6TiSCH
Internet-Draft
Intended status: Informational
Expires: April 30, 2015

Q. Wang, Ed.
Univ. of Sci. and Tech. Beijing
X. Vilajosana
Universitat Oberta de Catalunya
T. Watteyne
Linear Technology
October 27, 2014

6TiSCH Operation Sublayer (6top) Interface
draft-ietf-6tisch-6top-interface-02

Abstract

This document defines a generic data model for the 6TiSCH Operation Sublayer (6top), using the YANG data modeling language. This data model can be used for future network management solutions defined by the 6TiSCH working group. This document also defines a list of commands for the internal use of the 6top sublayer and or to be used by implementers as an API guideline or basic specification.

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 April 30, 2015.

Copyright Notice

Table of Contents

1. Introduction

This document defines a generic data model for the 6TiSCH Operation Sublayer (6top), using the YANG data modeling language defined in [RFC6020]. This data model can be used for future network management solutions defined by the 6TiSCH working group. This document also defines a list commands internal to the 6top sublayer. This data model gives access to metrics (e.g. cell state), TSCH configuration and control procedures, and support for the different scheduling mechanisms described in [I-D.ietf-6tisch-architecture]. The 6top sublayer addresses the set of management information and functionalities described in [I-D.ietf-6tisch-tsch].

For example, network formation in a TSCH network is handled by the use of Enhanced Beacons (EB). EBs include information for joining nodes to be able to synchronize and set up an initial network topology. However, [IEEE802154e] does not specify how the period of EBs is configured, nor the rules for a node to select a particular node to join. 6top offers a set of commands so control mechanisms can be introduced on top of TSCH to configure nodes to join a specific node and obtain a unique 16-bit identifier from the network. Once a network is formed, 6top maintains the network's health, allowing for nodes to stay synchronized. It supplies mechanisms to manage each node's time source neighbor and configure the EB interval. Network

layers running on top of 6top take advantage of the TSCH MAC layer information so routing metrics, topological information, energy consumption and latency requirements can be adjusted to TSCH, and adapted to application requirements.

TSCH requires a mechanism to manage its schedule; 6top provides a set of commands for upper layers to set up specific schedules, either explicitly by detailing specific cell information, or by allowing 6top to establish a schedule given a bandwidth or latency requirement. 6top is designed to enable decentralized, centralized or hybrid scheduling solutions. 6top enables internal TSCH queuing configuration, size of buffers, packet priorities, transmission failure behavior, and defines mechanisms to encrypt and authenticate MAC slotframes.

As described in [morell04label], due to the slotted nature of a TSCH network, it is possible to use a label switched architecture on top of TSCH cells. As a cell belongs to a specific track, a label header is not needed at each packet; the input cell (or bundle) and the output cell (or bundle) uniquely identify the data flow. The 6top sublayer provides operations to manage the cell mappings.

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 [RFC2119].

3. 6TiSCH Operation Sublayer (6top) Overview

6top is a sublayer which is the next-higher layer for TSCH (Figure 1), as detailed in [I-D.ietf-6tisch-architecture]. 6top offers both management and data interfaces to an upper layer, and includes monitoring and statistics collection, both of which are configurable through its management interface. The detail of 6top-sublayer is described in [I-D.wang-6tisch-6top-sublayer]

Protocol Stack

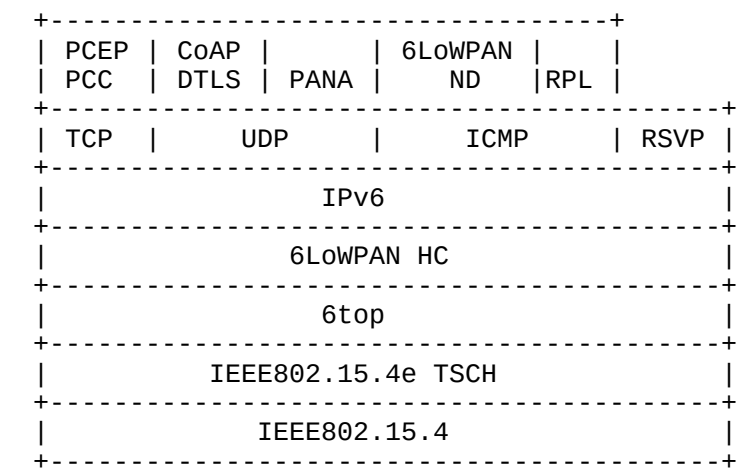

Figure 1

6top distinguishes between hard cells and soft cells. It therefore requires an extra flag to all cells in the TSCH schedule, as detailed in Section 3.1.

When a higher layer gives 6top a 6LoWPAN packet for transmission, 6top maps it to the appropriate outgoing priority-based queue, as detailed in Section 3.2.

Section 4 contains a generic data model for the 6top sublayer, described in the YANG data modeling language.

The commands of the management and data interfaces are listed in Section 5. This set of commands is designed to support decentralized, centralized and hybrid scheduling solutions.

3.1. Cell Model

[IEEE802154e] defines a set of options attached to each cell. A cell can be a Transmit cell, a Receive cell, a Shared cell or a Timekeeping cell. These options are not exclusive, as a cell can be qualified with more than one of them. The MLME-SET-LINK.request command defined in [IEEE802154e] uses a linkOptions bitmap to specify the options of a cell. Acceptable values are:

b0 = Transmit

b1 = Receive

b2 = Shared

b3 = Timekeeping

b4-b7 = Reserved

Only Transmit cells can also be marked as Shared cells. When the shared bit is set, a back-off procedure is applied to handle collisions. Shared behavior does not apply to Receive cells.

6top allows an upper layer to schedule a cell at a specific slotOffset and channelOffset, in a specific slotframe.

In addition, 6top allows an upper layer to schedule a certain amount of bandwidth to a neighbor, without having to specify the exact slotOffset(s) and channelOffset(s). Once bandwidth is reserved, 6top is in charge of ensuring that this requirement is continuously satisfied. 6top dynamically reallocates cells if needed, and over-provisions if required.

6top allows an upper layer to associate a cell with a specific track by using a TrackID. A TrackID is a tuple (TrackOwnerAddr,InstanceID), where TrackOwnerAddr is the address of the node which initializes the process of creating the track, i.e., the owner of the track; and InstanceID is an instance identifier given by the owner of the track. InstanceID comes from upper layer; InstanceID could for example be the local instance ID defined in RPL.

If the TrackID is set to (0,0), the cell can be used by the best-effort QoS configuration or as a Shared cell. If the TrackID is not set to (0,0), i.e., the cell belongs to a specific track, the cell MUST not be set as Shared cell.

6top allows an upper layer to ask a node to manage a portion of a slotframe, which is named as chunk. Chunks can be delegated explicitly by the PCE to a node, or claimed automatically by any node that participates to the distributed cell scheduling process. The resource in a chunk can be appropriated by the node, i.e. the owner of the chunk.

Given this mechanism, 6top defines hard cells (which have been requested specifically) and soft cells (which can be reallocated dynamically). The hard/soft flag is introduced by the 6top sublayer named as CellType, 0: soft cell, 1: hard cell. This option is mandatory; all cells are either hard or soft.

3.1.1. hard cells

A hard cell is a cell that cannot be dynamically reallocated by 6top. The CellType MUST be set to 1. The cell is installed by 6top given specific slotframe ID, slotOffset, and channelOffset.

3.1.2. soft cells

A soft cell is a cell that can be reallocated by 6top dynamically. The CellType MUST be set to 0. This cell is installed by 6top given a specific bandwidth requirement. Soft cells are installed through the soft cell negotiation procedure described in [I-D.wang-6tisch-6top-sublayer].

3.2. Data Transfer Model

Once a TSCH schedule is established, 6top is responsible for feeding the data from the upper layer into TSCH. This section describes how 6top shapes data from the upper layer (e.g., RPL, 6LoWPAN), and feeds it to TSCH. Since 6top is a sublayer between TSCH and 6LoWPAN, the properties associated with a packet/fragment from the upper layer includes the next hop neighbor (DestAddr) and expected sending priority of the packet (Priority), and/or TrackID(s). The output to TSCH is the fragment corresponding to the next active cell in the TSCH schedule.

6top Data Transfer Model

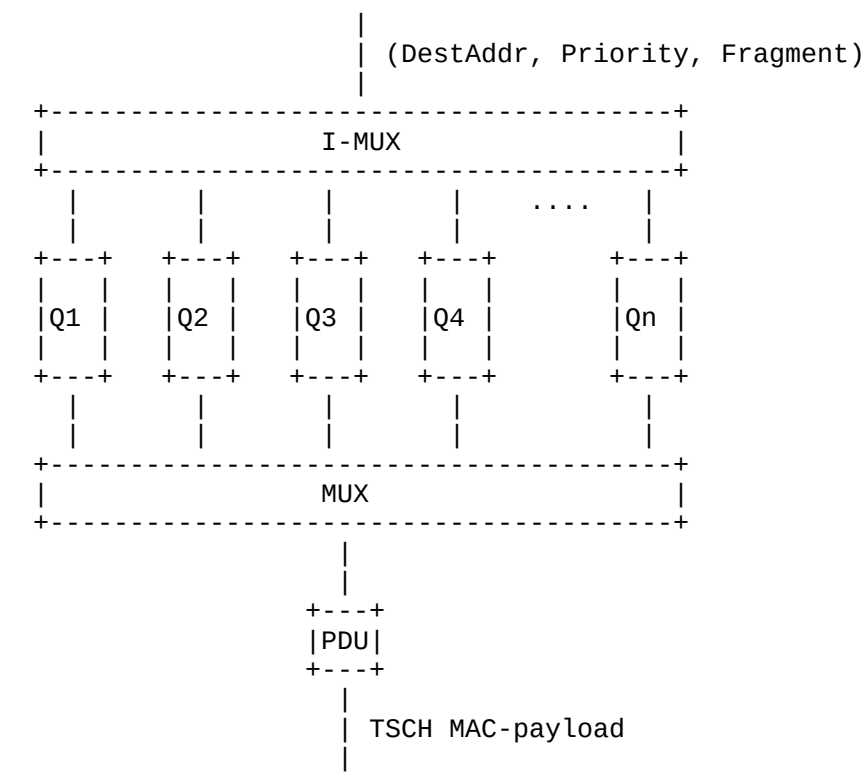

Figure 2

In Figure 2, Qi represents a queue, which is either broadcast or unicast, and has an assigned priority. The number of queues is configurable. The relationship between queues and tracks is configurable. For example, for a given queue, only one specific track can be used, all of the tracks can be used, or a subset of the tracks can be used.

When 6top receives a packet to transmit through a Send.data command (Section 5), the I-MUX module selects a queue in which to insert it. If the packet's destination address is a unicast (resp. broadcast) address, it will be inserted into a unicast (resp. broadcast) queue.

The MUX module is invoked at each scheduled transmit cell by TSCH. When invoked, the MUX module goes through the queues, looking for the best matching frame to send. If it finds a frame, it hands it over to TSCH for transmission. If the next active cell is a broadcast cell, it selects a fragment only from broadcast queues.

How the MUX module selects the best frame is configurable. The following rules are a typical example:

> The frame's layer 2 destination address MUST match the neighbor address associated with the transmit cell.
>
> If the transmit cell is associated with a specific track, the frames in the queue corresponding to the TrackID have the highest priority.
>
> If the transmit cell is not associated with a specific track, i.e., TrackID=(0,0), frames from a queue with a higher priority MUST be sent before frames from a queue with a lower priority.

Further rules can be configured to satisfy specific QoS requirements.

4. Generic Data Model

This section presents the generic data model of the 6top sublayer, using the YANG data modeling langage. This data model can be used for future network management solutions defined by the 6TiSCH working group. The data model consists of the MIB (management information base) defined in 6top, and part of the PIB (personal area network information base) defined in [IEEE802154e] and [IEEE802154].

4.1. YANG model of the 6top MIB

```
list CellList {
   key "CellID";
   description
   "List of scheduled cells of a node with all of its neighbors,
   in all of its slotframes.";

   leaf CellID {
      type uint16;
      description
      "Equal to Linkhandle in the linkTable of TSCH";
      reference
      "IEEE802154e";
   }
   leaf SlotframeID {
      type uint8;
      description
      "SlotframeID, one in SlotframeList, indicates the slotframe
      the cell belongs to.";
      reference
      "IEEE802154e";
   }
```

```
         leaf SlotOffset {
            type uint16;
            description
            "Defined in IEEE802154e.";
            reference
            "IEEE802154e";
         }
         leaf ChannelOffset {
            type uint16;
            description
            "Defined in IEEE802154e.";
            reference
            "IEEE802154e";
         }
         leaf LinkOption {
            type bits {
               bit Transmit {
                  position 0;
               }
               bit Receive {
                  position 1;
               }
               bit Share {
                  position 2;
               }
               bit Timekeeping {
                  position 3;
               }
               bit Reserved1 {
                  position 4;
               }
               bit Reserved2 {
                  position 5;
               }
               bit Reserved3 {
                  position 6;
               }
               bit Reserved4 {
                  position 7;
               }
            }
            description
            "Defined in IEEE802154e.";
            reference
            "IEEE802154e";
         }
         leaf LinkType {
            type enumeration {
```

```
               enum NORMAL;
               enum ADVERTISING;
            }
            description
            "Defined in IEEE802154";
            reference
            "IEEE802154";
         }
         leaf CellType {
            type enumeration {
               enum SOFT;
               enum HARD;
            }
            description
            "Defined in 6top";
         }
         leaf TargetNodeAddress {
            type uint64;
            description
            "Defined by 6top, but being constrained by TSCH
            macNodeAddress size, 2-octets. If using TSCH as MAC,
            higher 6-octets should be filled with 0, and lowest
            2-octets is neighbor address";
         }
         leaf TrackID {
            type uint16;
            description
            "A TrackID is one in the TrackList, pointing to a tuple
            (TrackOwnerAddr,InstanceID) , where TrackOwnerAddr is the
            address of the node which initializes the process of
            creating the track, i.e., the owner of the track; and
            InstanceID is an instance identifier given by the owner of
            the track.";
         }
         container Statistic {
            leaf NumOfStatistic {
               type uint8;
               description
               "Number of statistics collected on the cell";
            }
            list MeasureList {
               key "StatisticsMetricsID";
               leaf StatisticsMetricsID{
                  type uint16;
                  description
                  "An index of StatisticsMetricList, which defines how
                  to collect data and get the statistics value";
               }
```

```
               leaf StatisticsValue{
                  type uint16;
                  config false;
                  description
                  "updated by 6top according to the statistics method
                  specified by StatisticsMetricsID";
               }
            }
         }
      }

   list SlotframeList {
      key "SlotframeID";
      description
      "List of all of the slotframes used by the node.";

      leaf SlotframeID {
         type uint8;
         description
         "Equal to SlotframeHandle defined in TSCH";
         reference
         "IEEE802154e";
      }
      leaf NumOfSlots {
         type uint16;
         description
         "indicates how many timeslots in the slotframe";
      }
   }

      list MonitoringStatusList {
         key "MonitoringStatusID";
         description
         "List of the monitoring configuration and results per
         slotframe and neighbor. Basically, it is used for Monitoring
         Function of 6top to re-allocate softcells or initial the
         softcell negotiation process to increase/decrease number of
         softcells. Upper layer can use it also.";

         leaf MonitoringStatusID {
            type uint16;
         }
         leaf SlotframeID {
            type uint8;
            description
            "SlotframeID, one in SlotframeList, indicates the slotframe
            being monitored";
            reference
```

```
            "IEEE802154e";
          }
          leaf TargetNodeAddress {
             type uint64;
             description
             "Defined by 6top, but being constrained by TSCH
             macNodeAddress size, 2-octets. If using TSCH as MAC,
             higher 6-octets should be filled with 0, and lowest
             2-octets is neighbor address. It indicates the communication
             link being mornitored";
          }
          leaf EnforcePolicy {
             type enumeration {
                enum DISABLE;
                enum BESTEFFORT;
                enum STRICT;
                enum OVERPROVISION;
             }
             description
             "Currently enforced QoS policy. DISABLE-no QoS;
             BESTEFFORT- best effort policy is used; STRICT- Strict
             Priority Queueing; OVERPROVISION- cell overprovision";
          }
          leaf AllocatedHard {
             type uint16;
             config false;
             description
             "Number of hard cells allocated";
          }
          leaf AllocatedSoft {
             type uint16;
             config false;
             description
             "Number of soft cells allocated";
          }
          leaf OverProvision {
             type uint16;
             config false;
             description
             "Overprovisioned cells. 0 if EnforcePolicy is
             DISABLE";
          }
          leaf QoS {
             type uint16;
             config false;
             description
             "Current QoS including overprovisioned cells, i.e. the
             bandwidth obtained including the overprovisioned cells.";
```

```
         }
         leaf NQoS {
            type uint16;
            config false;
            description
            "Real QoS without over provisioned cells, i.e. the actual
            bandwidth without taking into account the overprovisioned
            cells.";
         }
      }

list StatisticsMetricsList {
   key "StatisticsMetricsID";
   description
   "List of Statistics Metrics used in the node. Statistics can be set and queri
ed.";

   leaf StatisticsMetricsID {
      type uint16;
   }
   leaf SlotframeID {
      type uint16;
      description
      "SlotframeID, one in SlotframeList, specifies the slotframe to
      which the statistics metrics applies to. If empty, applies to
      all slotframes";
      reference
      "IEEE802154e";
   }
   leaf SlotOffset {
      type uint16;
      description
      "Specific slotOffset to which the statistics metrics applies
      to. If empty, applies to all timeslots";
      reference
      "IEEE802154e";
   }
   leaf ChannelOffset {
      type uint8;
      description
      "Specific channelOffset to which the statistics metrics applies
      to. If empty, applies to all channels";
      reference
      "IEEE802154e";
   }
   leaf TargetNodeAddress {
      type uint64;
      description
      "Specific neighbor nodes to which the statistics metrics
```

```
      applies to. If empty, applies to all neighbor nodes.";
   }
   leaf Metrics {
      type enumeration {
         enum macCounterOctets
         enum macRetryCount
         enum macMultipleRetryCount
         enum macTXFailCount
         enum macTXSuccessCount
         enum macFCSErrorCount
         enum macSecurityFailure
         enum macDuplicateFrameCount
         enum macRXSuccessCount
         enum macNACKcount
         enum PDR;
         enum ETX;
         enum RSSI;
         enum LQI;
      }
      description
      "The metric to be monitored. Include those provided by underlying IEEE 802
.15.4e TSCH -- see table 4i (2012). PDR,ETX,RSSI,LQI are maintained by 6top. ";
   }
   leaf Window {
      type uint16;
      description
      "measurement period, in Number of the slotframes. If not specified the met
rics are updated continuously in time. If a period is specified the metric value
s are given for the specified time-window";
   }
   leaf Enable {
      type enumeration {
         enum DISABLE;
         enum ENABLE;
      }
      description
      "indicates the StatisticsMetric is active or not";
   }
}
```

```
      list EBList {
         key "EbID";
         description
         "List of information related with the EBs used by the node";

         leaf EbID {
            type uint8;
         }

         leaf CellID {
            type uint16;
            description
            "CellID, one in CellList, indicates the cell used to send
            EB";
         }
         leaf SlotframeId{
               type uint8;
               description
               "SlotframeID, one in SlotframeList, indicates the
               slotframe to which the EB is send";
         }
         leaf Period {
            type uint16;
            description
            "The EBs period, in seconds, indicates the interval between
            two EB sends";
         }
         leaf Expiration {
            type enumeration {
               enum NEVERSTOP;
               enum EXPIRATION;
            }
            description
            "NEVERSTOP- the period of the EB never stops; EXPIRATION-
            when the Period arrives, the EB will stop.";
         }
         leaf Priority {
            type uint8;
            description
            "The joining priority model that will be used for
            advertisements. Joining priority MAY be for example
            SAME_AS_PARENT, RANDOM, BEST_PARENT+1 or
            DAGRANK(rank).";
         }
      }
```

```
   container TimeSource {
      description
      "specify the timesource selection policy and some relative
      statistics.";

      leaf policy {
         type enumeration {
            enum ALLPARENT;
            enum BESTCONNECTED;
            enum LOWESTJOINPRIORITY;
         }
         description
         "indicates the policy to choose timesource. ALLPARENT- choose
         from all parents; BESTCONNECTED- choose the best-connected
         node; LOWESTJOINPRIORITY- choose the node with lowest priority
         in its EB.";
      }
      leaf TargetNodeAddress {
         type uint64;
         description
         "Address of the time source neighbor";
      }
      leaf MinTimeCorrection {
         type uint16;
         config false;
         description
         "measured in microsecond";
      }
      leaf MaxTimeCorrection {
         type uint16;
         config false;
         description
         "measured in microsecond";
      }
      leaf AveTimeCorrection {
         type uint16;
         config false;
         description
         "measured and computed in microsecond";
      }
   }
```

```
   typedef asntype {
      description
         "The type to store ASN. String of 5 bytes";
      type string {
         length "0..5";
      }
   }

   list NeighborList {
      key "TargetNodeAddress";
      description
      "statistics per communication link.";

      leaf TargetNodeAddress {
         type uint64;
         description
         "Address of the time source neighbor";
      }
      leaf RSSI {
         type uint8;
         config false;
         description
         "The received signal strength";
      }
      leaf LinkQuality {
         type uint8;
         config false;
         description
         "The LQI metric";
      }
      leaf ASN {
         type asntype;
         config false;
         description
         "The 5 ASN bytes, indicates the most recent timeslot when a
         packet from the neighbor was received";
      }
   }

   list QueueList {
      key "QueueId";
      description
      "List of Queues, including configuration and statistics.";

      leaf QueueId {
         type uint8;
         description
         "Queue Identifier";
```

```
      }
      leaf TxqLength {
         type uint8;
         description
         "The TX queue length in number of packets";
      }
      leaf RxqLength {
         type uint8;
         description
         "The RX queue length in number of packets";
      }
      leaf NumrTx {
         type uint8;
         description
         "Number of allowed retransmissions.";
      }
      leaf Age {
         type uint16;
         description
         "In seconds. Discard packet according to its age
          on the queue. 0 if no discards are allowed.";
      }
      leaf RTXbackoff {
         type uint8;
         description
         "retransmission backoff in number of slotframes.
          0 if next available timeslot wants to be used.";
      }
      leaf StatsWindow {
         type uint16;
         description
         "In second, window of time used to compute stats.";
      }
      leaf QueuePriority {
         type uint8;
         description
         "The priority for this queue.";
      }
      list TrackIds {
         key "TrackID";
         leaf TrackID{
            type uint16;
            description
            "The TrackID, one in TrackList, indicates the Track is
            associated with the Queue.";
         }
      }
      leaf MinLenTXQueue {
```

```
         type uint8;
         config false;
         description
         "Statistics, lowest TX queue length registered in the window.";
      }
      leaf MaxLenTXQueue {
         type uint8;
         config false;
         description
         "Statistics, largest TX queue length registered in the
         window.";
      }
      leaf AvgLenTXQueue {
         type uint8;
         config false;
         description
         "Statistics, avg TX queue length registered in the window.";
      }
      leaf MinLenRXQueue {
         type uint8;
         config false;
         description
         "Statistics, lowest RX queue length registered in the window.";
      }
      leaf MaxLenRXQueue {
         type uint8;
         config false;
         description
         "Statistics, largest RX queue len registered in the window.";
      }
      leaf AvgLenRXQueue {
         type uint8;
         config false;
         description
         "Statistics, avg RX queue length registered in the window.";
      }
      leaf MinRetransmissions {
         type uint8;
         config false;
         description
         "Statistics, lowest number of retransmissions registered in
         the window.";
      }
      leaf MaxRetransmissions {
         type uint8;
         config false;
         description
         "Statistics, largest number of retransmissions registered
```

```
          in the window.";
       }
       leaf AvgRetransmissions {
          type uint8;
          config false;
          description
          "Statistics, average number of retransmissions registered
           in the window.";
       }
       leaf MinPacketAge {
          type uint16;
          config false;
          description
          "Statistics, in seconds, minimum time a packet stayed in
           the queue during the observed window.";
       }
       leaf MaxPacketAge {
          type uint16;
          config false;
          description
          "Statistics, in seconds, maximum time a packet stayed
           in the queue during the observed window.";
       }
       leaf AvgPacketAge {
          type uint16;
          config false;
          description
          "Statistics, in seconds, average time a packet stayed in
           the queue during the observed window.";
       }
       leaf MinBackoff {
          type uint8;
          config false;
          description
          "Statistics, in number of slotframes, minimum Backoff
           for a packet in the queue during the observed window.";
       }
       leaf MaxBackoff {
          type uint8;
          config false;
          description
          "Statistics, in number of slotframes, maximum Backoff
           for a packet in the queue during the observed window.";
       }
       leaf AvgBackoff {
          type uint8;
          config false;
          description
```

```
         "Statistics, in number of slotframes, average Backoff
          for a packet in the queue during the observed window.";
      }
   }

   list LabelSwitchList {
      key "LabelSwitchID";
      description
      "List of Label switch' configuration on the node";

      leaf LabelSwitchID {
         type uint16;
      }
      list InputCellIds {
         key "CellID";
         leaf CellID{
            type uint16;
            description
            "The CellID, indicates the Rx cell on which the packet will
            come in.";
         }
      }
      list OutputCellIds {
         key "CellID";
         leaf CellID{
            type uint16;
            description
            "The CellID, indicates the Tx cell on which the received
            packet should be sent out.";
         }
      }
      leaf LoadBalancingPolicy {
         type enumeration {
            enum ROUNDROBIN;
            enum OTHER;
         }
         description
         "The load-balancing policy. ROUNDROBIN- Round robin algorithm
         is used for forwarding scheduling.";
      }
   }
```

```
   list TrackList {
      key "TrackId";
      description
      "List of the tracks through the node.";

      leaf TrackId {
         type uint16;
         description
         "Track Identifier, named locally. It is used to refer to the
         tuple (TrackOwnerAddr, InstanceID).";
      }
      leaf TrackOwnerAddr {
         type uint64;
         description
         "The address of the node which initializes the process of
         creating the track, i.e., the owner of the track;";
      }
      leaf InstanceID {
         type uint16;
         description
         "InstanceID is an instance identifier given by the owner of
         the track. InstanceID comes from upper layer; InstanceID could
         for example be the local instance ID defined in RPL.";
      }
   }
```

```
         list ChunkList {
            key "ChunkId";
            description
            "List of the chunks assigned to the node.";

            leaf ChunkId{
               type uint16;
               description
               "The identifier of a chunk";
            }
            leaf SlotframeId{
               type uint8;
               description
               "SlotframeID, one in SlotframeList, indicates the
               slotframe to which the chunk belongs";
            }
            leaf SlotBase {
               type uint16;
               description
               "the base slotOffset of the chunk in the slotframe";
            }
            leaf SlotStep {
               type uint8;
               description
               "the slot incremental of the chunk";
            }
            leaf ChannelBase {
               type uint8;
               description
               "the base channelOffset of the chunk";
            }
            leaf ChannelStep {
               type uint8;
               description
               "the channel incremental of the chunk";
            }
            leaf ChunkSize {
               type uint8;
               description
               "the number of cells in the chunk. The chunk is the set
               of (slotOffset(i), channelOffset(i)),
               i=0..Chunksize-1,
               slotOffset(i)= (slotBase + i * slotStep) % slotframeLen,
               channelOffset(i) = (channelBase + i * channelStep) % 16";
            }
         }
```

```
         list ChunkCellList {
            key "SlotOffset ChannelOffset";
            description
            "List of all of the cells assigned to the node via the
            assignment of chunks.";

            leaf SlotOffset{
               type uint16;
               description
               "The slotoffset of a cell which belongs to a Chunk";
            }
            leaf ChannelOffset{
               type uint16;
               description
               "The channeloffset of a cell which belongs to a chunk.";
            }
            leaf ChunkId {
               type uint16;
               description
               "Identifier of the chunk the cell belongs to";
            }
            leaf CellID{
               type uint16;
               description
               "Initial value of CellID is 0xFFFF. When the cell is
               scheduled, the value of CellID is same as that in
               CellList";
            }
            leaf ChunkCellStatus {
               type enumeration {
                  enum UNSCHEDULED;
                  enum SCHEDULED;
               }
            }
         }
```

4.2. YANG model of the IEEE802.15.4 PIB

This section describes the YANG model of the part of PIB ([IEEE802154] and [IEEE802154e]) used by 6top, such as security related attributes, TSCH related attributes. This part of data will be accessed through the MLME-GET and MLME-SET primitive [IEEE802154] directly, instead of using 6top comannds.

TODO the security related attributes will be added after 6TiSCH WG has consensus on the security scheme of 6top

```
         container TSCHSpecificPIBAttributes {
```

```
                description
                "TSCH specific MAC PIB attributes.";
                reference
                "table 52b in IEEE802.15.4e-2012.";

                leaf macMinBE {
                   type uint8;
                   description
                   "defined in Table 52b of IEEE802.15.4e-2012,
                   The minimum value of the backoff exponent (BE) in the
                   CSMA-CA algorithm or the TSCH-CA algorithm. default:
                   3-CSMA-CA, 1-TSCH-CA";
                }
                leaf macMaxBE {
                   type uint8;
                   description
                   "defined in Table 52b of IEEE802.15.4e-2012,
                   The maximum value of the backoff exponent (BE) in the
                   CSMA-CA algorithm or the TSCH-CA algorithm. default:
                   5-CSMA-CA, 7-TSCH-CA";
                }
                leaf macDisconnectTime {
                   type uint16;
                   description
                   "defined in Table 52b of IEEE802.15.4e-2012,
                   Time (in Timeslots) to send out Disassociate frames
                   before disconnecting, default: 0x00ff";
                }
                leaf macJoinPriority {
                   type uint8;
                   description
                   "defined in Table 52b of IEEE802.15.4e-2012,
                   The lowest join priority from the TSCH Synchronization
                   IE in an Enhanced beacon, default: 1";
                }
                leaf macASN {
                   type asntype;
                   description
                   "defined in Table 52b of IEEE802.15.4e-2012,
                   The Absolute Slot Number, i.e., the number of slots
                   that ha elapsed since the start of the network.";
                }
                leaf macNoHLBuffers {
                   type enumeration {
                      enum TRUE;
                      enum FALSE;
                   }
                   description
```

```
                "defined in Table 52b of IEEE802.15.4e-2012,
                If the value is TRUE, the higher layer receiving the
                frame payload cannot buffer it, and the device should
                acknowledge frames with a NACK; If FALSE, the higher
                layer can accept the frame payload. default: FALSE";
             }
          }

          list TSCHmacTimeslotTemplate {
             key "macTimeslotTemplateId";
             description
             "List of all timeslot templates used in the node.";
             reference
             "table 52e in IEEE802.15.4e-2012.";

             leaf macTimeslotTemplateId {
                type uint8;
                description
                "defined in Table 52e of IEEE802.15.4e-2012.
                Identifier of Timeslot Template. default: 0";
             }
             leaf macTsCCAOffset {
                type uint16;
                description
                "The time between the beginning of timeslot and start
                of CCA operation, in microsecond. default: 1800";
             }
             leaf macTsCCA {
                type uint16;
                description
                "Duration of CCA, in microsecond. default: 128";
             }
             leaf macTsTxOffset {
                type uint16;
                description
                "The time between the beginning of the timeslot and
                the start of frame transmission, in microsecond.
                default: 2120";
             }
             leaf macTsRxOffset {
                type uint16;
                description
                "Beginning of the timeslot to when the receiver shall
                be listening, in microsecond. default: 1120";
             }
             leaf macTsRxAckDelay {
                type uint16;
                description
```

```
                "End of frame to when the transmitter shall listen for
                Acknowledgment, in microsecond. default: 800";
             }
             leaf macTsTxAckDelay {
                type uint16;
                description
                "End of frame to start of Acknowledgment, in
                microsecond.
                default: 1000";
             }
             leaf macTsRxWait {
                type uint16;
                description
                "The time to wait for start of frame, in microsecond.
                default: 2200";
             }
             leaf macTsAckWait {
                type uint16;
                description
                "The minimum time to wait for start of an
                Acknowledgment, in microsecond. default: 400";
             }
             leaf macTsRxTx {
                type uint16;
                description
                "Transmit to Receive turnaround, in microsecond.
                default: 192";
             }
             leaf macTsMaxAck {
                type uint16;
                description
                "Transmission time to send Acknowledgment,in
                microsecond. default: 2400";
             }
             leaf macTsMaxTx {
                type uint16;
                description
                "Transmission time to send the maximum length frame,
                in microsecond. default: 4256";
             }
             leaf macTsTimeslotLength {
                type uint16;
                description
                "The total length of the timeslot including any unused
                time after frame transmission and Acknowledgment,
                in microsecond. default: 10000";
             }
          }
```

```
            list TSCHHoppingSequence {
               key "macHoppingSequenceID";
               description
               "List of all channel hopping sequences used in the
               nodes";
               reference
               "Table 52f of IEEE802.15.4e-2012";

               leaf macHoppingSequenceID {
                  type uint8;
                  description
                  "defined in Table 52f of IEEE802.15.4e-2012.
                  Each hopping sequence has a unique ID. default: 0";
               }
               leaf macChannelPage {
                  type uint8;
                  description
                  "Corresponds to the 5 MSBs (b27, ..., b31) of a row
                  in phyChannelsSupported. Note this may not correspond
                  to the current channelPage in use.";
               }
               leaf macNumberOfChannels {
                  type uint16;
                  description
                  "Number of channels supported by the PHY on this
                  channelPage.";
               }
               leaf macPhyConfiguration {
                  type uint32;
                  description
                  "For channel pages 0 to 6, the 27 LSBs(b0, b1, ...,
                  b26) indicate the status (1 = to be used, 0 = not to
                  be used) for each of the up to 27 valid channels
                  available to the PHY. For pages 7 and 8, the 27 LSBs
                  indicate the configuration of the PHY, and the channel
                  list is contained in the extendedBitmap.";
               }
               leaf macExtendedBitmap {
                  type uint64;
                  description
                  "For pages 7 and 8, a bitmap of numberOfChannels bits,
                  where bk shall indicate the status of channel k for
                  each of the up to numberOfChannels valid channels
                  supported by that channel page and phyConfiguration.
                  Otherwise field is empty.";
               }
               leaf macHoppingSequenceLength {
                  type uint16;
```

```
                description
                "The number of channels in the Hopping Sequence.
                Does not necessarily equal numberOfChannels.";
             }
             list macHoppingSequenceList {
                key "HoppingChannelID";
                leaf HoppingChannelID {
                   type uint16;
                   description
                   "channels to be hopped over";
                }
             }
             leaf macCurrentHop {
                type uint16;
                config false;
                description
                "Index of the current position in the hopping sequence
                list.";
             }
          }
```

5. Commands

6top provides a set of commands as the interface with the higher layer. Most of these commands are related to the management of slotframes, cells and scheduling information. 6top also provides an interface allowing an upper layer to retrieve status information and statistics. The command set aims to facilitate 6top implementation by describing the main operations that higher layers may use to interact with 6top. The listed commands aim at providing semantics to manipulate 6top MIB, IEEE802.15.4 PIB and IEEE802.15.4e PIB programmatically.

CREATE.hardcell: Creates one or more hard cells in the schedule. Fails if the cell already exists. A cell is uniquely identified by the tuple (slotframe ID, slotOffset, channelOffset). 6top schedules the cell and marks it as a hard cell, indicating that it cannot reschedule this cell. The return value is CellID and the created cell is also filled in CellList(Section 4.1).

CREATE.softcell: To create soft cell(s). 6top is responsible for picking the exact slotOffset and channelOffset in the schedule, and ensure that the target node chooses the same cell and TrackID. 6top marks these cells as soft cell, indicating that it will continuously monitor their performance and reschedule if needed. The return value is CellID, and the created cell is also filled in CellList (Section 4.1).

READ.cell: Given a (slotframe ID, slotOffset, channelOffset), retrieves the cell information. A read command can be issued for any cell, hard or soft. 6top gets cell information from CellList (Section 4.1).

UPDATE.cell: Update a hard cell, i.e., re-allocate it to a different slotOffset and/or channelOffset. Fails if the cell does not exist. CellList (Section 4.1) will be modified.

DELETE.hardcell: To remove a hard cell. This removes the hard cell from the node's schedule, from CellList (Section 4.1).

DELETE.softcell: To remove a (number of) soft cell(s). This command leads the pair of nodes figure out the specific cell(s) to be removed. After that, the cell(s) will be removed from the CellLists (Section 4.1) on both sides.

REALLOCATE.softcell: To force a re-allocation of a soft cell. The reallocated cell will be installed in a different slotOffset, channelOffset but slotframe and TrackID remain the same. Hard cells MUST NOT be reallocated. This command will result in the modificaition of CellLists (Section 4.1) on both sides.

CREATE.slotframe: Creates a new slotframe. Adds a entry to the SlotframeList (Section 4.1).

READ.slotframe: Returns the information of a slotframe given its slotframeID from SlotframeList (Section 4.1).

UPDATE.slotframe: Change the number of timeslots in a slotframe given its slotframeID in SlotframeList (Section 4.1).

DELETE.slotframe: Deletes a slotframe, remove it from SlotframeList (Section 4.1).

CONFIGURE.monitoring: Configures the level of QoS the Monitoring process MUST enforce, i.e. config MonitoringStatusList (Section 4.1).

READ.monitoring: Reads the current Monitoring status from MonitoringStatusList (Section 4.1).

CONFIGURE.statistics: Configures the statistics process in StatisticsMetricsList(Section 4.1). The CONFIGURE.statistics enables flexible configuration and supports empty parameters that will force 6top to conduct statistics on all members of that dimension. For example, if ChannelOffset is empty and metric is

set as PDR, then, 6top will conduct the statistics of PDR on all of channels.

READ.statistics: Reads a metric for the specified dimension. Information is aggregated according to the parameters from CellList (Section 4.1).

RESET.statistics: Resets the gathered statistics in CellList (Section 4.1).

CONFIGURE.eb: Configures EBs, i.e. configures EBlist (Section 4.1).

READ.eb: Reads the EBs configuration from EBList (Section 4.1).

CONFIGURE.timesource: Configures the Time Source Neighbor selection process, i.e. configure TimeSource (Section 4.1).

READ.timesource: Retrieves information about the time source neighbors of that node from TimeSource (Section 4.1).

CREATE.neighbor: Creates an entry for a neighbor in the neighbor table, i.e. NeighborList (Section 4.1).

READ.all.neighbor: Returns the list of neighbors of that node according to NeighborList (Section 4.1).

READ.neighbor: Returns the information of a specific neighbor of that node specified by its neighbor address according to NeighborList (Section 4.1).

UPDATE.neighbor: Updates the last status for a given TargetNodeAddress in the NeighborList (Section 4.1).

DELETE.neighbor: Deletes a neighbor given its address from NeighborList (Section 4.1).

CREATE.queue: Creates and Configures a queue in QueueList (Section 4.1).

READ.queue: Reads the queue configuration for given QueueId from QueueList (Section 4.1).

READ.queue.stats: For a given QueueId, reads the queue statistics information from the QueueList (Section 4.1).

UPDATE.queue: For a given QueueId, update its configuration in the QueueList (Section 4.1).

DELETE.queue: Deletes a Queue for a given QueueId from the QueueList (Section 4.1).

LabelSwitching.map: Maps an input cell or a bundle of input cells to an output cell or a bundle of output cells, i.e. adds a entry to the LabelSwitchList (Section 4.1).

LabelSwitching.unmap: Unmap one input cell or a bundle of input cells to an output cell or a bundle of output cells, i.e. modifies the LabelSwitchList (Section 4.1).

CREATE.chunk: Creates a chunk which consists of one or more unscheduled cells, i.e. add an entry to the ChunkList (Section 4.1).

READ.chunk: Returns the information of a chunk given its ChunkID from ChunkList (Section 4.1).

DELETE.chunk: For given ChunkId, removes a chunk from the ChunkList (Section 4.1), which also causes all of the scheduled cells in the chunk to be deleted from the TSCH schedule and CellList (Section 4.1).

CREATE.hardcell.fromchunk: Creates one or more hard cells from a chunk. 6top schedules the cell and marks it as a hard cell, indicating that it cannot reschedule this cell. The cell will be added into the CellList (Section 4.1). In addition, 6top will change the attributes corresponding to the cell in the ChunkCellList (Section 4.1), i.e. its CellID is changed to the same CellID in the CellList, and its Status is changed to SCHEDULED.

READ.chunkcell: Returns the information of all cells in a chunk given its ChunkID from ChunkCellList (Section 4.1).

DELETE.hardcell.fromchunk: To remove a hard cell which comes from a chunk. This removes the hard cell from the node’s schedule and CellList (Section 4.1). In addition, it changes the attributes corresponding to the cell in the ChunkCellList (Section 4.1), i.e. its CellID is changed back to 0xFFFF, and its Status is changed to UNSCHEDULED.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2. Informative References

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[I-D.ietf-6tisch-tsch] Watteyne, T., Palattella, M., and L. Grieco, "Using IEEE802.15.4e TSCH in an IoT context: Overview, Problem Statement and Goals", draft-ietf-6tisch-tsch-02 (work in progress), October 2014.

[I-D.ietf-6tisch-architecture] Thubert, P., Watteyne, T., and R. Assimiti, "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4e", draft-ietf-6tisch-architecture-03 (work in progress), July 2014.

[I-D.ietf-6tisch-terminology] Palattella, M., Thubert, P., Watteyne, T., and Q. Wang, "Terminology in IPv6 over the TSCH mode of IEEE 802.15.4e", draft-ietf-6tisch-terminology-02 (work in progress), July 2014.

[I-D.ietf-6tisch-minimal] Vilajosana, X. and K. Pister, "Minimal 6TiSCH Configuration", draft-ietf-6tisch-minimal-03 (work in progress), October 2014.

[I-D.wang-6tisch-6top-sublayer] Wang, Q., Vilajosana, X., and T. Watteyne, "6TiSCH Operation Sublayer (6top)", draft-wang-6tisch-6top-sublayer-01 (work in progress), July 2014.

[I-D.ietf-6tisch-coap] Sudhaakar, R. and P. Zand, "6TiSCH Resource Management and Interaction using CoAP", draft-ietf-6tisch-coap-01 (work in progress), July 2014.

6.3. External Informative References

[IEEE802154e]
IEEE standard for Information Technology, "IEEE std. 802.15.4e, Part. 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer", April 2012.

[IEEE802154]
IEEE standard for Information Technology, "IEEE std. 802.15.4, Part. 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks", June 2011.

[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.

[morell04label]
Morell, A., Vilajosana, X., Lopez-Vicario, J., and T. Watteyne, "Label Switching over IEEE802.15.4e Networks. Transactions on Emerging Telecommunications Technologies", June 2013.

Authors' Addresses

Qin Wang (editor)
Univ. of Sci. and Tech. Beijing
30 Xueyuan Road
Beijing, Hebei 100083
China

Phone: +86 (10) 6233 4781
Email: wangqin@ies.ustb.edu.cn

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

Phone: +34 (646) 633 681
Email: xvilajosana@uoc.edu

Thomas Watteyne
Linear Technology
30695 Huntwood Avenue
Hayward, CA 94544
USA

Phone: +1 (510) 400-2978
Email: twatteyne@linear.com