C-TSCH: A Centralized Scheduler for TSCH

C-TSCH: A Centralized Scheduler for TSCH

Laura Harms
Department of Computer Science and Engineering
Chalmers University of Technology, Sweden
harms@chalmers.se

Abstract

We present a centralized scheduler for reliable communications in low-power wireless TSCH networks. This scheduler enables us to introduce a novel centralized routing approach not dependent on existing distributed solutions like RPL.

1 Introduction

The growing amount of wireless Internet of Things (IoT) applications and particularly the Industrial Internet of Things (IIoT) imposes harsh requirements on the communication especially in regards to reliability and latency. As most IoT devices are battery powered a low energy consumption is required as well. The requirements led to the development of many distributed solutions trying to maximize reliability and minimize latency in non-predefined networks. These solutions include distributed flooding protocols like Glossy [5] or Chaos [6] as well as protocols based on the distributed routing protocol RPL [8], like Orchestra [3]. All of these protocols can achieve high reliability but can't guarantee holding predefined deadlines. A centralized approach on the other hand is able to give these guarantees, as the latest delivery time of a packet is known beforehand. This knowledge exists because the centralized scheduler has a central view on the topology of the network as well as which data has to be sent at which point in time and can therefore create flows (scheduled routes) with exact timing knowledge holding the predefined deadlines. However, existing centralized schedulers like C-LLF [7] focus on high schedulability and create schedules for (almost) interference-free networks with highly reliable links. This neglection of interference and the resulting lack of retransmissions leads to low latency but also to low reliability in non-interference-free environments.

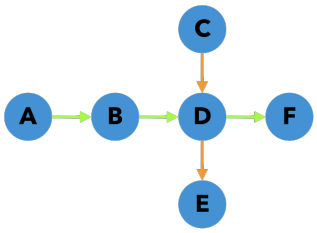
We present a centralized scheduler for TSCH [4] combining the scheduling of communications as well as its routing, avoiding the overhead of a routing protocol like RPL. Moreover, we introduce a novel routing strategy named *sliding windows* to keep latency as low as possible while still being able to use retransmissions and hold predefined deadlines. This strategy allows our scheduler to work in networks susceptible to interference combining the advantages of both distributed and centralized scheduling approaches. Moreover, we introduce flow-based queues for TSCH to overcome the limitations of neighbor-based queues as currently found in the TSCH implementation of Contiki-NG [1].

2 Centralized Scheduler (C-TSCH)

The centralized scheduler, we present in this paper operates as a network layer on top of the layer implementation of TSCH in Contiki-NG [1]. Contrary to previous works on centralized schedulers, like C-LLF [7] focusing mainly on the schedulability of the proposed scheduler, the focus of our scheduler lies on the reliability of communications in networks susceptible to interference.

The scheduler consists of a Contiki network layer and a scheduling software including state-of-the-art scheduling algorithms as well as modifications to these to accomplish higher reliabilities using the concept of sliding windows. The functions of the network layer consist of a neighbor discovery, the implementation of the TSCH schedule and the main functions of the network layer, sending, receiving and forwarding of data packets. The neighbor discovery can either be performed before run-time, using broadcasts and individual transmission slots for each node in the network with all other nodes listening, similar to sender-based dedicated slots (SBD) in Orchestra, or during run-time using modified TSCH beacons. The standard way in the current implementation is performing the neighbor discovery before run-time to be able to create comparable schedules using different scheduling algorithms, but the goal is to switch to beacon-based neighbor discovery later.

Besides the mentioned network layer implementations, some layer modifications were necessary to overcome certain limitations. These modification include the TSCH beacons but more important the TSCH queues had to be changed to enable the forwarding of a packet to a certain neighbor prior to forwarding an earlier received packet to the same neighbor. To allow this behaviour, we added flow-based queues, in addition to the neighbor-based queues of TSCH. These flow-based queues are possible because the traffic of



(a) Topology

	1	2	3	4	5	6	7	8	9	10
A					TX	TX	TX	TX		
B					RX	RXTX	RXTX	RXTX	TX	
C	TX	TX	TX	TX						
D	RX	RXTX	RXTX	RXTX	TX	RX	RXTX	RXTX	RXTX	TX
E		RX	RX	RX	RX					
F							RX	RX	RX	RX

(b) interleaving-free SPF with sliding windows

Figure 1. An example of a schedule combining the sliding window strategy with an interleaving-free shortest path first scheduler. One packet is to be transmitted from node A to node F and another packet from node C to node E. The size of the sliding window is 5, as seen for nodes B and D.

each slot is defined by the scheduler and belongs to a certain communication flow.

The scheduling software, which is written in Python to allow an easier implementation of new schedulers and routing or retransmission strategies, computes the schedule based on the output of the neighbor discovery using the retransmission strategy of sliding windows and one of the implemented scheduling algorithms including interleaving-free shortest path first. Currently, the scheduler generates the schedule as C-code which then can be uploaded to the nodes, but for a future version of the scheduling software, we plan to generate and upload the schedule to the nodes during run-time.

2.1 Sliding Windows

The scheduling strategy of sliding windows introduces a variable number of retransmissions for a flow/route from a transmitter to a receiver to add robustness of the system towards interference while having the least possible increase of latency added by retransmissions. An important parameter of this strategy is the window size which is the number of timeslots a node is maximally involved in for communication. This window size also determines how many timeslots a transmission maximally takes. In the best case, without interference, it takes no more timeslots than the number of hops like traditional scheduling algorithms, whereas it can take a maximum of 2 less than the window size additional timeslots to perform the end-to-end transmission. The window size can be dependant on the expected number of transmissions or can be fixed.

Figure 1 shows a possible schedule for two communication flows from node A to F and from node C to E with a fixed window size of 5, involving each node of a flow for up to 5 timeslots. Every node which might be sending or receiving in a given timeslot (marked with RXTX) has a shared slot of both modes executing one or the other based on the previous reception of the packet sent in this communication. If a transmission was successful, a node stops communicating for the rest of its window and the next communication of the flow can take place.

3 Conclusion and Future Work

The Contiki scheduler module as well as the external centralized scheduler are already implemented including sliding windows with a fixed or ETX-dependant size using interleaving-free shortest path first as scheduler. Initial experiments showed that high reliabilities are achievable given that an optimal window size is chosen. The current work using flow-based queues will presumably increase the reliability even further. Furthermore, the current state of the scheduler is tested at this years EWSN Dependability Competition.

In the near future, we want to extend our scheduler by implementing additional state-of-the-art scheduling algorithms and use them in combination with sliding windows. Moreover, we will analyze how to avoid long waiting times at intersections of flows to increase the schedulability of sliding window based flows. As mentioned before, we will also create a way to upload a schedule during run-time. Furthermore, we plan an additional routing strategy called Autobahn enabling multi-path routing.

Our long term vision is to include battery-free nodes in our system which should be possible due to the schedule providing timeslots in which transmissions can happen. Moreover, we might use a software defined networking architecture for our scheduler similar to μSDN. [2]

4 Acknowledgments

This work was supported by the Swedish Foundation for Strategic Research (SSF) through the project LoWi, reference FFL15-0062. Moreover, I would like to thank my supervisor Olaf Landsiedel for his valuable comments and suggestions to improve this work.

5 References

[1] Contiki-NG: The OS for Next Generation IoT Devices. http://contiki-ng.org/.

[2] M. Baddeley et al. Evolving SDN for Low-Power IoT Networks. In *IEEE NetSoft*, 2018.

[3] S. Duquennoy et al. Orchestra: Robust mesh networks through autonomously scheduled tsch. In *ACM SenSys*, 2015.

[4] S. Duquennoy et al. TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation. In *IEEE DCOSS*, 2017.

[5] F. Ferrari et al. Efficient network flooding and time synchronization with Glossy. In *ACM/IEEE IPSN*, 2011.

[6] O. Landsiedel et al. Chaos: Versatile and Efficient All-to-All Data Sharing and In-Network Processing at Scale. In *ACM SenSys*, 2013.

[7] A. Saifullah et al. Real-Time Scheduling for WirelessHART Networks. In *IEEE RTSS*, 2010.

[8] T. Winter et al. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. Technical report, 2012.