Stripe: a Distributed Scheduling Protocol for 802.15.4e TSCH Networks

Iacob Juc, Olivier Alphand, Roberto Guizzetti, Michel Favre, Andrzej Duda

Stripe: a Distributed Scheduling Protocol for 802.15.4e TSCH Networks

Iacob Juc*¶, Olivier Alphand*, Roberto Guizzetti¶, Michel Favre¶, and Andrzej Duda*

*Grenoble Alpes University, Grenoble Institute of Technology, Grenoble Informatics Laboratory, Grenoble, France
¶STMicroelectronics, Crolles, France
Email: {firstname.lastname}@imag.fr, {firstname.lastname}@st.com

***Abstract*—The 802.15.4e Time Slotted Channel Hopping (TSCH) mode defines how nodes operate according to a common shared schedule that determines which device may transmit frames on a given channel and during a given timeslot. By taking advantage of channel diversity, TSCH increases reliability and cell allocation to two nodes for a given transmission results in reduced collision probability. As the standard does not specify how to construct the common schedule, there is a need for finding adequate scheduling algorithms. In this paper, we propose Stripe, a *distributed scheduling* protocol that allocates timeslots in temporal alignments having the property that multi hop forwarding of packets benefits from the minimal delay. Stripe ensures short delays both for upward and downward traffic. The protocol comprises two phases: the *relocation phase* that reconfigures the random pre-allocated cells in a schedule fitting convergecast traffic and a *reinforcement phase* that schedules additional cells to support the traffic generated and relayed by each node towards the sink. We evaluate Stripe with an enhanced 6TiSCH simulator and compare its performance with Orchestra [1]. The results from extensive simulations show that Stripe presents fast convergence, short delays, and improved packet delivery ratio.**

***Index Terms*—802.15.4e, TSCH, distributed scheduling, WSN, LLN**

I. Introduction

The IEEE 802.15.4 standard [2] specified the Time Slotted Channel Hopping (TSCH) mode in the 802.15.4e extension [3], [4]. TSCH takes advantage of channel diversity through *channel hopping* to increase reliability: devices switch channels according to a predefined sequence for each communication in a reserved *timeslot*. Devices operate according to a *common shared schedule* that determines which device may transmit frames on a given channel and during a given timeslot, which is long enough to accomodate an acked transmission of a maximum size packet. Timeslots have a fixed length across the entire network, typically $10\,\mathrm{ms}$. A *(timeslot, channel offset)* tuple is called a *cell*, the units of allocation within a fixed-length structure called *slotframe* that repeats over time.

The schedule of a node is defined as the collection of all active cells. More formally, a schedule is a sequence of 3-tuples *(timeslot, channel offset, associated action)* (action can be a transmission or a reception). By extension, the global schedule is the sum of all individual schedules. To follow the schedule, nodes need to be synchronized, so they can benefit from the deterministic behavior and some level of the required quality of service resulting from the schedule allocation: in the ideal situation, there is only one transmission during a slot, so TSCH eliminates collisions.

TSCH has attracted considerable attention from the Wireless Sensor Network community with the establishment of the IETF 6TiSCH Working Group [5] whose goal is to define the operation of Low-power and Lossy Networks (LLN) under RPL routing (Routing Protocol for Low-power and lossy networks) [6] over TSCH and specify essential protocols for elaboration of scheduling policies [7]. The 802.15.4e standard defines how a collection of nodes is able to operate in a multi-hop network based on a TSCH schedule, but does not specify how the schedule is constructed. The 6TiSCH working group began to specify a schedule construction algorithm [8] and several authors proposed algorithms for constructing schedules.

In this paper, we introduce Stripe, a distributed scheduling protocol that builds a schedule over a TSCH network in a decentralized way. The protocol allocates timeslots in *stripes*, temporal alignments of timeslots having the property that multi hop forwarding of packets benefits from the minimal delay. Stripe ensures short delays both for upward and downward traffic. To achieve efficient forwarding, the protocol builds a tree-like topology in which nodes obtains an optimized bandwidth allocation (in terms of the number of active cells) based on their position in the tree and the weight of the associated sub-tree. The protocol comprises two phases: the *relocation phase* that reconfigures the random pre-allocated cells in a schedule fitting convergecast traffic and a *reinforcement phase* that schedules additional cells to support the traffic generated and relayed by each node towards the sink.

We evaluate Stripe with an enhanced 6TiSCH simulator[1] and compare its performance with Orchestra [1]. The results from extensive simulations show that Stripe presents fast convergence, short delays, and better packet delivery ratio.

The rest of the paper is structured as follows: we start with a discussion of the related work in Section II and give the details of the protocol in Section III. Section IV reports on the simulation comparison with other algorithms and we conclude in Section V.

[1] https://bitbucket.org/6tisch/simulator/

II. Related Work

Several authors proposed scheduling algorithms for TSCH adopting either a centralized or a distributed approach to establishing a schedule. We focus on distributed protocols that compare with Stripe.

We discuss first DeTAS (Decentralized Traffic Aware Scheduling) [9] and Wave [10] because they share a lot of features in what concerns the required input and the operation of the allocation algorithm. They build a collision-free schedule for one or several convergecast routing graphs constructed by RPL or other routing protocols and operate on an initial allocation resulting from running the 6TiSCH minimal configuration—all nodes are synchronized and they have a basic allocation of cells available for the protocols to operate. The allocation algorithms aims at building a schedule based on the traffic generated by source nodes.

Each node *i* computes a global queue level *Qi* as the sum of the traffic contribution of itself and its children. In both protocols, nodes require some form of input from their parents to start. They both apply a greedy heuristic, prioritizing the child with a bigger traffic requirement. Starting from a sorted list of children *Qc*, a node running DeTAS allocates contiguous consecutive chunks to each of its children, whereas the allocations in Wave are interleaved. The resulting allocations are similarly compact, yet Wave spreads the delay more evenly among child nodes. DeTAS avoids buffer overflow by alternating the sequence of transmit/receive slots for each node. However, if other links exist in addition to the convergecast links, collisions may occur.

A node joining or leaving the network can modify the corresponding queue levels on the path linking the node to the sink. Because of the way DeTAS works, the network re-organization would then potentially concern the entire network, whereas Wave can apply more conservative adjustments.

Both algorithms are not entirely distributed, as each child needs some input from its parent to locally execute the algorithm, which results in a wave-like propagation of the execution front across the tree. By construction, DeTAS avoids interference between devices, as it temporally separates the branches of the tree and re-uses channels 3 hops away. Wave requires the knowledge of the conflict graph, but Wave itself does not specify how it should be built, it only gives the rules to apply in its construction.

The downside of Wave is that it requires packets to be exchanged on top of the conflict graph, often not a tree, which presents the risk of creating loops when several nodes take the same decision based on their local information and communicate the decision to their conflicting neighbors. DeTAS requires no transversal communications across the starting tree.

The recent Orchestra protocol assumes fully desynchronized and disorganized nodes, but requires some pre-programmed information to be flashed on nodes prior to the deployment.

Orchestra is based on the idea that a slotframe can be used as a *codomain* to a function. The function is globally known and nodes advertise to their neighbours a subset of the function domain. Neighbours apply this input to the function and compute in what slot(s) the corresponding node will be active.

The simplest function that can be used is *identity* on the $(1, N)$ domain. For example, all nodes are counted and are assigned the corresponding index. Then, each node will be active in the *slot* corresponding to their index, modulo the size of the slotframe. In a slotframe of size greater than N, all nodes would have their individual slot.

In the implementation, Orchestra uses a more advanced hash function, which also returns a channel offset. Moreover, additional information is propagated, concerning the type of activity in the corresponding (Timeslot, Channel) cell, i.e., transmitting, listening, or providing shared access. The function and the identifiers used as inputs can be modified at runtime by an external entity via a CoAP [11] interface. Similarly to Orchestra, Stripe does not require the establishment of a routing structure before operating and provides a means for allocation of contiguous cells resulting in short delays.

III. Stripe protocol

The Stripe protocol implements distributed scheduling of TSCH cells to reduce latency and improve packet delivery ratio for convergecast traffic.

Stripe assumes that nodes already benefit from random pairwise allocations with some or all of their 1-hop neighbours. Thus, in addition to already scheduled shared cells inherent to the slotframe in the minimal 6TiSCH configuration [12], nodes also have two scheduled dedicated cells (a Tx cell and a Rx one) per neighbour. The channel and time offsets of those cells are randomly allocated either by an autonomous approach [13] like Orchestra [1] or by a specific neighbour discovery protocol run before or in parallel of the TSCH network construction. One benefit from such an approach is to speed up the schedule setup process by avoiding the inherent contention due to a low ratio of shared cells in slotframes.

Stripe operates in two phases: the *relocation phase* and the *reinforcement phase* that may overlap in time. Figure 1 gives an overview of the cells scheduled by each phase. The traffic assumption for Fig. 1 and hereunder is that each node generates one packet to the sink per slotframe. We adopt a simple topology presented in Fig. 4 to illustrate the operation of the protocol.

In the *relocation phase*, upstream pre-allocated cells are reassigned to move from a random schedule to *stripes*, temporal alignments of consecutive contiguous timeslots along the collection tree (DODAG). Concerning the downstream cells, Stripe merges all the pre-allocated dedicated cells from one parent towards its children into one broadcast cell as indicated in Fig. 1. All those broadcast cells are also relocated to have the form of a downstream stripe pattern of contiguous cells. We can proceed in this way bacause the downstream traffic is usually low.

In the *reinforcement phase*, every parent negotiates cells with its own parents to support the additional traffic that cannot

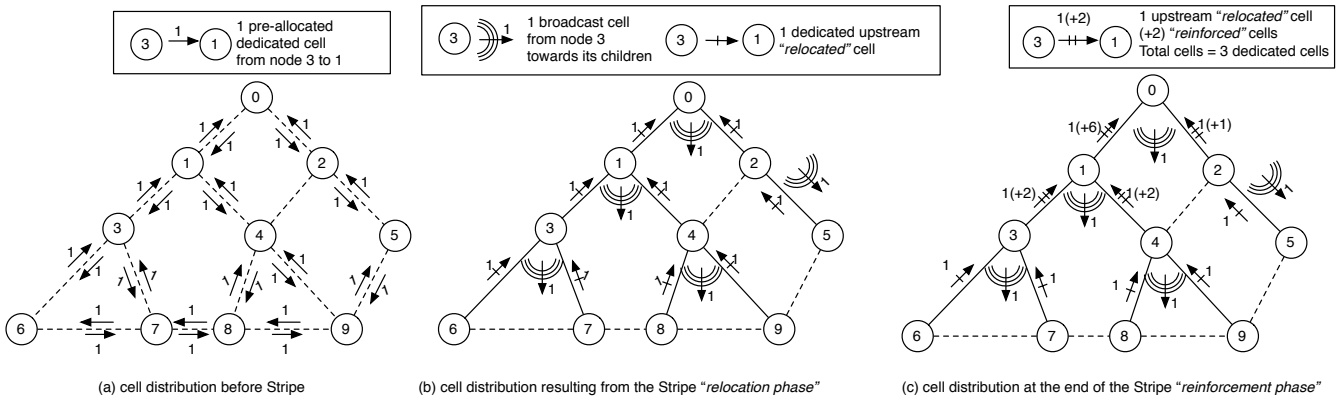

Figure 1. Cell scheduling resulting from each phase of Stripe under assumption of 1 packet per node per slotframe data traffic.

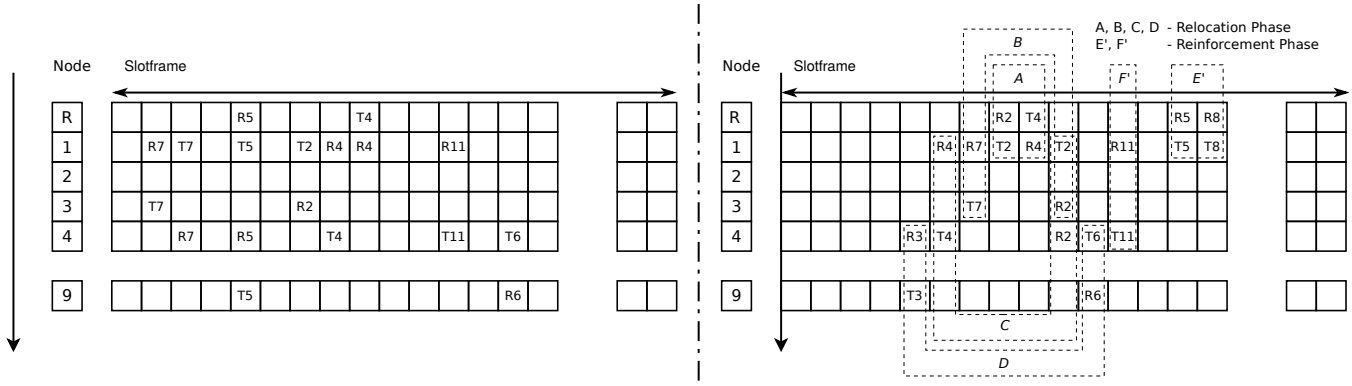

Figure 2. Cell allocation in a slotframe before Stripe and after its execution

be handled by the relocated upstream cells alone. For instance, Node 4 in Fig. 1 will request two additional cells from Node 1 to forward the trafic from its children (Nodes 8 and 9). Similarly, Node 1 will request six additional cells from the sink (Node 0), which accounts for the total trafic in the subtree below Node 1 less the already relocated cell with the sink.

Stripe defines three types of messages: Stripe Request (used in the *relocation phase*), Stripe Confirm, and Stripe Reply (used in the *reinforcement phase*).

Initially, the sink bootstraps the Stripe protocol. It sends a unicast Stripe Request in each Tx pre-allocated cell dedicated to one of its neighbours and proposes in its payload the relocation of the Rx and Tx cells to two new positions defined by the tuple [(TxSlot, TxChannelOffset), (RxSlot, RxChannelOffset)].
The sink chooses these positions randomly. Once a neighbour accepts a relocation, it automatically becomes a child of the proposing node. It also becomes part of the Stripe schedule and therefore, it is then able to send its own Stripe Requests to its neighbours. Thus, step by step, the Stripe Requests gradually reach all the nodes. Fig. 2 shows the cell allocation before Stripe and after its execution.

Nodes that accept the relocation notify the sender by acknowledging the Stripe Request with a specific MAC layer Enhanced ACK[2] called Stripe ACK while the nodes not interested in the relocation reply with a Negative Stripe Ack (NACK). This negative NACK triggers unscheduling of pre-allocated cells on both sides. Fig. 1(b) illustrates the mechanism: pre-allocated cells on links 6-7, 7-8, 8-9, and 9-5 are unscheduled. Explicit NACKs also prevent loops and stop further retransmissions of Stripe Requests towards those nodes. The obvious interest of such an exchange is that it takes place in one cell instead of two or three cells for respectively 2-step or 3-step 6top transactions [7]. Nodes that already belong to the Stripe schedule can also generate Stripe Requests when a new node attaches to them. Events A, B, C, and D in Fig. 3 illustrate the successive steps of the *relocation phase* for Nodes 0, 1, 3, 4, and 9 arranged according to the topology presented in Fig. 4.

The *reinforcement phase* takes place in parallel to the *relocation phase*. More specifically, when a node receives Stripe ACKs from its neighbours to its Stripe Requests, it will have to forward their traffic to its own parent. The negotiation of additional cells to communicate with its parent is carried out by exchanging the Stripe Confirm and Stripe Reply messages.

[2]The Enhanced ACK is in fact a MAC response frame that can either indicate ACK or NACK, as well as other information. See the full discussion at https://www.ietf.org/mail-archive/web/6tisch/current/msg02263.html

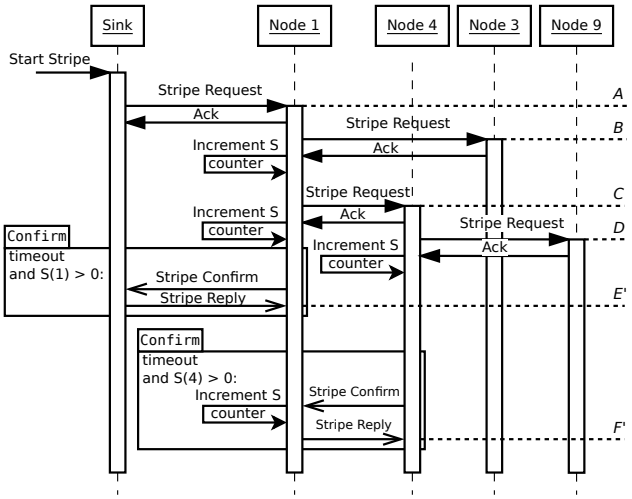

Figure 3. Message time diagram in example Stripe execution

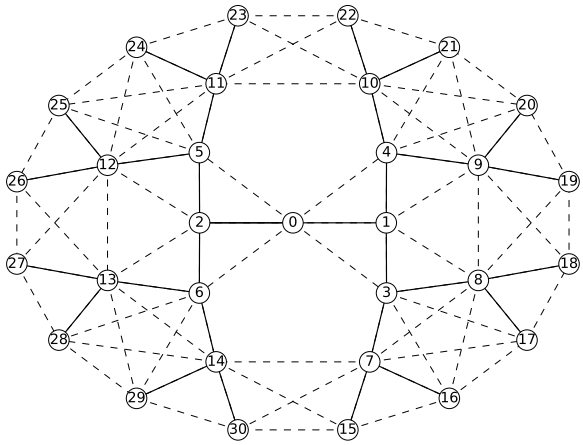

Figure 4. Topology used in the protocol description (dashed lines: connectivity, plain lines: established links; actual positions of nodes on the simulation plane)

This exchange will follow the 2-step whitelist 6P Transaction scheme [7] with whitelisted [8] cells. Thus, each child includes a list of candidate slots in the Stripe Confirm (see Fig. 3, D event) based on the number of required slots. The parent then uses this information to propose an allocation (see Fig. 3, E event). The counter S determines the number of cells requested in the Stripe Confirm (see Fig. 3, B, C, D, F events) incremented by the number of cells requested by each node placed lower in the subtree.

Then, a parent sends a Stripe Confirm message to its own parent requesting an additional slot allocation. The parent replies with a Stripe Reply message and generates a Stripe Confirm message to its parent. Step by step, Stripe Confirm messages propagate up to the sink.

IV. Evaluation of Stripe

We have implemented Stripe in the discrete-time 6TiSCH simulator. The simulator has the time resolution of one timeslot. Table I presents the parameters of the simulation. For Stripe, we measure the performance for the relocation phase and for the complete execution.

Table I
Simulation parameters

Number of nodes, unless stated otherwise	50
Slotframe size in slots	151
Slotframes per run	170
Number of runs	100
Packet buffer size	10
Simulation surface	$1\,\mathrm{km}^2$
Packet generation rate (per node)	1 every 12 slotframes

We consider two cases: i) random topologies with uniformly distributed nodes (marked random in the figures) and ii) a structured binary tree, similar to the one used in the DeTAS experimental setup [14] (marked binary). The simulator generates random topologies subject to the constraint that each node needs to have at least four neighbors with good connectivity. For comparisons, we reuse the random topologies between runs, so each mesurement point is generated in the same topology. An example of the random topology is shown in Fig.5.

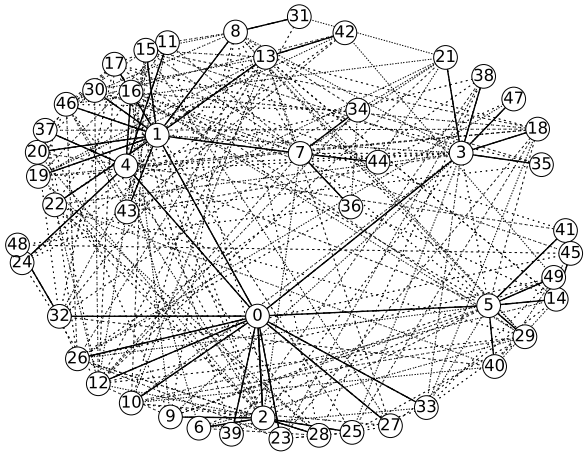

Figure 5. Example of a random topology with 50 nodes (dashed lines: connectivity, plain lines: established links)

Table II
Convergence time

Network size/ Topology	Convergence time in timeslots	Allocated timeslots
15 binary	570 ± 21	36
15 random	709 ± 59	36
31 binary	1567 ± 64	114
31 random	1810 ± 277	100
50 random	3837 ± 393	180

Table II gives the convergence time in timeslots for the Stripe protocol and the resulting allocated TxSlots. The time is exponential in function of the number of nodes and in fact is function of the network density. Because of the way the

simulator operates, it is difficult to set the desired network density beforehand (however, we can assess the density a posteriori).

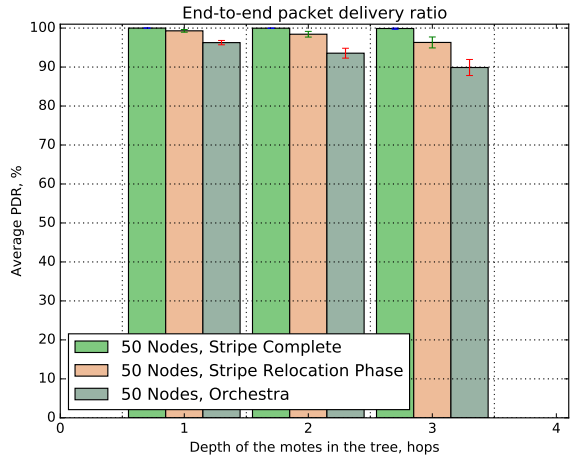

Figure 6. Average Packet Delivery Ratio for each depth of the tree (DODAG)

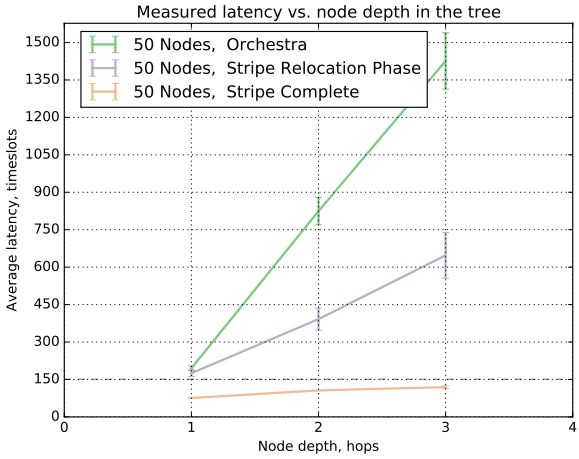

Figure 7. Average end-to-end latency measured at each depth of the network

Figure 6 compares the end-to-end Packet Delivery Ratio for a network configured according to the parameters in Table I. Fig.7 presents the resulting latency.

Next, we evaluate the performance of Stripe in different topologies and for various traffic rates. Figure 8 illustrates the end-to-end delay measured at the leaves of the network for the binary tree and random topologies. We can notice an increased confidence interval for random topologies—for the binary topology, the network density is fixed by construction and the distribution of the protocol convergence time is narrower compared to random topologies. Figure 9 shows the end-to-end PDR for leaves running in the same configuration. The binary topology of 31 nodes shows a slight drop when increasing the data rate. As illustrated in Fig. 4, we impose a considerable amount of interference between leaves.

Finally, Fig. 10 presents the evolution of the total number of allocated Tx cells versus the number of exchanged Stripe packets as well as the number of Stripe Packets that receive no Acks (we consider that the Stripe packets that receive a Stripe Negative Ack as successfully transmitted). The initial drop in the number of the reserved Tx cells is due to the nodes dropping initially allocated cells with neighbors that reply with Negative Acks, as well as merging the downstream cells. We detect a bottleneck during the early bootstrapping phase with a peak in failed Stripe packet transmissions.

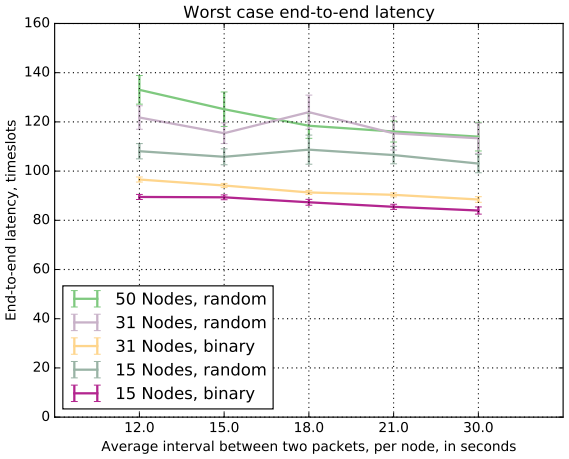

Figure 8. End-to-end delay

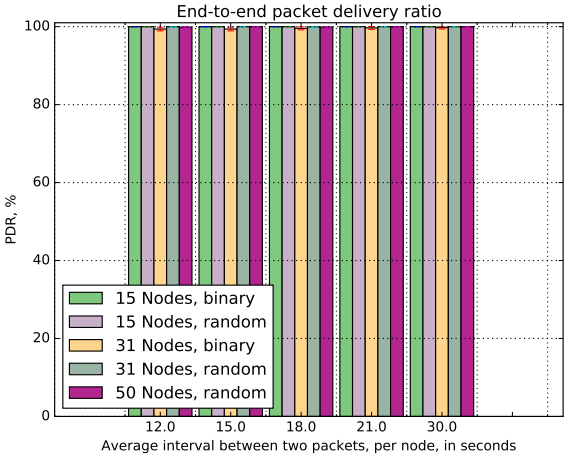

Figure 9. End-to-end Packet Delivery Rate

V. Conclusion

In this paper, we have proposed Stripe, a protocol for constructing a schedule of a TSCH network in a decentralized way. The idea is to allocate timeslots in *stripes*, contiguous

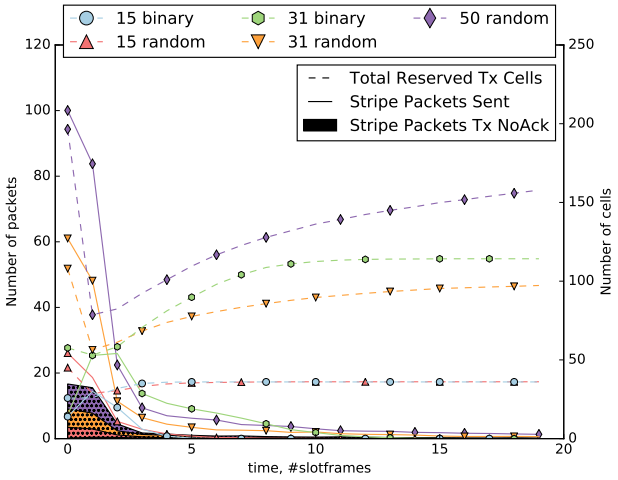

Figure 10. Cell allocation versus Stripe traffic over time

temporal alignments of timeslots enabling fast multi hop forwarding of packets. The protocol starts with an initial non-optimized allocation of slots and proceeds with reallocation to obtain upstream and downstream stripes. Moreover, the protocol achieves a funnel-like allocation in which nodes close to the sink obtain more active cells to be able to forward converging traffic. The allocation is based on the node position in the collection tree (DODAG) and corresponds to the weight of the associated sub-tree.

We evaluate Stripe through simulations in the extended 6TiSCH simulator. We compare Stripe with Orchestra with respect to the delay and packet delivery ratio. The simulations show that Stripe exhibits much shorter delays and improved packet delivery ratio.

REFERENCES

[1] S. Duquennoy, B. A. Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust Mesh Networks through Autonomously Scheduled TSCH," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems, SenSys 2015, Seoul, South Korea, November 1-4, 2015*, 2015, pp. 337–350.

[2] "IEEE Standard for Information technology–Local and metropolitan area networks–Specific requirements–Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pp. 1–320, Sep. 2006.

[3] "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 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, pp. 1–225, Apr. 2012.

[4] "IEEE Standard for Low-Rate Wireless Networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, Apr. 2016.

[5] 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 RFC 7554, Oct. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7554.txt

[6] T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks." [Online]. Available: https://tools.ietf.org/html/rfc6550

[7] Q. Wang and X. Vilajosana, "6top Protocol (6P)," IETF, Internet Draft draft-ietf-6tisch-6top-protocol-03, Oct. 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-6top-protocol-03

[8] E. D. Dujovne, L. Grieco, M. Palattella, and N. Accettura, "6TiSCH 6top Scheduling Function Zero (SF0)," Internet Draft draft-ietf-6tisch-6top-sf0-02. [Online]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-6top-sf0-02

[9] N. Accettura, M. Palattella, G. Boggia, L. Grieco, and M. Dohler, "Decentralized Traffic Aware Scheduling for Multi-Hop Low Power Lossy Networks in the Internet of Things," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, Jun. 2013, pp. 1–6.

[10] R. Soua, P. Minet, and E. Livolant, "Wave: a Distributed Scheduling Algorithm for Convergecast in IEEE 802.15.4e Networks (Extended Version)," Inria, report, Jan. 2015. [Online]. Available: https://hal.inria.fr/hal-01100420/document

[11] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)." [Online]. Available: https://tools.ietf.org/html/rfc7252

[12] X. Vilajosana, K. Pister, and T. Watteyne, "Minimal 6TiSCH Configuration," IETF, Internet Draft draft-ietf-6tisch-minimal-21, Feb. 2017. [Online]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-minimal-21

[13] S. H. Kim, N. E. Kim, N. D. Lam, and C. K. Kim, "Autonomous Link-based TSCH Cell Scheduling," IETF, Internet Draft draft-3k1n-6tisch-alice0-01, Dec. 2016. [Online]. Available: https://tools.ietf.org/html/draft-3k1n-6tisch-alice0-01

[14] N. Accettura, E. Vogli, M. R. Palattella, L. A. Grieco, G. Boggia, and M. Dohler, "Decentralized Traffic Aware Scheduling in 6TiSCH Networks: Design and Experimental Evaluation," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 455–470, Dec. 2015.