Distributed Approach of Cross-Layer Resource Allocator in Wireless Sensor Networks

Franck Rousseau and **Olivier Alphand** and **Rodolphe Bertolini**
Grenoble Informatics Laboratory (LIG), France

Karine Altisen and **Stephane Devismes**
INRIA Grenoble, France

Abstract

Wireless Sensor Networks (WSN) are resource-constrained by their architecture: low memory, low powerage, low calculation capabilities, low connectivity. Despite those constraints, they need to organize themselves in a tree-like topology along which all the data will be relayed towards the root of the network. To increase the life expectancy of the network, nodes should spend most of their time sleeping while still ensuring an acceptable delay for data to reach the root. In this paper, we present the sketch of the distributed version of a WSN simulator from the state of the art, that was initally using centralization. We show that distribution creates contention in the shared slot, and that increasing the number of shared slots heals this contention.

1 Introduction and state of the art

Wired sensor networks installation cost can be up to 10 times the market share value of sensors. WSN lower this cost, and can also provide an ubiquity of the information gathering, that we call the Internet of Things (IoT). Those reasons among others can explain the growth of WSN usage.

Great effort were made by the IETF to standardize a complete IP-compliant IoT stack : a specific transport layer (CoAP [Shelby *et al.*, 2014]) as well as a routing protocol (RPL [Winter, 2012]) and some specific schemes (6LowPAN [Hui *et al.*, 2010]) to compress IPv6 addresses. For the physical layer, the 802.15.4 technology was the most widespread one because of its low consumption capabilities. However no MAC layer was clearly supported by the IETF. The WSN we are interested in this article specifically aim at addressing the requirements of a harsh industrial environment. Therefore, beside an enhanced WSN life expectancy achieved by sleeping as much as possible, robustness and time contraints are vital requirements. To do so, the MAC layer of the original 802.15.4 standard was not robust or deterministic enough. To overcome those problems, additional behaviors were introduced at the MAC layer in the 802.15.4e standard [Group and others, 2011]. One main feature is channel hopping which allows to mitigate multi path fading and external interferences experienced in 802.15.4 that uses a unique channel. Several mode were defined, including synchronous and asynchronous mode.

We focus on Time Slot Channel Hopping (TSCH), a protocol defined by IEEE 802.15.4e standard [Mirzoev and others, 2014]. TSCH is a Medium Access Control (MAC) layer protocol that provides a mean to sensors so that they can communicate properly.

Orchestra [Duquennoy *et al.*, 2015], a routing-aware, autonomous, random-access TSCH allocator, gives a solution for ressource allocation at MAC layer level: it allocates time slots to a sensor in order to communicate longer with its parent when the incoming data is too large. But the randomness of this algorithm does not prevent from a sensor from a branch of the graph to allocate the same slot that a sensor from an other branch of the graph also allocated. Non preventing this would, in some cases, lead to interferences and collisions. For instance, a sensor can give to its child the same cell that has been allocated in a close neighborhood. If they communicate in the same time, and because the medium used is radio wave, the signal cannot be read by any of both receivers.

We worked on an open-source simulator, developped at Berkely University by the 6tisch working group [Palattella *et al.*, 2016], that focuses on the On-The-Fly (OTF) Bandwith Reservation algorithm implementation [Dujovne *et al.*, 2014]. OTF is an algorithm that determines how many time slots are needed for a sensor in order to communicate with its parent(s) to minimize congestion in its packet queue.

This simulator has a centralized view of the network: each node has the knowledge of neighbors and parents properties. For instance, to add a cell in its scheduler, a node calls a function that gets the free cells from its parent and picks randomly the needed number of slot.

Moreover, the simulated network traffic only consists of data packets while control packets (request for a new cell, reply, DIO (see section 2.1) ...) are considered non-existent. The centralization is made possible with calls from a cell of its parents' / neighbor's function. In addition, the fact that slots are randomly picked induces the same drawback as the one Orchestra induces.

We would like to compare the actual probabilistic approach of allocation, caused by the random choice of a cell, to a deterministic algorithm. But to have interpretable results, we first need to make the simulator more realistic concerning the simulated network traffic.

We thus modified the simulator in two main ways:

- so that it also simulates some control traffic (requests and answers triggered by OTF, and later DIO)
- so that this algorithm operates in a distributed manner: instead of having an omniscient view of the network that allows a sensor to get the needed resource instantaneously, there are communications that occur between sensors, and sensors use the information in those communications to make decision about resource allocation.

We start in section 2 with definition of the graph generated by the sensors connectivity, and an overview of TSCH protocol. Then, we present in section 3 the simulator we modified, and we explain the modifications that we did. We see in 4 that those modifications impact the bootstrap duration, and we explain in section 5 some improvements that can be done in order to reduce the bootstrap duration when using shared slots. In 6 we analyse and discuss the results using improved communication through shared slot.

In section 7, we finally conclude about the effects of the decentralization of the overall behavior, and bring ideas on other improvements to speed up the bootstrap and to reduce the number of collisions in data packet transmission that occur more often as the number of node increases.

2 Communication between sensors

2.1 Through a DODAG ...

To collect data, sensors need to structure a Destination Oriented Direct Acyclic Graph (DODAG). The graph construction is handled by Routing Protocol for Low-Power and Lossy Networks (RPL). The aim of RPL is to build the shortest route from any sensor to the root gateway sensor that will transmit information to a server that will process this information.

RPL states that a sensor should send broadcast to its neighbors containing at least its own rank in the DODAG. Because the traffic is from leaf to root, the root sets its DODAG rank to 0 (because there is no hop between the root and the server) and starts sending the DODAG Information Object (DIO) [Winter, 2012], in which its rank information is contained, to all of its neighbors. When a sensor receives a DIO that has a lower rank than its own, it updates its rank as $received\_rank + 1$ and broadcasts a new DIO containing the updated rank information. If the rank in the received DIO is higher, it ignores it. In the first case, a sensor can also decides to ignore the DIO if the medium has a poor quality (low RSSI for instance).

To send a data packet to the route through the shortest path, a sensor has to send its packet to the sensor that has a lower rank.

Figure 1 shows the different paths (set up by DIO) that data may follow to reach the root (R). The node rank is put inside brackets.

$$\begin{tikzcd}
A\ (2) \arrow[r] \arrow[ddr] & C\ (1) \arrow[dr] & \\
 & & R\ (0) \\
B\ (2) \arrow[r] & D\ (1) \arrow[ur] &
\end{tikzcd}$$

Figure 1: Example of a DODAG with 5 nodes

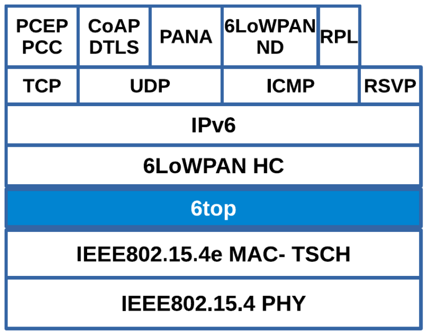

Figure 2: TSCH is a MAC layer protocol [Palattella *et al.*, 2016]

2.2 ... using TSCH

The medium through which sensors communicate is defined by TSCH, a MAC layer (Figure 2 [Palattella *et al.*, 2016]) protocol first outlined in IEEE 802.15.4e standard. A sensor X is assigned a time slot and a channel that corresponds to a specific communication with sensor Y : from X to Y (Tx: transmission), from Y to X (Rx: reception), or unicast and broadcast. Let's take sensor D from Figure 1 and see Table 1. The couple of time slot and channel offset (1,2) and (2,3) correspond to Rx cells but are Tx cells for respectively A and B, and (4,0) to a Tx cell for D but Rx cell for the root.

We initially use the couple (0,0) as the mean for unicast and broadcast informations (DIO messages, control packets). This cell corresponds to the shared cell.

		Time slot				
		0	1	2	3	4
Channel offset	0	DIO Control	$A \to C$			$D \to R$
	1				$C \to R$	
	2		$B \to D$			
	3			$A \to D$		
	4					

Table 1: Time slot & Channel offset matrix

3 Presentation of the simulator

We modified an open-source WSN simulator that has been developped at Berkeley University [Palattella *et al.*, 2016]. The standard parameters of a simulation follow the Minimal 6TiSCH Configuration [Vilajosana and Pister, 2016]: a time slot lasts 10 ms, a slot frame contains 101 time slots , and 16 frequency channels are available. The nodes are located in a 1km^2 wide square.

3.1 Original simulator

The topology of the network is created randomly: a sensor is placed at a random position, the number of neighbor(s) is computed using RSSI calculation, and if there are too few neighbors, the sensor is replaced until the number of neighbors reaches a certain threshold. The first placed sensor is the DODAG root, the destination of all the data collected by the sensors, and is to be at the center of the network: if we consider that the topology is bordered by a rectangle, then its coordinates are $x = width/2, y = height/2$.

The DIO messages are not simulated in the network traffic, sensor notifies its neighbors of its rank using a centralized function. This function is called each slot frame by all of the sensors, whereas they are supposed to be sent depending on a trickle timer : while a sensor does not receive a DIO that proposes a lower rank, it doubles its DIO period. When a lower rank DIO is received, it starts again with a period of 1 slot frame.

When a sensor scheduler activates a cell in the matrix shown in Figure 1, it gathers information about what to do: if the cell is Rx then it has to listen for possible incoming packets; if the cell is Tx then it has to send the first packet in its data queue; if the cell is a shared cell it sends the first packet from its control packets queue or listen to control if its queue is empty.

When a sensor detects that it needs more time slot to communicate with its parents (thanks to OTF algorithm), it calls a function of a parent sensor. This parent randomly selects the needed number of slots that are not used in both itself's scheduler and requester's (child) scheduler, it adds them into its scheduler as Rx cells, and it tells the requester to also add them as Tx cells.

Most of the functionalities are using a centralized abstraction, that permits instantaneous transactions between nodes. To bring a more realistic model, we modified this centralized behavior for some of the transactions.

3.2 Actual simulator

Modification in the behavior

We first added a new type of cell: the shared cell, that corresponds to the cell (0,0) on which every sensor can listen to and send information.

This special cell allows us to make the resource allocator distributed: instead of a parent to get the information of the slot requester's scheduler through omniscient function calls, the requester sends through the shared cell all of the information that the parent needs to process the request: MAC address of the requester, a list of its already used time slots (so that to compute the available slots, the parent can subtract it from its non-used time slots list), and the number of needed slots. The parent's answer is also sent through the shared cell. We implemented the 3-way transaction [Wang and Vilajosana, 2016], thus the requester sends a confirmation packet after it added the cells into its scheduler. If the answer from the parent does not contain enough slots, it adds the allocated slots and send a new request with the new needed number of slots and the updated list of slots being in use by the requester.

To make communications through the shared slot the most coherent possible, we added a transaction aborting system. If one of the node is dropped, it aborts the transaction and reverse the changes that have been done during the aborted transaction. We also added a sequence number for control packets. If a mote receives from a neighbor a control packet that has a different number as it expected, it drops the packet and aborts the currently handled transaction if any.

To be coherent with the 6top protocol, a node cannot have two pending transactions at the same time, which means that it has to wait for the transaction to be ended or aborted before starting a new transaction.

We modified the simulation engine so that a callback can take parameters. Thus, the same callback can be used to send both request and answer in the shared cell.

Modification of the launcher

We modified the simulation launcher as following:

- Addition of parameter `queuing`: if 0 then the simulation will be as original simulator (nothing through cell 0), 1 then simulation will be as new simulator behavior (slot request and answer using shared cell)
- Addition of parameter `bootstrap`: if used with `queuing 0` $\rightarrow$ no effect. If used with `queuing 1`, then the bootstrap of the network will be managed through shared cell (no Rx / Tx cell at the beginning). If not used, and `queuing 1` is used, then the bootstrap time slot request and answer will be managed by the omniscient function, then use the shared cell.
- Addition of parameter `topology`: if used, the simulator can save the topology into a file (with parameter `rw w`) or read a topology from a file (with parameter `rw r`). The file name is `topologyX.txt`, X being the number of sensors.
- Addition of parameter `numSharedSlots`: it determines the number of shared slots per slot frame. We quickly understood that only one shared slot restricts the communications during bootstrap when there are more than 10 nodes in the square area. The slot number is given by the following formula : if *i* is the shared slot number (from 0 to *numSharedSlots* excluded), n the number of shared slots and l the slot frame length, then $i * \lfloor l/n \rfloor$

As we explain more deeply in **??**, the use of this new cell brings more realism in the simulation, but also brings delay and latency in the communications.

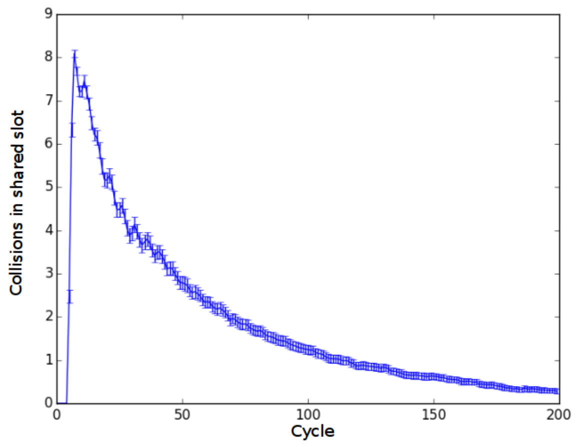

Figure 3: Collisions in the shared cell $(0, 0)$, 10 sensors

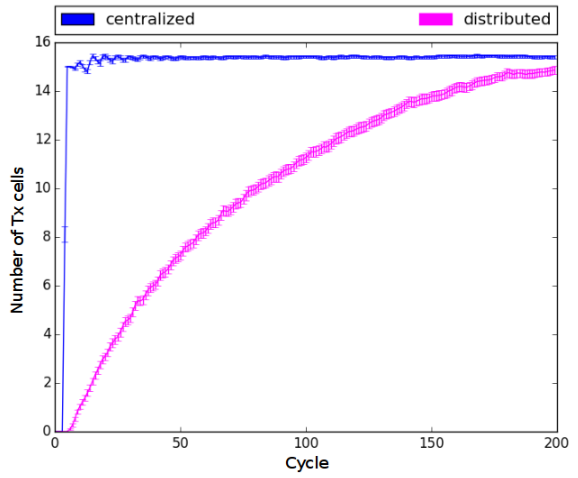

Figure 4: Evolution of cells allocated as Tx, 10 sensors

4 Performance of communication through shared slot

For all simulations using the new version of the simulator, we compared the results with the original version using the same parameters to understand what does distribution change in the network behavior.

We first used a topology of 10 sensors. The simulation ran for 200 cycles (slot frame), that is to say $200 * 101 * 0.010 = 202$ seconds. We writed in a file the topology using a first run, then we read the topology file for 600 runs: 300 without distribution (parameter $queuing$ 0) and 300 with distribution (parameters $queuing$ 1 and $bootstrap$). We separated the results into two categories: without and with distribution. Then, we calculated mean and confidence interval of: collisions in the shared cell (Figure 3)(does concern only distributed version); number of Tx cells allocated in all the network (Figure 4).

We then ran 100 times the same protocol but we increased the number of sensors from 10 to 25.

First of all, we focus on the new shared cell $(0, 0)$, through which all of the requests and answers are transmitted. We can see in Figure 3 that at the beginning of the bootstrap, there is a peak of 8 collisions in one cycle. Then the number of collisions decreases progressively to tend to 0 after 200 seconds.

Figure 4 shows the difference in the evolution of the number of cells that are allocated as Tx (transmission cells) without and with distribution. We can see that the steady state of 15 cells, that is reached after the 5th cycle without distribution, is now reached around the 175th cycle using distribution.

We can see in Figure 5 and 6 that performances decreases when we increase the number of sensors : even after 200 cycles, there are still around 10 collisions per slot frame in the shared slot, thus network has trouble allocating more than 15 Tx cells, whereas original simulator allocates 50 cells around the 10th cycle.

The high number of collisions in the shared cell at boot-

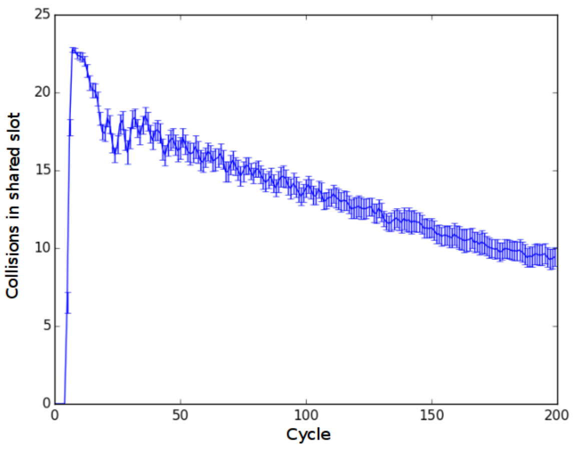

Figure 5: Collisions in the shared cell $(0, 0)$, 25 sensors

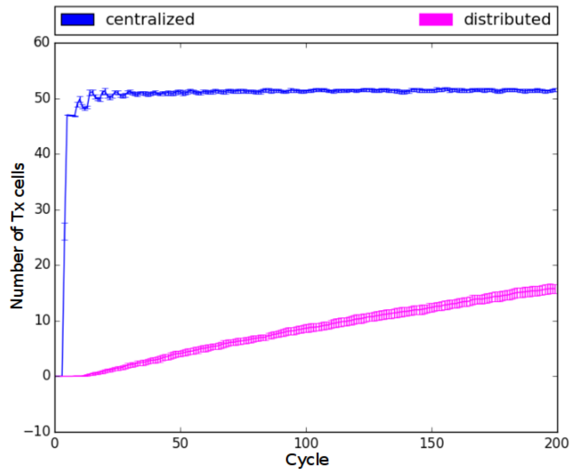

Figure 6: Evolution of cells allocated as Tx, 25 sensors

strap is easily explained by the fact that none of the sensors has Tx cell to communicate with its parent. Thus, OTF considers that they need cells, they all use the shared cell to ask for a cell to their parents, resulting in an heavy contention in this cell.

The delay that it takes to the network in its globality to reach a steady state in the distributed version of the simulator can be explained by the lack of omniscient view of the network. Indeed, instead of a sensor to get all of the resources needed at the next slotframe, exchanges through the network have to be done. But the medium used for this resource allocation request is the same for all of the sensors: the shared cell. We implemented TSCH CSMA/CA algorithm provided by IEEE standard, but as we can see it is not enough to allow a fluent communication in the shared cell.

Indeed, the backoff algorithm is just a window that increases progressively while the control packets is not received. When a sensor sends a control packet, it waits for a random number (picked in the windows) of shared slots before sending again the packet. At the beginning, the windows is $2^{number\_of\_try} - 1$. Thus, it takes at least 4 tries before having a windows that allows a reasonable collision probability for 10 sensors to chose the same delay (window after 4 tries : $[0;15]$).

Reducing contention in shared slot would reduce the time needed for the network to reach a consistent network. It would also allow us to add other important traffic in the shared slot such as enhanced beacon [Li *et al.*, 2012], DIO, and OTF deletion request that is triggered when a sensor considers that it has too many cells that are not used.

To accelerate the process of resource allocation, there is a priority for answers over queries: if a sensor has both queries to be sent to a parent and answer to be sent to a child ((i.e. both query and answer are present in the control paquet queue), then it will send the answer. Indeed, if no priority is handled, the shared cell would be even more flooded by request packet. This flood would dramatically increase the delay for a node to get the answer.

5 Improving communications through shared slot

5.1 Increasing the number of shared slots

As we showed in previous sections, the shared slot becomes quickly overloaded as the number of nodes increases. A study from the Polytechnic Institute of Porto [Koubaa *et al.*, 2006] showed that increasing *macMinBE* (the backoff exponent used in the CSMA/CA retransmission algorithm when collision occurs) enhances the throughput of the network. In our case, the throughput is limited by the number of shared slots, that is set to one slot according to Minimal 6TiSCH Configuration [Vilajosana and Pister, 2016].

We show in this section that the number of shared slots has a significant influence in the proceedings of transactions.

To have an overview of this impact, we chosed the following parameters for the simulations :

- *numMotes* : 25, 50 and 100

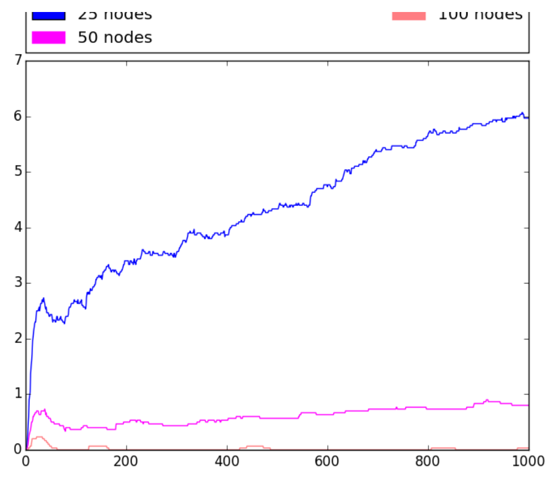

Figure 7: Evolution of cells allocated as Tx, 3 shared slots

- *numSharedSlots* : 3, 10 and 20

We kept the slot frame to a length of 101, because even with 100 nodes the slot frame is not fullfilled thanks to the 16 different channels.

5.2 Two-third inband communication

A 6top cell request transaction can be a 2-way or 3-way transaction. We decided to implement the 3-way because it ensure a coherent scheduler state on both two nodes.

We implemented a two-third inband communication. It means that when a node has a Tx cell allocated to its parent, the next cell request transaction will take place for two-third in the allocated cell.

A full inband communication cannot be handled by the receiver of the request. Indeed, in the 6TiSCH matrix, a cell cannot be tagged as Tx and Rx by the same node, thus the receiver is not able to send its answer through the allocated cell to its child. Hence, the answer is sent through a shared slot.

6 Results and discussions

Figures 7, 8 and 9 show the allocated cells in the 6TiSCH matrix. The most important thing that comes from those figures is that as the number of shared slots increases, the speed at which the cells are allocated also increases. An other thing to notice is the presence of a shared slot number lower bound depending on the number of nodes. Indeed, we can see that the 25 nodes topologies reach 60 allocated cells in 100 cycles or less with 10 and 25 shared slots, while it has a tough time when there are only 3 shared slots.

We can also point that the cells are allocated in the same proportion with 50 nodes in a 10 shared slots configuration and 100 nodes in a 20 shared slots configuration.

This cells allocation pace impacts directly the flows the data goes through. We saw that the more shared slots are present, the fastest the cell allocation is proceeded. Now if we focus on figures 10, 11 and 12, we can see that the more

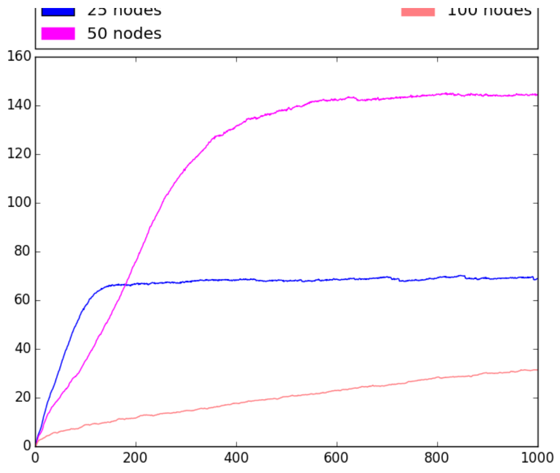

Figure 8: Evolution of cells allocated as Tx, 10 shared slots

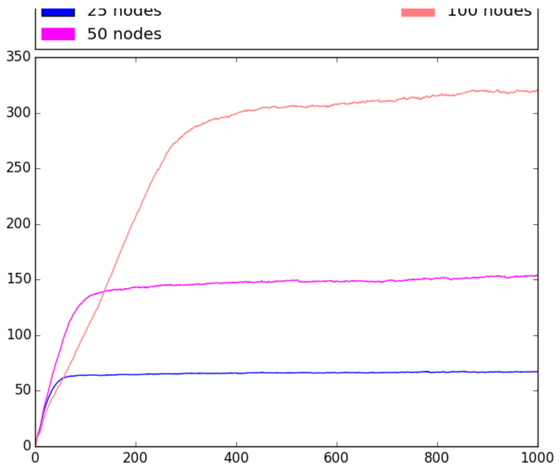

Figure 9: Evolution of cells allocated as Tx, 20 shared slots sensors

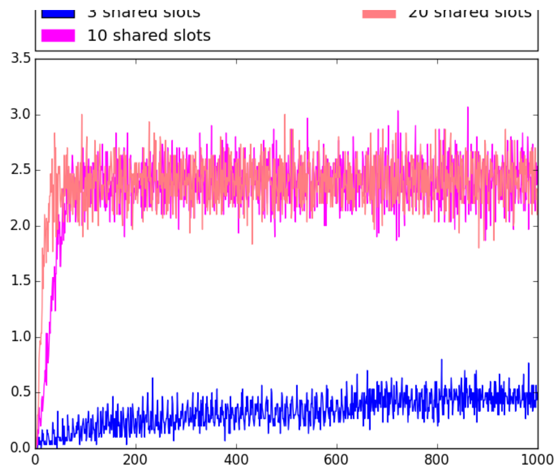

Figure 10: Data packet that reaches the dagroot, 25 nodes

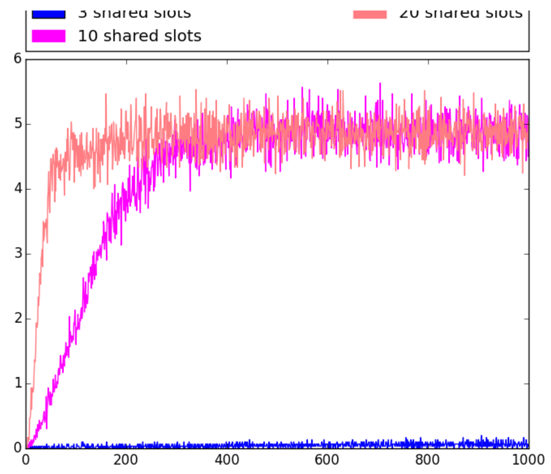

Figure 11: Data packet that reaches the dagroot, 50 nodes

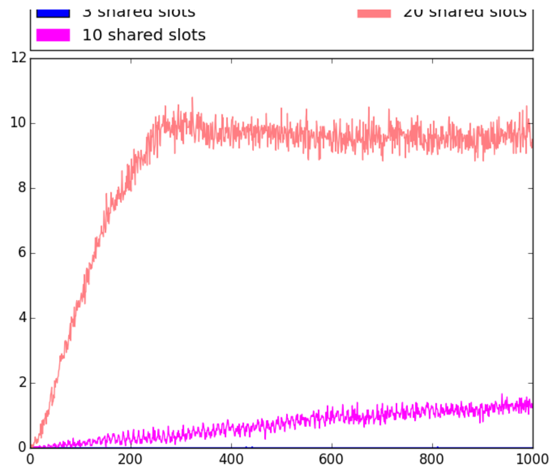

Figure 12: Data packet that reaches the dagroot, 100 nodes

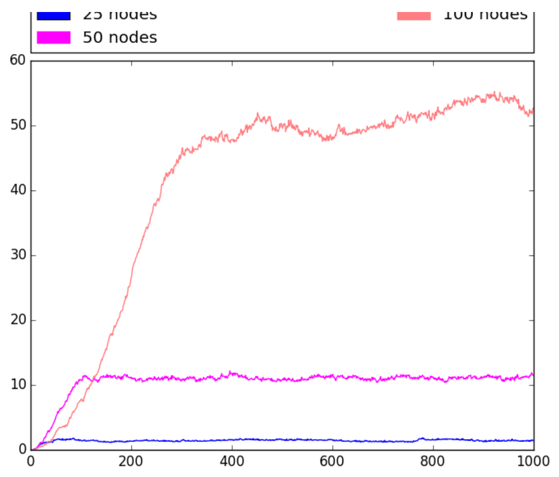

Figure 13: Collisions in the Tx cells

shared slots are present, the fastest the data packets are transmitted to the root of the DODAG.

Indeed, we saw in figure 7 that with 25 nodes and 3 shared slots, some cells are allocated but it growths at a slow pace. We can see in 10 that in this case, only a few data packets reach the root of the dag : 0.5 packets per slot frame in average; while it reaches 2.5 packets per slot frame for the 10 and 20 share slots cases.

We understand from 11 that the faster the Tx cells are allocated, the faster the path from a leaf to the root is built : there is an average of 5 data packets that reach root of the dag from the 25th slot frame with 20 shared slots while this average is reached after 300 slot frames with 10 shared slots. This is directly linked with the Tx cells allocation pace. Indeed, from figure 8 and 9 we see that the 25 nodes topology allocates around 65 cells. This state of 65 allocated cells is reached faster with 20 than with 10 shared slot.

Figure 13 shows the number of collisions that occur per cycle, depending on the number of nodes. We see that as the number of nodes increases, the number of collisions increases dramaticaly. The collisions are caused by the lack of knowledge that a node has about its physical neighbors scheduler. Indeed, there is no control that are broadcasted to prevent a node to allocate a cell that has already been allocated by one of its neighbor. It is thus totally free to allocate this already allocated cell to a communication with its child, inducing collisions. There is no CSMA/CA retransmission algorithm in the allocated slots, so when a collision occurs, it will keep occuring until one or both nodes drop the packet.

We saw that to have a distributed behavior of the cell allocation, the network has to have a medium through which nodes can all communicate. Since this medium, the shared slot, is shared between all of the nodes, there is a very high contention, mostly during network bootstrap. The solution of this contention we brought is to increase the number of occurence of this shared medium.

With the use of the shared slots, it is possible and intended to implement the Lamport Local Mutual Exclusion (LLME) algorithm, to prevent a node to start a transaction until all its neighbors transactions ended. The ”transaction end” broadcast packet would contain the cells that have been allocated. All the nodes that receive this packet would tag those cells as ”allocated by neighbor”, and will not be able to allocate them (or in a last resort : if it has to allocate a cell and it does not have any free cell).

7 Conclusion

Transforming the resource allocator into a distributed version has needed to add the shared cell. This cell is shared among all nodes to broadcast informations and when communication has to occur between nodes that do not have any other communication medium. We saw that the shared cell is subject to contention, mostly during bootstrap.

This contention induces collisions, that themselves induce a delay in the request/reply transaction. This delay affects the time needed for the network to reach its steady state, but also reduces interferences between sensors during this delay. We solved the congestion problem by adding more shared slots, depending on the number of nodes in the network.

We believe that those results are reliable in a 6TiSCH context, but also for any WSN that makes use of open slots on which a considerable number of nodes can communicates, thus on which a non negligeable number of collisions may occur.

Having a distributed version of the simulator and resource allocator might permit further works: replacing the random allocator by an allocator that would gather in a certain locality, depending on the power and the distance of the sensors, the list of already allocated slots. Thanks to this knowledge, a sensor would give a slot that is not allocated or that has been allocated in a far enough place that has few probabilities to induce interferences. An idea would be an algorithm based on LLME algorithm.

The LLME algorithm gives a local control of a critical section, but also allows nodes to reach the critical section if they are distant enough (i.e. their signal do not overlap). We believe that this algorithm would increase the contention in the open slots - that can be healed as we saw in this paper - but would prevent from most of the collisions in the Tx slots.

Acknowledgments

I would like to thanks Mr. Franck Rousseau, Mr Olivier Alphan, Ms Karin Alisten and Mr. Stephane Devismes. They always took time to listen to my questions and answer to them. They also asked me questions that deepened and widened a lot my view of the problem.

References

[Dujovne *et al.*, 2014] D Dujovne, LA Grieco, M Palattella, and N Accettura. 6tisch on-the-fly scheduling draft-dujovne-6tisch-on-the-fly-04 (work in progress). Technical report, IETF, Internet Draft, Jan. 2015.[Online]. Available: http://tools. ietf. org/html/draft-dujovne-6tisch-on-the-fly-04, 2014.

[Duquennoy *et al.*, 2015] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. Orchestra: Robust mesh networks through autonomously scheduled tsch. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 337–350. ACM, 2015.

[Group and others, 2011] IEEE 802 Working Group et al. Ieee standard for local and metropolitan area networkspart 15.4: Low-rate wireless personal area networks (lr-wpans). *IEEE Std*, 802:4–2011, 2011.

[Hui *et al.*, 2010] Jonathan Hui, Pascal Thubert, et al. Compression format for ipv6 datagrams in 6lowpan networks. *draft-ietf-6lowpan-hc-13 (work in progress)*, 2010.

[Koubaa *et al.*, 2006] Anis Koubaa, Mário Alves, and Eduardo Tovar. A comprehensive simulation study of slotted csma/ca for ieee 802.15. 4 wireless sensor networks. In *5th IEEE International Workshop on Factory Communication Systems*, pages 183–192. IEEE, 2006.

[Li *et al.*, 2012] Xiaoyun Li, Chris J Bleakley, and Wojciech Bober. Enhanced beacon-enabled mode for improved ieee 802.15. 4 low data rate performance. *Wireless Networks*, 18(1):59–74, 2012.

[Mirzoev and others, 2014] Dr Mirzoev et al. Low rate wireless personal area networks (lr-wpan 802.15. 4 standard). *arXiv preprint arXiv:1404.2345*, 2014.

[Palattella *et al.*, 2016] Maria Rita Palattella, Thomas Watteyne, Qin Wang, Kazushi Muraoka, Nicola Accettura, Diego Dujovne, Luigi Alfredo Grieco, and Thomas Engel. On-the-fly bandwidth reservation for 6tisch wireless industrial networks. *Sensors Journal, IEEE*, 16(2):550–560, 2016.

[Shelby *et al.*, 2014] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). 2014.

[Vilajosana and Pister, 2016] Xavier Vilajosana and Kris Pister. Minimal 6TiSCH Configuration. Internet-Draft draft-ietf-6tisch-minimal-16, Internet Engineering Task Force, June 2016. Work in Progress.

[Wang and Vilajosana, 2016] Qin Wang and Xavier Vilajosana. 6top Protocol (6P). Internet-Draft draft-ietf-6tisch-6top-protocol-02, Internet Engineering Task Force, July 2016. Work in Progress.

[Winter, 2012] Tim Winter. Rpl: Ipv6 routing protocol for low-power and lossy networks. 2012.