6TiSCH
Internet-Draft
Intended status: Informational
Expires: September 28, 2014

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

March 27, 2014

6TiSCH Operation Sublayer (6top) Interface
draft-ietf-6tisch-6top-interface-00

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 commands internal to the 6top sublayer.

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 September 28, 2014.

Table of Contents

1. Introduction

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 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. 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

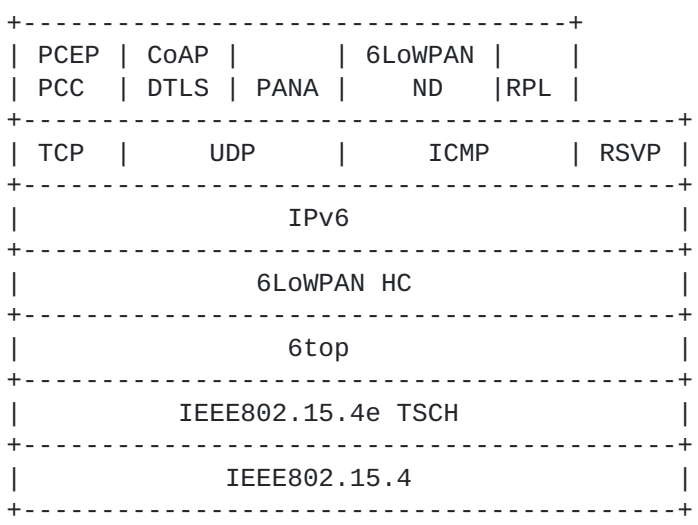

Figure 1

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 2.1.

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 2.2.

Section 3 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 4. This set of commands is designed to support decentralized, centralized and hybrid scheduling solutions.

2.1. 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 ask a node manage a 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.

2.1.1. hard cells

A hard cell is a cell that cannot be dynamically reallocated by 6top. A hard cell is uniquely identified by the following tuple:

slotframe ID: ID of the slotframe this cell is part of.

slotOffset: the slotOffset for the cell.

channelOffset: the channelOffset for the cell.

LinkOption bitmap: bitmap as defined in [IEEE802154].

CellType: MUST be set to 1.

2.1.2. 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].

2.2. 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

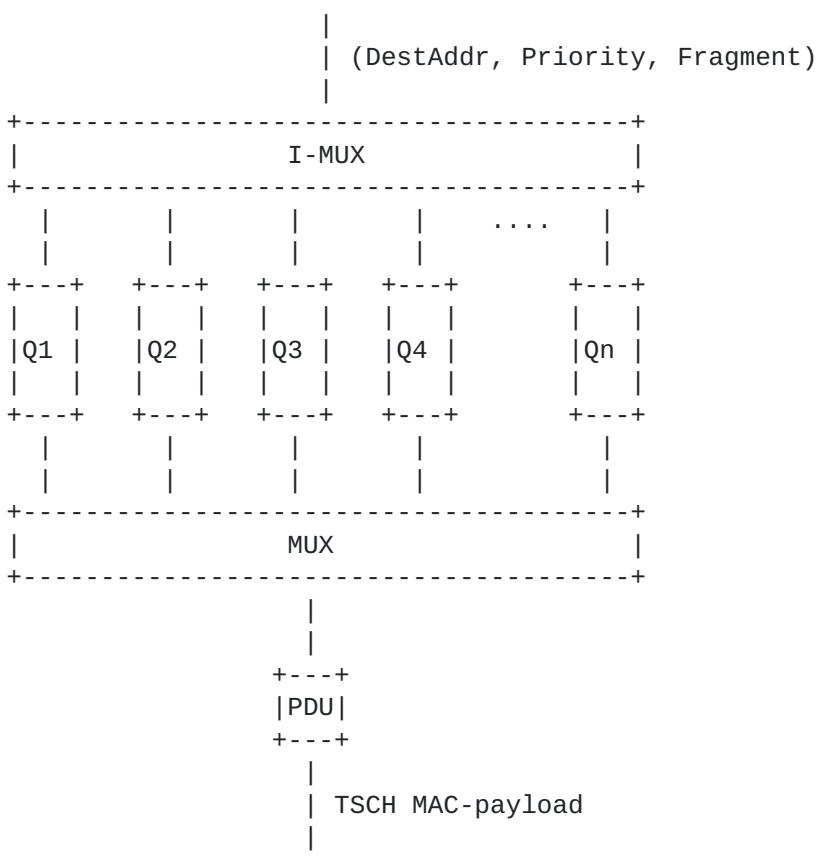

Figure 2

In Figure 2, Qi represents a queue, which is either broadcast or unicast, and is assigned a 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 4), 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.

3. 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 dada model consists of three parts: 6top MIB, part of the [IEEE802154e] PIB, and part of the [IEEE802154] PIB.

3.1. 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
      "Equal to SlotframeHandle defined in TSCH";
      reference
      "IEEE802154e";
   }
   leaf SlotOffset {
      type uint16;
      description
```

```
            "Defined in IEEE802154e.";
            reference
            "IEEE802154e";
         }
         leaf ChannelOffset {
            type uint8;
            description
            "Defined in IEEE802154e. Value range is 0..15";
            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 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;
               }
               leaf StatisticsValue{
                  type uint16;
                  config false;
               }
            }
         }
      }
```

```
   list SlotframeList {
      key "SlotframeID";
      leaf SlotframeID {
         type uint8;
      }
      leaf NumOfSlots {
         type uint16;
      }
   }

      list MonitoringStatusList {
         key "MonitoringStatusID";
         leaf MonitoringStatusID {
            type uint16;
         }
         leaf SlotframeID {
            type uint8;
         }
         leaf TargetNodeAddress {
            type uint64;
         }
         leaf EnforcePolicy {
            type enumeration {
               enum DISABLE;
               enum BESTEFFORT;
               enum STRICT;
               enum OVERPROVISION;
            }
            description
            "Currently enforced QoS policy";
         }
         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 CONFIGURE.qos enforce 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 provisioned cells, i.e. the actual
            bandwidth without taking into account the overprovisioned
            cells.";
         }
      }

   list StatisticsMetricsList {
      key "StatisticsMetricsID";
      leaf StatisticsMetricsID {
         type uint16;
      }
      leaf SlotframeID {
         type uint16;
         description
         "ID of the slotframe.  If empty, monitors all slotframe IDs";
         reference
         "IEEE802154e";
      }
      leaf SlotOffset {
         type uint16;
         description
         "Specific slotOffset to be monitored.  If empty all timeslots
         are monitored";
         reference
         "IEEE802154e";
      }
      leaf ChannelOffset {
         type uint8;
         description
         "Specific channelOffset to be monitored.  If empty all
         channels are monitored";
         reference
         "IEEE802154e";
      }
      leaf TargetNodeAddress {
```

```
         type uint64;
         description
         "If empty, all neighbor nodes are monitored.";
      }
      leaf Metrics {
         type enumeration {
            enum PDR;
            enum ETX;
            enum RSSI;
            enum LQI;
         }
         description
         "The metric to be monitored.";
      }
      leaf Window {
         type uint16;
         description
         "measurement period, in Number of the slotframe size";
      }
      leaf Enable {
         type enumeration {
            enum DISABLE;
            enum ENABLE;
         }
      }
   }
```

```
      list EBList {
         key "EbID";
         leaf EbID {
            type uint8;
         }
         leaf CellID {
            type uint16;
            description
            "Equal to LinkHandle in IEEE802154e";
         }
         leaf Peroid {
            type uint16;
            description
            "The EBs period, in seconds";
         }
         leaf Expiration {
            type enumeration {
               enum NEVERSTOP;
               enum EXPIRATION;
            }
            description
            "Which Period to indicate when the EBs periodicity will
            stop. If Zero the period never stops.";
         }
         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 {
      leaf policy {
         type enumeration {
            enum ALLPARENT;
            enum BESTCONNECTED;
            enum LOWESTJOINPRIORITY;
         }
      }
      leaf TargetNodeAddress {
         type uint64;
         description
         "Address of the time source neighbor";
      }
      leaf MinTimeCorrection {
         type uint16;
         description
         "In microsecond";
      }
      leaf MaxTimeCorrection {
         type uint16;
         description
         "In microsecond";
      }
      leaf AveTimeCorrection {
         type uint16;
         description
         "In microsecond";
      }
   }
```

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

   list NeighborList {
      key "TargetNodeAddress";
      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";
      }
   }

   list QueueList {
      key "QueueId";
      leaf QueueId {
         type uint8;
         description
         "Address of the time source neighbor";
      }
      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.";
         }
      }
      leaf MinLenTXQueue {
         type uint8;
         config false;
         description
         "Statistics, lowest TX queue len registered in the window.";
      }
      leaf MaxLenTXQueue {
         type uint8;
         config false;
```

```
         description
         "Statistics, largest TX queue len registered in the window.";
      }
      leaf AvgLenTXQueue {
         type uint8;
         config false;
         description
         "Statistics, avg TX queue len registered in the window.";
      }
      leaf MinLenRXQueue {
         type uint8;
         config false;
         description
         "Statistics, lowest RX queue len 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 len
          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";
      leaf LabelSwitchID {
         type uint16;
      }
      list InputCellIds {
         key "CellID";
         leaf CellID{
            type uint16;
            description
            "The CellID.";
         }
      }
      list OutputCellIds {
         key "CellID";
         leaf CellID{
            type uint16;
            description
            "The CellID.";
         }
      }
      leaf LoadBalancingPolicy {
         type enumeration {
            enum ROUNDROBIN;
            enum OTHER;
         }
         description
         "The load-balancing policy.";
      }
   }
```

```
   list TrackList {
      key "TrackId";
      leaf TrackId {
         type uint16;
      }
      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";
            leaf ChunkId{
               type uint16;
               description
               "The id of a chunk";
            }
            leaf SlotframeId{
               type uint8;
               description
               "The id of the slotframe that is mapped to this chunk";
            }
            leaf SlotBase {
               type uint16;
               description
               "the base slotOffset of the chunk";
            }
            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";
           leaf SlotOffset{
              type uint16;
              description
              "The slotoffset.";
           }
           leaf ChannelOffset{
              type uint16;
              description
              "The channeloffset.";
           }
           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 UNUSED;
                 enum USED;
              }
           }
        }
```

3.2. YANG model of the IEEE802.15.4 PIB

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

TODO

3.3. YANG model of the IEEE802.15.4e PIB

This section describes the YANG model of the part of [IEEE802154e] PIB used in 6top, such as TSCH related attributes. This part of data will be accessed through the MLME-GET and MLME-SET [IEEE802154] primitive.

TODO

4. 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 3.1).

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 3.1).

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 3.1).

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 3.1) will be modified.

DELETE.hardcell: To remove a hard cell. This removes the hard cell from the node's schedule, from CellList (Section 3.1).

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 3.1) 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 3.1) on both sides.

CREATE.slotframe: Creates a new slotframe. Adds a entry to the SlotframeList (Section 3.1).

READ.slotframe: Returns the information of a slotframe given its slotframeID from SlotframeList (Section 3.1).

UPDATE.slotframe: Change the number of timeslots in a slotframe given its slotframeID in SlotframeList (Section 3.1).

DELETE.slotframe: Deletes a slotframe, remove it from SlotframeList (Section 3.1).

CONFIGURE.monitoring: Configures the level of QoS the Monitoring process MUST enforce, i.e. config MonitoringStatusList (Section 3.1).

READ.monitoring: Reads the current Monitoring status from MonitoringStatusList (Section 3.1).

CONFIGURE.statistics: Configures the statistics process in StatisticsMetricsList(Section 3.1). 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 3.1).

RESET.statistics: Resets the gathered statistics in CellList (Section 3.1).

CONFIGURE.eb: Configures EBs, i.e. configures EBlist (Section 3.1).

READ.eb: Reads the EBs configuration from EBList (Section 3.1).

CONFIGURE.timesource: Configures the Time Source Neighbor selection process, i.e. configure TimeSource (Section 3.1).

READ.timesource: Retrieves information about the time source neighbors of that node from TimeSource (Section 3.1).

CREATE.neighbor: Creates an entry for a neighbor in the neighbor table, i.e. NeighborList (Section 3.1).

READ.all.neighbor: Returns the list of neighbors of that node according to NeighborList (Section 3.1).

READ.neighbor: Returns the information of a specific neighbor of that node specified by its neighbor address according to NeighborList (Section 3.1).

UPDATE.neighbor: Updates the last status for a given TargetNodeAddress in the NeighborList (Section 3.1).

DELETE.neighbor: Deletes a neighbor given its address from NeighborList (Section 3.1).

CREATE.queue: Creates and Configures a queue in QueueList (Section 3.1).

READ.queue: Reads the queue configuration for given QueueId from QueueList (Section 3.1).

READ.queue.stats: For a given QueueId, reads the queue statistics information from the QueueList (Section 3.1).

UPDATE.queue: For a given QueueId, update its configuration in the QueueList (Section 3.1).

DELETE.queue: Deletes a Queue for a given QueueId from the QueueList (Section 3.1).

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 3.1).

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 3.1).

CREATE.chunk: Creates a chunk which consists of one or more unused cells, i.e. add an entry to the ChunkList (Section 3.1).

READ.chunk: Returns the information of a chunk given its ChunkID from ChunkList (Section 3.1).

DELETE.chunk: For given ChunkId, removes a chunk from the ChunkList (Section 3.1), which also causes all of the scheduled

cells in the chunk to be deleted from the TSCH schedule and CellList (Section 3.1).

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 3.1). In addition, 6top will change the attributes corresponding to the cell in the ChunkCellList (Section 3.1), i.e. its CellID is changed to the same CellID in the CellList, and its Status is changed to USED.

READ.chunkcell: Returns the information of all cells in a chunk given its ChunkID from ChunkCellList (Section 3.1).

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 3.1). In addition, it changes the attributes corresponding to the cell in the ChunkCellList (Section 3.1), i.e. its CellID is changed back to 0xFFFF, and its Status is changed to UNUSED.

5. References

5.1. Normative References

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

5.2. 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 LLN context: Overview, Problem Statement and Goals", draft-ietf-6tisch-tsch-00 (work in progress), November 2013.

[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-01 (work in progress), February 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-01 (work in progress), February 2014.

[I-D.ietf-6tisch-minimal]
Vilajosana, X. and K. Pister, "Minimal 6TiSCH Configuration", draft-ietf-6tisch-minimal-00 (work in progress), November 2013.

[I-D.wang-6tisch-6top-sublayer]
Wang, Q., Vilajosana, X., and T. Watteyne, "6TiSCH Operation Sublayer (6top)", draft-wang-6tisch-6top-sublayer-00 (work in progress), February 2014.

5.3. 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] "Berkeley's OpenWSN Project Homepage", <http://www.openwsn.org/>.

[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