                  6TiSCH Minimal Scheduling Function (MSF)
                         draft-ietf-6tisch-msf-00

Abstract

   This specification defines the 6TiSCH Minimal Scheduling Function
   (MSF). This Scheduling Function describes both the behavior of a
   node when joining the network, and how the communication schedule is
   managed in a distributed fashion. MSF builds upon the 6TiSCH
   Operation Sublayer Protocol (6P) and the Minimal Security Framework
   for 6TiSCH.

Requirements Language

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and
   "OPTIONAL" in this document are to be interpreted as described in
   [RFC2119].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   The 6TiSCH Minimal Scheduling Function (MSF), defined in this
   specification, is a 6TiSCH Scheduling Function (SF). The role of an
   SF is entirely defined in [I-D.ietf-6tisch-6top-protocol]: it
   complements [I-D.ietf-6tisch-6top-protocol] by providing the rules of
   when to add/delete cells in the communication schedule. The SF
   defined in this document follows that definition, and satisfies all
   the requirements for an SF listed in Section 4.2 of
   [I-D.ietf-6tisch-6top-protocol].

   MSF builds on top of the following specifications: the Minimal IPv6
   over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration
   [RFC8180], the 6TiSCH Operation Sublayer Protocol (6P)
   [I-D.ietf-6tisch-6top-protocol], and the Minimal Security Framework
   for 6TiSCH [I-D.ietf-6tisch-minimal-security].

   MSF defines both the behavior of a node when joining the network, and
   how the communication schedule is managed in a distributed fashion.
   When a node running MSF boots up, it joins the network by following
   the 7 steps described in Section 4. The end state of the join
   process is that the node is synchronized to the network, has mutually
   authenticated to the network, has identified a preferred routing
   parent, has scheduled one default unicast cell to/from each of its
   neighbors. After the join process, the node can continuously
   add/delete/relocate cells, as described in Section 5. It does so for
   3 reasons: to match the link-layer resources to the traffic, to
   handle changing parent, to handle a schedule collision.

   MSF is designed to operate in a wide range of application domains.
   It is optimized for applications with regular upstream traffic (from
   the nodes to the root). Appendix C contains a performance evaluation
   of MSF.

   This specification follows the recommended structure of an SF
   specification in Appendix A of [I-D.ietf-6tisch-6top-protocol], with
   the following adaptations:

   o We have reordered part of the sections, in particular to have the
      section on the node behavior at boot [Section 4](#) appear early in
      this specification.
   o We added sections on the interface to the minimal 6TiSCH
      configuration [Section 2](#), the use of the SIGNAL command [Section 6](#),
      the MSF constants [Section 14](#), the MSF statistics [Section 15](#), the
      performance of MSF [Appendix C](#).
   o This specification does not include an examples section.

2. Interface to the Minimal 6TiSCH Configuration

   A node implementing MSF MUST implement the Minimal 6TiSCH
   Configuration [RFC8180], which defines the "minimal cell", a single
   shared cell providing minimal connectivity between the nodes in the
   network.

   MSF uses the minimal cell to exchange the following packets:

   1. Enhanced Beacons (EBs), defined by [IEEE802154-2015]. These are
       broadcast frames.
   2. DODAG Information Objects (DIOs), defined by [RFC6550]. These
       are broadcast frames.

   Because the minimal cell is SHARED, the back-off algorithm defined in
   [IEEE802154-2015] is used to resolve collisions. To ensure there is
   enough bandwidth available on the minimal cell, a node implementing
   MSF SHOULD enforce the following rules for broadcast frames:

   1. send EBs on a portion of the minimal cells not exceeding
       $1/(3(N+1))$, where N is the number of neighbors of the node.
   2. send DIOs on a portion of the minimal cells not exceeding
       $1/(3(N+1))$, where N is the number of neighbors of the node.

   The RECOMMENDED behavior for sending EBs is to have a node send EBs
   with a probability of $1/(3(N+1))$. The RECOMMENDED behavior for
   sending DIOs is to use a Trickle timer with rate-limiting.

   [Section 4.3](#) describes how to evaluate the number of neighbors during
   the joining process. After the joining process, how to evaluate the
   number of neighbors is implementation-specific.

   As detailed in Section 2.2 of [I-D.ietf-6tisch-6top-protocol], MSF
   MUST schedule cells from Slotframe 1, while Slotframe 0 is used for
   traffic defined in the Minimal 6TiSCH Configuration. The length of
   Slotframe 0 and Slotframe 1 SHOULD be the same value. The default of
   SLOTFRAME_LENGTH is RECOMMENDED, although any value can be advertised
   in the EBs.

3. **Autonomous Unicast Cells**

   MSF nodes MUST initialize Slotframe 1 with a set of default cells for
   unicast communication with their neighbors. These cells are referred
   to as 'autonomous cells', because they are maintained autonomously by
   each node. Each node has:

   1. One cell to receive, at a [slotOffset,channelOffset] computed as
       a hash of the node's EUI64 (detailed next). The cell options for
       this cell are RX=1.
   2. For each neighbor in the IPv6 neighbor table, one cell to
       transmit, at a [slotOffset,channelOffset] computed as a hash of
       the neighbor's EUI64 (detailed next). The cell options for this
       cell are TX=1, SHARED=1.

   To compute a [slotOffset,channelOffset] from an EUI64 address, nodes
   MUST use the hash function SAX [SAX-DASFAA]. The coordinates are
   computed to distribute the cells across all 16 channel offsets, and
   all but the first time offsets of Slotframe 1. The first time offset
   is skipped to avoid colliding with the minimal cell in Slotframe 0.
   The slot coordinates derived from a given EUI64 address are computed
   as follows:

      slotOffset(MAC) = 1 + hash(EUI64) % (length(Slotframe_1) - 1)
      channelOffset(MAC) = hash(EUI64) % 16

   Because of hash collisions, there are cases where one node has
   multiple cells scheduled at the same time offset and/or channel
   offset. Note that nodes have only one autonomous RX cell and
   potentially multiple TX cells. Hash collisions among a set of cells
   at a given time offset is resolved at run-time as follows:

   1. The TX cell with the most packets in outgoing queue takes
       precedence.
   2. If all TX cells have empty outgoing queues, the RX cell takes
       precedence.

   Throughout the network lifetime, nodes MUST maintain the autonomous
   cells as follows:

   1. The receive cell MUST always remain scheduled.
   2. Whenever a new neighbor is discovered, add a transmit cell for
       it.
   3. Whenever a new neighbor is removed, remove transmit cell that was
       assigned to it.
   4. 6P CLEAR MUST NOT erase autonomous cells.

[4](#). Node Behavior at Boot

This section details the behavior the node SHOULD follow from the
moment it is switched on, until it has successfully joined the
network. [Section 4.1](#) details the start state; [Section 4.9](#) details
the end state. The other sections detail the 7 steps of the joining
process. We use the term "pledge" and "joined node", as defined in
[[I-D.ietf-6tisch-minimal-security](#)].

[4.1](#). Start State

A node implementing MSF MUST implement the Minimal Security Framework
for 6TiSCH [[I-D.ietf-6tisch-minimal-security](#)]. As a corollary, this
means that a pledge, before being switched on, is pre-configured with
the Pre-Shared Key (PSK) for joining, as well as any other
configuration detailed in [[I-D.ietf-6tisch-minimal-security](#)].

[4.2](#). Step 1 - Choosing Frequency

When switched on, the pledge SHOULD randomly choose a frequency among
the available frequencies, and start listening for EBs on that
frequency.

[4.3](#). Step 2 - Receiving EBs

Upon receiving the first EB, the pledge SHOULD continue listening for
additional EBs to learn:

1. the number of neighbors N in its vicinity
2. which neighbor to choose as a Join Proxy (JP) for the joining
    process

While the exact behavior is implementation-specific, the RECOMMENDED
behavior is to follow [[RFC8180](#)], and listen until EBs sent by
NUM_NEIGHBOURS_TO_WAIT nodes (defined in [[RFC8180](#)]) have been
received.

During this step, the pledge MAY synchronize to any EB it receives
from the network it wishes to join. How to decide whether an EB
originates from a node from the network it wishes to join is
implementation-specific, but MAY involve filtering EBs by the PAN ID
field it contains, the presence and contents of the IE defined in
[[I-D.richardson-6tisch-join-enhanced-beacon](#)], or the key used to
authenticate it.

The decision of which neighbor to use as a JP is implementation-
specific, and discussed in [[I-D.ietf-6tisch-minimal-security](#)].

4.4. **Step 3 - Setting up Autonomous Unicast Cells**

   After joining, nodes MUST set up their autonomous unicast cells, as
   described in Section 3. This enables unicast communication in
   Slotframe 1, until more cells are added with 6P as defined in
   Section 5.

4.5. **Step 4 - Join Request/Response**

   As per [I-D.ietf-6tisch-minimal-security], after having selected a
   JP, the pledge sends a Join Request to its JP. Because no dedicated
   cells are in place at this point, this happens on the autonomous
   unicast cell. The JP then forwards the Join Request to the JRC,
   possibly over multiple hops. When forwarding this Join Request, a
   node MUST use a unicast cell (autonomous or dedicated) it has with
   its preferred parent. How dedicated cells are installed is detailed
   in Section 5.

   As per [I-D.ietf-6tisch-minimal-security], the JRC sends back a Join
   Response to the pledge, through the JP. When forwarding this Join
   Response, a node MUST use a unicast (autonomous or dedicated) cell it
   has with its child (not the minimal cell).

   As per [I-D.ietf-6tisch-minimal-security], after receiving the Join
   Response, the pledge learns the keying material used in the network,
   as well as other configurations, and becomes a "joined node".

4.6. **Step 5 - Acquiring a RPL rank**

   Because it has learned the link-layer keying material used in the
   network, the joined node can now decrypt the DIO packets sent by its
   neighbors. Per [RFC6550], the joined node receives DIOs, computes
   its own rank, and selects a preferred parent.

4.7. **Step 6 - Send EBs and DIOs**

   The node SHOULD start sending EBs and DIOs on the minimal cell, while
   following the transmit rules for broadcast frames from Section 2.

4.8. **Step 7 - Neighbor Polling**

   The node SHOULD send some form of keep-alive messages to all its
   neighbors it has unicast cells with. The Keep-Alive (KA) mechanism
   is detailed in [RFC7554]. It uses the keep-alive messages to its
   preferred parent to stay synchronized. It uses the keep-alive
   messages to its children (with which it has a unicast cell to) to
   ensure the child is still reachable. The RECOMMENDED period for
   sending keep-alive messages is KA_PERIOD.

   If the keep-alive message to a child fails at the link layer (i.e.
   the maximum number of link-layer retries is reached), the node SHOULD
   declare the child as unreachable. This can happen for example when
   the child node is switched off.

   When a neighbor is declared unreachable, the node MUST remove all
   dedicated cells with that neighbor from its own schedule. In
   addition, it MAY issue a 6P CLEAR to that neighbor (which can fail at
   the link-layer). If the node has autonmous cells to the unreachable
   neighbor those cells will be removed following the procedure
   described in Section 3.

4.9. End State

   For a new node, the end state of the joining process is:

   o it is synchronized to the network
   o it is using the link-layer keying material it learned through the
      secure joining process
   o it has identified its preferred routing parent
   o it has a set of autonomous unicast cells to/from its neighbors
   o it is periodically sending DIOs, potentially serving as a router
      for other nodes' traffic
   o it is periodically sending EBs, potentially serving as a JP for
      new joining nodes

5. Rules for Adding/Deleting Cells

   Once a node has joined the 6TiSCH network, it adds/deletes/relocates
   cells with its preferred parent for three reasons:

   o to match the link-layer resources to the traffic between the node
      and its preferred parent (Section 5.1)
   o to handle switching preferred parent (Section 5.2)
   o to handle a schedule collision (Section 5.3)

5.1. Adapting to Traffic

   A node implementing MSF MUST implement the behavior described in this
   section.

   In order to handle transient traffic bursts, MSF uses the
   [IEEE802154-2015] frame pending bit (page 152, Section 7.2.1.3). By
   setting the bit, a node can transmit a series of packets to a given
   neighbor in consecutive time offsets. The next paragraphs define how
   to handle longer-term fluctuations in traffic, using 6P.

The goal of MSF is to manage the communication schedule in the 6TiSCH
schedule in a distributed manner. For a node, this translates into
monitoring the current usage of the cells it has to its preferred
parent:

o If the node determines that the number of link-layer frames it is
   attempting to exchange with its preferred parent per unit of time
   is larger than the capacity offered by the TSCH unicast cells
   (dedicated and autonmous cells) it has scheduled with it, the node
   triggers a 6P Transaction with its preferred parent to add
   dedicated cells to the TSCH schedule of both nodes.
o If the traffic is lower than the capacity, the node triggers a 6P
   Transaction with its preferred parent to delete dedicated cells
   from the TSCH schedule of both nodes.

From the join process, the node already has a set of autonmous
unicast cells, as defined in [Section 3](#). The autonmous cells MUST
NOT be removed by 6P, so that there always exists a unicast cell
between a node and its preferred parent, even if no frames are being
exchanged between them. Autonomous cells are used indistinguishably
together with dedicated cells, for broadcast or unicast traffic with
the target neighbor. The procedure to remove autonomous cells is
described in [Section 3](#).

Adding/removing/relocating cells involves exchanging frames that
contain 6P commands. All 6P frames MUST be sent on the unicast cells
(and not the minimal cell).

The node MUST maintain the following counters for its preferred
parent:

NumCellsPassed: Counts the number of unicast cells (dedicated and
    autonmous cells) that have passed since the counter was
    initialized. This counter is initialized at 0. Each time the
    TSCH state machine indicates that the current cell is a unicast
    cell to the preferred parent, NumCellsPassed is incremented by
    exactly 1, regardless of whether the cell is used to transmit/
    receive a frame.
NumCellsUsed: Counts the number of unicast cells that have been
    used. This counter is initialized at 0. NumCellsUsed is
    incremented by exactly 1 when, during a unicast cell to the
    preferred parent, either of the following happens:

    * The node sends a frame to its preferred parent. The counter
       increments regardless of whether a link-layer acknowledgment
       was received or not.
    * The node receives a frame from its preferred parent.

   Implementors MAY choose to create the same counters for each
   neighbor, and add them as additional statistics in the neighbor
   table.

   The counters are used as follows:

   1. Both NumCellsPassed and NumCellsUsed are initialized to 0 when
       the node boots.
   2. When the value of NumCellsPassed reaches MAX_NUMCELLS:

       * If NumCellsUsed > LIM_NUMCELLSUSED_HIGH, trigger 6P to add a
          single cell to the preferred parent
       * If NumCellsUsed < LIM_NUMCELLSUSED_LOW, trigger 6P to remove a
          single cell to the preferred parent
       * Reset both NumCellsPassed and NumCellsUsed to 0 and go to step
          2.

5.2. Switching Parent

   A node implementing MSF MUST implement the behavior described in this
   section.

   Part of its normal operation, the RPL routing protocol can have a
   node switch preferred parents. The procedure for switching from the
   old preferred parent to the new preferred parent is:

   1. the node counts the number of dedicated (unicast but not
       autonomous) cells it has per slotframe to the old preferred
       parent
   2. the node triggers one or more 6P ADD commands to schedule the
       same number of dedicated cells to the new preferred parent
   3. when that successfully completes, the node issues a 6P CLEAR
       command to its old preferred parent

5.3. Handling Schedule Collisions

   A node implementing MSF SHOULD implement the behavior described in
   this section. The "MUST" statements in this section hence only apply
   if the node implements schedule collision handling.

   Since scheduling is entirely distributed, there is a non-zero
   probability that two pairs of nearby neighbor nodes schedule a cell
   at the same [slotOffset,channelOffset] location in the TSCH schedule.
   In that case, data exchanged by the two pairs may collide on that
   cell. We call this case a "schedule collision".

   The node MUST maintain the following counters for each cell to its
   preferred parent:

NumTx: Counts the number of transmission attempts on that cell.
    Each time the node attempts to transmit a frame on that cell,
    NumTx is incremented by exactly 1.
NumTxAck: Counts the number of successful transmission attempts on
    that cell. Each time the node receives an acknowledgment for a
    transmission attempt, NumTxAck is incremented by exactly 1.

Implementors MAY choose to maintain the same counters for each cell
in the schedule.

Since both NumTx and NumTxAck are initialized to 0, we necessarily
have NumTxAck <= NumTx. We call Packet Delivery Ratio (PDR) the
ratio NumTxAck/NumTx; and represent it as a percentage. A cell with
PDR=50% means that half of the frames transmitted are not
acknowledged (and need to be retransmitted).

Each time the node switches preferred parent (or during the join
process when the node selects a preferred parent for the first time),
both NumTx and NumTxAck MUST be reset to 0. They increment over
time, as the schedule is executed and the node sends frames to its
preferred parent. When NumTx reaches 256, both NumTx and NumTxAck
MUST be divided by 2. That is, for example, from NumTx=256 and
NumTxAck=128, they become NumTx=128 and NumTxAck=64. This operation
does not change the value of the PDR, but allows the counters to keep
incrementing.

The key for detecting a schedule collision is that, if a node has
several cells to the same preferred parent, all cells should exhibit
the same PDR. A cell which exhibits a PDR significantly lower than
the others indicates than there are collisions on that cell.

Every HOUSEKEEPINGCOLLISION_PERIOD, the node executes the following
steps:

1. It computes, for each dedicated cell with its preferred parent
    (not for the autonomous cell), that cell's PDR.
2. Any cell that hasn't yet had NumTx divided by 2 since it was last
    reset is skipped in steps 3 and 4. This avoids triggering cell
    relocation when the values of NumTx and NumTxAck are not
    statistically significant yet.
3. It identifies the cell with the highest PDR.
4. For each other cell, it compares its PDR against that of the cell
    with the highest PDR. If it's less than RELOCATE_PDRTHRES, it
    triggers the relocation of that cell using a 6P RELOCATE command.

[6](#). **6P SIGNAL command**

The 6P SIGNAL command is not used by MSF.

[7](#). **Scheduling Function Identifier**

The Scheduling Function Identifier (SFID) of MSF is
IANA_6TISCH_SFID_MSF.

[8](#). **Rules for CellList**

MSF uses 2-step 6P Transactions exclusively. 6P Transactions are
only initiated by a node towards it preferred parent. As a result,
the cells to put in the CellList of a 6P ADD command, and in the
candidate CellList of a RELOCATE command, are chosen by the node
initiating the 6P Transaction. In both cases, the same rules apply:

o The CellList SHOULD contain 5 or more cells.
o Each cell in the CellList MUST have a different slotOffset value.
o For each cell in the CellList, the node MUST NOT have any
   scheduled cell on the same slotOffset.
o The slotOffset value of any cell in the CellList MUST NOT be the
   same as the slotOffset of the minimal cell (slotOffset=0).
o The slotOffset of a cell in the CellList SHOULD be randomly and
   uniformly chosen among all the slotOffset values that satisfy the
   restrictions above.
o The channelOffset of a cell in the CellList SHOULD be randomly and
   uniformly chosen in [0..numFrequencies], where numFrequencies
   represents the number of frequencies a node can communicate on.

[9](#). **6P Timeout Value**

The 6P Timeout is not a constant value. It is calculated as
(1/C)*(1/PDR)*SIXP_TIMEOUT_SEC_FACTOR, where:

o C represents the number of cells per second scheduled to that
   neighbor
o PDR represents the average PDR of those cells
o SIXP_TIMEOUT_SEC_FACTOR is a security factor, a constant

[10](#). **Rule for Ordering Cells**

Cells are ordered slotOffset first, channelOffset second.

The following sequence is correctly ordered (each element represents
the [slottOffset,channelOffset] of a cell in the schedule):

[1,3],[1,4],[2,0],[5,3],[6,0],[6,3],[7,9]

11. Meaning of the Metadata Field

   The Metadata field is not used by MSF.

12. 6P Error Handling

   Section 6.2.4 of [I-D.ietf-6tisch-6top-protocol] lists the 6P Return
   Codes. Figure 1 lists the same error codes, and the behavior a node
   implementing MSF SHOULD follow.

```
        +-----------------+----------------------+
        | Code            | RECOMMENDED behavior |
        +-----------------+----------------------+
        | RC_SUCCESS      | nothing              |
        | RC_EOL          | nothing              |
        | RC_ERR          | quarantine           |
        | RC_RESET        | quarantine           |
        | RC_ERR_VERSION  | quarantine           |
        | RC_ERR_SFID     | quarantine           |
        | RC_ERR_SEQNUM   | clear                |
        | RC_ERR_CELLLIST | clear                |
        | RC_ERR_BUSY     | waitretry            |
        | RC_ERR_LOCKED   | waitretry            |
        +-----------------+----------------------+
```

          Figure 1: Recommended behavior for each 6P Error Code.

   The meaning of each behavior from Figure 1 is:

   nothing: Indicates that this Return Code is not an error. No error
      handling behavior is triggered.
   clear: Abort the 6P Transaction. Issue a 6P CLEAR command to that
      neighbor (this command may fail at the link layer). Remove all
      cells scheduled with that neighbor from the local schedule. Keep
      that node in the neighbor and routing tables.
   quarantine: Same behavior as for "clear". In addition, remove the
      node from the neighbor and routing tables. Place the node's
      identifier in a quarantine list for QUARANTINE_DURATION. When in
      quarantine, drop all frames received from that node.
   waitretry: Abort the 6P Transaction. Wait for a duration randomly
      and uniformly chosen in [WAITDURATION_MIN,WAITDURATION_MAX].
      Retry the same transaction.

13. Schedule Inconsistency Handling

   The behavior when schedule inconsistency is detected is explained in
   Figure 1, for 6P Return Code RC_ERR_SEQNUM.

14. MSF Constants

   Figure 2 lists MSF Constants and their RECOMMENDED values.

```
        +------------------------------+-------------------+
        | Name                         | RECOMMENDED value |
        +------------------------------+-------------------+
        | KA_PERIOD                    |        10 s        |
        | LIM_NUMCELLSUSED_HIGH        |        75 %        |
        | LIM_NUMCELLSUSED_LOW         |        25 %        |
        | HOUSEKEEPINGCOLLISION_PERIOD |         1 min      |
        | RELOCATE_PDRTHRES            |        50 %        |
        | SIXP_TIMEOUT_SEC_FACTOR      |         3 x        |
        | SLOTFRAME_LENGTH             |       101 slots    |
        | QUARANTINE_DURATION          |         5 min      |
        | WAITDURATION_MIN             |        30 s        |
        | WAITDURATION_MAX             |        60 s        |
        +------------------------------+-------------------+
```

           Figure 2: MSF Constants and their RECOMMENDED values.

15. MSF Statistics

   Figure 3 lists MSF Statistics and their RECOMMENDED width.

```
        +-----------------+-------------------+
        | Name            | RECOMMENDED width |
        +-----------------+-------------------+
        | NumCellsPassed  |       1 byte      |
        | NumCellsUsed    |       1 byte      |
        | NumTx           |       1 byte      |
        | NumTxAck        |       1 byte      |
        +-----------------+-------------------+
```

           Figure 3: MSF Statistics and their RECOMMENDED width.

16. Security Considerations

   MSF defines a series of "rules" for the node to follow. It triggers
   several actions, that are carried out by the protocols defined in the
   following specifications: the Minimal IPv6 over the TSCH Mode of IEEE
   802.15.4e (6TiSCH) Configuration [RFC8180], the 6TiSCH Operation
   Sublayer Protocol (6P) [I-D.ietf-6tisch-6top-protocol], and the
   Minimal Security Framework for 6TiSCH
   [I-D.ietf-6tisch-minimal-security]. In particular, MSF does not
   define a new protocol or packet format.

MSF relies entirely on the security mechanisms defined in the
specifications listed above.

17. IANA Considerations

17.1. MSF Scheduling Function Identifiers

This document adds the following number to the "6P Scheduling
Function Identifiers" sub-registry, part of the "IPv6 over the TSCH
mode of IEEE 802.15.4e (6TiSCH) parameters" registry, as defined by
[I-D.ietf-6tisch-6top-protocol]:

```
+----------------------+----------------------------+-------------+
|  SFID                | Name                       | Reference   |
+----------------------+----------------------------+-------------+
| IANA_6TISCH_SFID_MSF | Minimal Scheduling Function | RFCXXXX    |
|                      | (MSF)                      | (NOTE:this) |
+----------------------+----------------------------+-------------+
```

Figure 4: IETF IE Subtype '6P'.

18. References

18.1. Normative References

[I-D.ietf-6tisch-6top-protocol]
          Wang, Q., Vilajosana, X., and T. Watteyne, "6TiSCH
          Operation Sublayer Protocol (6P)", draft-ietf-6tisch-6top-
          protocol-12 (work in progress), June 2018.

[I-D.ietf-6tisch-minimal-security]
          Vucinic, M., Simon, J., Pister, K., and M. Richardson,
          "Minimal Security Framework for 6TiSCH", draft-ietf-
          6tisch-minimal-security-06 (work in progress), May 2018.

[I-D.richardson-6tisch-join-enhanced-beacon]
          Dujovne, D. and M. Richardson, "IEEE802.15.4 Informational
          Element encapsulation of 6tisch Join Information", draft-
          richardson-6tisch-join-enhanced-beacon-03 (work in
          progress), January 2018.

[IEEE802154-2015]
          IEEE standard for Information Technology, "IEEE Std
          802.15.4-2015 Standard for Low-Rate Wireless Personal Area
          Networks (WPANs)", December 2015.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997,
              <https://www.rfc-editor.org/info/rfc2119>.

   [RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J.,
              Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur,
              JP., and R. Alexander, "RPL: IPv6 Routing Protocol for
              Low-Power and Lossy Networks", RFC 6550,
              DOI 10.17487/RFC6550, March 2012,
              <https://www.rfc-editor.org/info/rfc6550>.

   [RFC7554] Watteyne, T., Ed., Palattella, M., and L. Grieco, "Using
              IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the
              Internet of Things (IoT): Problem Statement", RFC 7554,
              DOI 10.17487/RFC7554, May 2015,
              <https://www.rfc-editor.org/info/rfc7554>.

   [RFC8180] Vilajosana, X., Ed., Pister, K., and T. Watteyne, "Minimal
              IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH)
              Configuration", BCP 210, RFC 8180, DOI 10.17487/RFC8180,
              May 2017, <https://www.rfc-editor.org/info/rfc8180>.

18.2. Informative References

   [OpenWSN] Watteyne, T., Vilajosana, X., Kerkez, B., Chraim, F.,
              Weekly, K., Wang, Q., Glaser, S., and K. Pister, "OpenWSN:
              a Standards-Based Low-Power Wireless Development
              Environment", Transactions on Emerging Telecommunications
              Technologies , August 2012.

   [RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running
              Code: The Implementation Status Section", RFC 6982,
              DOI 10.17487/RFC6982, July 2013,
              <https://www.rfc-editor.org/info/rfc6982>.

   [SAX-DASFAA]
              Ramakrishna, M. and J. Zobel, "Performance in Practice of
              String Hashing Functions", DASFAA , 1997.

Appendix A. Contributors

   Beshr Al Nahas (Chalmers University, beshr@chalmers.se) and Olaf
   Landsiedel (Chalmers University, olafl@chalmers.se) contributed to
   the design and evaluation of autonomous unicast cells.

Appendix B. Implementation Status

   This section records the status of known implementations of the
   protocol defined by this specification at the time of posting of this
   Internet-Draft, and is based on a proposal described in [RFC6982].
   The description of implementations in this section is intended to
   assist the IETF in its decision processes in progressing drafts to
   RFCs. Please note that the listing of any individual implementation
   here does not imply endorsement by the IETF. Furthermore, no effort
   has been spent to verify the information presented here that was
   supplied by IETF contributors. This is not intended as, and must not
   be construed to be, a catalog of available implementations or their
   features. Readers are advised to note that other implementations may
   exist.

   According to [RFC6982], "this will allow reviewers and working groups
   to assign due consideration to documents that have the benefit of
   running code, which may serve as evidence of valuable experimentation
   and feedback that have made the implemented protocols more mature.
   It is up to the individual working groups to use this information as
   they see fit".

   OpenWSN: MSF is being implemented in the OpenWSN project [OpenWSN]
      under a BSD open-source license. The authors of this document are
      collaborating with the OpenWSN community to gather feedback about
      the status and performance of the protocols described in this
      document. Results from that discussion will appear in this
      section in future revision of this specification. More
      information about this implementation at http://www.openwsn.org/.
   6TiSCH simulator The 6TiSCH simulator is a Python-based high-level
      simulator on which MSF is being implemented. More information at
      https://bitbucket.org/6tisch/simulator/.

Appendix C. Performance Evaluation

   The performance of MSF may be published as companion documents to
   this specification, possibly under the form a applicability
   statements.

Appendix D. [TEMPORARY] Changelog

   o draft-chang-6tisch-msf-02

      * Added autonomous cell.
   o draft-chang-6tisch-msf-01

      * When neighbor is unreachable, sending a CLEAR command was a
         MUST, now a MAY.

        * Fixing 6P Timeout calculation.
        * Clearer text for "Handling Schedule Collisions" section.
        * Typos.
        * Input from Yasuyuki Tanaka's review (https://www.ietf.org/mail-archive/web/6tisch/current/msg05723.html).
    o draft-chang-6tisch-msf-00

        * Initial submission.

Authors' Addresses

    Tengfei Chang (editor)
    Inria
    2 rue Simone Iff
    Paris 75012
    France

    Email: tengfei.chang@inria.fr

    Malisa Vucinic
    University of Montenegro
    Dzordza Vasingtona bb
    Podgorica 81000
    Montenegro

    Email: malisav@ac.me

    Xavier Vilajosana
    Universitat Oberta de Catalunya
    156 Rambla Poblenou
    Barcelona, Catalonia 08018
    Spain

    Email: xvilajosana@uoc.edu

    Simon Duquennoy
    RISE SICS
    Isafjordsgatan 22
    164 29 Kista
    Sweden

    Email: simon.duquennoy@ri.se

Diego Dujovne (editor)
Universidad Diego Portales
Escuela de Informatica y Telecomunicaciones
Av. Ejercito 441
Santiago, Region Metropolitana
Chile

Phone: +56 (2) 676-8121
Email: diego.dujovne@mail.udp.cl