6TiSCH Minimal Scheduling Function: Performance Evaluation

Tengfei Chang, Mališa Vučinić, Xavi Vilajosana Guillén, Diego Dujovne, Thomas Watteyne

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

6TiSCH Minimal Scheduling Function: Performance Evaluation

Tengfei Chang*[1] | Mališa Vučinić[1] | Xavi Vilajosana Guillén[2] | Diego Dujovne[3] | Thomas Watteyne*[1]

[1]EVA, Inria, Parise, France
[2]Internet Interdisciplinary Institute (IN3), Universitat Oberta de Catalunya, Barcelona, Spain
[3]Org Division, Universidad Diego Portales, Santiago, Chile

Correspondence
*Tengfei Chang, Thomas Watteyne Email: {tengfei.chang, thomas.watteyne}@inria.com

6TiSCH is a standardization group within the Internet Engineering Task Force (IETF) that works on IPv6-enabled Time-slotted Channel Hopping (TSCH) networks. The 6TiSCH protocol stack, designed by the standardization work at the IETF, has direct applicability to low-power Internet of Things (IoT) use cases, including smart factory, building, infrastructure and home applications. A key component of the 6TiSCH stack is the Minimal Scheduling Function (MSF). MSF implements a traffic adaptation algorithm which allocates link-layer resources, i.e. cells in the TSCH schedule, according to the traffic load. `MAX_NUMCELL` is an important parameter defined in the MSF draft standard which determines the length of the running window used to measure cell usage. MSF draft standard does not recommend a value of `MAX_NUMCELL` to use. This paper provides recommendations on how to choose the value of `MAX_NUMCELL`, validated through simulation. For periodic traffic, setting `MAX_NUMCELL` to at least 4 times the traffic load is recommended to increase efficiency. For bursty traffic, we show that setting `MAX_NUMCELL` to a small value achieves a low end-to-end latency but at high communication overhead. In addition, an improved version of MSF is implemented and tested, which shows a 44% reduction in the communication overhead, considering `MAX_NUMCELL` = 4, while maintaining the same end-to-end latency.

KEYWORDS:
6TiSCH, Scheduling, Latency, Overhead, SF

1 | INTRODUCTION

The Internet Engineering Task Force (IETF) and the 6TiSCH working group have produced a set of specifications that constitute the 6TiSCH protocol stack[1]. The stack is designed for critical industrial automation, and is hence also applicable to smart building, infrastructure and home automation applications. The stack foundation is the Time-Slotted Channel Hopping (TSCH) mode of operation at the MAC layer, which enables ultra low-power operation. As a time-frequency division multiple access technique, the quantum of TSCH bandwidth is called a `cell`. A cell is uniquely defined by a time and channel offset, describing the time instant and the frequency at which the transmission to a neighbor node takes place. The set of all cells that a 6TiSCH network node, the `mote`, has installed is called a `schedule`. The allocated cells provide a deterministic communication pattern for the motes to schedule bandwidth according to the traffic being transmitted.

The 6top Protocol[2] (6P) enables the motes to manage the schedule in a distributed manner through a handshake mechanism. The Scheduling Function (SF) defines the rules that trigger 6P transactions. As a successful outcome of these 6P exchanges, the schedule of the neighboring motes is populated with cells. A number Scheduling Functions have already been proposed[3,4,5,6,7].

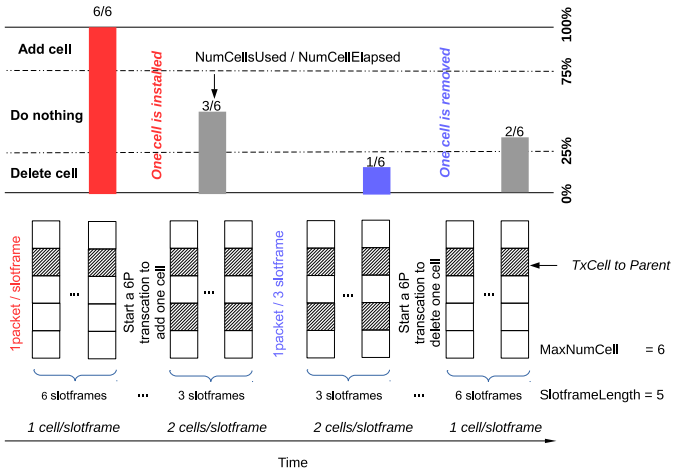


FIGURE 1 MSF adapts the number of cells a mote allocates to a neighbor according to the number of frames they exchange. Here, `MAX_NUMCELL` = 6. A cell is added when the cell usage is over 75%, and deleted when below 25%.

They were designed to address different application types and particular network requirements. The Minimal Scheduling Function (MSF)[3] is the only scheduling function adopted for standardization at the IETF. MSF is designed for best-effort traffic, where most traffic is periodic monitoring, with occasional bursts. MSF uses two types of cells: `autonomous` and `negotiated`. Autonomous cells provide the minimal bandwidth required for bidirectional communication between two neighbors. Negotiated cells are installed in response to dynamic bandwidth requirements. Bandwidth requirements are estimated based on the traffic load, by measuring the usage of the installed cells over some sliding window. If the load is above a pre-defined threshold, a new cell is added. If the load is below some other threshold, a cell is removed from the schedule. If the load is between those two thresholds, no action is taken. This is illustrated in Fig. 1.

The MSF draft standard[3] defines a number of parameters that influence network performance. It does not, however, provide any recommendation on the duration of the sliding window, a parameter that is critical for responsiveness to variation in traffic. The MSF draft standard refers to the sliding window length as `MAX_NUMCELL`, the term we use throughout this paper.

The contribution of this paper is twofold. First, we provide recommendations on how to choose the value of `MAX_NUMCELL`, based on simulation results. Second, we propose an advanced version of MSF which reduces the number of 6P exchanges to adapt to bursty traffic.

The remainder of this paper is organized as follows. Section 2 gives on overview of the MSF's traffic adaptation algorithm. Section 3 provides simulation results and insights on the performance of MSF using different `MAX_NUMCELL` values. Finally, Section 4 summarizes and concludes this paper.

2 | TRAFFIC ADAPTATION OF MSF

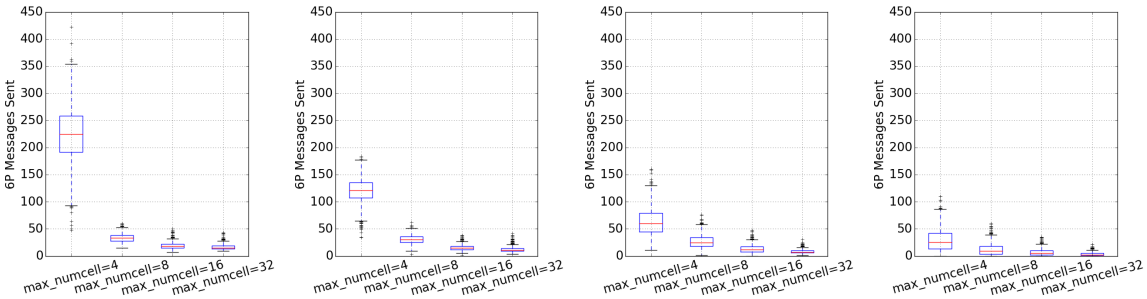
The MSF draft standard defines the rules for motes in the network to follow, both when first booting and when adapting to traffic. In this section, we only focus on the MSF's adaptation algorithm.

MSF adds or deletes a cell according to the cell usage. Cell usage is calculated as `NumCellUsed` divided by `NumCellElapsed`. MSF maintains those two counters. When the current cell is a negotiated transmit cell, `NumCellElapsed` increments. When a frame is sent on that transmit cell, regardless of whether it is acknowledged, `NumCellUsed` increments. When `NumCellElapsed` hits the pre-defined value `MAX_NUMCELL`, MSF calculates the cell usage. It decides to issue a 6P request to add a cell if it is over 75%, or delete a cell if it is below 25%. After computing cell usage (and possibly adding/deleting cells), `NumCellElapsed` and `NumCellUsed` reset to zero and the process repeats.

Fig. 1 shows an example of how MSF adds and deletes negotiated cells to adapt to traffic changes. Assuming the slotframe is 5 slots long, and `MAX_NUMCELL` =6. The mote already has one transmit cell to its parent. At the beginning, the traffic load of the mote is 1 packet/slotframe. After 6 slotframes, `NumCellElapsed` reaches 6 and cell usage is 100% (6/6). MSF starts a 6P transaction with its parent and adds one transmit cell. 3 slotframes later, after the second transmit cell is scheduled, `NumCellElapsed`

name	settings
event to log	`app.tx`, `app.rx`, `tsch.add_cell`, `tsch.delete_cell`, `sixp.tx`
traffic pattern	`periodic`, `bursty`
KPIs	num. of cells scheduled, num. of 6P messages sent, end-to-end latency
`MAX_NUMCELL`	4, 8, 16, 32
duration per run	1 hour (3600 slotframes, each slotframe 101 slots, each slot 10 ms)
number of runs	1,000 runs for each value of `MAX_NUMCELL`

TABLE 1 Simulation Settings.



(a) Mote 1 (b) Mote 2 (c) Mote 3 (d) Mote 4

FIGURE 2 *Periodic Traffic*. Number of 6P messages sent through each simulation. One plot per mote.

reaches 6 and cell usage is 50% (3/6). No action is required. At some point later, the traffic load reduces to 1 packet every 3 slotframes. 3 slotframes after, `NumCellElapsed` reaches 6 again, and the cell usage drops to 17% (1/6). MSF starts a 6P transaction with to delete one transmit cell to its parent. 6 slotframes later, after deleting one of the transmit cells, `NumCellElapsed` reaches 6 and cell usage increments to 33% (2/6), which triggers no action from MSF.

The goal of this paper is to show how the value of `MAX_NUMCELL` influences the performance of MSF, and to recommend a range of values meeting the expected performance for specific scenarios. Performance results are obtained using the 6TiSCH simulator[8], and are presented in Section 3.

3 | EVALUATION

3.1 | Simulation Results

The 6TiSCH simulator implements the complete 6TiSCH protocol stack; it is written in Python and includes an implementation of the latest version of the MSF draft. The log files generated by the simulator record each event from each mote. Events include: an app packet generated by a mote, an app packet received by the root, a 6P request frame sent by a mote, a cell added in the schedule and a cell deleted from the schedule.

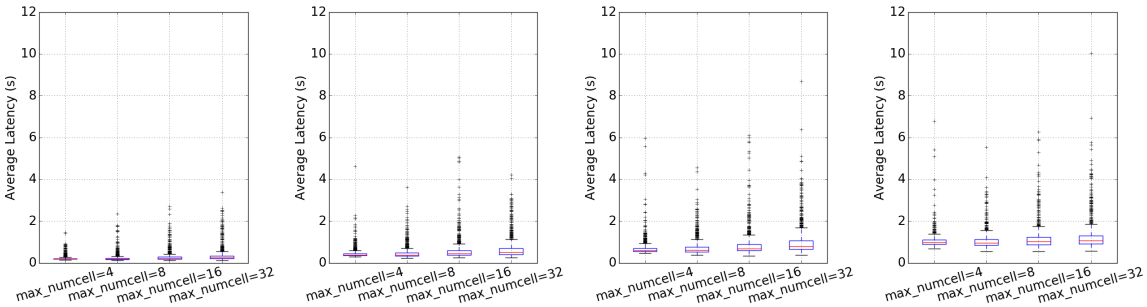
The simulator supports multi-processing, so we run it on many cores in a cluster system, reducing total simulation time.

We target two traffic patterns: `periodic`, where each mote generates one packet per slotframe and `bursty`, where one mote generates 20 packets per minute at the start of each minute.

We consider the two Key Performance Indicators (KPIs) for MSF:

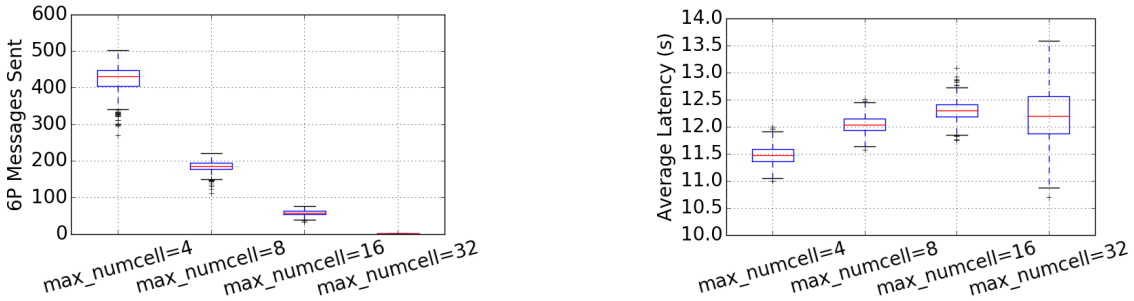
- the number of 6P messages sent by each mote, which we aim to reduce,
- the end-to-end latency of each packet, which we aim to shorten.

Without lack of generality, we consider the canonical case of 4-hop linear topology (5 motes). This allows us to observe only the impact of the scheduling approach, and presents a worst case scenario compared to for example load-balancing traffic across



(a) Mote 1 (b) Mote 2 (c) Mote 3 (d) Mote 4

FIGURE 3 *Periodic Traffic*. End-to-end latency. One plot per mote.



(a) number of 6P sent (b) average latency

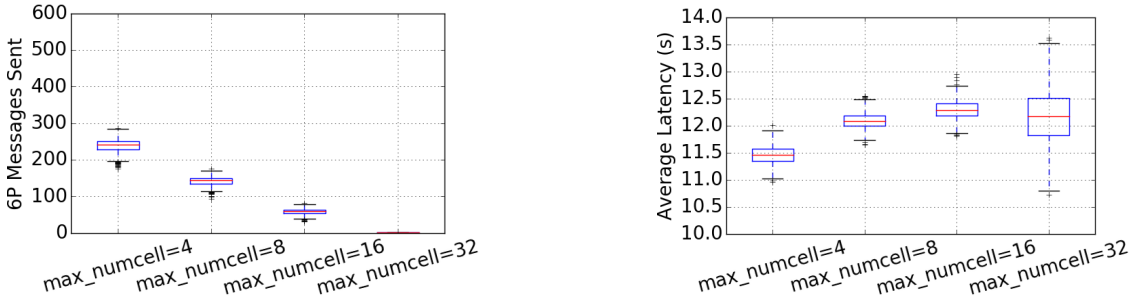
FIGURE 4 *Bursty traffic, using (regular) MSF*. Number of 6P messages sent *(left)* and end-to-end latency *(right)*. Plots only for Mote 4.

multiple routing parents. Mote 0 is the root node, mote 1 is the first hop node, etc. In the `periodic` traffic case, each of the 4 motes generates data. This results in the first-hop mote having to send 4 frames per slotframe to the root. In the `bursty` traffic case, only the fourth hop mote generates traffic.

The generality is because of the link local nature of MSF protocol. For most of the application of 6TiSCH network, they are upstream traffic. Each mote sends packet periodically to the root. From the view of routing, it is a linear topology for each of the motes. Though two motes may select a common mote as their parent, which turns to a star topology, the common mote does not differentiate the source of the traffic, either from previous hop or generated by itself. In this case, from the point of MSF, it equals to the second hop mote in a linear topology. The `MAX_NUMCELL` is an input parameter of link local decision for the mote who is trying to send packets. Hence the performance of different values of `MAX_NUMCELL` does not vary depending on the topology.

We repeat the simulation for `MAX_NUMCELL` set to 4, 8, 16 and 32. For each value of `MAX_NUMCELL`, we run a simulation for a period equivalent to 3600 slotframes, with slot duration set to 10 ms and slotframe length 101 slots. To ensure our results are statistically valid, we conduct 1000 runs for each of the example values of `MAX_NUMCELL`, and present all results with 99.3% confidence intervals. Table 1 summarizes the simulation settings.

The typical KPI of a scheduling function, such as packet delivery rate (PDR) or energy consumption, are not chosen directly to evaluate the result. In fact, the `MAX_NUMCELL` is not affected by the packet loss because of the nature of MSF scheduling mechanism. The MSF adds/removes cells according to number of packet sent. In case of a $PDR < 100\%$, the number of packet sent will be translated as number of packet expected to send divides PDR. For example, the packet sending rate of 1 packet per slotframe at $PDR = 50\%$, the performance then can be referred to packet sending rate of 2 packets per slotframe at $PDR = 100\%$. The simulation in the paper fixed the PDR to 100% but added different packet sending rates at each hop of the network.



(a) number of 6P sent

(b) average latency

FIGURE 5 *Bursty traffic, using Advanced MSF*. Number of 6P messages sent by Mote 4 *(left)* and end-to-end latency *(right)*. Plots only for Mote 4. Comparing to regular MSF (Fig. 4), end-to-end latency is equivalent, but the number of 6P messages is halved.

The energy consumption of MSF mostly comes from two parts: the number of cells scheduled and the overhead. As the scheduling mechanism of MSF indicated it schedules more cells than required to compensate latency performance, at the cost of additional energy. The `MAX_NUMCELL` has a limited effect on the number of cells scheduled. The energy consumption related to `MAX_NUMCELL` is mostly introduced by the transaction overhead, which is the number of 6P messages sent. This is discussed following.

3.1.1 | Periodic Traffic Pattern

Fig. 2 shows the number of 6P messages sent by each mote. Using `MAX_NUMCELL` $= 4$ results in a higher number of 6P messages, compared to larger values of `MAX_NUMCELL`. For mote 1, the traffic load is 4 packets per slotframe, thus the mote needs to schedule 8 cells to meet the ideal 50% cell usage. By using `MAX_NUMCELL` $= 4$, MSF deletes a cell when 4 cells are unused, representing a 0% cell usage. However, the following 4 cells could be used for transmitting purposes, thus generating a 100% cell usage. As a consequence, MSF may trigger a command to add a cell right after. The probability of adding a cell right after increases when the value of `MAX_NUMCELL` is reduced.

Fig. 3 shows the result of end-to-end latency for motes at each hop. For each mote, changing `MAX_NUMCELL` has a very limited effect on improving end-to-end latency. This is because MSF reserves additional cells for the case when traffic load increases. Hence, the delay caused by the 6P transaction is very small for the periodic traffic application. Notice that the average latency using `MAX_NUMCELL` $= 32$ is a little bit lower than using `MAX_NUMCELL` $= 16$, but with a larger variation. This is because the packets are generated at rate of 20 packets per minutes which is less than 60 slotframes. The generating interval has a variation of 20%, ranging of 48 to 72 slotframes interval. When using larger `MAX_NUMCELL` such 32, the 20 packets generated at first time may be not all sent out yet when the next 20 packets are generated. Hence the previous added cells in schedule are kept, which reduces part of the latency caused by 6P transactions overhead.

Assuming the traffic load is L [packets per slotframe], the recommended value of `MAX_NUMCELL` setting should be at least $2 \times (L \div 50\%)$, which corresponds to $4 \times L$. Here, 50% is the average of cell usage and the $2\times$ is double the sample rate to increase the accuracy. For example, for mote 1, its link to the root has a traffic load of 4 packets per slotframe. To reach a stable cell usage of 50%, it needs to schedule 8 cells in the schedule. So the final `MAX_NUMCELL` value for mote 1 should be set to > 16 to have a higher sampling precision of cell usage among the 8 cells.

3.1.2 | Bursty Traffic Pattern

For the bursty traffic pattern, the delay on 6P transactions shows up as indicated in Fig. 4. Fig. 4(b) shows that the end-to-end latency increases when increasing the `MAX_NUMCELL` value. Fig. 4(a) shows the number of 6P messages sent by mote 4. When `MAX_NUMCELL` is set to 4, the end-to-end latency is reduced to 0.5 s compared to using `MAX_NUMCELL` $= 8$, which doubles the 6P frame overhead.

3.2 | An Improved MSF

One of the main issues of MSF is the generation of a large amount of 6P messages because MSF only adds or deletes one cell at a time. The mote may need several 6P transactions to reach the stable status (25% to 75% of cell usage).

One possible improvement to MSF is to let the mote add or delete multiple cells at a time to reach the ideal 50% cell usage. Base on this core idea, we design an advanced version of MSF: A-MSF, which adapts to traffic as follows:

- when cell usage > 75%, add X transmit cells, with $X = numTxCellScheduled \cdot (cellUsage/0.5 - 1)$
- when cell usage < 25%, delete X transmit cells, with $X = numTxCellScheduled \cdot (1 - cellUsage/0.5)$

To compare the performance to MSF, we implement A-MSF and conduct the same experiment for the bursty traffic pattern case. Fig. 5 shows the result of A-MSF using the same setting of Fig. 4. A-MSF yields the same end-to-end latency as MSF, while reducing the number of 6P messages by 44% (resp. 26%) when using `MAX_NUMCELL` = 4 (resp. `MAX_NUMCELL` = 8). For other values of `MAX_NUMCELL`, the number 6P messages sent by A-MSF and MSF is equivalent.

4 | CONCLUSION

This paper provides recommendations on how to set the value of `MAX_NUMCELL` when using MSF. It considers two common traffic patterns. For periodic traffic, it is recommended to use a larger `MAX_NUMCELL` so as to reduce 6P traffic. The value should be set at least 4 times the link traffic load measured in packets per slotframe. For bursty traffic, a small `MAX_NUMCELL` value reduces the end-to-end latency, but exhibits a large numerb of 6P messages. We proposed A-MSF, extension of MSF which yields 44% less 6P messages than MSF, with comparable end-to-end latency. Based on these results, we are working with the 6TiSCH standardization team to introduce the changes of A-MSF into the next version of the 6TiSCH MSF standard.

References

1. Vilajosana X, Watteyne T, Vučinić M, Chang T, Pister K. 6TiSCH: Industrial Performance for IPv6 Internet-of-Things Networks. *Proceedings of the IEEE* 2019: 1–13. doi: 10.1109/JPROC.2019.2906404

2. Wang Q, Vilajosana X, Watteyne T. 6TiSCH Operation Sublayer (6top) Protocol (6P). tech. rep., Internet Engineering Task Force (IETF); 2018.

3. Chang T, Vučinić M, Vilajosana X, Duquennoy S, Dujovne D. 6TiSCH Minimal Scheduling Function (MSF). Tech. Rep. 06, Internet Engineering Task Force (IETF); 2019.

4. Duquennoy S, Nahas BA, Landsiedel O, Watteyne T. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In: ACM. ACM; 2015; Seoul, South Korea: 337–350

5. Domingo-Prieto M, Chang T, Vilajosana X, Watteyne T. Distributed PID-Based Scheduling for 6TiSCH Networks. *IEEE Communications Letters* 2016: 1006 – 1009. doi: 10.1109/LCOMM.2016.2546880

6. Palattella MR, Accettura N, Dohle M, Grieco LA, Boggia , Gennaro . Traffic Aware Scheduling Algorithm for reliable low-power multi-hop IEEE 802.15.4e networks. In: IEEE. IEEE; 2012; Sydney, NSW, Australia.

7. Accettura N, Palattella MR, Boggia G, Grieco LA, Dohler M. Decentralized Traffic Aware Scheduling for multi-hop Low power Lossy Networks in the Internet of Things. In: IEEE. IEEE; 2013; Madrid, Spain.

8. Municio E, Daneels G, Vučinić M, et al. Simulating 6TiSCH networks. *Emerging Telecommunications Technologies* 2018; 30(Issue 3). doi: https://doi.org/10.1002/ett.3494