Accelerating 6TiSCH Network Formation

Yasuyuki Tanaka
Corporate R&D Center
Toshiba Corporation
Kawasaki, Japan
Email: yatch1.tanaka@toshiba.co.jp

Pascale Minet, Thomas Watteyne
EVA Team
Inria
Paris, France
Email: {pascale.minet, thomas.watteyne}@inria.fr

Fumio Teraoka
Faculty of Science and Technology
Keio University
Yokohama, Japan
Email: tera@keio.jp

***Abstract*—Wireless sensor networking is a key enabler of Industrial IoT. IETF (Internet Engineering Task Force) has standardized a protocol suite called 6TiSCH (IPv6 over the TSCH mode of IEEE802.15.4e). 6TiSCH builds an IPv6 multi-hop wireless network with the IEEE802.15.4 radio, which achieves low energy consumption and high reliability. Although network formation time is one of key performance indicators of wireless sensor networks, it has not been studied well with 6TiSCH standard protocols such as MSF (6TiSCH Minimal Scheduling Function) and CoJP (Constrained Join Protocol). In this paper, we propose a scheduling function called SF-Fastboot which shortens network formation time of 6TiSCH. We evaluate SF-Fastboot by simulation comparing with MSF, the state-of-the-art scheduling function. The simulation shows SF-Fastboot reduces network formation time by 41 % – 80 %.**

***Index Terms*—6TiSCH, TSCH, IEEE802.15.4, Network formation, Scheduling, Lifetime.**

I. Introduction

Industrial IoT applications such as plant monitoring need low-power and high reliability wireless multi-hop networks. Since battery-powered wireless sensors do not need cables, they are easy to deploy in complicated structured equipment. The "multi-hop" nature lowers deployment costs since a network covering tens of thousands of square meters can be built with a single base station.

6TiSCH (IPv6 over the TSCH mode of IEEE802.15.4e) [1] is best suited for such applications. 6TiSCH is an IETF standard protocol suite enabling an IPv6 wireless multi-hop network on top of TSCH (Time Slotted Channel Hopping), which is a channel access method defined in IEEE802.15.4 [2].

The core idea of TSCH is to combine Time Division Multiple Access and Frequency Division Multiple Access. The atomic resource for TSCH communication is called "cell", which is defined as a slot offset and a channel offset in a communication schedule. The slot offset is the relative time within the schedule of that cell. The channel offset is used to compute the communication frequency to use at the cell. The schedule tells each node what to do at each timeslot: transmit, listen, or sleep. The scheduling function is the algorithm that builds and maintains the communication schedule of the network. How the schedule is built determines several key performance indicators of the network, including network formation time, end-to-end latency, end-to-end reliability, and the battery lifetime of the nodes.

IETF standardizes a scheduling function called the 6TiSCH Minimal Scheduling Function (MSF) [3]. MSF is the first full-featured standardized 6TiSCH scheduling function that covers the following aspects, all of which are vital for a real-world deployment:

- Network dynamics: the routing topology adapts to changes in the radio environment.
- Security: the TSCH schedule is protected against unauthorized nodes including joining nodes and attackers.
- Control traffic handling: control traffic is in charge of keeping the network operational and secure.

In addition, MSF adapts to traffic changes dynamically modifying the number of scheduled cells. However, MSF is not optimized for network formation.

In this paper, we propose a scheduling function called SF-Fastboot which shortens network formation time of 6TiSCH. Since SF-Fastboot is a plug-in type scheduling function, it can work with another scheduling function like MSF.

The remainder of this paper is organized as follows. Section II introduces related work. Section III provides overview of 6TiSCH. Section IV explains how SF-Fastboot works, and interacts with MSF. Section V shows through an extensive simulation campaign how SF-Fastboot outperforms vanilla MSF. Section VI discusses possible extensions of SF-Fastboot. Finally, Section VII concludes this paper.

II. Related Work

Since on the one hand, the network formation can be long or even in large deployments some nodes may fail to join the network [4], and on the other hand, the nodes wanting to join the network stay active until they succeed. Many authors propose to reduce the network formation time and as a consequence the energy consumption of joining nodes to prolong their battery lifetime. We distinguish two classes of solutions depending on the initial schedule used. Some start from a schedule adapted to the network deployed, whereas others start from the minimal configuration defined by the 6TiSCH working group at IETF, where only one cell is a shared one in a slotframe of 101 slots.

In the first class of solutions, the authors of [5] propose a Random-based Advertisement (RA) algorithm where each device already in the network randomly selects an advertising slot among a given set to advertise its beacon with a probability equal to the inverse of its number of joined neighbors. They show that the time needed by a device to join the network

mainly depends on the number of channel offsets used for the advertisement of Enhanced Beacons.

In [6], two algorithms are presented: Random Vertical (RV) filling and Random Horizontal (RH) filling. In both algorithms, the coordinator of a network sends its beacons in the first advertisement slot of the slotframe, with channel offset 0. In RV, the other nodes transmit their beacon in the same advertisement slot but with a random channel offset. In RH, the other nodes transmit their beacon with channel offset 0 but in an advertisement slot randomly chosen in the slotframe. These two algorithms exhibit similar performances.

In [7], the authors model the behavior of a joining node as a Markov chain. They derive the joining time from this model. The optimal beacon schedule is formalized as an optimization problem where the joining time is minimized. The authors finally propose the Model-based Beacon Scheduling algorithm (MBS), where the coordinator of a network broadcasts the optimal cells (channel offset, slot offset) for beacon advertising. Each network node randomly selects one cell among the optimal cells to transmit its beacon. MBS outperforms RA [8], RV and RH [6], even if it is subject to collisions.

In [9], collisions between beacons are avoided. The authors propose an Enhanced Deterministic Beacon Advertising (EDBA), where the coordinator of a network assigns a beacon index that ensures collision-free transmissions, to each joining node during its association. Simulation results show that EDBA outperforms MBS.

In the second class of solutions starting from the minimal configuration, an original approach to cope with this minimal configuration during network formation is given in [10], where a Bayesian broadcast algorithm is adopted. The authors show how to set different transmission probabilities for each type of traffic, i.e. beacons, RPL routing messages or bootstrapping to reduce the network formation duration.

In [11], the authors show how to tune the TSCH and RPL parameters to decrease the network formation duration and the energy consumed, and make the interplay easier. They also recommend to allocate at least one additional shared slot to the one defined in the minimal configuration of the standard. This is beneficial for both TSCH network formation and routing.

In Orchestra proposed in [12], nodes autonomously build their local schedules, without negotiation and without a centralized scheduler. Several schedules coexist on each node, each schedule is assigned to a traffic plane such as MAC, routing and application. Cells are allocated in such a way that they can be automatically inserted in or removed from the schedule to adapt to the evolving RPL topology.

The authors of [4] explain the poor performance of network formation in large deployments by the limitations enforced by the minimal configuration and a static cell allocation. Hence, they propose a dynamic cell allocation performed periodically. The number of shared slots is computed from an estimation of the beacons and the routing messages in the neighborhood. This solution improves both reliability and efficiency of the network formation and provides a smaller network formation duration than Orchestra.

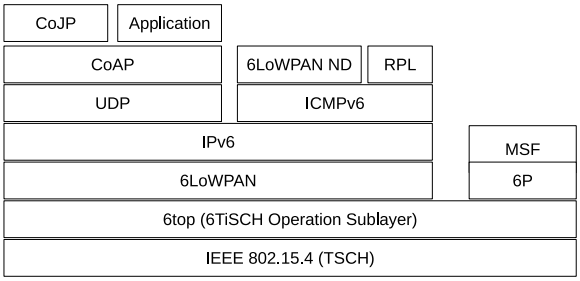


Fig. 1: 6TiSCH protocol stack architecture.

III. Overview of 6TiSCH

6TiSCH builds a multihop IPv6 network on IEEE802.15.4 TSCH MAC. Fig. 1 shows the protocol stack architecture of 6TiSCH [13]. 6top (6TiSCH Operation Sublayer) is a logical link control sublayer sitting on top of IEEE802.15.4.

6top manages a communication schedule. In a network using the minimal schedule defined by RFC8180 [14], 6top of each device configures TSCH with a slotframe of 101 timeslots and 16 channel offsets. Then, 6top installs one cell to the slotframe, at slot offset 0 and channel offset 0, which we call the "minimal shared cell". The minimal shared cell is used mostly for broadcast traffic.

One of direct upper layers of 6top is 6LoWPAN (IPv6 over Low-power Wireless Personal Area Network). 6LoWPAN enables IPv6 communication with 127-octet IEEE802.15.4 frames. RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) [15] is an IPv6 routing protocol that builds a tree-like routing structure. DIO (Destination Oriented Directed Acyclic Graph Information Object) is a control message advertising the RPL network, and broadcast on the minimal shared cell. CoJP (Constrained Join Protocol) [16] is an access authentication protocol implemented on CoAP (Constrained Application Protocol).

The other direct upper layer of 6top is 6P (6top Protocol) [17]. 6P is a control protocol for cell allocation between link neighbors. MSF (6TiSCH Minimal Scheduling Function) [3] is the default standard scheduling function that schedules cells dynamically using 6P.

MSF defines two types of cells: autonomous cells and negotiated cells. Autonomous cells do not need 6P involvements for allocation. Negotiated cells are scheduled using 6P.

A device installs an autonomous cell to its join proxy[1] or its parent, whose MAC address is learned through broadcast beacons or DIOs. In other words, beacons and DIOs are used to determine the slot offset and the channel offset of the autonomous cell. CoJP messages are exchanged over the

[1]A join proxy is a node which has already been part of the network and serves as a relay providing connectivity between a joining node and an authentication server.

autonomous cell. 6P also uses the autonomous cell to allocate negotiated cells between a parent device and a child device on the RPL routing structure, if there is no negotiated cell scheduled. After initial negotiated cells are scheduled, MSF keeps allocating or deallocating negotiated cells in accordance to amount of traffic going over the negotiated cells. For instance, under heavy application traffic, MSF of a device allocates additional negotiated cells with its parent in order to handle the traffic. Conversely, when the amount of traffic decreases, MSF deallocates negotiated cells and keeps some for the current traffic.

Fig. 2 depicts a typical 6TiSCH bootstrap sequence. Firstly, a joining node listens on a randomly chosen channel and waits for beacons. After the joining node receives beacons, it starts the CoJP join process with the sender of the beacon, which is either the root or a join proxy. Once the join process is completed, the node listens on the minimal shared cell and waits for DIOs. The node identifies its parent by DIOs. MSF of the node schedules negotiated cells with the parent through 6P communication. At this moment, the node becomes fully operational and advertises the network with beacons and DIOs.

The synchronization phase in the bootstrap sequence could be long because of channel hopping. Assuming the root broadcasts a beacon on the minimal shared cell every 1 min, it takes 16 min to make its beacon transmission happen on all the 16 channels of 2.4 GHz. This means that a joining node in this case may need to wait for 16 min before getting synchronized. Although a naive way to improve the synchronization phase is to make the beacon interval shorter, there is a trade-off between joining time and energy consumption as pointed out by RFC8180.

The authentication phase is another part to consider regarding network formation time. There could be multiple joining nodes which receive the same beacon. If they start the CoJP join process at the same time, join request packets are transmitted on the same autonomous cell. This causes retransmissions of the request packets, which prolong their join processes. Introducing a random period of wait to start the join process is an idea to avoid such a situation, however, which affects the network formation time as well.

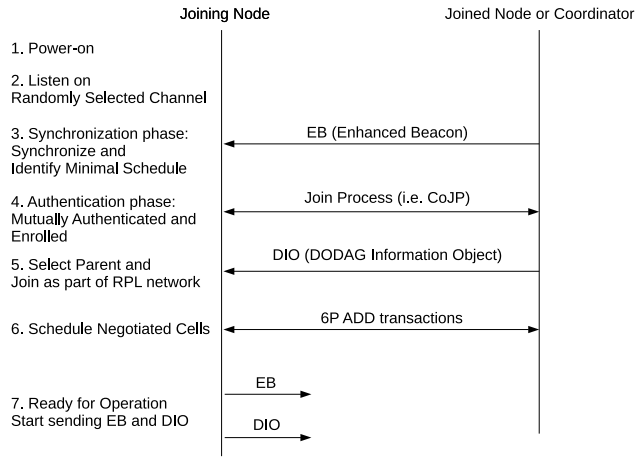


Fig. 2: A typical 6TiSCH bootstrap sequence.

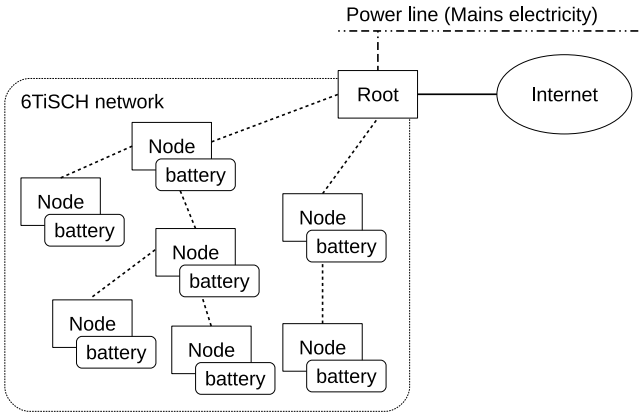


Fig. 3: A typical system architecture.

IV. SF-Fastboot

A. Design Principles

Our proposing scheduling function, SF-Fastboot, aims to improve the synchronization phase and the authentication phase of the bootstrap sequence. Fig. 3 depicts a typical system architecture of 6TiSCH where the root connects a 6TiSCH network with Internet. The root is mains-powered. Nodes of the 6TiSCH network are battery-powered. The intuitive idea of SF-Fastboot is exploiting the asymmetry between the root and the other nodes, as it is commonly done between the "base station" and the "terminals" in wireless network systems. This approach is found more recently in SmartMesh IP [18] that configures different beacon intervals depending on the node type. In SF-Fastboot, the root is allowed to send beacons more frequently than any other node having already joined the 6TiSCH network, since this higher beacon transmission frequency for the root is harmful to neither energy consumption, nor network lifetime.

Fig. 4 depicts the schedule of the root. The red cell is the minimal shared cell defined by RFC8180. Grey cells are unused cells. White cells are used by MSF. Yellow cells and green cells are allocated by SF-Fastboot.

A specific channel offset, $ch\_off_{fastboot}$, is reserved for network formation. It is equal to 15 in Fig. 4. At $ch\_off_{fastboot}$, the root schedules beacon TX cells continuously as many as the number of channels, which are the yellow cells in Fig. 4. In a certain slotframe, the root generates beacons, and transmits them repeatedly at all the available channels. This gives more chances to receive beacons, to a joining node which listens on a randomly chosen channel. In other words, transmitting beacons continuously at the same channel offset, $ch\_off_{fastboot}$, contributes shorter time for a joining node to get synchronized with the network.

After a joining node gets synchronized by receiving a beacon from the root, it starts the CoJP joining process by sending a join request to the root. In order to avoid congestion by join

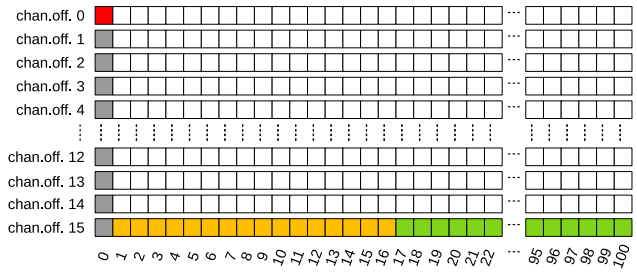


Fig. 4: SF-Fastboot schedule layout at the root.

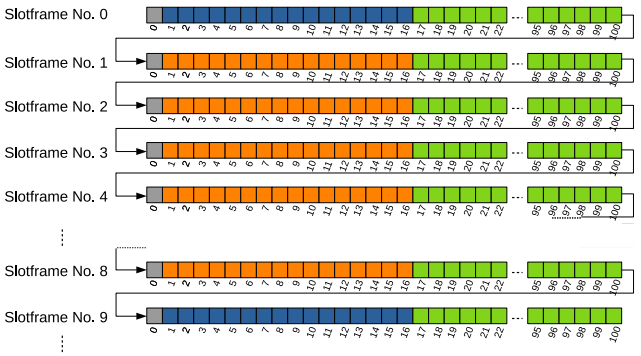


Fig. 5: A view of the schedule at channel offset 15 of the root, in a timeline.

requests from multiple joining nodes, the root schedules RX cells, the yellow cells and the green cells in Fig. 4, at all the slot offsets other than the slot offset of the minimal shared cell so that join requests can be distributed over the RX cells. As mentioned above, the yellow cells are used for beacon transmissions as well. When a beacon is in the TX queue, a yellow cell is used for a beacon transmission. Otherwise, a yellow cell is used for RX. The green cells are pure RX cells.

Fig. 5 shows how cells at $ch\_off_{fastboot}$ are used along the time. Cells where beacon transmissions happen are marked in blue. SF-Fastboot controls timings of beacon generation so that a set of beacon transmissions happens with a fixed interval. In the example of Fig. 5, beacon transmissions happen at slot offset 1–16 in every ninth slotframe, Slotframe No. 0 and Slotframe No. 9. In other slotframe, cells at slot offset 1–16 are treated as RX cells, which are marked in orange in Fig. 5.

Once a node joins the network, the node schedules a beacon TX cell at $ch\_off_{fastboot}$. At any node, a beacon RX cell is allocated to receive from its parent. This contributes to reduce the keep-alive traffic. Furthermore, thanks to the beacon TX cell allocation, the minimal shared cell can be used for more important traffic than beacons.

B. Application to MSF

MSF can be integrated with SF-Fastboot with minor changes. As shown Table I, we introduce a slotframe for SF-Fastboot with slotframe handle 3, which is the lowest priority.

Slotframe Handle	Usage
0	Minimal shared cell (RFC8180)
1	Autnomous cells (MSF)
2	Negotiated cells (MSF)
3	SF-Fastboot

TABLE I: The slotframe configuration in MSF with SF-Fastboot.

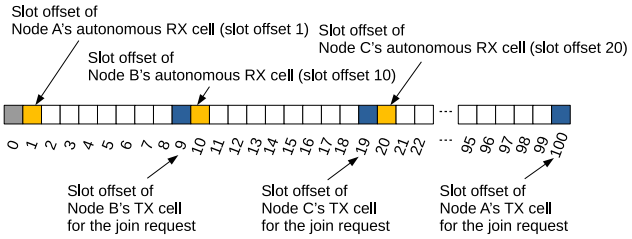


Fig. 6: An example of TX cell allocations for CoJP join process.

If multiple cells are scheduled at a same slot offset, TX cells in Slotframe 1 is processed with the highest priority, followed by RX cells in Slotframe 1, TX cells in Slotframe 2, RX cells in Slotframe 2, TX cells in Slotframe 3, and RX cells in Slotframe 3. No cells are scheduled at slot offset 0 except the minimal shared cell, that is scheduled in Slotframe 0.

The root at the startup schedules beacon TX cells and RX cells to SF-Fastboot slotframe in the same manner as explained with Fig. 4 and Fig. 5. The root decreases the number of beacon TX cells up to the lower bound of three in accordance with the number of 1-hop nodes, which the root is aware of through the negotiated cell scheduling of MSF. In the end, the root keeps three beacon TX cells at slot offset 1, 2, and 3, which are considered as well-known beacon TX cells of the root.

When a joining node receives a beacon from the root, it schedules a TX cell at unused slot offset right before its autonomous RX cell as shown in Fig. 6. The channel offset of the TX cell is $ch\_off_{fastboot}$. The TX cell is used to transmit a CoJP join request, and will be removed when the join process is completed. This behavior naturally distributes join request traffic along the slotframe. Since the root sends back a join response at the autonomous RX cell of a joining node, the latency of the secure join process is minimized, that is, 10ms in RFC8180 configuration.

Once a node completes the join process, it schedules a beacon TX cell at the slot offset right before the autonomous RX cell of the node. In addition, the node schedules a beacon RX cell where its parent will transmit beacons. If the parent is the root, the node schedules a beacon RX cell at unused slot offset from slot offset 1 to slot offset 3, and the channel offset of $ch\_off_{fastboot}$. If the parent is a non-root node, the node schedules a beacon RX cell, the position of which is computed based on the MAC address of the parent in the same manner for the autonomous RX cell of the parent.

Fig. 7 shows an example of a global view of beacon TX

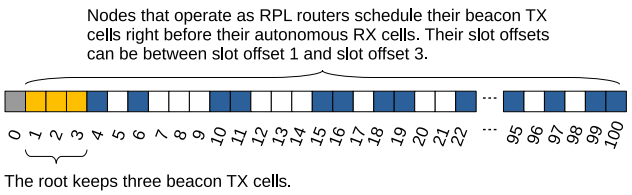


Fig. 7: An example of a global view of beacon TX cell allocations.

cell allocations. Yellow cells are the beacon TX cells of the root. Blue cells are beacon TX cells of other nodes which have joined the network successfully. Since there are more than 12 joined nodes, the root has only three beacon TX cells, which are the well-known beacon TX cells.

MSF specifies that the minimal shared cell is used for broadcast frames such as beacons. However, with SF-Fastboot, beacons are never transmitted on the minimal shared cell. Since each node has its own beacon TX cell at a different position in the schedule, there is no chance for beacons to collide if the slotframe is large enough. Therefore, the beacon interval can be configured with a fixed value.

C. Exploit DIS

DIS (DODAG Information Solicitation) is a control message defined by RFC6550 [15]. There are two modes of DIS: unicast DIS and broadcast DIS. Unicast DIS makes a recipient send back a DIO. Broadcast DIS makes recipients reset the DIO trickle timer.

SF-Fastboot uses the unicast DIS to reduce network formation time further. A node sends a DIS to its join proxy just after the join process is completed. This helps the node join the RPL network quickly.

V. Performance Evaluation

We evaluate SF-Fastboot by simulation. We implement SF-Fastboot in version 1.3.0 of the 6TiSCH Simulator [19].

A. Configuration

We used two algorithms of node placement: Fully-Meshed and Random. The fully-Meshed algorithm places nodes within a radio communication range. Every node has 100 % connectivity to other nodes. The random algorithm determines the x, y coordinates of the nodes randomly in a 2 km $\times$ 2 km space, for every simulation run. We use the Pister-hack connectivity model [20]. Each node has at least three links to neighbors whose link PDR (Packet Delivery Ratio) values are above 50%. In the evaluation, we observed 5 hops in a 25-node network, 6 hops in a 50-node network, and 7 hops in a 100-node network at the deepest.

Table II shows the simulation parameters. Note that beacons are transmitted at every ninth slotframe when SF-Fastboot is enabled.

TABLE II: Simulation parameters.

Parameter	Settings
Number of nodes	25, 50, 100
SF-Fastboot	disabled, enabled
DIS	disabled, unicast, multicast
Scheduling Function	MSF
Application packet interval	60 s
Application packet interval randomization	$\pm$5%
RPL DAO interval	60 s
TSCH slotframe length	101 slots
TSCH slot duration	10 ms
TSCH TX queue length	10 frames
TSCH keep-alive interval	10 s
TSCH EB transmission probability	0.33
TSCH max. number of retransmissions	5
Number of radio channels	16
Channel offset reserved for SF-Fastboot	15

B. TSCH Synchronization Time

We call TSCH synchronization time the duration between the time simulation starts, and the time the last node gets synchronized with the 6TiSCH network. Fig. 8 shows the TSCH synchronization time. In the fully-meshed placement, the average TSCH synchronization time by SF-Fastboot is less than 30 s regardless of the network size. In the random placement, SF-Fastboot reduces the average TSCH synchronization time by 40 % – 80 %.

C. Network Formation Time

We call network formation time the duration between the time simulation starts, and the time the last node joins the RPL network. Fig. 9 shows the network formation time.

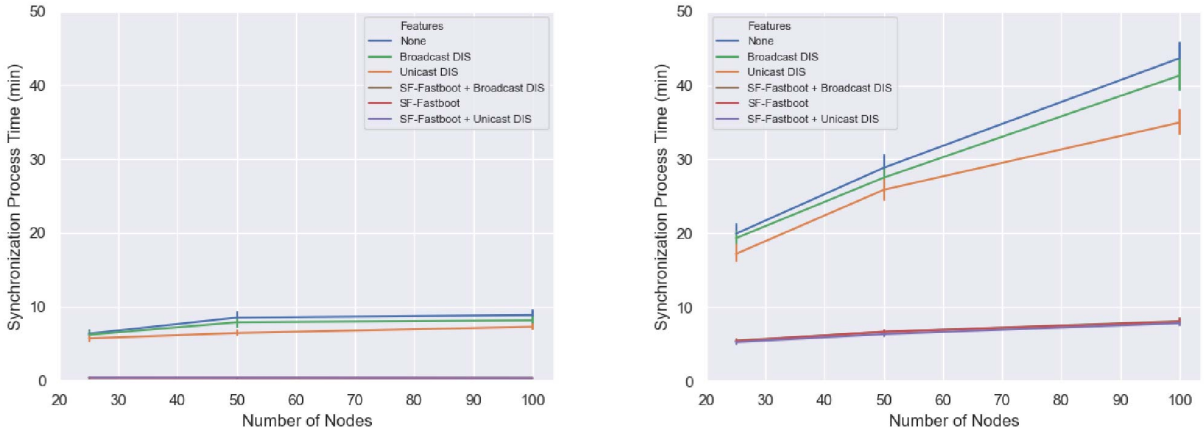
Overall, SF-Fastboot yields shorter network formation time than vanilla MSF, denoted by "None" in Fig. 9. SF-Fastboot reduces network formation time comparing vanilla MSF by 41% – 80% in the fully-meshed placement, and by 71% – 77% in the random placement. The result also shows DIS shortens network formation time. With 100 nodes and the fully-meshed placement, using unicast DIS alone performs closely to SF-Fastboot. When SF-Fastboot is enabled, differences by the DIS mode are marginal.

D. Energy Consumption

We compute energy consumption of each node but the root using the energy consumption model proposed by Vilajosana *et al.* [21]. Fig. 10 shows the energy consumption during the first one hour in which the network formation process happens. SF-Fastboot reduces energy consumption comparing vanilla MSF by 30% – 32% in the fully-meshed placement, and by 29% – 38% in the random placement.

VI. Discussion

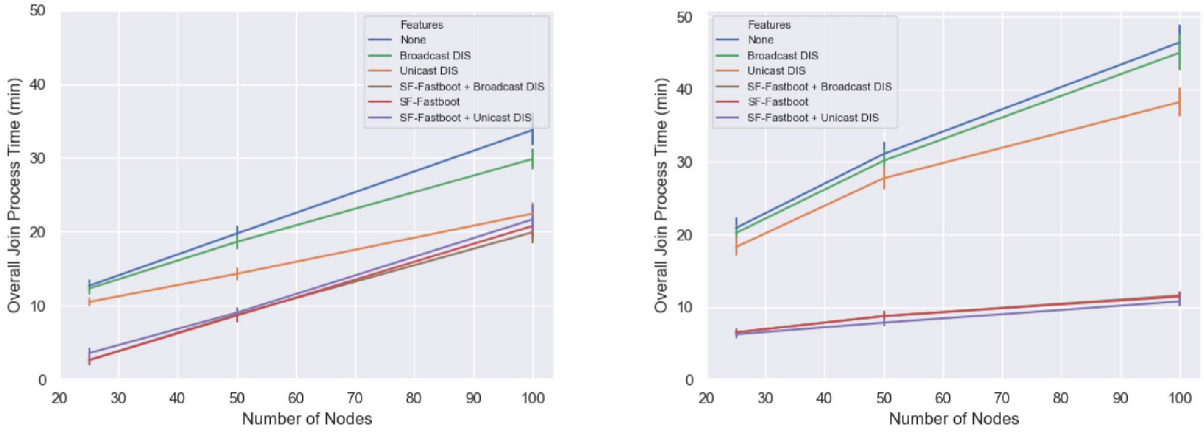
The results in Section V show SF-Fastboot does improve network formation time. Comparing Fig. 9a to Fig. 9b, SF-Fastboot is more effective in a sparse topology. For a very dense topology, a network with 100 nodes of fully-meshed placement, the schedule by SF-Fastboot seems to have little impact on overall network formation time since MSF with



(a) Fully-Meshed placement.

(b) Random placement.

Fig. 8: TSCH synchronization time. Results are averaged over 100 simulation runs and plotted with a 95 % confidence interval.



(a) Fully-Meshed placement.

(b) Random placement.

Fig. 9: Network formation time. Results are averaged over 100 simulation runs and plotted with a 95 % confidence interval.

unicast DIS yields almost the same performance as SF-Fastboot. This implies there is room for improvement on SF-Fastboot for dense topologies.

The evaluation also shows applying DIS alone can accelerate network formation to some extent. Unicast DIS performs better than broadcast DIS since broadcast DIS causes DIO floods on the minimal shared cell, which could bother network advertisement by beacons. Unicast DIS is easy to implement with a small code footprint. We recommend unicast DIS to 6TiSCH implementations with any scheduling function.

SF-Fastboot reduces energy consumption as well, however, with smaller improvement comparing with reduction of network formation time. In addition, one interesting finding in Fig. 10b is that with SF-Fastboot, the maximum energy consumption changes little regardless of the network size or the number of nodes.

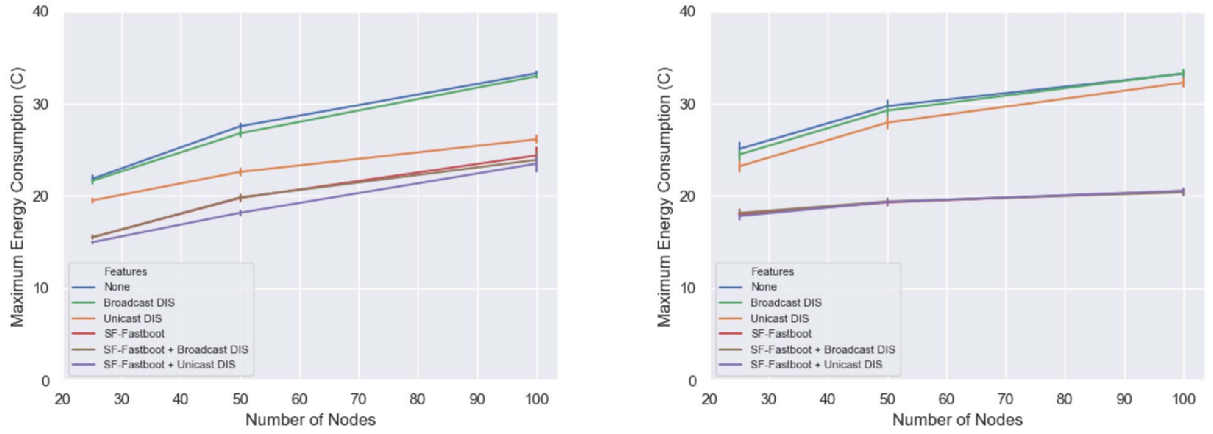
VII. Conclusion

In this paper, we propose a scheduling function, SF-Fastboot, which shortens network formation time and reduces energy consumption. SF-Fastboot is a plug-in type scheduling function that can be used with MSF. Simulation results show that SF-Fastboot improves network formation time by 80% at the maximum, comparing vanilla MSF. We will implement SF-Fastboot on an actual 6TiSCH stack in future work.

References

[1] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A Tutorial," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 595–615, 2019.

[2] *802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks*, IEEE Std., April 2016.



(a) Fully-Meshed placement.

(b) Random placement.

Fig. 10: Maximum energy consumption during the first one hour. Results are averaged over 100 simulation runs and plotted with a 95 % confidence interval.

[3] T. Chang, M. Vučinić, X. Vilajosana, S. Duquennoy, and D. Dujovne, *6TiSCH Minimal Scheduling Function (MSF)*, IETF Std. draft-ietf-6tisch-msf-18 [work-in-progress], 2020.

[4] C. Vallati, S. Brienza, G. Anastasi, and S. K. Das, "Improving network formation in 6TiSCH networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 98–110, 2018.

[5] C. M. G. Algora, V. A. Reguera, E. M. G. Fernández, and K. Steenhaut, "Parallel Rendezvous-Based Association for IEEE 802.15. 4 TSCH Networks," *IEEE Sensors Journal*, vol. 18, no. 21, pp. 9005–9020, 2018.

[6] E. Vogli, G. Ribezzo, L. A. Grieco, and G. Boggia, "Fast network joining algorithms in industrial IEEE 802.15. 4 deployments," *Elsevier Ad Hoc Networks*, vol. 69, pp. 65–75, 2018.

[7] D. De Guglielmo and S. Brienza and G. Anastasi, "A Model-based Beacon Scheduling algorithm for IEEE 802.15.4e TSCH networks," in *2016 IEEE 17th international symposium on a World of Wieless, Mobile and Multimedia Networks (WoWMoM)*, Coimbra, Portugal, 2016.

[8] D. De Guglielmo, A. Seghetti, G. Anastasi, and M. Conti, "A performance analysis of the network formation process in IEEE 802.15. 4e TSCH wireless sensor/actuator networks," in *IEEE Symposium on Computers and Communications (ISCC)*, 2014, pp. 1–6.

[9] I. Khoufi and P. Minet, "An enhanced deterministic beacon advertising algorithm for building {TSCH} networks," *Annals of Telecommunications*, vol. 73, pp. 745 – 757, 2018. [Online]. Available: https://link.springer.com/article/10.1007/s12243-018-0638-3

[10] M. Vučinić, T. Watteyne, and X. Vilajosana, "Broadcasting strategies in 6TiSCH networks," *Internet Technology Letters*, vol. 1, no. 1, p. e15, 2018.

[11] D. Fanucchi, B. Staehle, and R. Knorr, "Network Formation for Industrial IoT: Evaluation, Limits and Recommendations," in *IEEE 23rd International Conference n Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 227–234.

[12] S. Duquennoy, B. Al 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*, Seoul, South Korea, November 2015, pp. 337–350.

[13] P. Thubert, *An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4*, IETF Std. draft-ietf-6tisch-architecture-20 [work-in-progress], 2019.

[14] X. Vilajosana, K. Pister, and T. Watteyne, *Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration*, IETF Std. RFC8180, May 2017.

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

[16] M. Vučinić, J. Simon, K. Pister, and M. Richardson, *Constrained Join Protocol (CoJP) for 6TiSCH*, IETF Std. draft-ietf-6tisch-minimal-security-15 [work-in-progress], 2019.

[17] Q. Wang, X. Vilajosana, and T. Watteyne, *6TiSCH Operation Sublayer (6top) Protocol (6P)*, IETF Std. RFC8480, 2019.

[18] T. Watteyne, L. Doherty, J. Simon, and K. Pister, "Technical overview of SmartMesh IP," in *Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2013, pp. 547–551.

[19] Y. Tanaka, K. Brun-Laguna, and T. Watteyne, "Trace-based simulation for 6TiSCH," *Internet Technology Letters*, 2020.

[20] E. Municio, G. Daneels, M. Vučinić, S. Latre, J. Famaey, Y. Tanaka, K. Brun-Laguna, K. Muraoka, X. Vilajosana, and T. Watteyne, "Simulating 6TiSCH networks," *Transactions on Emerging Telecommunications Technologies, Wiley*, vol. 30, no. 3, Mar. 2019.

[21] X. Vilajosana, Q. Wang, F. Chraim, T. Watteyne, T. Chang, and K. S. Pister, "A realistic energy consumption model for TSCH networks," *IEEE Sensors Journal*, vol. 4, no. 2, pp. 482–489, 2013.