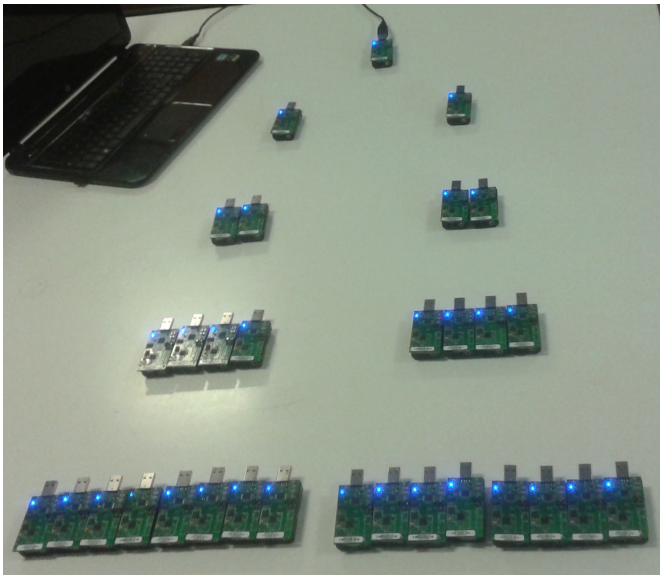
of parameters used in the experiments.

Nodes with the same $DAGrank$ have been positioned as close as possible, in order to increase the effect of mutual interference. In fact, we have investigated the effect of channel reuse. Furthermore, the transmission power of the cc2420 radio present in the TelosB mote has been reduced in order to position the motes in a closer distance (between 25-30cm) and to have complete control of the deployed network. The experiments have been conducted without interference from other wireless technologies, since we are interested in evaluating the effect of the mutual interference among nodes in the same network. As a consequence, the reduction of the transmission power is acceptable for this experimental environment. Fig. 9(b) and Fig. 10(b) show some pictures of the actual testbed deployed.

For each experimental scenario, we have collected 5 40-minutes long traces. In the remaining part of this section, we present the plot related to some performance indices, specifying also the 95% confidence interval. In particular, we have evaluated: (i) the end-to-end delay, i.e., the latency between the data generation and its reception at the $DODAGroot$; (ii)




(a) Topology.



(b) Testbed.

Fig. 10: Binary tree (a) Topology and (b) Testbed.

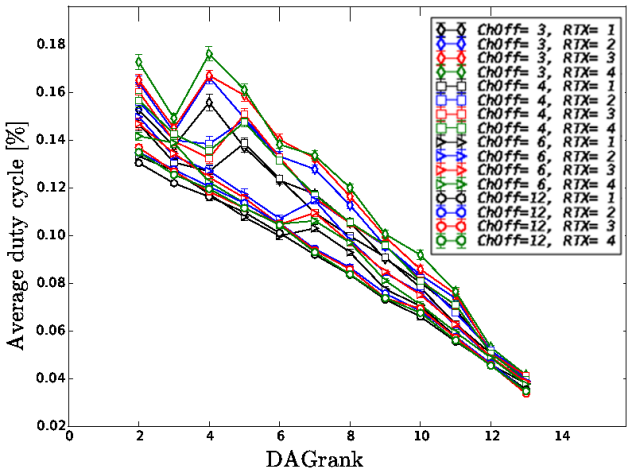



Fig. 11: Duty cycle as a function of the $DAGrank$ a node has inside the network in the Double Chain Topology.

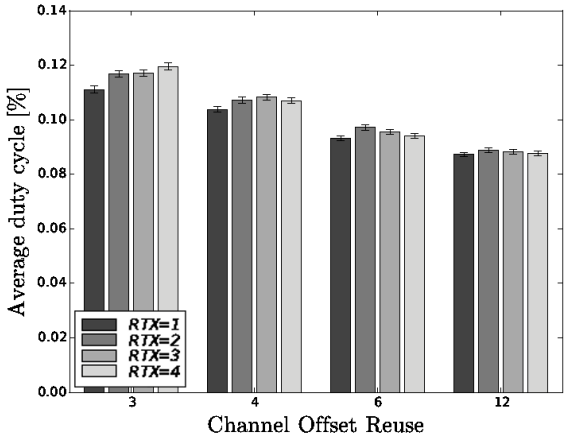



Fig. 12: Average network duty cycle as a function of $W$ in the Double Chain Topology.

the end-to-end and link Packet Loss Ratio (PLR); (iii) the node duty cycle, calculated considering for each active slot only the time when the radio is on (i.e., TX or RX mode).

### A. *Experiments results: double chain topology*

The results related to the experiments on a double chain topology are shown in Figs. 11–18.

Fig. 11 shows the average duty cycle as a function of the $DAGrank$, for any value of channel offsets used and retransmissions allowed. As a general expected behavior, it can be seen that the average duty cycle linearly decreases as the $DAGrank$ increases. In fact, nodes with smaller $DAGrank$ are those closer to the $DODAGroot$, hence bottlenecks for the traffic directed to the $DODAGroot$.

In Fig. 12, the average duty cycle (regardless of $DAGrank$) has been plotted with histograms. In details, we have grouped results according to the value $W$ of the number of channel offsets used. As it can be seen, as the $W$ increases, the duty cycle decreases, because with more channel offsets available there will be less mutual interference. For instance, with $W = 3, 4$ couple of nodes in the double chain topology will be allowed to interfere in the same cell. Although some of these couples will not be exchanging data (1 cell every 4 will be used for data packet exchange), the receiving side of each couple will detect a transmission and will continue receiving the packet. Therefore, some nodes will increase their duty cycle because overhearing the radio medium, even though they will later realize that the packet was not addressed to themselves.

In addition, it can be also noticed that for lower values of $W$, the duty cycle increases as the maximum number of retransmissions increases. For higher values of $W$, results are almost identical when varying the retransmissions number. This behavior was expected too, since with higher $W$ values, the mutual interference is lower, thus making the retransmission mechanism useless.

The end-to-end average delay as a function of the $DAGrank$ is shown in Fig. 13. As expected, the delay linearly

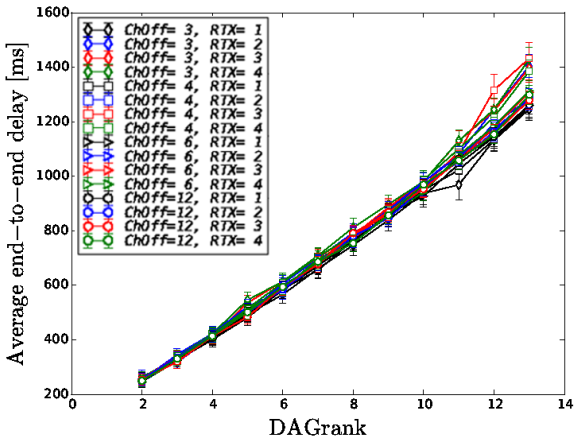



Fig. 13: Network Delay as a function of the $DAGrank$ that a node has inside the network in the Double Chain Topology.

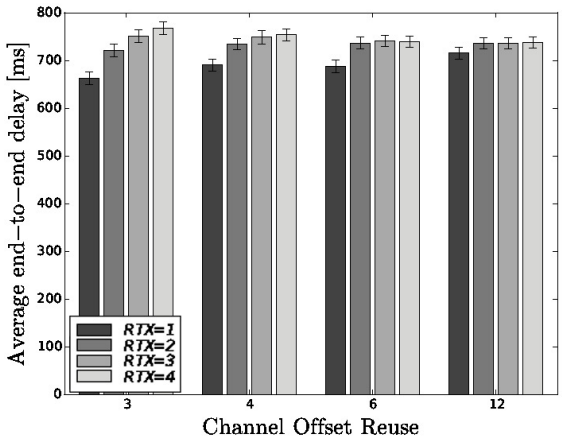



Fig. 14: Average network Delay as a function of $W$ in the Double Chain Topology.

increases as the number of hops (which is directly correlated to the $DAGrank$) augments. It is also worth noting that the average maximum delay is lower than 1.5 seconds. On average, a data packet will reach the $DODAGroot$ in less than half of the slotframe duration.[6] This feature is inherent to DeTAS: given a path between a source node and the $DODAGroot$, the transmission on a link belonging to that path will be always scheduled before the transmission on the following link in the same path toward the $DODAGroot$. In other words, if a node has to relay packet toward the $DODAGroot$, a given receiving cell in the schedule of that node will be always followed by a transmitting cell. This feature is very important with reference to time-critical monitoring application in industrial plants.

Fig. 14 clearly shows that augmenting the number $W$ of channel offsets available, the number of maximum retransmission has a lower effect. With $W = 3$, more collision can happen, so more allowed retransmissions give more reliability to the network at the cost of bigger delays.

Obviously, the end-to-end PLR increases as the hop distance between a node and the $DODAGroot$ augments. This is confirmed by the results plotted in Fig. 15. Such increase depends on the link PLR measured at each hop, which is also plotted in Fig. 16. The average link PLR does not depend on the $DAGrank$, and the not aligned values for the link PLR (e.g., the average link PLR at $DAGrank = 10$) can be explained as due to device misbehavior.

Finally, in Fig. 17 it can be seen that the end-to-end PLR is significantly reduced with at least a re-transmission (RTX) allowed then a regular transmission (RTX$\geq$ 2). Increasing the number of available channel offsets there is an additional improvement of the reliability. Similar arguments can be used when considering the link PLR (as pictured in Fig. 18).

### *B. Experiments results: binary tree topology*

The performance results related to the experiments performed for exploring the efficiency of DeTAS on a binary

[6]In OpenWSN a timeslot is 15 milliseconds long, therefore a 101-sized slotframe is 1.515 seconds long).

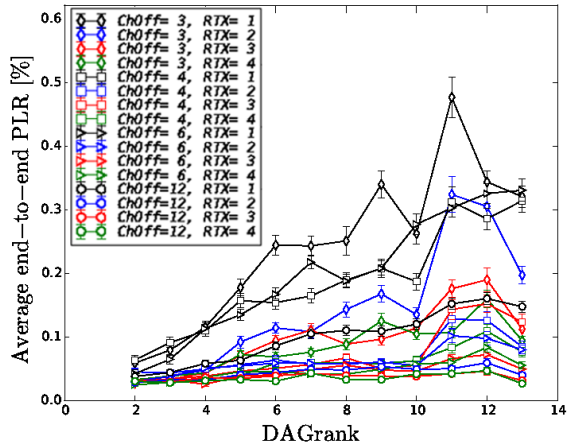



Fig. 15: End-to-end PLR as a function of the $DAGrank$ that a node has inside the network in the Double Chain Topology.

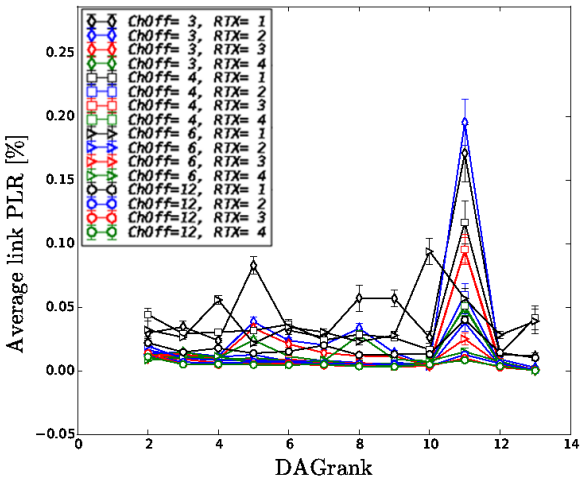



Fig. 16: Link PLR as a function of the $DAGrank$ that a node has inside the network in the Double Chain Topology.

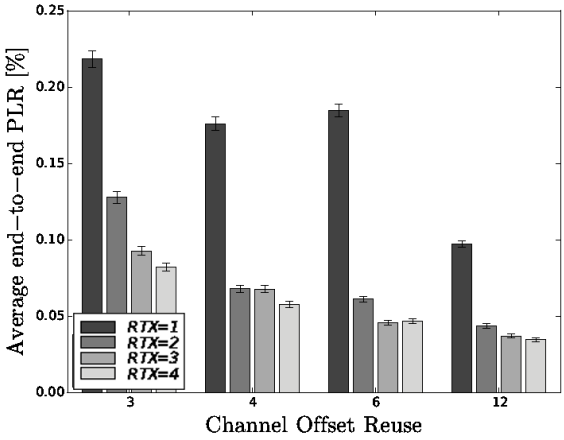



Fig. 17: Average end-to-end PLR of the network as a function of $W$ in the Double Chain Topology.

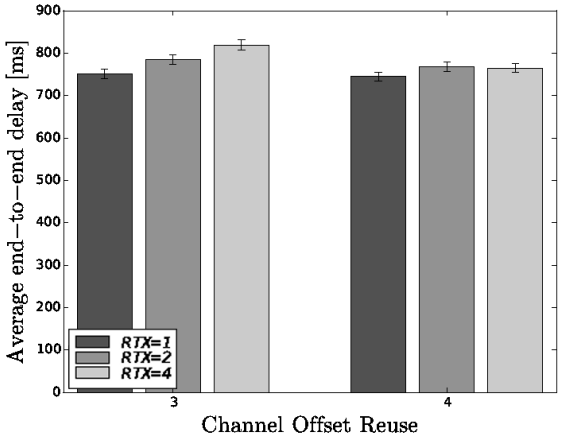



Fig. 20: Average network Delay as a function of $W$ in the Binary Tree Topology.

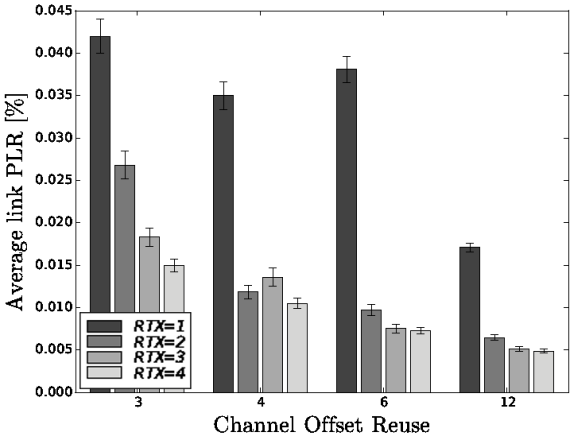



Fig. 18: Average link PLR of the network as a function of $W$ in the Double Chain Topology.

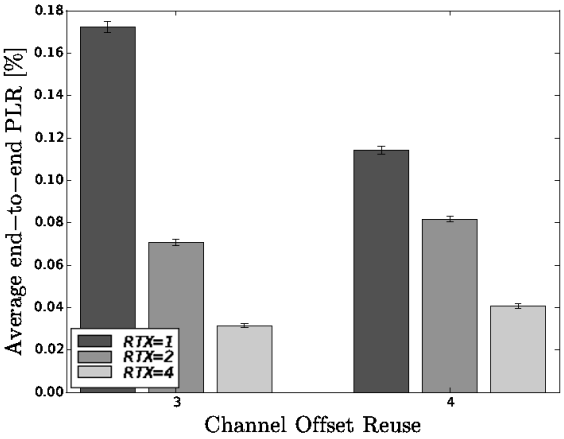



Fig. 21: Average end-to-end PLR of the network as a function of $W$ in the Binary Tree Topology.

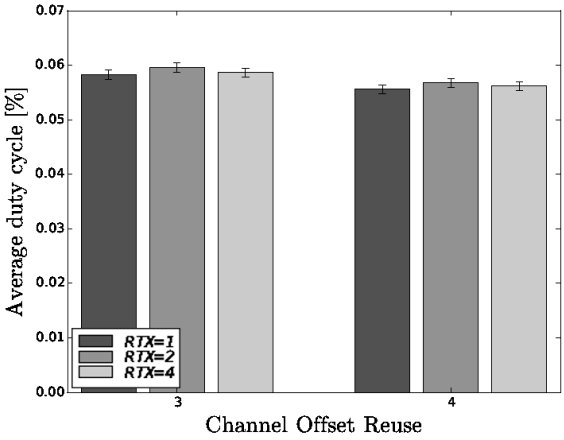



Fig. 19: Average network duty cycle as a function of $W$ in the Binary Tree Topology.

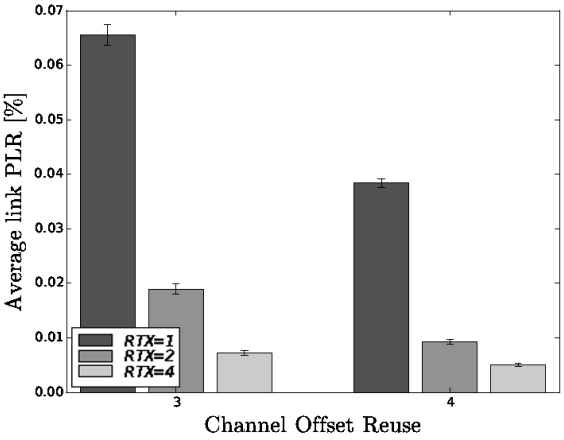



Fig. 22: Average link PLR of the network as a function of the $W$ in the Binary Tree Topology.

tree topology are shown in Figs. 19–22.

In this topology, a number of available channel offset equal to 4 is sufficient for avoiding collisions and radio overhearing. In fact, the maximum $DAGrank$ is 4 with a binary tree topology made by 30 source nodes (as in our experiments). The results clearly confirm that DeTAS has the same performance independently from the network topology. This is what we expected, since DeTAS has been designed to manage all kinds of topology and to be scalable in all scenarios.

## VI. Conclusion

In this paper, we have described in more details the very first implementation of the Decentralized Traffic Aware Scheduling algorithm in the OpenWSN protocol stack. Some experimental results related to real network deployments have been assessed confirming the effectiveness of DeTAS in time-critical applications especially needed in industrial environment for monitoring and control purposes.

In details, we have described the efforts being spent within the IETF 6TiSCH working group to the aim of standardizing an adaptation layer which can let IETF standards be employed on top of the novel IEEE802.15.4e Timeslotted Channel Hopping MAC protocol.

Then, we have described the DeTAS scheduling technique, highlighting its theoretical effectiveness in building a multi-hop schedule in a distributed fashion.

We have also reported details of the real implementation of DeTAS, by picturing the signaling required and explaining how it can be integrated into 6TiSCH-enabled networks.

The experimental results confirm what we already expected for the DeTAS performance in terms of duty cycle, end-to-end delay, end-to-end and link Packet Loss Ratio.

We strongly believe that the strength of DeTAS relies in its design: it enables a fast communication between the $DODAGroot$ and any node in the network; it avoids queues being congested; and it reduces the packet loss ratio through a proper scheduling of resources.

In future work, we will extend DeTAS to manage topologies where each node can route traffic to more than one parent.

## References


[1] M. R. Palattella, P. Thubert, X. Vilajosana, T. Watteyne, Q. Wang, and T. Engel, "6TiSCH Wireless Industrial Networks: Determinism Meets IPv6," in *Internet of Things: Challenges and Opportunities. Lecture series of Smart Sensors, Measurement, and Instrumentation*, S. C. Mukhopadhyay, Ed. Springer-Verlag, 2014, vol. 9.

[2] S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 2460, Internet Engineering Task Force RFC 2460, December 1998.

[3] M. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the internet of (important) things," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 3, pp. 1389–1406, Third 2013.

[4] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. P. Vasseur, and R. Alexander, *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, RFC 6550, IETF RFC 6550, March 2012.

[5] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, RFC 7252, Internet Engineering Task Force RFC 7252, June 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[6] 6tisch mailing list. [Online]. Available: https://www.ietf.org/mailman/listinfo/6tsch

[7] 6tisch homepage. [Online]. Available: https://bitbucket.org/6tsch/

[8] *802.15.4e-2012: IEEE Standard for Local and Metropolitan Area Networks – Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC Sublayer*, IEEE Std., 16 April 2012.

[9] P. Thubert, T. Watteyne, M. Palattella, X. Vilajosana, and Q. Wang, "Ietf 6tsch: Combining ipv6 connectivity with industrial performance," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, July 2013, pp. 541–546.

[10] T. Watteyne, M. Palattella, and L. A. Grieco, "Using IEEE802.15.4e TSCH in an IoT context: Overview, Problem Statement and Goals draft-ietf-6tisch-tsch-05 (work in progress)," IETF, Internet Draft, January 2015. [Online]. Available: http://tools.ietf.org/html/draft-ietf-6tisch-tsch-05

[11] X. Vilajosana and K. Pister, "Minimal 6TiSCH Configuration draft-ietf-6tisch-minimal-05(work in progress)," IETF, Internet Draft, January 2015. [Online]. Available: http://tools.ietf.org/html/draft-ietf-6tisch-minimal-05

[12] D. Dujovne, L. A. Grieco, M. Palattella, and N. Accettura, "6TiSCH On-the-Fly Scheduling draft-dujovne-6tisch-on-the-fly-04 (work in progress)," IETF, Internet Draft, January 2015. [Online]. Available: http://tools.ietf.org/html/draft-dujovne-6tisch-on-the-fly-04

[13] M. Palattella, N. Accettura, M. Dohler, L. Grieco, and G. Boggia, "Traffic aware scheduling algorithm for reliable low-power multi-hop ieee 802.15.4e networks," in *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*, Sept 2012, pp. 327–332.

[14] M. Palattella, N. Accettura, L. Grieco, G. Boggia, M. Dohler, and T. Engel, "On optimal scheduling in duty-cycled industrial iot applications using ieee802.15.4e tsch," *Sensors Journal, IEEE*, vol. 13, no. 10, pp. 3655–3666, Oct 2013.

[15] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wank, S. Glaser, and K. Pister, "OpenWSN: a standards-based low-power wireless development environment," *Tansactions an Emerging Telecomunications Technologies IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, p. 480493, 2012.

[16] ures. [Online]. Available: https://openwsn.atlassian.net/wiki/display/OW/uRES

[17] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*, RFC 2205, Internet Engineering Task Force RFC 2205, September 1997.

[18] N. Accettura, M. R. Palattella, G. Boggia, L. A. Grieco, and M. Dohler, "Decentralized traffic aware scheduling for multi-hop low power lossy networks in the internet of things," in *Proc. of IEEE Int. Symp. on a World of Wireless Mobile and Multimedia Networks, WoWMoM*, Madrid, Spain, Jun. 2013.

[19] A. Morell, X. Vilajosana, J. L. Vicario, and T. Watteyne, "Label switching over ieee802.15.4e networks," *Transactions on Emerging Telecommunications Technologies*, vol. 24, no. 5, pp. 458–475, 2013. [Online]. Available: http://dx.doi.org/10.1002/ett.2650

[20] IEEE std. 802.15.4, *IEEE Standard for Local and metropolitan area networks –Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, Standard for Information Technology Std., September 2011.

[21] N. Kushalnagar, G. Montenegro, and C. Schumacher, *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*, RFC 4919, Internet Engineering Task Force RFC 4919, August 2007.

[22] B. C., *6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*, RFC 7400, IETF RFC 7400, November 2014.

[23] Highway addressable remote transducer, a group of specifications for industrial process and control devices administered by the hart foundation. [Online]. Available: www.hartcomm.org

[24] Isa, isa100, wireless systems for automation, may 2008. [Online]. Available: http://www.isa.org/Community/SP100WirelessSystemsforAutomation

[25] M. Nixon, "A Comparison of WirelessHART and ISA100.11a," Tech. Rep., 2012.

[26] P. Thubert, T. Watteyne, and R. A. Assimiti, "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4e draft-ietf-6tisch-architecture-04 (work in progress)," IETF, Internet Draft, October 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-6tisch-architecture-04

[27] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, *Neighbor Discovery for IP version 6 (IPv6)*, RFC 4861, Internet Engineering Task Force RFC 4861, September 2007. [Online]. Available: http://tools.ietf.org/html/rfc4861

[28] M. Palattella, T. Watteyne, and Q. Wang, "Terminology in IPv6 over the TSCH mode of IEEE 802.15.4e draft-ietf-6tisch-terminology-03 (work in progress)," IETF, Internet Draft, January 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-terminology-03

[29] J. Ben Slimane, Y.-Q. Song, and A. Koubaa, "Control and data channels allocation for large-scale uwb-based wsns," in *First International Conference on Communications and Networking, 2009. ComNet 2009.*, Nov 2009, pp. 1–8.

[30] P. Namboothiri and K. Sivalingam, "Capacity analysis of multi-hop wireless sensor networks using multiple transmission channels: A case study using ieee 802.15.4 based networks," in *IEEE 35th Conference on Local Computer Networks (LCN), 2010*, Oct 2010, pp. 168–171.

[31] V. Sarkar, *Partitioning and scheduling parallel programs for multiprocessors*. MIT press, 1989.

[32] R. E. Korf, "Multi-way number partitioning," in *Proceedings of the 21st international jont conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., 2009, pp. 538–543.

[33] X. Wang, Q.and Vilajosana and T. Watteyne, "6TiSCH Operation Sublayer (6top) draft-wang-6tisch-6top-sublayer-01 (work in progress)," IETF, Internet Draft, July 2014. [Online]. Available: https://tools.ietf.org/html/draft-wang-6tisch-6top-sublayer-01

[34] Telosb datasheet. [Online]. Available: http://www.willow.co.uk/TelosB˙Datasheet.pdf