Distributed PID-based Scheduling for 6TiSCH Networks

Marc Domingo-Prieto, Tengfei Chang, Xavier Vilajosana, Thomas Watteyne

Distributed PID-based Scheduling for 6TiSCH Networks

Marc Domingo-Prieto, Tengfei Chang, Xavier Vilajosana *Senior Member, IEEE*, Thomas Watteyne *Senior Member, IEEE*

***Abstract*—Industrial low power networks are becoming the nexus of operational technologies and the Internet thanks to the standardization of networking layer interfaces. One of the main promoters of this shift is the IETF 6TiSCH WG, which addresses network management and IP integration of Time Synchronized Channel Hopping (TSCH) networks as those developed by the IEEE802.15.4 TG. The 6TiSCH WG is defining the operational interface and mechanism by which the network schedule can be distributed amongst the devices in the network. This operational sub-layer, called 6top, supports distributed scheduling and enables implementers to define the scheduling policy, only standardizing the distribution mechanism. This letter proposes a novel distributed scheduling policy based on the well-known industrial control paradigm referred as Proportional, Integral and Derivative (PID) control. The proposed technique is completely decentralized, enabling each node to determine the number of cells to schedule to one another, according to its traffic demand. The mechanism is reactive to sudden or bursty traffic patterns, while staying conservative in over-provisioning cells.**

***Index Terms*—IEEE802.15.4e, 6TiSCH, Wireless Sensor Networks, Distributed Network Scheduling, Industrial IoT, TSCH.**

I. Introduction

The IETF 6TiSCH Working Group is defining the architecture that enables the convergence of IPv6 and low-power industrial deterministic networks. At its core is a management layer called 6top which is in charge of handling and managing the underlying MAC layer resources. Fig. 1 illustrates a Time Synchronized Channel Hopping (TSCH) network structure. 6top exposes clear management interfaces and mechanisms so distributed management entities can operate the network. Scheduling policies are left open to implementers, enabling vendors to develop their optimal solutions, while still being standards-compliant.

This letter presents a distributed and efficient scheduling policy. This policy is based on the well-known Proportional, Integral and Derivative (PID) control algorithm, which we use to drive the communication schedule in a TSCH network, in a distributed manner. We demonstrate that with PID scheduling, a node in the network dynamically manages its TSCH schedule to its neighbors, without requiring a central management entity or network-wide information.

This letter is organized as follow. Section II relates our work to the current state of the art, and introduces the scheduling operation in TSCH/6TiSCH networks. Section III introduces the concept of PID scheduling and its integration to OpenWSN. Section IV presents the implementation and evaluation details. Section V concludes this letter.

M. Domingo and X. Vilajosana are with Worldsensing and Universitat Oberta de Catalunya, Barcelona, Spain.

T. Watteyne is with Inria-Paris, EVA Team, France.

T. Chang is with both Inria-Paris, EVA Team, France and University of Science and Technology, Beijing, China.

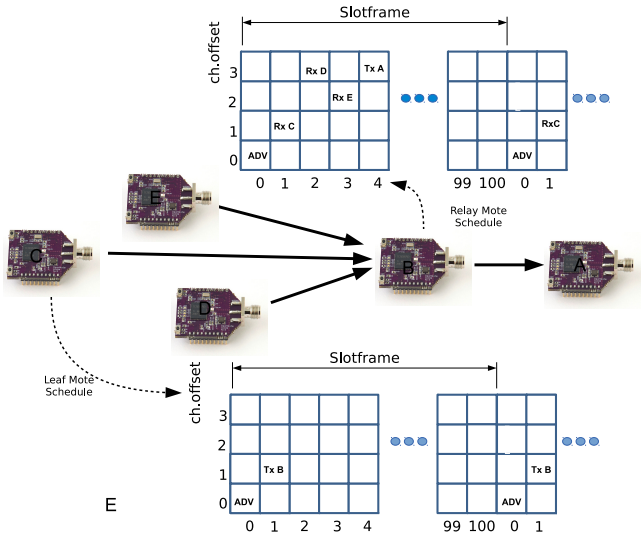

Fig. 1. Example TSCH network. "TxD" stands for "sending to node D". "RxD" stands for "receiving from node D". Cells marked "ADV" are used to advertise the presence of the network [1].

II. Scheduling in TSCH networks

A. Related Work

TSCH networks use synchronization combined with time slotted medium access to enable collision-free communication. This slotted structure is also multiplexed in frequency to be able to scale up communications and to improve reliability [2]. The IETF 6TiSCH working group is standardizing IP convergence and control plane management for these networks [3].

According to RFC7554 [1], a missing component in the TSCH architecture is an entity in charge of scheduling the TSCH cells for the nodes in the network. This entity is referred to as "Logical Link Control" (LLC), and manages and controls the schedule of the network.

The IEEE802.15.4e standard defines the mechanisms for a TSCH node to communicate; 6TiSCH defines the basic configuration [4]. No standard defines the policy to compute the communication schedule, match that schedule to the multi-hop paths maintained by the routing protocol or adapt the link-layer resources allocated between neighbor nodes to the data traffic flows.

Several approaches have been proposed to schedule TSCH networks, few of them being implemented in real hardware/software ecosystems. Palattella *et. al.* developed TASA [5], a centralized scheduler that calculates optimal schedules at the cost of intensive signaling. Tinka *et. al.* [6] developed a decentralized algorithm to schedule the network, while requiring little peer-to-peer signaling between nodes. Morell *et. al.* [7], developed an hybrid approach with the idea of label-switching in TSCH networks which proposes the use of end-to-end path reservation signals to transport

bandwidth requirements to the nodes. Allocation is done between parents and children recursively along the path. A similar approach was taken later by Accettura *et. al.* [8]. Recently, Duquennoy *et. al.* presented Orchestra [9], a best-effort decentralized approach to schedule the network. Orchestra uses randomization to allocate slots without requiring node communication. While the approach yields 99.999% packet delivery, it does not address bursty traffic as is shown later.

B. Network Scheduling in 6TiSCH Networks

According to the 6TiSCH minimal configuration [4], any 6TiSCH node must provide one shared slot to bootstrap and advertise the network. Once a node has joined the network, it uses that slot to agree on a schedule with its neighbor. The 6top sublayer is in charge of handling slot reservation requests between neighbors, i.e. install the required slots in the node and its target neighbor schedule. 6top contains an algorithm, called Scheduling Function (SF), which triggers when to add/delete one or more cells to a neighbor. The SF can pre-provision extra cells to cope with sudden bandwidth increases without loosing packets. Similarly, the SF can monitor the performance and utilization of each cell and possibly relocate the cell within the schedule in case it detects collisions are happening on that cell. The SF uses local information to take decisions about the schedule. The PID-based SF we propose can be plugged directly into the 6top sublayer.

III. PID SCHEDULING

A PID controller is a well-known control loop feedback mechanism used for stabilizing industrial control loops. The error minimization is done by adjusting the process variables according to the current state and the desired end point. The further off-target the process variable is, the more aggressively the PID controller corrects it.

A PID controller is generally defined as in (1), where e_t stands for the error between the current state and the desired objective. e_t is multiplied by a K_p constant which is used to fine-tune the process. The integral term of the PID, controlled by K_i, is used to take into account the error evolution along time (based on past values) and introduce certain inertia to the system. This prevents the control to be reacting proportionally to the immediate error and to maintain the correction trend according to the previous iterations. The derivative term, controlled by a K_d constant, provides a sense of the speed of the error variation (predicts future values of the error) and works in opposite direction to the integral term by reducing the inertia.

$$p_t = K_p * e_t + K_i \int_0^t e_t dt + K_d * \frac{d}{dt} e_t \tag{1}$$

This letter looks at TSCH scheduling as a closed loop control problem, with the objective to optimize the schedule of the TSCH network so the number of cells in it accommodates to the node's demand. The approach is fully decentralized: scheduling decisions are taken according to the current state of the node, without requiring global or partial network information.

The variables we optimize are the size of the queue and the number of scheduled slots that are unused in a slotframe. The PID controller computes the number of required cells in the schedule, according to the traffic demand on that particular node. At each slotframe, the PID controller determines the number of cells that need to be added or removed in a per-neighbor basis, according to the state of the queues and the previous slotframes cells usage. For example, if the queue fills up, the controller determines that more cells in the schedule are needed. Similarly, if the number of unused cells increases, the controller decides to remove some of them. Controlling the inertia of the PID is fundamental to avoid constants increments/decrements to the slot allocation because this introduces unnecessary energy consumption and packet overhead due to unnecessary allocations.

We propose a PID controller which calculates the number of cells to be scheduled at every start of slotframe, for each of the possible traffic destinations. The controller is also in charge of re-scheduling under-performing cells given a configurable threshold, e.g. if a link has an average Packet Delivery Ratio (PDR) under the threshold, the cell is relocated after N PID iterations, with N configurable. Once the controller indicates the number of cells to add/delete, the 6top protocol [10] carries out the negotiation and reservation with the neighbor. The proposed cells are selected randomly from those not being allocated in the requester's schedule. The objective of the mechanism is twofold: (1) keep a low (or zero) number of packets in the queue, (2) minimize the number of cells allocated in the node's schedule in order to reduce unnecessary energy consumption.

The number of cells to be scheduled per target node (C_{Sj}) is calculated by the PID controller using (2) as a discrete approximation of (1).

The proportional error of the PID to a particular neighbor (e_{tj}) can be seen in (3) and is calculated by the current number of packets in queue (P_{cj}) to that particular neighbor j, minus the current number of cells in the schedule to that particular neighbor (C_{cj}), and minus the target number of packet in queue (P_{tj}).

$P_{tj} = 0$ is the objective of our PID controller to maintain the queue empty. $\sum_0^{nj-1} e_{t_{nj}} * \delta_{SF}$ is the integral of the errors, that is the sum of the errors in the last N slotframes. δ_{SF} represents the slotframe duration. The last term is a derivative term that indicates how fast the error changes. In addition, note that the total number of cells to be scheduled in one iteration of the PID must not exceed the total schedule capacity as noted in (2).

$$C_{Sj} = K_p * e_{tj} + K_i \sum_0^{n-1} e_{t_{nj}} * \delta_{SF} + K_d * \frac{e_{tj} - e_{tj-1}}{\delta_{SF}},$$
$$j \in source\ nodes,$$
$$\sum_j C_{Sj} \leq Total\ slotframe\ capacity \tag{2}$$

$$e_{tj} = P_{cj} - C_{cj} - P_{tj} \tag{3}$$

To illustrate, let's imagine we use $K_p = 1$. When multiplied by e_{tj}, having the rest of K's set to zero, we would have a PID result proportional to the current error. That is, if the queue is 2 packets too large, the PID result C_S is that 2 slots need to be allocated to the schedule. By reducing the weight of the proportional error and adding weight to the integral term, we smoothen the reaction of the controller by restricting the impact of sudden queue variations as we introduce certain inertia to the system. Note that *Ks* are represented as float values [0..1] that depend on the scheduling objective that wants to be achieved. K_p weights the proportional term; it determines the amount of cells to schedule given a certain error. It is desirable to have one extra cell per extra packet in the queue; therefore K_p should be close to 1.0 but giving some space to both the integral term to avoid excess proportionality,

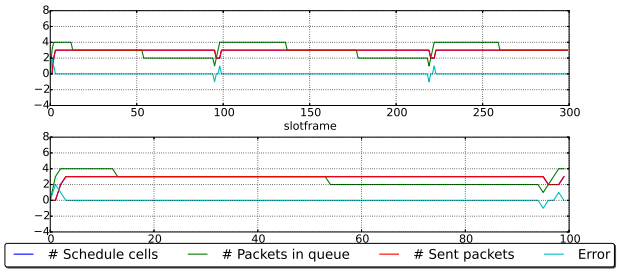

Fig. 2. Number of cells allocated for constant traffic. Top: results for 300 slotframes. Bottom: zoom on the first 100 slotframes.

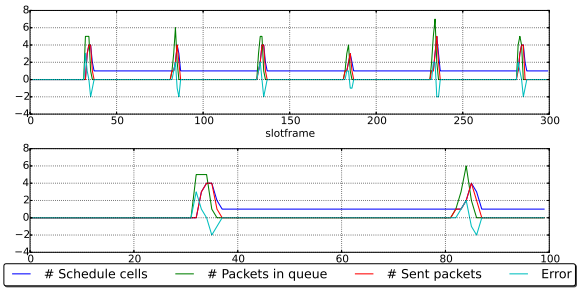

Fig. 3. Number of cells allocated for bursty traffic. Top: results for 300 slotframes. Bottom: zoom on the first 100 slotframes.

and therefore instability, and also to the derivative term. K_i is used to stabilize the controller, giving weight to the past actions. K_i depends on the window size and on the speed of the controller; its value should be approximately $\delta(SF)^n$ times smaller than K_p. Finally, K_d breaks the system inertia as it anticipates future values of the system and should have a value proportional to the speed of the system.

In the proposed approach, is important to recall that the 6top negotiation protocol does not ensure collision-free allocations, but it is the responsibility of the SF to address under-performing slots. A scheduling collision may cause a delay of several slots in the allocation procedure. These slots are dependent on the number of retries. The PID policy is simple in this case. The PID computes the required cells according to the current status. If an allocation failed in the previous attempt, the result of the PID calculation in the current iteration will be more aggressive to reach the objective as the system is drifting from it. If we are in the extreme case that no more cells in the schedule of the counterpart are available, the queue of the node grows and eventually starts dropping packets. This behavior is not a PID problem but a network planning problem as the required bandwidth is larger than the available bandwidth.

IV. Evaluation

We implemented the PID scheduling controller in the OpenWSN project [11]. OpenWSN is an open-source implementation of a standards-based protocol stack, including IEEE802.15.4e TSCH, 6LoWPAN, RPL, CoAP and the latest IETF 6TiSCH standards. OpenWSN implements the 6top sublayer [10]. The PID scheduling controller is a component that uses the 6top sublayer. Different SFs can be used in 6top to decide when to add/delete cells to each neighbor. Once the SF decides to add/delete cells, the 6top protocol locally negotiates with the neighbor. Negotiation occurs using

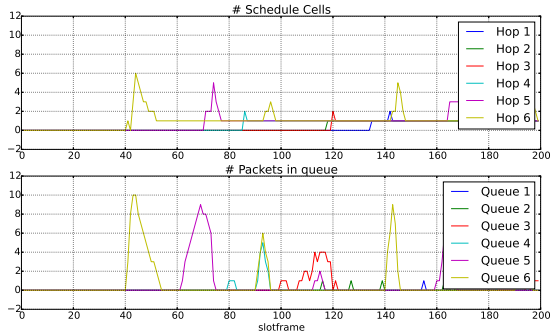

Fig. 4. Number of cells allocated for bursty traffic in a mulithop setting (6 hops). Top: For each of the hops, evolution of allocated cells in the schedule. Bottom: number of packets in the queue evolution at each hop.

pairwise communication between nodes exchanging allocation requests/responses. The PID controller is an SF which determines the number of cells that should be scheduled to a particular neighbor and invokes the corresponding 6top protocol function to request the cells. 6top is standardized in such a way that a SF can be plugged-in.

To test the performance of the PID under various types of traffic, we utilized `cstorm` application in OpenWSN. This application enables us to control the packet generation rate and therefore emulate different scenarios such as those with constant traffic and others with bursty traffic.

The evaluation aims to demonstrate that the mechanism is able to dynamically and in a distributed manner manage the schedule of the network without requiring centralized information.

A. Description of the test environment

OpenSim is the emulator of the OpenWSN project. It compiles the firmware running on real nodes as a Python extension module and enables to run them in a single computer emulating a real network. The simulated code is exactly the same as the one that runs on real devices.

The first network built with OpenSim contains two emulated nodes: node A configured as root node, node B configured as slave. The slot duration and length of the slotframe of this network are set to 15 ms and 101 slots long, respectively. Both nodes use 5 pre-configured shared cells for best effort and 6top reservation traffic. The cells reserved through 6top negotiation are dedicated cells between both nodes and hence without collision. The `cstorm` application is used to generate traffic from node B to A. The rate of the traffic is controlled by manipulating the packet generation interval. The smaller the interval, the higher the traffic. To accurately test the performance of the PID controller, `cstorm` packets are forced to be sent on reserved cells. All other packets (beacons, routing signaling traffic, etc.) can only be sent through shared cells. Only the packets in the queue generated by `cstorm` and the 6top layer are used for the PID calculation.

The experimental data is obtained at the last slot in the slotframe. The recorded information for each slotframe consists in a tuple with six elements: the number of scheduled cells, the number of packets in the queue, the number of sent packets, the traffic, the average duty cycle for the slotframe and the error (as e_t in (3)). The traffic represents the number of packets generated every slotframe.

A second setting is used to evaluate the performance of the PID scheduling in a multihop network. We use a 7 node linear

network, or 6 hops. Only the last node is generating `cstorm` traffic. In addition, we limit the number of shared slots to 2 between each pair of nodes, aiming to see more cells being reserved all along the 6 hops.

B. Experiments

In our experiments, we aimed to evaluate the performance of the PID scheduler under constant and bursty traffic and compared it to a well-known state of the art mechanism referred as Orchestra [9]. Two experiments have been conducted with a predetermined PID configuration with values ($K_p = 0.7$, $K_i = 0.075$, $K_d = 0$, $n = 4$). The PID configuration dependents on the application objective and part of the predeployment adjustment and tuning. We plot the variation of the number of scheduled cells in the schedule, packets in the queue, the sent packets and the error, as shown in Fig. 2 and Fig. 3. Fig. 2 presents the cases for constant traffic. For those, the `cstorm` packet generation rate is set to 3 packet every slotframe. Per Fig. 2, the scheduled cells are stable at 3 allocated cells. Despite the fact that packets in the queue eventually oscillate from 4 to 2, the scheduled cells stay at 3 and do not drop to 2 until the number of packets in the queue drop to 1; then, when the packets in the queue increase again the PID reacts in just 2 slotframes to reallocate the 3 slots.

To evaluate a bursty traffic scenario, we use a period of 50 slotframes to generate traffic. During the first 3 of the 50 slotframes, the packet generation rate is set randomly between 3 and 5 packets per slotframe. No packets are generated in the remaining 47 slotframes. As seen in Fig. 3, every time a traffic burst occurs, the PID controller allocates the determined number of cells through the 6top management interface, and recycle them after the end of the burst within 4 to 5 slotframes (due to the integral weight and the overhead of the negotiation protocol). The number of allocated cells is reduced to one after the end of the burst. At that point, the number of cells to be removed is less than one; thus, no cell can be removed, and the allocated cells are kept at one until next traffic burst.

A second experiment, depicted in Fig. 4 presents the evolution of cell allocations along a multihop path. Only the leaf node (hop 6) generates traffic using `cstorm` every 50 slotframes. We can see the filtering effect caused by progressive allocation of cells (smooth ramp due to inertia introduced by the PID integral term) which causes the next hop to require less "extra" slots to relay the traffic as their queue size grows more progressively.

Finally, in Fig. 5, the duty cycle of a network with a static schedule (Orchestra [9]) is compared to the PID scheduler. In the experiment, we use bursts of 8 packets/slotframe during 3 slotframes every 50 slots for a three-node network. Three different cases are considered following the Orchestra approach. Receiver oriented (Rx cells), Sender oriented (Tx cells) and Relay schedules (Both Tx and Rx cells). The static scheduler has been configured to deal with that burst using a fixed capacity of 8 dedicated slots per slotframe according to the evaluated case (Tx, Rx or Relay). The figure presents the duty cycle per slotframe according to the node's role (Sender, Receiver or Relay). In the top, we see the overhead caused by the signaling of the PID which causes a slightly higher duty cycle that tends to zero as the traffic demand decreases. In this case, the static schedule configuration outperforms in average slightly the PID scheduler as transmission cells are not used if queues are empty; the PID introduces an overhead to remove them from the schedule. However, the middle and bottom figures, showing the receiver and relay oriented measurements,

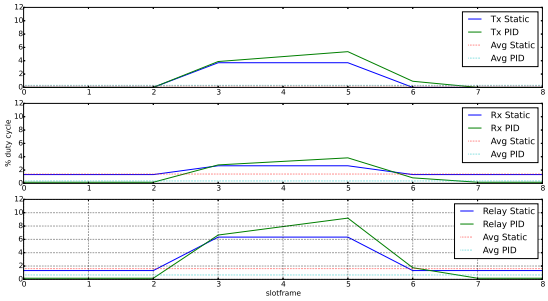

Fig. 5. Duty cycle comparison between the transmission oriented, sender oriented and relay scenarios during a burst. Average traffic in dotted lines.

indicate that the PID removes Rx cells not being used after a burst, saving the nodes to idle-listen in those slots. Average doted lines are provided for the three cases, showing that the PID approach outperforms static scheduling in terms of duty cycle when the network activity has a bursty nature.

V. Conclusions

This letter presents a distributed and efficient scheduling policy for 6TiSCH networks. This policy is based on the PID industrial control paradigm and uses the 6top management layer being standardized at the IETF. The mechanism targets schedule stabilization for coping with dynamic application demands in a distributed manner, and uses pairwise communication between neighbor nodes. We demonstrate that a PID schedule is able to cope with different types of traffic and react autonomously to sudden demand variations, targeting stabilization and minimization of cells in the schedule and the queue.

References

[1] T. Watteyne, M. R. Palattella, and L. A. Grieco, *Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement*, IETF Std. RFC7554, May 2015.

[2] T. Watteyne, A. Mehta, and K. S. Pister, "Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense," in *Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN)*, Tenerife, Canary Islands, Spain, 29-30 October 2009.

[3] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: Deterministic IP-enabled Industrial Internet (of Things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, 2014.

[4] X. Vilajosana and K. Pister, *Minimal 6TiSCH Configuration*, IETF Std. draft-ietf-6tisch-minimal-12 [work-in-progress], 21 September 2015.

[5] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic Aware Scheduling Algorithm for Reliable Low-power Multi-hop IEEE 802.15.4e Networks," in *International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*. Sydney, Australia: IEEE, 9-12 September 2012, pp. 327–332.

[6] A. Tinka, T. Watteyne, K. S. Pister, and A. M. Bayen, "A Decentralized Scheduling Algorithm for Time Synchronized Channel Hopping," *Transactions on Mobile Communications and Applications*, vol. 11, no. 1, pp. 201–216, September 2011.

[7] A. Morell, X. Vilajosana, J. L. Vicario, and T. Watteyne, "Label Switching over IEEE802.15.4e Networks," *Wiley Transactions on Emerging Telecommunications Technologies*, vol. 24, no. 5, pp. 458–475, 3 May 2013.

[8] 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 *International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 4-7 June 2013, pp. 1–6.

[9] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH," in *International Conference on Embedded Networked Sensor Systems (SenSys)*. Seoul, South Korea: ACM, 1-4 November 2015.

[10] Q. Wang and X. Vilajosana, *6TiSCH Operation Sublayer (6top)*, IETF Std. draft-wang-6tisch-6top-sublayer-03 [work-in-progress], 6 July 2015.

[11] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. D. Glaser, and K. Pister, "OpenWSN: a Standards-based Low-power Wireless Development Environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.