Abstract

This document provides the method that autonomously and dynamically schedules TSCH slotframe based on the current traffic load. In the introduction part, we introduce some basic knowledge on TSCH and RPL. After this, we introduce some existing TSCH schedulers and their limitations. They are classified as centralized, distributed, and autonomous. Then, we propose a novel autonomous and dynamic TSCH cell scheduler. With this, each node schedules its own TSCH slotframe and allocates additional cells based on the current traffic load without any negotiation or traffic overhead. Since they allocate additional cells only when needed, energy efficient operation can be achieved simultaneously.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 27, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

TSCH is one mode on IEEE 802.15.4e standard and it is an acronym of Time Slotted Channel Hopping, which has the feature of Time Division Multiple Access with channel hopping. From 2003, IEEE 802.15.4 standard has been introduced and the standard was extended with 2006 version. Zigbee is one of the mostly used protocol which is on the basis of 802.15.4 standard. However, the standard was not enough to support industrial network which requires higher reliability. To better support industrial markets by increasing robustness against external interference, the MAC portion of the standard has been amended. Among 5 modes it provides, TSCH is receiving most attentions from WSN researchers and industry market. Moreover, by Internet Engineering Task Force (IETF) 6TiSCH working group, it has been selected as a mac protocol to connect sensor nodes with Internet over IPv6.

RPL is a routing protocol for low power and lossy network and has been standardized by the IETF RoLL WG in 2012. RPL builds tree
topology on the basis of DODAG structure. To build and manage routing topology, RPL control packets are exchanged. The routing topology is maintained by using Rank and Objective Function. Each node has its own rank and rank can be understood as the path cost of each node to the root. Definition of rank, rank calculation or parent selection rules are defined in the used objective function (OF). Every node in the RPL network has a preferred parent except the root. The rank of parent must be always lower than its own rank and the rank of its children must be always higher than its own rank. By checking this value, routing loop is avoided or detected.

In this document, the point is that each node has preferred parent and children as its RPL neighbor. To maintain routing topology, they exchange RPL control messages periodically. Though other routing topologies can be used over TSCH network, RPL is used mostly. Moreover, to build IETF 6TiSCH compliant network, RPL and TSCH should be used together.

Let’s move on to network bootstrap phase on TSCH and RPL network. Since TSCH is a link-layer protocol and RPL is a routing-layer protocol, a node should join TSCH network first and join RPL next.

To build TSCH network, each node sends Enhanced Beacon (EB) which includes TSCH network information such as frequency hopping sequence, used frequency, the length of slotframe and Absolute Slot Number (ASN). A node who received EB joins TSCH network and it also broadcasts EB periodically. When a node joins TSCH network, a node sets a node who sent EB as its time source. Since TSCH is a time synchronized network, each node synchronizes its time to its time source node.

When a node joins TSCH network, it should join RPL network to exchange application-level messages. However, though IEEE 802.15.4e standard defines how mac executes TSCH schedule, it does not provide how each node should build TSCH schedule. In general, simple TSCH scheduling method is used at the initial network bootstrap phase. Here, a cell (0,0) on a TSCH slotframe can be used by all the nodes as a shared slot. All the nodes wake up at this slot and sleep at the other timeslots synchronously. RPL control packets are exchanged on this shared slot and the routing topology is constructed.

However, just using only one cell on a TSCH slotframe does not seem to make good use of TSCH technique. In order to take full advantage of the benefits of TSCH technique, more number of cells on a slotframe should be used for efficient communication. Here, how each node schedules the slotframe directly affects network performance including PDR, latency, duty cycle and throughput. However, the standard does not answer this question.

When we design TSCH scheduling algorithm, we should consider various aspects. For example, each node should have as many transmission opportunity as the traffic load. A node close to the root requires
more communication opportunity compared to the leaf node, since it should forward more number of packets between the root and its sub-network. A node should receive packets from different neighbors on different time slots to avoid contention at a cell. A node cannot receive packets from multiple nodes at the same timeslot. With only one radio interface, a node cannot listen and transmit at the same timeslot.

It is not simple to set up a good schedule. Moreover, we should remember that the RPL topology evolves over time. Even if a node has an optimal schedule now, when the topology changes, new schedule for the changed topology is needed.

Moreover, since the schedule of slotframe iterates over time, links assigned to a crowded cell continue to suffer disadvantages. Thereby, how we schedule slotframe directly affects performance of TSCH, which again affects RPL performance.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

3. Terminology

Cell : This document follows the same definition used in [RFC 7554]
ASN : This document follows the same definition used in [RFC 7554]
Slotframe : This document follows the same definition used in [RFC 7554]

4. TSCH Schedulers

From now on, we will explore the existing works regarding TSCH schedule. Depending upon their approach, scheduling methods are categorized into 3 types: centralized, decentralized and autonomous.

4.1. Overview

Recall that IEEE 802.15.4e was standardized in 2012. At the first year, centralized approaches have been proposed. With this approach, a root node receives all network information from every node. After obtaining a global view, this central node sets up optimal schedule.
However, this method requires high traffic overhead. Moreover, when a new node joins network, the schedule of the whole nodes in the network should be changed.

With this problem, naturally, the scheduling approach has been changed from centralized to distributed approach. In this methods, the scheduling load at the central node has been distributed to all the nodes in the network. Though more flexible and scalable scheduling could be achieved with this approach, further traffic overhead for TSCH schedule still exists.

In 2015, a scheduling approach named orchestra have been proposed, which does not require any traffic overhead or negotiation process. With this method, each node can autonomously schedule its own TSCH slotframe. Since it does not require any negotiation for TSCH scheduling, it could achieve highly flexible and scalable schedule regardless of topology variation. In April 2018, two more autonomous scheduling method, which have similar approach to Orchestra have been introduced. [Orchestra]

Since autonomous scheduling method does not require any traffic overhead for TSCH scheduling, though it does not provide optimal schedule, it provides high reliability, high flexibility, high scalability with low scheduling overhead.

### 4.2. Centralized schedulers

TASA and MODESA are mostly cited TSCH centralized schedulers. These two methods are very similar except only a few differences. Both methods have very strong assumptions. They assume that the central node knows complete topology information including routing topology, traffic load of each node, conflict relations among nodes. They assume that traffic load per each node does not change over time and routing topology does not change over time. Since they assume multi-point-to-point traffic scenario, they schedule only the upstream links. Moreover, they assume no message loss with 0% link error rate. [TASA][MODESA]

The difference between these two schedulers is that MOESA assumes more complex network. For example, it considers queue-length of each node and assumes that the root node can have multiple radio interfaces. With strong assumptions described above, they could build collision-free schedule. They evaluated the proposed methods by using simple simulation. Moreover, since there is no message loss at link-level, they only evaluated delay, duty cycle and throughput.

### 4.3. Distributed schedulers

DeTAS and Wave are the early schedulers based on distributed approach. Though the load of scheduling at the central node has been distributed to all the nodes in the network, only the
centralized node can initiate the start of the schedule of all
the nodes in the network and each node’s schedule is strongly
dependent on the schedule of its parent node. As a result, any
topology or traffic change results in a scheduling change of the
entire network as centralized schedule does. [DeTAS1][DeTAS2]
[Wave1][Wave2]
Both methods assume that each node has a constant traffic load and
multipoint-to-point scenario. The root node initiates the start of
the scheduling process. The node closer to the root selects cells
first. The cell selection rule is simple. Each node selects
timeslots those were not selected by its parent.
The difference is that, DeTAS selects channel offset on the basis of
hop-count, and Wave considers conflicting nodes group on its
channel offset schedule. To have knowledge on each node’s conflicting
node group, Wave requires more control packets to exchange that
information.
However, these methods cannot be considered as a fully distributed
scheduling method in that each node’s schedule strongly depends on
parent’s choice.

From 2014, negotiation protocol for distributed TSCH schedule have
been proposed. The proposed negotiation technique have been
gradually developed through active open discussions in 6TiSCH
working group and has been standardized recently. They named the
negotiation protocol as 6P. [RFC 8480] By using 6P transactions, two
neighboring nodes can add, delete or relocate cells in their TSCH
slotframe for their unicast communication. However, 6P is just
used as a negotiation protocol. When to trigger 6P transaction
and how to schedule slotframe is determined by scheduling function.
OTF, SF0 and MSF are negotiation-based distributed scheduling method
that use 6P transaction. [OTF][SF0][MSF] With this methods, each node
flexibly schedules its own slotframe based on the current traffic
load. Whenever the required traffic load changes, each node adds or
delete cells. To avoid frequent schedule changes, they use
threshold.
MSF has been recently proposed, which schedules slotframe
autonomously by following Orchestra receiver based mode. If
additional cell is needed, MSF uses 6P transactions for further
cell allocation.

4.4. Autonomous schedulers

[Orchestra] was introduced on Sensys 2015 and it was the first and
the only one autonomous scheduler until April 2018. For three
years, no autonomous scheduler have been introduced. Although
two more autonomous schedulers have been introduced recently,
their fundamental design framework is the same as that of
orchestra: they calculate timeslots by hashing node ID.
The key idea of orchestra is that each node can obtain node IDs
of its parent and children from RPL level. Orchestra provides two
operation mode: sender-based and receiver-based. In sender-based
mode, each node schedules its transmission cell by hashing its own node ID. By using this rule, it calculates receive cells by hashing its neighbor’s node ID, which are transmission cells of its neighbor nodes. In the same way, in receiver-based mode, each node calculates its receive cell by hashing its own node ID and calculates its transmission cells by hashing its neighbor nodes’ ID. It is very simple and does not require any transmission overhead for TSCH schedule. No negotiation is needed. Its reliable performance was proved on a real-world large public testbed. However, the limitation of Orchestra is that it allocates only one TX or RX cell per slotframe, which limits throughput, causing congestion near the root node. Besides, the hashed values of different nodes may be the same value, which makes the problem worse.

[Escalator] pointed out that Orchestra gives the nodes close to the root and leaf nodes the same communication opportunities. It focuses on the fact that a node with a larger sub-network has more packets to forward. Thereby, the key idea is that each node allocates different cells on its slotframe for all the nodes in its sub-network. With this method, the node closer to the root has more cells to forward packets. With the increased communication opportunity, it could achieve higher throughput. However, since a node closer to the root has large sub-tree, too many cells may be allocated on its slotframe. Consequently, this node should be awake for a longer time, regardless of the actual traffic load. However, if the traffic load on the network is low, this node does not need to stay awake for long. Moreover, since this method considers hop-count information on its scheduling, with varying routing topology, it may not work properly.

To overcome limitation of Orchestra which allocates only one TX or RX cell per a slotframe, [e-TSCH-Orch] proposes different solution. The key idea of the proposed method is that it allocates additional multiple cells dynamically to clean the transmission queue quickly. With this method, end-to-end delay is reduced with the increased communication opportunity. Moreover, since cells are added only when needed, nodes can use energy more efficiently.

In this method, slotframe is first scheduled based on Orchestra receiver-based mode. Whenever each node transmits message, it sends the number of packets in its transmission queue together with the message, by using TSCH header. Then, a node who received that message allocates additional cells consecutively from the immediately next time slot. The sender sends message sequentially and quickly cleans its queue. However, rules for additive cell allocation are too simple, thereby communication on additional cells may interfere with communication in the existing cells scheduled by orchestra method, which may cause network to be broken down.
5. Autonomous and Dynamic TSCH Cell Allocation Method

The trend of recent TSCH schedule research has changed from centralized to distributed and has changed from a decentralized approach to an autonomous approach. Regarding autonomous scheduler, the most influential method to date is the orchestra, and all subsequent studies are based on the design structure of the orchestra. They hash node ID to calculate cells for communication.

Recall that each node has multiple neighbors and one node is connected with multiple links. With node-based scheduling method, multiple different links are scheduled on a same cell. As a result, scheduled links are concentrated in only a few cells wasting other time slots or channel offsets. Each node SHOULD utilize more time slots and more channel offsets in more efficient way.

In the proposed method, we change the scheduling paradigm from node-based to link-based. Moreover, it dynamically and autonomously adds and deletes cells on a slotframe based on the current traffic load.

Here, we use 4 different slotframes: 1) TSCH EB, 2) Broadcast/Default, 3) Unicast, 4) Supplementary slotframes.

Scheduling rules for TSCH EB and Broadcast/Default slotframes follow those of orchestra. To schedule unicast and supplementary slotframes, we use a link-based scheduling approach.

5.1. Schedule of Unicast Slotframe

On the unicast slotframe, each node autonomously schedules all directional links to its RPL neighbors in different cells. As a result, the number of transmission (outgoing) links is equal to the number of its RPL neighbors (preferred parent and children) and the number of receive (incoming) links is also equal to the number of its RPL neighbors. We define a link \((X,Y)\) as the link from node \(X\) to node \(Y\). Then, the link ID of the link \((X,Y)\) is calculated by the following equation.

\[
\text{linkID}(X,Y) = b \times \text{nodeID}(X) + \text{nodeID}(Y)
\]

Here, the coefficient \(b\) is used to distinguish the direction of the link, and the value \(b\) can be the maximum value of available node IDs. For example, if we use nodeID as the last byte of the mac address, the value of coefficient \(b\) is 256. Thereby, we can distinguish each directional link by using this equation.

Moreover, we also use time information by adopting ASFN. ASFN is
Absolute Slotframe Number that can be calculated using the following equation.

\[ \text{ASFN} = \text{floor}( \text{ASN} / \text{SlotframeLength} ) \]

The proposed method hashes the directional link ID and the ASFN together. As a result, the TSCH schedule of the unicast slotframe is changed every slotframe cycle. Therefore, even if multiple links are scheduled in a same cell, they are fully distributed in the next cycle.

Moreover, the proposed method uses multiple channel offsets so as to take the benefits of channel diversity. Here we use the same rule for both time offset and channel offset calculation. The hashed value is projected to one offset and the projection area depends on the number of available offset.

For hash function, a combination of integer mix function and modulo operation can be used as the following form.

\[ \text{Hash}(X,Y) = \text{IntegerMixFunction}(X) \mod Y \]

\( \text{Hash}(X,Y) \) therefore outputs a value in the range \([0, Y - 1]\) from input \(X\). Thereby, we use the following equations to calculate a cell for the link \((A,B)\) at the current ASFN.

\[
\begin{align*}
\text{timeOffset} & = \text{Hash}\left( \text{linkID}(A,B)+\text{ASFN} , \text{Nt}_{-}\text{uc} \right) \\
\text{channelOffset} & = \text{Hash}\left( \text{linkID}(A,B)+\text{ASFN} , \text{Nc}_{-}\text{uc} \right) + 1
\end{align*}
\]

Where \(\text{Nt}_{-}\text{uc}\) and \(\text{Nc}_{-}\text{uc}\) are the number of time offsets and channel offsets of the unicast slotframe, respectively. Thus, the ranges for time offset and channel offset are \([0, \text{Nt}_{-}\text{uc} - 1]\) and \([1, \text{Nc}_{-}\text{uc}]\), respectively. We do not use channel offset 0 since it is used by TSCH EB slotframe. Due to priority settings between slotframes, though channel offset 1 is used in the broadcast/default slotframe, this offset can also be used in the unicast slot frame.

For example, when the shared slot of the broadcast/default slotframe is activated, the channel offset 1 is used only for performing the corresponding operation. At this time, although a cell in the unicast slotframe is simultaneously scheduled at the same time slot, the shared slot of the broadcast/default slotframe is activated because of the priority setting between the slot frames. In the remaining case, since broadcast/default slotframe is not activated, the channel offset 1 is used only for the unicast slotframe.

As described above, we use time information in scheduling and we also use the integer mix function to achieve the result that the
output value is completely different when the input value is changed by 1. Consequently, if the ASFN increases by 1 for every slot frame cycle, the next unicast slot frame schedule will be completely changed from the current schedule.

With this scheduling method, on the unicast slotframe, each node allocated a cell for each directional link between the node itself and its RPL neighbors. When the network traffic load is low enough, allocating only one cell for each directional link per slotframe is enough to support reliable communication. However, as the traffic load increases, just allocating only one cell may not be enough. Moreover, a node close to the root has more packets to forward to support multi-hop communication between the root and the nodes in its sub-network.

Depending on the current traffic load, each node SHOULD dynamically and flexibly manage the number of unicast cells it can use.

5.2. Schedule of Supplementary Slotframe

We can think two approaches. One is static additive cell allocation based on the sub-tree size of each node. The other is dynamic and additive cell allocation based on the current traffic load to its RPL neighbors. In the former method, if the traffic load is low, the additional allocated cells might not be used, resulting in energy wastage. So we use the latter approach when we allocate additional cells to achieve energy efficiency. Because additional cells are allocated only when needed, we don’t have to worry about unnecessary wake-up period.

Moreover, the additional allocated cells MUST NOT interfere with communication scheduled at the previous three (TSCH EB, Broadcast /Default, Unicast) slotframes. So we use completely different channels on supplementary slotframe with the lowest priority among all the slotframes used.

To recognize the current traffic load and detect the traffic change in real time, each node manages 4 parameters (myTxCount, myNumTx, NumTx, NumRx) per each neighbor node X.

myTxCount(X): A counter that counts the number of required transmission cell per slotframe from the node itself to its neighbor X. The value is used to update myNumTx(X). Using myTxCount(X), myNumTx(X) is updated at the last active cell of the current slotframe schedule. After myNumTx(X) is updated, myTxCount(X) SHOULD be set as 0 to count the number of required transmission cell for the next slotframe cycle. When counting how many cells are needed, it SHOULD consider NOT only the number of enqueued packets in the
transmission queue but also mac-layer retransmissions.

myNumTx(X): The required number of transmission cells per slotframe for node X. The value is updated at the last active cell of the current slotframe schedule of the current ASFN. Update equation can be as follows: myNumTx(X) = (1-e)*myNumTx(X)+e*myTxCount(X). Here, a coefficient e is used to implement exponentially weighted moving average (EWMA). After the calculation of new myNumTx(X) at the last active cell of the current slotframe, myTxCount(X) is set as 0 to count the number of required transmission cell for the next slotframe cycle.

NumTx(X): The number of transmission slots per supplementary slotframe for node X. Whenever a node sends a unicast packet to its neighbor node X, it sends the number myNumTx(X) on the TSCH header together with the message. After receiving the link-layer acknowledgement from node X, the node updates NumTx(X) to myNumTx(X). The node can not update the value NumTx(X) to myNumTx(X) until it receives link-layer acknowledgement to the sent unicast message to node X.

NumRx(X): The number of listen slots per supplementary slotframe for node X. Each time a unicast packet is received from the node X, the value is updated to the latest value.

Example scenario: There are node A and B. Node A sends a unicast message to node B periodically. Every slotframe cycle, node A counts the number of required transmission cells for node B by managing myTxCount(B). Based on this value, myNumTx(B) is updated at the last cell of each slotframe cycle. Whenever node A sends a unicast packet to node B, it puts myTxCount(B) in the TSCH header. When it receives mac-layer acknowledgement from node B, it updates NumTx(X) to the current myTxCount(B). Since node B received unicast message sent from node A successfully, it knows how much additional transmission cells A will allocate through the information in the TSCH header. So node B allocates that amount of additional listen cells. All these additional cells are allocated only on the supplementary slotframe.

Each node has NumTx(X) and NumRx(X) for each RPL neighbors. It SHOULD allocate NumTx(X) more transmission cells for the transmission link (NodeID,X) and NumRx(X) more listen cells for the receive link (X,NodeID). Thereby each node knows the number of additional cells that SHOULD be scheduled for link (X,Y). According to this value, each link (X,Y) has the number of additional cells that SHOULD be scheduled on supplementary slotframe and we assign different traffic ID for each additional requirement for each link. For example, if N additional cell assignments are required for the link (X,Y), we have N different trfID(X,Y) and which are 1, 2, 3, ... , N.
By using trfID(X,Y), linkID(X,Y) and ASFN, each additional cell on supplementary slotframe for link (X,Y) is scheduled by using the following equations.

\[
\text{timeOffset} = \text{Hash}(a \ast \text{trfID}(X,Y) + \text{linkID}(X,Y) + \text{ASFN}, N_t_{sc})
\]

\[
\text{channelOffset} = \text{Hash}(a \ast \text{trfID}(X,Y) + \text{linkID}(X,Y) + \text{ASFN}, N_c_{sc}) + 1 + N_c_{uc}
\]

Where $N_t_{sc}$ and $N_c_{sc}$ are the number of time offset and channel offset in supplementary slotframe. Here, coefficient $a$ is used to distinguish a tuple of traffic ID and link ID. The value of coefficient $a$ is the maximum value of link ID. Therefore, if we use the last byte of the mac address as node ID, the maximum value of node ID is 256 and the maximum value of link ID is $256 \ast 256$. Thereby, we use $256 \ast 256 = 65,536$ as coefficient $a$.

For channel offset, we add $1 + N_c_{uc}$ to the hashed output not to use the same channels used by TSCH EB, Broadcast/Default and Unicast slotframes. As a result, cells scheduled on supplementary slotframe does not interfere with communication in the other three slotframes.

5.3. Summary

Here, we summarize the proposed method. The key idea is simple. The proposed method is dynamic and autonomous TSCH cell scheduling method which does not require any negotiation process or additional packet exchange. We use directional link ID to allocate different cell for each link and they are scheduled on unicast slotframe. We use ASFN to periodically change the slotframe schedule not to disadvantage links scheduled on a crowded cell. If communication opportunities provided by unicast slotframe is not enough, we use supplementary slotframe for additional cell allocation. Supplementary slotframe does not use the channels used by the other slotframes and has the lowest priority among all the used slotframes, not to interfere default schedule. Since the number of further allocated cells is based on the current traffic load, additional cells are allocated only when needed and are automatically deleted if not needed.

6. IANA Considerations

There are no IANA considerations related to this document.

7. Security Considerations

Autonomous and dynamic TSCH cell allocation method for time-varying
traffic load and evolving topology has similar requirements on security as [RFC 7554].

8. ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIP) (No.2016R1A5A1012966), MSIP support program (IITP-2018-2015-0-00378) supervised by the IITP, IITP grant funded by the Korea government (MSIT) (No.2015-0-00557, Resilient/Fault-Tolerant Autonomic Networking Based on Physicality, Relationship and Service Semantic of IoT Devices) and Institute for Industrial Systems Innovation of Seoul National University.

9. References

9.1. Normative References


9.2. Informative References


Authors’ Addresses

Seohyang Kim
Department of Computer Science and Engineering
Seoul National University
Seoul, Republic of Korea
Phone: +82 (0)2 880 1858
Email: shkim@popeye.snu.ac.kr

Hyung-Sin Kim
Computer Science Division
University of California, Berkeley
Berkeley, CA, USA
Phone: +82 (0)2 880 1858
Email: hs.kim@cs.berkeley.edu

Chongkwon Kim
Department of Computer Science and Engineering
Seoul National University
Seoul, Republic of Korea
Phone: +82 (0)2 880 1858
Email: ckim@snu.ac.kr