6TiSCH
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2018

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

October 25, 2017

# 6top Protocol (6P)
## draft-ietf-6tisch-6top-protocol-09

## Abstract

This document defines the 6top Protocol (6P), which enables distributed scheduling in 6TiSCH networks. 6P allows neighbor nodes to add/delete TSCH cells to one another. 6P is part of the 6TiSCH Operation Sublayer (6top), the next higher layer to the IEEE Std 802.15.4 TSCH medium access control layer. The 6P layer is formed by the 6top Protocol defined in this document and 6top Scheduling Function(s). A 6top Scheduling Function (SF) decides when to add/delete cells, and triggers 6P Transactions. This document lists the requirements for an SF, but leaves the definition of SFs out of scope.

## 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 RFC 2119 [RFC2119].

## 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 https://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 28, 2018.

Table of Contents

## 1. Introduction

All communication in a 6TiSCH network is orchestrated by a schedule [RFC7554]. The schedule is composed of cells, each identified by a [slotOffset,channelOffset]. This specification defines the 6top Protocol (6P), part of the 6TiSCH Operation sublayer (6top). 6P allows a node to communicate with a neighbor to add/delete TSCH cells to one another. This results in distributed schedule management in a 6TiSCH network. The 6top layer is composed of the 6top Protocol and one of more 6top Scheduling Functions (SFs) that decide when to add/delete cells and triggers 6P Transactions. The SF is out of the scope of this document but the requirements for an SF are defined here.

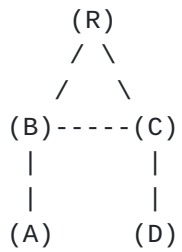


Figure 1: A simple 6TiSCH network.

The example network depicted in Figure 1 is used to describe the interaction between nodes. We consider the canonical case where node "A" issues 6P requests to node "B". We keep this example throughout this document. Throughout the document, node A will always represent

the node that issues a 6P request; node B the node that receives this request.

We consider that node A monitors the communication cells it has in its schedule to node B:

- If node A determines that the number of link-layer frames it is sending to B per unit of time is larger than the capacity offered by the TSCH cells it has scheduled to B, it triggers a 6P Transaction with node B to add one or more cells to the TSCH schedule of both nodes.
- If the traffic is lower than the capacity, node A triggers a 6P Transaction with node B to delete one or more cells in the TSCH schedule of both nodes.
- Node A MAY also monitor statistics to determine whether collisions are happening on a particular cell to node B. If this feature is enabled, node A communicates with node B to "relocate" the cell which suffered from collisions to a different [slotOffset,channelOffset] location in the TSCH schedule.

This results in distributed schedule management in a 6TiSCH network.

The 6top Scheduling Function (SF) defines when to add/delete a cell to a neighbor. Different applications require different SFs, so the SF is left out of scope of this document. Different SFs are expected to be defined in future companion specifications. A node MAY implement multiple SFs and run them at the same time. At least one SF MUST be running. The SFID field contained in all 6P messages allows a node to invoke the appropriate SF on a per-transaction basis.

Section 2 describes the 6TiSCH Operation Sublayer (6top). Section 3 defines the 6top Protocol (6P). Section 4 provides guidelines on how to design an SF.

## 2. 6TiSCH Operation Sublayer (6top)

As depicted in Figure 2, the 6TiSCH Operation Sublayer (6top) is the next higher layer to the IEEE Std 802.15.4 TSCH medium access control (MAC) layer [IEEE802154]. We use "802.15.4" as a short version of "IEEE Std 802.15.4" in this document.

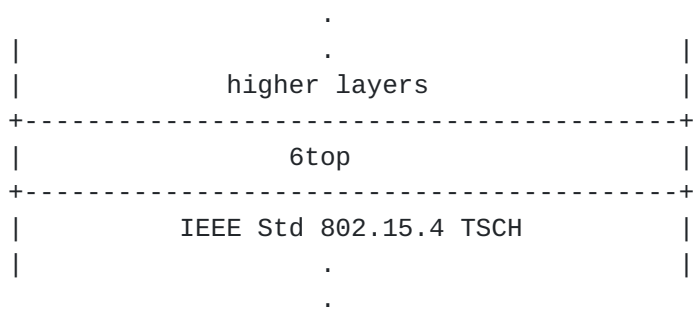


Figure 2: The 6top sublayer in the protocol stack.

The roles of the 6top sublayer are to:

- Implement and terminate the 6top Protocol (6P), which allows neighbor nodes to communicate to add/delete cells to one another.
- Run one or multiple 6top Scheduling Functions (SFs), which define the rules that decide when to add/delete cells.

## 2.1. Hard/Soft Cells

Each cell in the schedule is either "hard" or "soft":

- a soft cell can be read, added, deleted or updated by 6top.
- a hard cell is read-only for 6top.

In the context of this specification, all the cells used by 6top are soft cells. Hard cells can be used for example when "hard-coding" a schedule [RFC8180].

## 2.2. Using 6P with the Minimal 6TiSCH Configuration

6P MAY be used alongside the Minimal 6TiSCH Configuration [RFC8180]. In this case, it is RECOMMENDED to use 2 slotframes, as depicted in Figure 3:

- Slotframe 0 is used for traffic defined in the Minimal 6TiSCH Configuration. In Figure 3, this slotframe is 5 slots long, but the slotframe can be shorter or longer.
- 6P allocates cells from Slotframe 1. In Figure 3, Slotframe 1 is 10 slots long, but the slotframe can be shorter or longer.

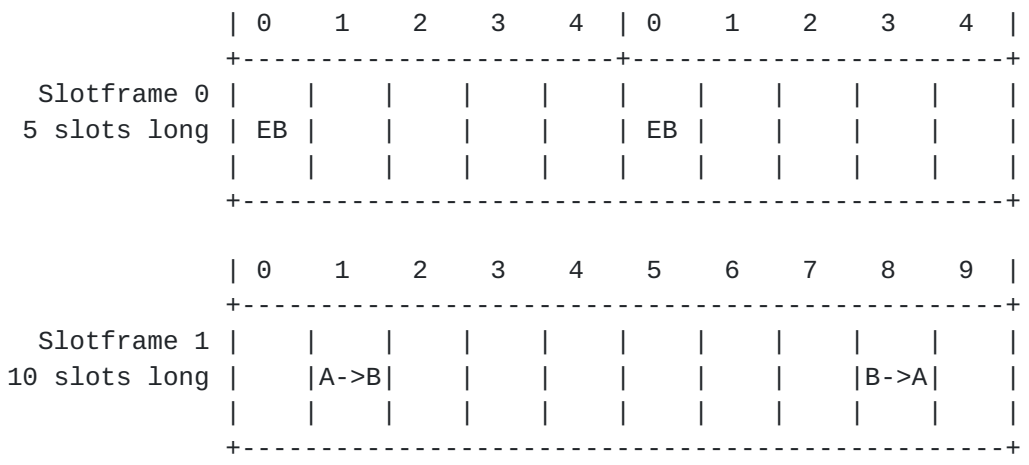


Figure 3: 2-slotframe structure when using 6P alongside the Minimal 6TiSCH Configuration.

The Minimal 6TiSCH Configuration cell SHOULD be allocated from a slotframe of higher priority than the slotframe used by 6P for dynamic cell allocation. This way, dynamically allocated cells cannot "mask" the cells used by the Minimal 6TiSCH Configuration. 6top MAY support additional slotframes; how to use additional slotframes is out of the scope for this document.

## 3. 6top Protocol (6P)

The 6top Protocol (6P) enables two neighbor nodes to add/delete/relocate cells in their TSCH schedule. Conceptually, two neighbor nodes "negotiate" the location of the cells to add, delete, or relocate in their TSCH schedule.

### 3.1. 6P Transactions

We call "6P Transaction" a complete negotiation between two neighbor nodes. A 6P Transaction starts when a node wishes to add/delete/relocate one or more cells with one of its neighbors. A 6P Transaction ends when the cell(s) have been added/deleted/relocated in the schedule of both nodes, or when the 6P Transaction fails.

The 6P messages exchanged between nodes A and B during a 6P Transaction SHOULD be exchanged on non-shared unicast cells ("dedicated" cells) between A and B. If no dedicated cells are scheduled between nodes A and B, shared cells MAY be used.

Keeping consistency between the schedules of the two neighbor nodes is important. A loss of consistency (e.g. node A has a transmit cell to node B, but node B does not have the corresponding reception cell) can cause loss of connectivity. To verify consistency, neighbor

nodes maintain a Sequence Number (SeqNum). Neighbor nodes exchange the SeqNum as part of each 6P Transaction to detect possible inconsistency. This mechanism is explained in Section 3.4.6.2.

An implementation MUST include a mechanism to associate each scheduled cell with the SF that scheduled it. This mechanism is implementation-specific and out of the scope of this document.

A 6P Transaction can consist of 2 or 3 steps. A 2-step transaction is used when node A selects the cells to be allocated. A 3-step transaction is used when node B selects the cells to be allocated. An SF MUST specify whether to use 2-step transactions, 3-step transactions, or both.

We illustrate 2-step and 3-step transactions using the topology in Figure 1.

### 3.1.1. 2-step 6P Transaction

Figure 4 shows an example 2-step 6P Transaction. In a 2-step transaction, node A selects the candidate cells. Several elements are left out to simplify understanding.

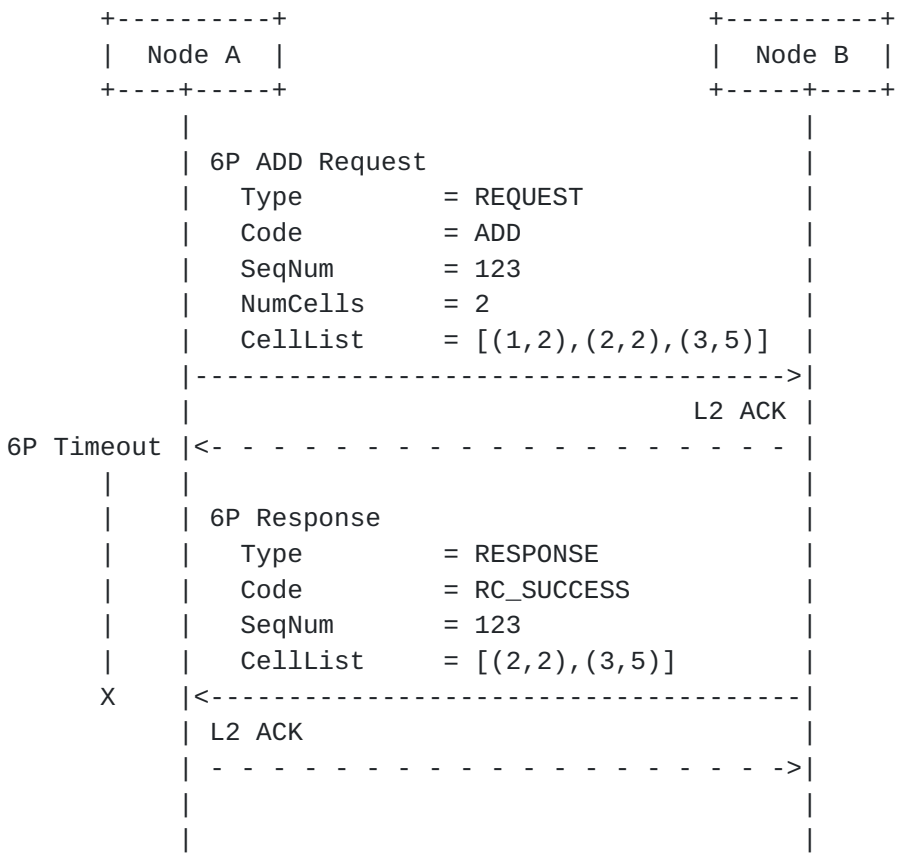


Figure 4: An example 2-step 6P Transaction.

In this example, the 2-step transaction occurs as follows:

1. The SF running on node A determines that 2 extra cells need to be scheduled to node B.
2. The SF running on node A selects 3 candidate cells.
3. Node A sends a 6P ADD Request to node B, indicating it wishes to add 2 cells (the "NumCells" value), and specifying the list of 3 candidate cells (the "CellList" value). Each cell in the CellList is a [slotOffset,channelOffset] tuple. This 6P ADD Request is link-layer acknowledged by node B (labeled "L2 ACK" in Figure 4).
4. After having successfully sent the 6P ADD Request, Node A starts a 6P Timeout to abort the transaction in case no response is received.
5. The SF running on node B selects 2 out of the 3 cells in the CellList of the 6P ADD Request. Node B sends back a 6P Response to node A, indicating the cells that node B has selected. The response is link-layer acknowledged by node A.
6. Upon completion of this 6P Transaction, 2 cells from A to B have been added to the TSCH schedule of both nodes A and B.

### 3.1.2. 3-step 6P Transaction

Figure 5 shows an example 3-step 6P Transaction. In a 3-step transaction, node B selects the candidate cells. Several elements are left out to simplify understanding.

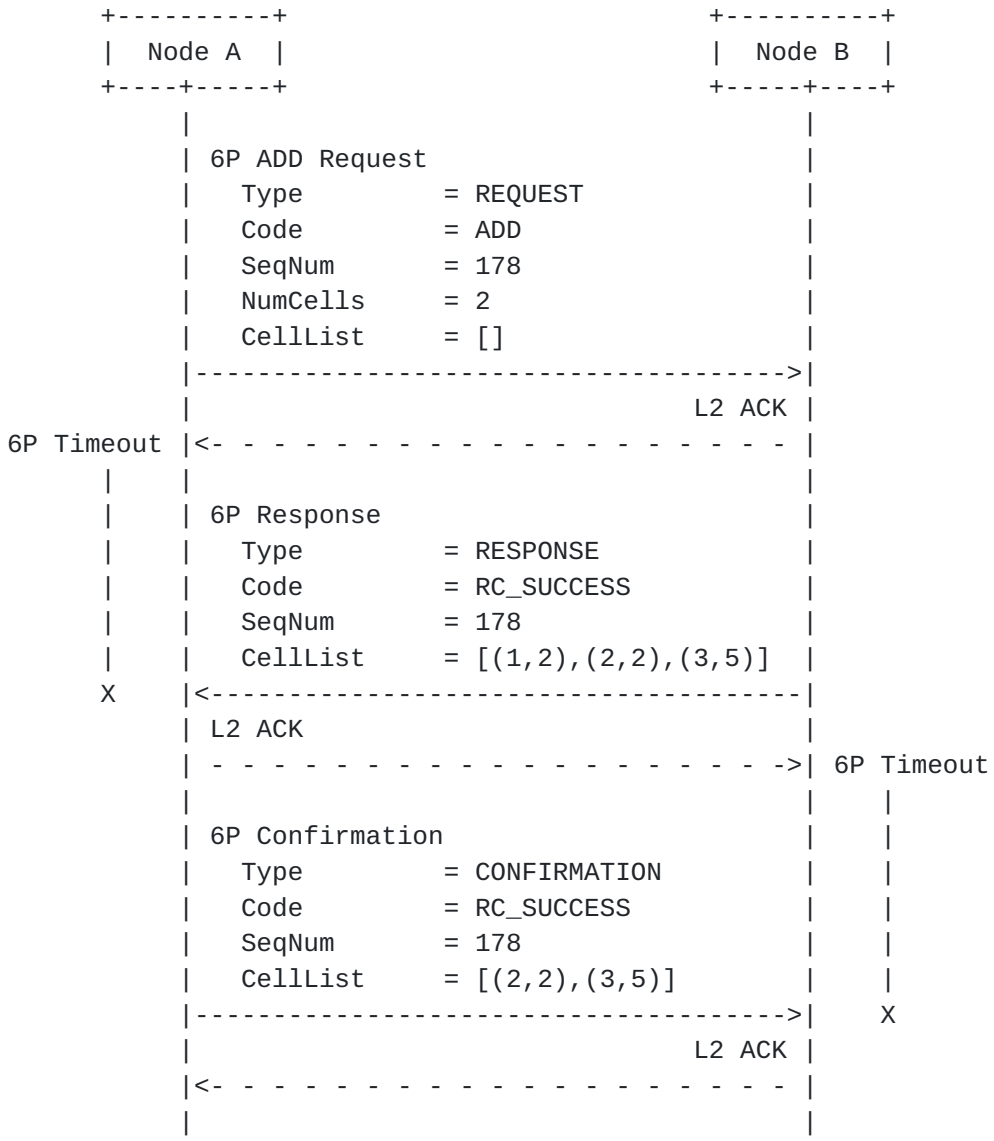


Figure 5: An example 3-step 6P Transaction.

In this example, the 3-step transaction occurs as follows:

1. The SF running on node A determines that 2 extra cells need to be scheduled to node B, but does not select candidate cells.
2. Node A sends a 6P ADD Request to node B, indicating it wishes to add 2 cells (the "NumCells" value), with an empty "CellList". This 6P ADD Request is link-layer acknowledged by node B.

3. After having successfully sent the 6P ADD Request, Node A starts a 6P Timeout to abort the transaction in case no 6P Response is received.
4. The SF running on node B selects 3 candidate cells. Node B sends back a 6P Response to node A, indicating the 3 cells it selected. The response is link-layer acknowledged by node A.
5. After having successfully sent the 6P Response, Node B starts a 6P Timeout to abort the transaction in case no 6P Confirmation is received.
6. The SF running on node A selects 2 cells. Node A sends back a 6P Confirmation to node B, indicating the cells it selected. The confirmation is link-layer acknowledged by node B.
7. Upon completion of this 6P Transaction, 2 cells from A to B have been added to the TSCH schedule of both nodes A and B.

## 3.2. Message Format

### 3.2.1. 6top Information Element (IE)

6P messages travel over a single hop. 6P messages are carried as payload of an IEEE 802.15.4 Payload Information Element (IE) [IEEE802154]. The messages are encapsulated with the Payload IE Header. The Group ID is set to the IETF IE value defined in [RFC8137]. The content is encapsulated by a SubType ID as defined in [RFC8137].

Since 6P messages are carried in IE, IEEE bit/byte ordering applies. Bits within each field in the 6top IE are numbered from 0 (leftmost and least significant) to k-1 (rightmost and most significant), where the length of the field is k bits. Fields that are longer than a single octet are copied to the packet in the order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits (little endian).

This document defines the "6top IE", a SubType of the IETF IE defined in [RFC8137], with subtype ID IANA_6TOP_SUBIE_ID. The SubType Content of the "6top IE" is defined in Section 3.2.2. The length of the "6top IE" content is variable.

### 3.2.2. Generic 6P Message Format

All 6P messages follow the generic format shown in Figure 6.

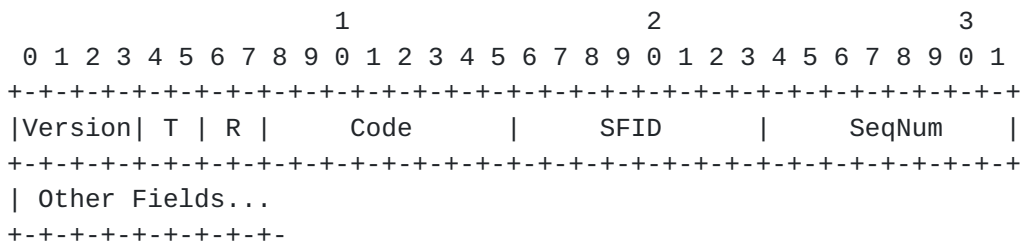


Figure 6: Generic 6P Message Format.

6P Version (Version): The version of the 6P protocol. Only version 0 is defined in this document. Future specifications MAY define further versions of the 6P protocol.

Type (T): Type of message. The message types are defined in Section 6.2.2.

Reserved (R): Reserved bits. These two bits SHOULD be set to zero when sending the message and MUST be ignored upon reception.

Code: The Code field contains a 6P Command Identifier when the 6P message is of Type REQUEST. Section 6.2.3 lists the 6P command identifiers. The Code field contains a 6P Return Code when the 6P message is of Type RESPONSE or CONFIRMATION. Section 6.2.4 lists the 6P Return Codes. The same return codes are used in both 6P Response and 6P Confirmation messages.

6top Scheduling Function Identifier (SFID): The identifier of the SF to use to handle this message. The SFID is defined in Section 4.1.

SeqNum: Sequence number associated with the 6P Transaction, used to match the 6P Request, 6P Response and 6P Confirmation of the same 6P Transaction. The value of SeqNum MUST be different at each new 6P request issued to the same neighbor. The SeqNum is also used to ensure consistency between the schedules of the two neighbors. Section 3.4.6 details how the SeqNum is managed.

Other Fields: The list of other fields and how they are used is detailed in Section 3.3.

### 3.2.3. 6P CellOptions

An 8-bit 6P CellOptions bitmap is present in the following 6P requests: ADD, DELETE, COUNT, LIST, RELOCATE.

- In the 6P ADD request, the 6P CellOptions bitmap is used to specify what type of cell to add.
- In the 6P DELETE request, the 6P CellOptions bitmap is used to specify what type of cell to delete.
- In the 6P COUNT and the 6P LIST requests, the 6P CellOptions bitmap is used as a selector of a particular type of cells.

- In the 6P RELOCATE request, the 6P CellOptions bitmap is used to specify what type of cell to relocate.

The contents of the 6P CellOptions bitmap apply to all elements in the CellList field. Section 6.2.6 contains the RECOMMENDED format of the 6P CellOptions bitmap. Figure 7 contains the RECOMMENDED meaning of the 6P CellOptions bitmap for the 6P COUNT and 6P LIST requests.

Note: assuming node A issues the 6P command to node B.

| CellOptions Value | the cells B selects from its schedule when receiving a 6P COUNT or LIST Request from A, from the cells it has scheduled with A |
|---|---|
| TX=0,RX=0,S=0 | all cells |
| TX=1,RX=0,S=0 | all cells marked as RX |
| TX=0,RX=1,S=0 | all cells marked as TX |
| TX=1,RX=1,S=0 | all cells marked as TX and RX |
| TX=0,RX=0,S=1 | all cells marked as SHARED |
| TX=1,RX=0,S=1 | all cells marked as RX and SHARED |
| TX=0,RX=1,S=1 | all cells marked as TX and SHARED |
| TX=1,RX=1,S=1 | all cells marked as TX and RX and SHARED |

Figure 7: Meaning of the 6P CellOptions bitmap for the 6P COUNT and the 6P LIST requests.

The CellOptions is an opaque set of bits, sent unmodified to the SF. The SF MAY redefine the format of the CellOptions bitmap. The SF MAY redefine the meaning of the CellOptions bitmap.

### 3.2.4. 6P CellList

A CellList field MAY be present in a 6P ADD Request, a 6P DELETE Request, a 6P RELOCATE Request, a 6P Response or a 6P Confirmation. It is composed of a concatenation of zero, one or more 6P Cells as defined in Figure 8. The contents of the CellOptions field specify the options associated with all cells in the CellList. This necessarily means that the same options are associated with all cells in the CellList.

The 6P Cell is a 4-byte field, its RECOMMENDED format is:

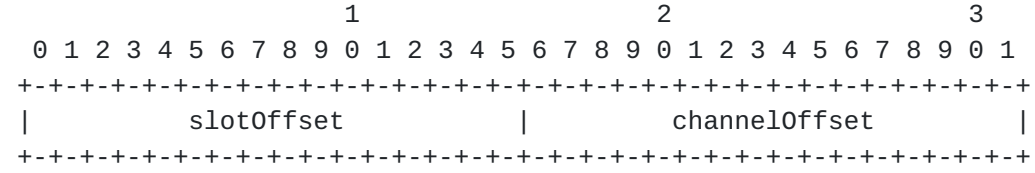


Figure 8: 6P Cell Format.

slotOffset: The slot offset of the cell.
channelOffset: The channel offset of the cell.

The CellList is an opaque set of bytes, sent unmodified to the SF. The SF MAY redefine the format of the CellList field.

## 3.3. 6P Commands and Operations

### 3.3.1. Adding Cells

Cells are added by using the 6P ADD command. The Type field (T) is set to REQUEST. The Code field is set to ADD. Figure 9 defines the format of a 6P ADD Request.

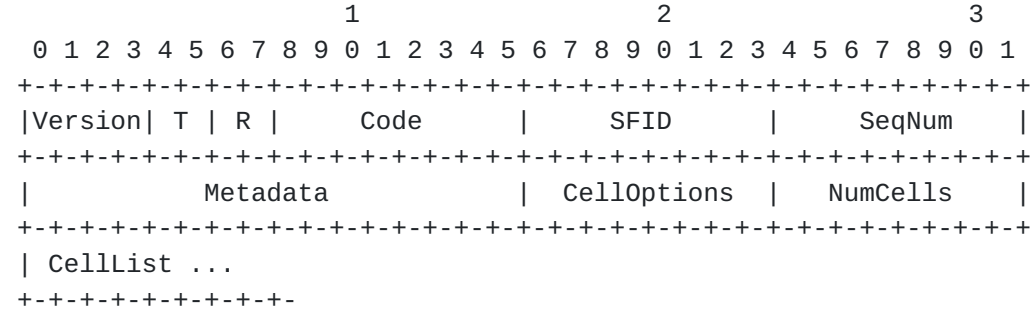


Figure 9: 6P ADD Request Format.

Metadata: Used as extra signaling to the SF. The contents of the Metadata field is an opaque set of bytes passed unmodified to the SF. The meaning of this field depends on the SF, and is out of scope of this document. For example, Metadata can specify in which slotframe to add the cells.

CellOptions: Indicates the options to associate with the cells to be added. If more than one cell is added (NumCells>1), the same options are associated with each one. This necessarily means that, if node A needs to add multiple cells with different options, it needs to initiate multiple 6P ADD Transactions.

NumCells: The number of additional cells the sender wants to schedule to the receiver.

CellList: A list of 0, 1 or multiple candidate cells.

Figure 10 defines the format of a 6P ADD Response and Confirmation.

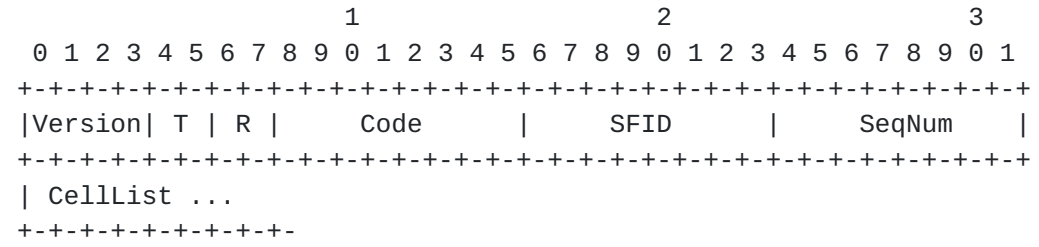


Figure 10: 6P ADD Response and Confirmation Formats.

CellList: A list of 0, 1 or multiple 6P Cells.

Consider the topology in Figure 1 where the SF on node A decides to add NumCells cells to node B.

Node A's SF selects NumCandidate cells from its schedule. These are cells that are candidates to be scheduled with node B. The CellOptions field specifies the type of these cells. NumCandidate MUST be larger or equal to NumCells. How many cells node A selects (NumCandidate) and how that selection is done is specified in the SF and out of scope of this document. Node A sends a 6P ADD Request to node B which contains the CellOptions, the value of NumCells and a selection of NumCandidate cells in the CellList. In case the NumCandidate cells do not fit in a single packet, this operation MUST be split into multiple independent 6P ADD Requests, each for a subset of the number of cells that eventually need to be added.

Upon receiving the request, node B's SF verifies which of the cells in the CellList it can install in node B's schedule, following the specified CellOptions field. How that selection is done is specified in the SF and out of scope of this document. The verification can succeed (NumCells cells from the CellList can be used), fail (none of the cells from the CellList can be used) or partially succeed (less than NumCells cells from the CellList can be used). When the allocation succeeds or partially succeeds, node B MUST send a 6P Response with return code set to RC_SUCCESS, and which specifies the list of cells that were scheduled following the CellOptions field. The returned list can contain NumCells elements (succeeded) or between 0 and NumCells elements (partially succeeded). In the case that none of the cells could be allocated node B MUST send a 6P Response with return code set to NOALLOC, indicating that cells could not be allocated in the schedule, for example because they are already used or reserved. The returned list in this case MUST contain 0 elements.

Upon receiving the response, node A adds the cells specified in the CellList according to the request CellOptions field.

### 3.3.2. Deleting Cells

Cells are deleted by using the 6P DELETE command. The Type field (T) is set to REQUEST. The Code field is set to DELETE. Figure 11 defines the format of a 6P DELETE Request.

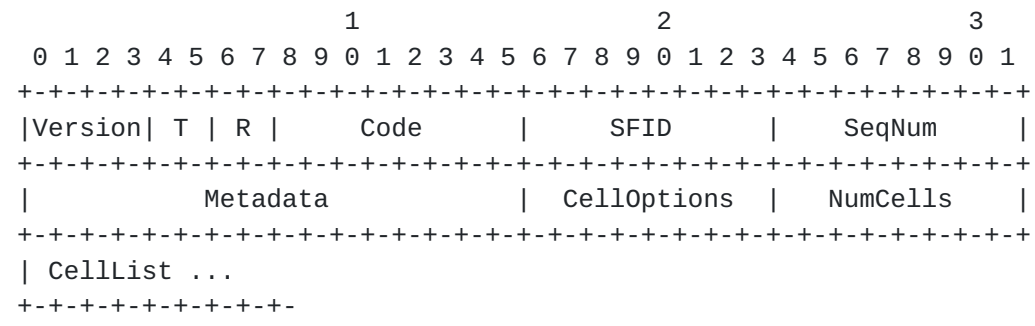


Figure 11: 6P DELETE Request Format.

Metadata: Same usage as for the 6P ADD command, see Section 3.3.1. Its format is the same as that in 6P ADD command, but its contents could be different.

CellOptions: Indicates the options that need to be associated to the cells to delete. Only the cells matching the CellOptions are deleted.

NumCells: The number of cells from the specified CellList the sender wants to delete from the schedule of both sender and receiver.

CellList: A list of 0, 1 or multiple 6P Cells.

Figure 12 defines the format of a 6P DELETE Response and Confirmation.

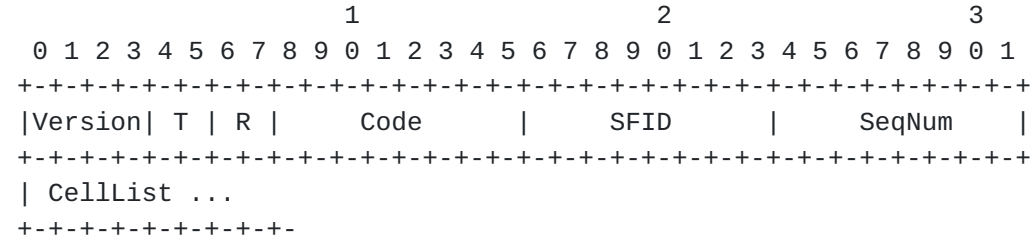


Figure 12: 6P DELETE Response and Confirmation Formats.

CellList: A list of 0, 1 or multiple 6P Cells.

The behavior for deleting cells is equivalent to that of adding cells except that:

- The nodes delete the cells they agree upon rather than adding them.
- All cells in the CellList MUST already be scheduled between the two nodes and MUST match the CellOptions field. If node A puts cells in its CellList that are not already scheduled between the two nodes and match the CellOptions field, node B MUST reply with a RC_CELLLIST return code.
- If the CellList in the 6P Request is empty, the SF on the receiving node SHOULD delete any cell from the sender, as long as it matches the CellOptions field.
- The CellList in a 6P Request (2-step transaction) or 6P Response (3-step transaction) MUST either be empty, contain exactly NumCells cells, or more than NumCells cells. The case where the CellList is not empty but contains less than NumCells cells is not supported.

### 3.3.3. Relocating Cells

Cell relocation consists in moving a cell to a different [slotOffset,channelOffset] location in the schedule. The Type field (T) is set to REQUEST. The Code is set to RELOCATE. Figure 13 defines the format of a 6P RELOCATE Request.

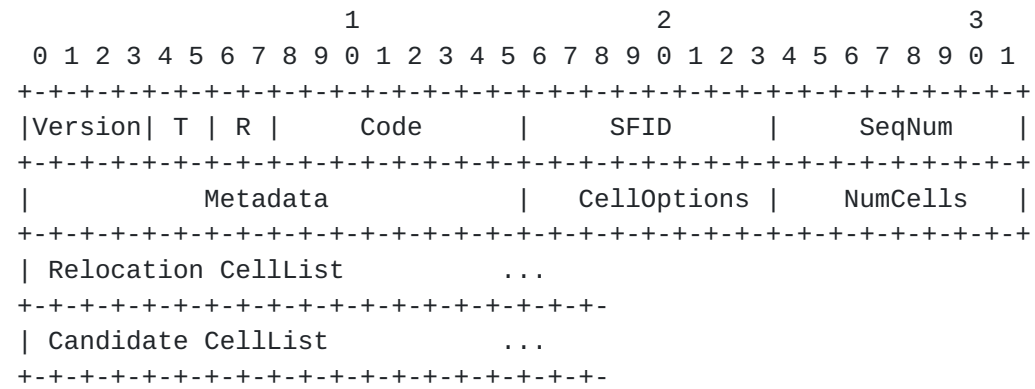


Figure 13: 6P RELOCATE Request Format.

Metadata: Same usage as for the 6P ADD command, see Section 3.3.1.
CellOptions: Indicates the options that need to be associated to the relocated cells.
NumCells: The number of cells to relocate, which MUST be equal or greater than 1.
Relocation CellList: The list of NumCells 6P Cells to relocate.
Candidate CellList: A list of NumCandidate candidate cells for node B to pick from. NumCandidate MUST be 0, equal to NumCells, or greater than NumCells.

In a 2-step 6P RELOCATE Transaction, node A specifies both the cells it needs to relocate, and the list of candidate cells to relocate to. The Relocation CellList MUST contain exactly NumCells entries. The Candidate CellList MUST contain at least NumCells entries.

In a 3-step 6P RELOCATE Transaction, node A only specifies the cells it needs to relocate, but not the list of candidate cells to relocate to. The Candidate CellList MUST therefore be empty.

Figure 14 defines the format of a 6P RELOCATE Response and Confirmation.

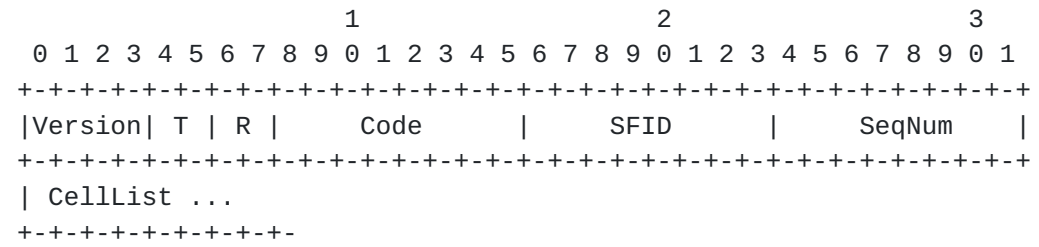


Figure 14: 6P RELOCATE Response and Confirmation Formats.

CellList: A list of 0, 1 or multiple 6P Cells.

Node A's SF wants to relocate NumCells cells. Node A creates a 6P RELOCATE Request, and indicates the cells to relocate in the Relocation CellList. It also selects NumCandidate cells from its schedule as candidate cells for node B, and puts those in the Candidate CellList. The CellOptions field specifies the type of the cell(s) to relocate. NumCandidate MUST be larger or equal to NumCells. How many cells it selects (NumCandidate) and how that selection is done is specified in the SF and out of scope of this document. Node A sends the 6P RELOCATE Request to node B.

Upon receiving the request, node B's SF verifies that all the cells in the Relocation CellList are indeed scheduled with node A, and are associate the options specified in the CellOptions field. If that check fails, node B MUST send a 6P Response to node A with return code RC_CELLLIST. If that check passes, node B's SF verifies which of the cells in the Candidate CellList it can install in its schedule. How that selection is done is specified in the SF and out of scope of this document. That verification on Candidate CellList can succeed (NumCells cells from the Candidate CellList can be used), fail (none of the cells from the Candidate CellList can be used) or partially succeed (less than NumCells cells from the Candidate CellList can be used). In all cases, node B MUST send a 6P Response with return code set to RC_SUCCESS, and which specifies the list of cells that were scheduled following the CellOptions field. That can

contain 0 elements (when the verification failed), NumCells elements (succeeded) or between 0 and NumCells elements (partially succeeded). If N < NumCells cells appear in the CellList, this means first N cells in the Relocation CellList have been relocated, the remainder have not.

Upon receiving the response, node A relocates the cells specified in Relocation CellList of its RELOCATE Request to the new location specified in the CellList of the 6P Response.

Figure 15 shows an example of a successful 2-step 6P RELOCATION Transaction.

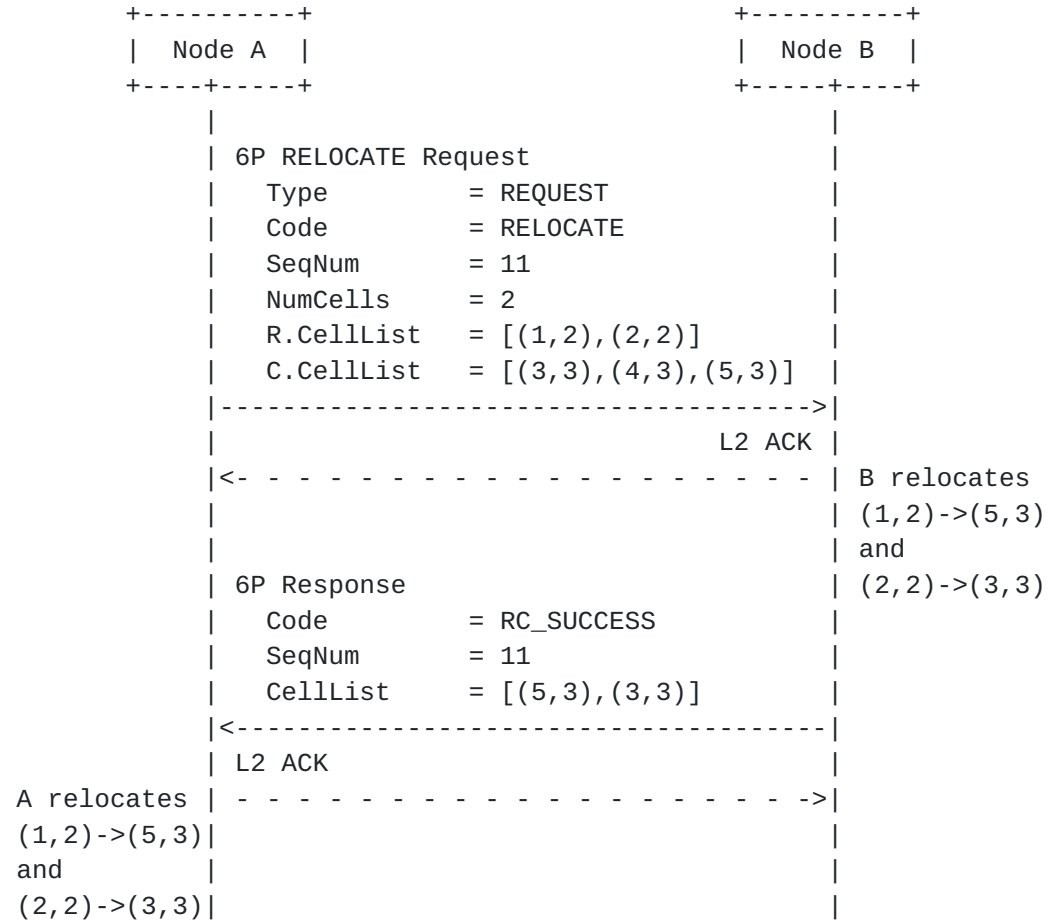


Figure 15: Example of a successful 2-step 6P RELOCATION Transaction.

Figure 16 shows an example of a partially successful 2-step 6P RELOCATION Transaction.

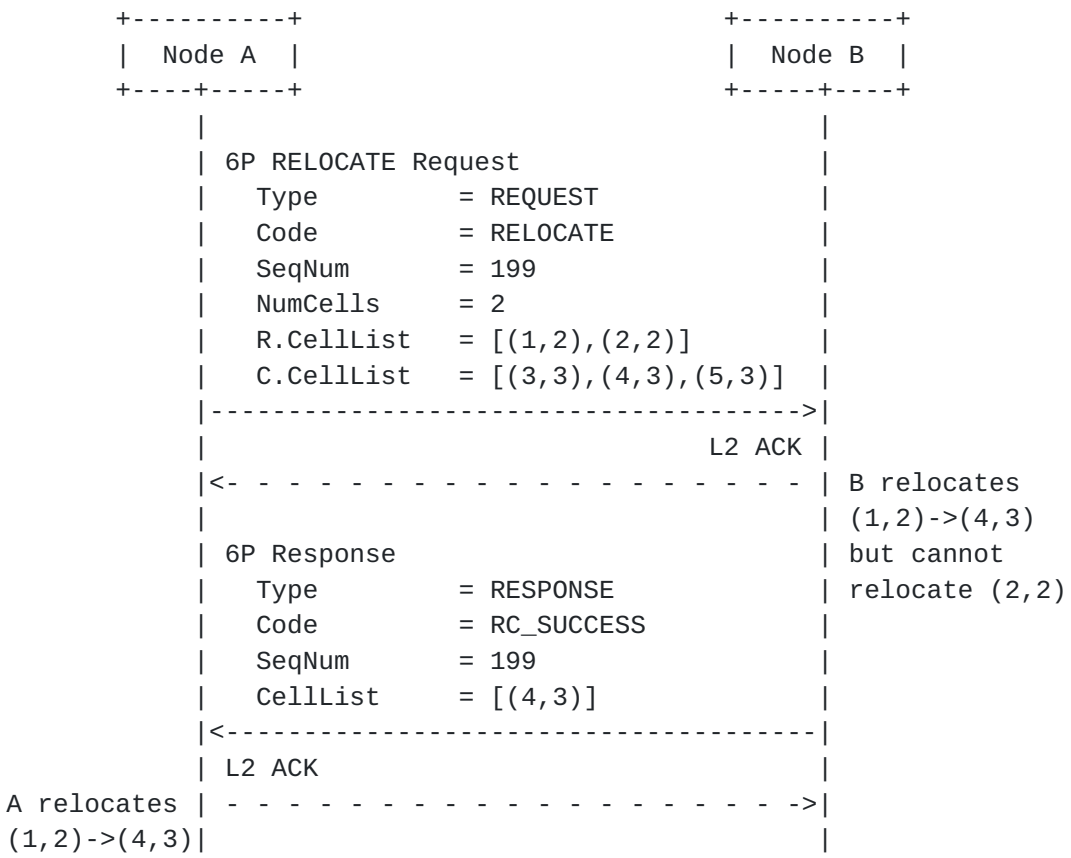


Figure 16: Example of a partially successful 2-step 6P RELOCATION Transaction.

Figure 17 shows an example of a failed 2-step 6P RELOCATION Transaction.

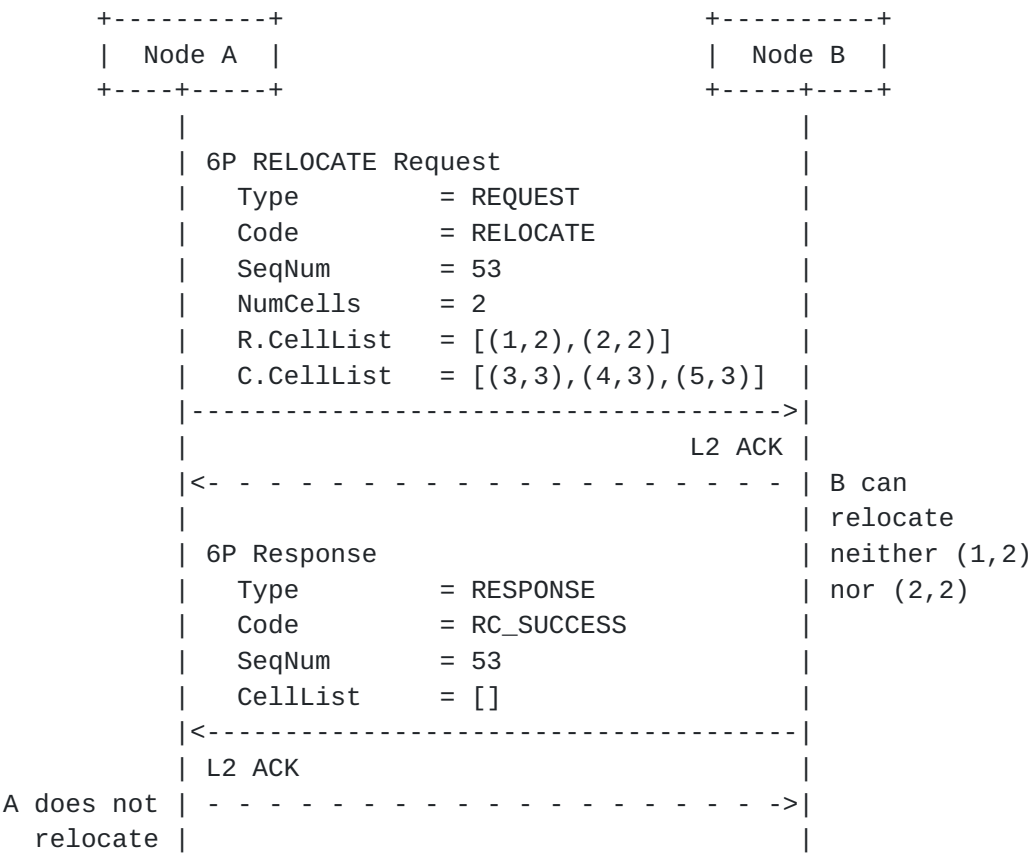


Figure 17: Failed 2-step 6P RELOCATION Transaction Example.

Figure 18 shows an example of a successful 3-step 6P RELOCATION Transaction.

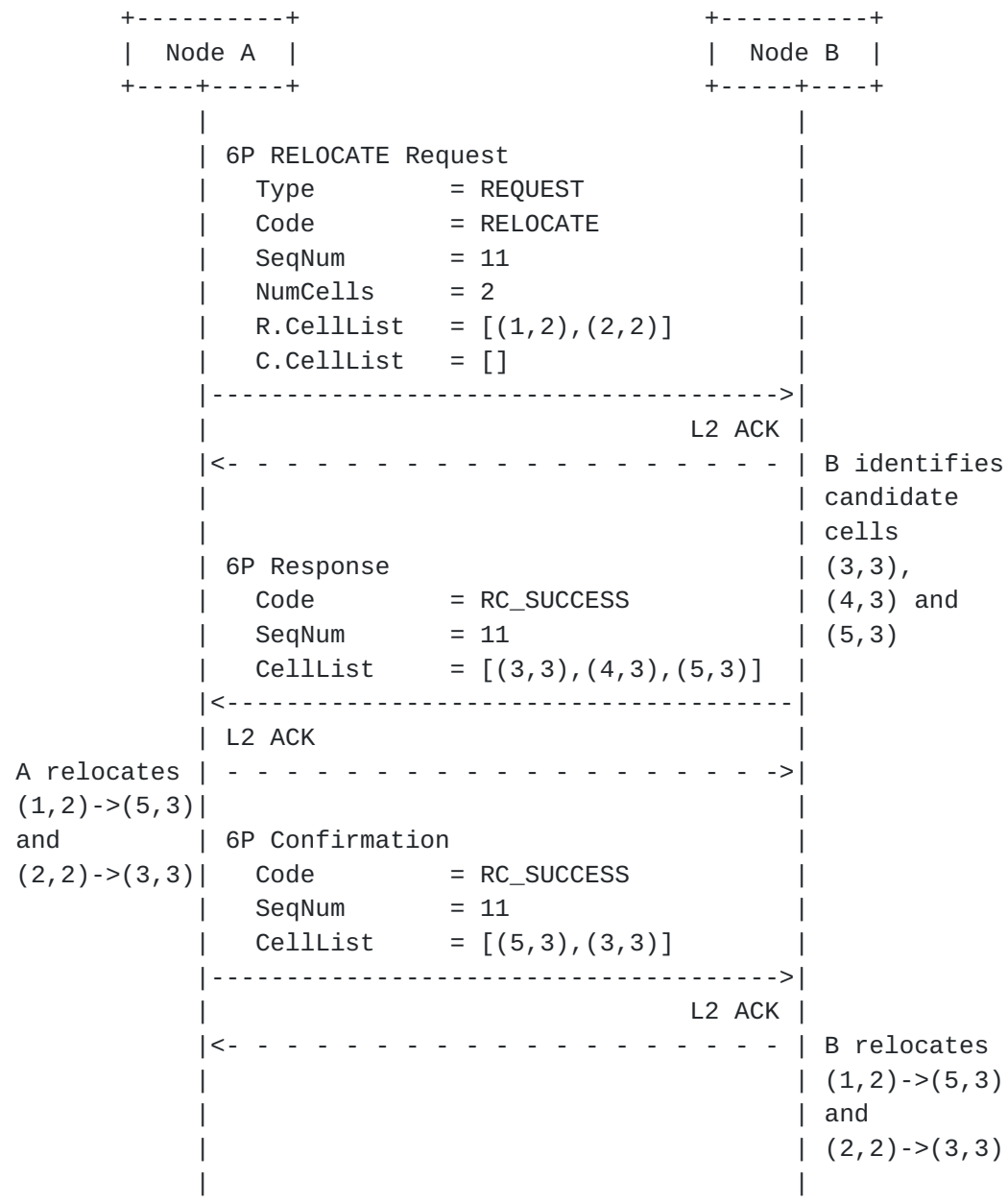


Figure 18: Example of a successful 3-step 6P RELOCATION Transaction.

### 3.3.4. Counting Cells

To retrieve the number of scheduled cells at B, node A issues a 6P COUNT command. The Type field (T) is set to REQUEST. The Code field is set to COUNT. Figure 19 defines the format of a 6P COUNT Request.

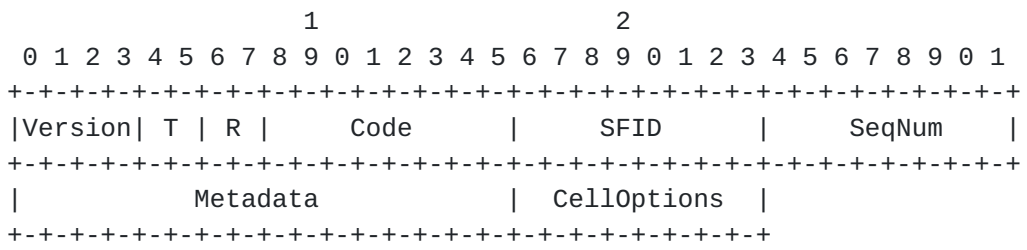


Figure 19: 6P COUNT Request Format.

Metadata: Same usage as for the 6P ADD command, see Section 3.3.1. Its format is the same as that in 6P ADD command, but its contents could be different.

CellOptions: Specifies which types of cells to be counted.

Figure 20 defines the format of a 6P COUNT Response.

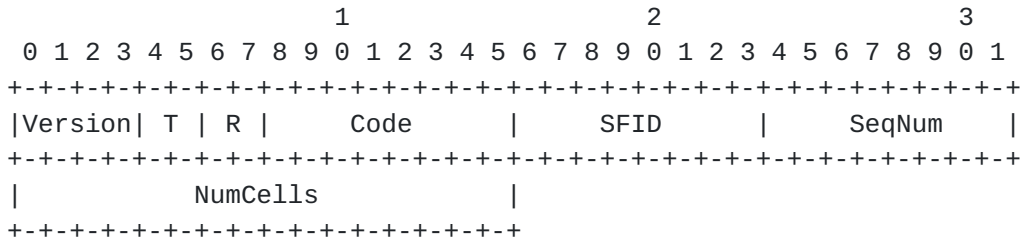


Figure 20: 6P COUNT Response Format.

NumCells: The number of cells which correspond to the fields of the request.

Node A issues a COUNT command to node B, specifying a set of cell options. Upon receiving the 6P COUNT request, node B goes through its schedule and counts the number of cells scheduled with node A in its own schedule, and which match the cell options in the CellOptions field of the request. Section 3.2.3 details the use of the CellOptions field.

Node B issues a 6P response to node A with return code set to RC_SUCCESS, and with NumCells containing the number of cells that match the request.

### 3.3.5. Listing Cells

To retrieve the list of scheduled cells at B, node A issues a 6P LIST command. The Type field (T) is set to REQUEST. The Code field is set to LIST. Figure 21 defines the format of a 6P LIST Request.

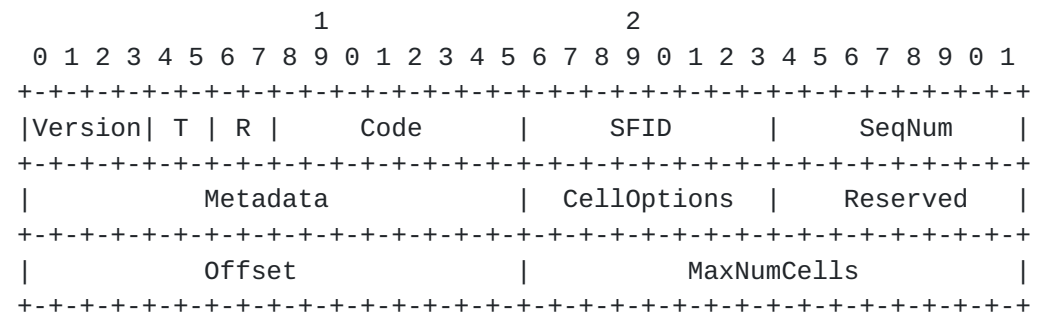


Figure 21: 6P LIST Request Format.

Metadata: Same usage as for the 6P ADD command, see Section 3.3.1. Its format is the same as that in 6P ADD command, but its contents could be different.

CellOptions: Specifies which types of cells to be listed.

Reserved: Reserved bits. These bits SHOULD be set to zero when sending the message and MUST be ignored upon reception.

Offset: The Offset of the first scheduled cell that is requested. The mechanism assumes cells are ordered according to a rule defined in the SF. The rule MUST always order the cells in the same way.

MaxNumCells: The maximum number of cells to be listed. Node B MAY returns less than MaxNumCells cells, for example if MaxNumCells cells do not fit in the frame.

Figure 22 defines the format of a 6P LIST Response.

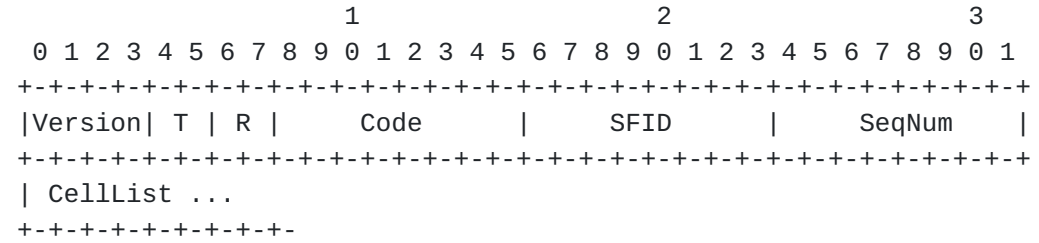


Figure 22: 6P LIST Response Format.

CellList: A list of 0, 1 or multiple 6P Cells.

When receiving a LIST command, node B returns the cells in its schedule that match the CellOptions field as specified in Section 3.2.3.

When node B receives a LIST request, the returned CellList in the 6P Response contains between 1 and MaxNumCells cells, starting from the specified offset. Node B SHOULD include as many cells as fit in the frame. If the response contains the last cell, Node B MUST set the

Code field in the response to RC_EOL (as per Figure 36), indicating to Node A that there no more cells that match the request. Node B MUST return at least one cell, unless the specified Offset is beyond the end of B's cell list in its schedule. If node B has less than Offset cells that match the request, node B returns an empty CellList and a Code field set to RC_EOL.

### 3.3.6. Clearing the Schedule

To clear the schedule between nodes A and B (for example after a schedule inconsistency is detected), node A issues a CLEAR command. The Type field (T) is set to 6P Request. The Code field is set to CLEAR. Figure 23 defines the format of a 6P CLEAR Request.

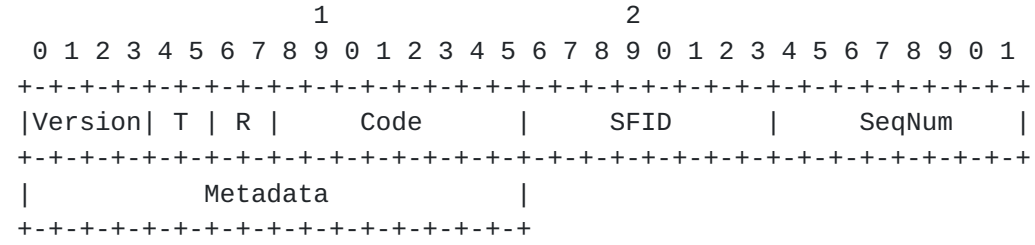


Figure 23: 6P CLEAR Request Format.

Metadata: Same usage as for the 6P ADD command, see Section 3.3.1. Its format is the same as that in 6P ADD command, but its contents could be different.

Figure 24 defines the format of a 6P CLEAR Response.

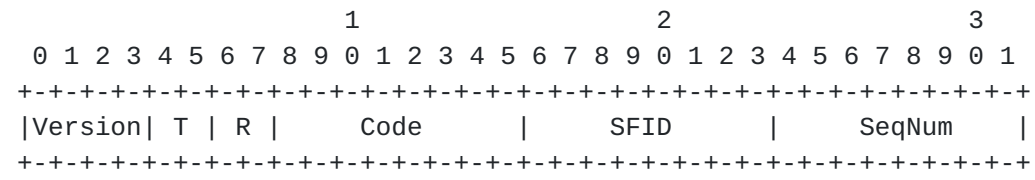


Figure 24: 6P CLEAR Response Format.

When a 6P CLEAR command is issued from node A to node B, both nodes A and B MUST remove all the cells scheduled between them. That is, node A MUST remove all the cells scheduled with node B, and node B MUST remove all the cells scheduled with node A. In a 6P CLEAR command, the SeqNum MUST NOT be checked. In particular, even if the request contains a SeqNum value that would normally cause node B to detect a schedule mismatch, the transaction MUST NOT be aborted. Upon 6P CLEAR completion, the value of SeqNum MUST be reset to 0.

### 3.3.7. Generic Signaling Between SFs

The 6P SIGNAL message allows the SF implementations on two neighbor nodes to exchange generic commands. The payload in a received SIGNAL message is an opaque set of bytes passed unmodified to the SF. How the generic SIGNAL command is used is specified by the SF, and outside the scope of this document. The Type field (T) is set to REQUEST. The Code field is set to SIGNAL. Figure 25 defines the format of a 6P SIGNAL Request.

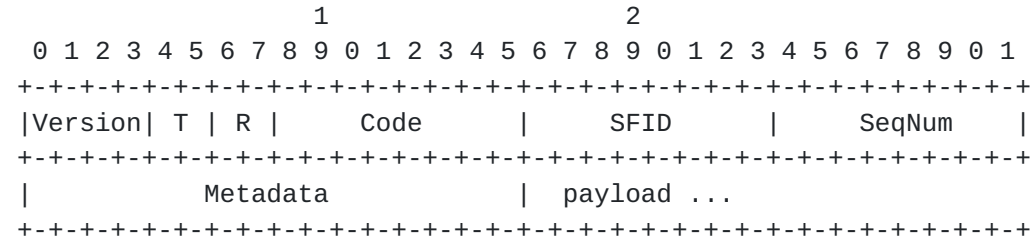


Figure 25: 6P SIGNAL Request Format.

Metadata: Same usage as for the 6P ADD command, see Section 3.3.1. Its format is the same as that in 6P ADD command, but its contents could be different.

Figure 26 defines the format of a 6P SIGNAL Response.

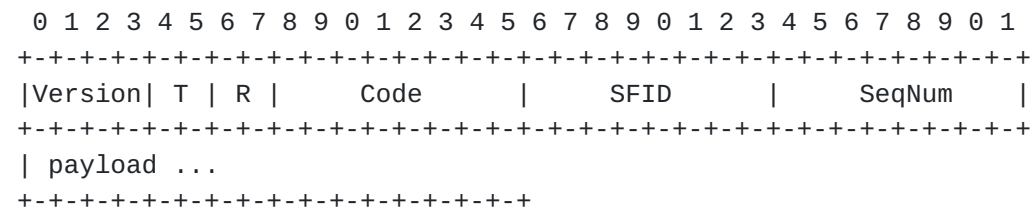


Figure 26: 6P SIGNAL Response Format.

## 3.4. Protocol Functional Details

### 3.4.1. Version Checking

All messages contain a Version field. If multiple Versions of the 6P protocol have been defined (in future specifications for Version values different from 0), a node MAY implement multiple protocol versions at the same time. When a node receives a 6P message with a Version number it does not implement, the node MUST reply with a 6P Response with a Return Code field set to RC_VERSION. The format of this 6P Response message MUST be compliant with Version 0 and MUST be supported by all future versions of the protocol. This ensures that,

when node B sends a 6P Response to node A indicating it does not implement the 6P version in the 6P Request, node A can successfully parse that response.

In case a node supports a version number received in a 6P Request message, the Version field in the 6P Response MUST be the same as the Version field in the corresponding 6P Request. Similarly, in a 3-step transaction, the Version field in the 6P Confirmation MUST match that of the 6P Request and 6P Response in the same transaction.

### 3.4.2. SFID Checking

All messages contain an SFID field. A node MAY support multiple SFs at the same time. When receiving a 6P message with an unsupported SFID, a node MUST reply with a 6P Response and a return code of RC_SFID. The SFID field in the 6P Response MUST be the same as the SFID field in the corresponding 6P Request. In a 3-step transaction, the SFID field in the 6P Confirmation MUST match that of the 6P Request and 6P Response in the same transaction.

### 3.4.3. Concurrent 6P Transactions

Only a single 6P Transaction between two neighbors, in a given direction, can take place at the same time. That is, a node MUST NOT issue a new 6P Request to a given neighbor before having received the 6P Response for a previous request to that neighbor, except when the previous 6P Transaction has timed out. If a node receives a 6P Request from a given neighbor before having sent the 6P Response to the previous 6P Request from that neighbor, it MUST send back a 6P Response with a return code of RC_RESET (as per Figure 36). A node receiving RC_RESET code MUST abort the transaction and consider it never happened.

Nodes A and B MAY support having two transactions going on at the same time, one in each direction. Similarly, a node MAY support concurrent 6P Transactions from different neighbors. In this case, the cells involved in an ongoing 6P Transaction MUST be locked until the transaction finishes. For example, in Figure 1, node C can have a different ongoing 6P Transaction with nodes B and R. In case a node does not have enough resources to handle concurrent 6P Transactions from different neighbors it MUST reply with a 6P Response with return code RC_BUSY (as per Figure 36). In case the requested cells are locked, it MUST reply to that request with a 6P Response with return code RC_LOCKED (as per Figure 36). The node receiving RC_BUSY or a RC_LOCKED MAY implement a retry mechanism, defined by the SF.

### 3.4.4. 6P Timeout

A timeout occurs when the node sending the 6P Request has not received the 6P Response within a specified amount of time determined by the SF. In a 3-step transaction, a timeout also occurs when the node sending the 6P Response has not received the 6P Confirmation. The value of the 6P Timeout should be larger than the longest possible time it can take for the exchange to finish. The value of the 6P Timeout hence depends on the number of cells scheduled between the neighbor nodes, the maximum number of link-layer retransmissions, etc. The SF MUST determine the value of the timeout. The value of the timeout is out of scope of this document.

### 3.4.5. Aborting a 6P Transaction

In case the receiver of a 6P Request fails during a 6P Transaction and it is unable to complete it, it SHOULD reply to that Request with a 6P Response with return code RC_RESET. Upon receiving this 6P Response, the initiator of the 6P Transaction MUST consider the 6P Transaction as failed.

Similarly, in the case of 3-step transaction, when the receiver of a 6P Response fails during the 6P Transaction and is unable to complete it, it MUST reply to that 6P Response with a 6P Confirmation with return code RC_RESET. Upon receiving this 6P Confirmation, the sender of the 6P Response MUST consider the 6P Transaction as failed.

### 3.4.6. SeqNum Management

The SeqNum is the field in the 6top IE header used to match Request, Response and Confirmation. The SeqNum is used to detect and handle duplicate commands (Section 3.4.6.1) and schedule inconsistencies (Section 3.4.6.2). Each node remembers the last used SeqNum for each neighbor. That is, a node stores as many SeqNum values as it has neighbors. In the remainder of this section, we describe the use of SeqNum between two neighbors; the same happens for each other neighbor, independently.

When a node resets or after a CLEAR transaction, it MUST reset SeqNum to 0. The 6P Response and 6P Confirmation for a transaction MUST use the same SeqNum value as that in the Request. After every transaction, the SeqNum MUST be incremented by exactly 1.

Specifically, if node A receives the link-layer acknowledgment for its 6P Request, it commits to incrementing the SeqNum by exactly 1 after the 6P Transaction ends. This ensure that, at the next 6P Transaction where it sends a 6P Request, that 6P Request will have a different SeqNum.

Similarly, node B increments the SeqNum by exactly 1 after having received the link-layer acknowledgment for the 6P Response (2-step 6P Transaction), or after having sent the link-layer acknowledgment for the 6P Confirmation (3-step 6P Transaction) .

The SeqNum MUST be implemented as a lollipop counter: it rolls over from 0xFF to 0x01 (not to 0x00). This is used to detect that a neighbor reset. Figure 27 lists the possible values of the SeqNum.

| Value | Meaning |
|---|---|
| 0x00 | Clear or Reset |
| 0x01-0xFF | Lollipop Counter values |

Figure 27: Possible values of SeqNum.

#### 3.4.6.1. Detecting and Handling Duplicate 6P Messages

All 6P commands are link-layer acknowledged. A duplicate message means that a node receives a second 6P Request, Response or Confirmation. This happens when the link-layer acknowledgment is not received, and a link-layer retransmission happens. Duplicate messages are normal and unavoidable.

Figure 28 shows an example 2-step transaction in which Node A receives a duplicate 6P Response.

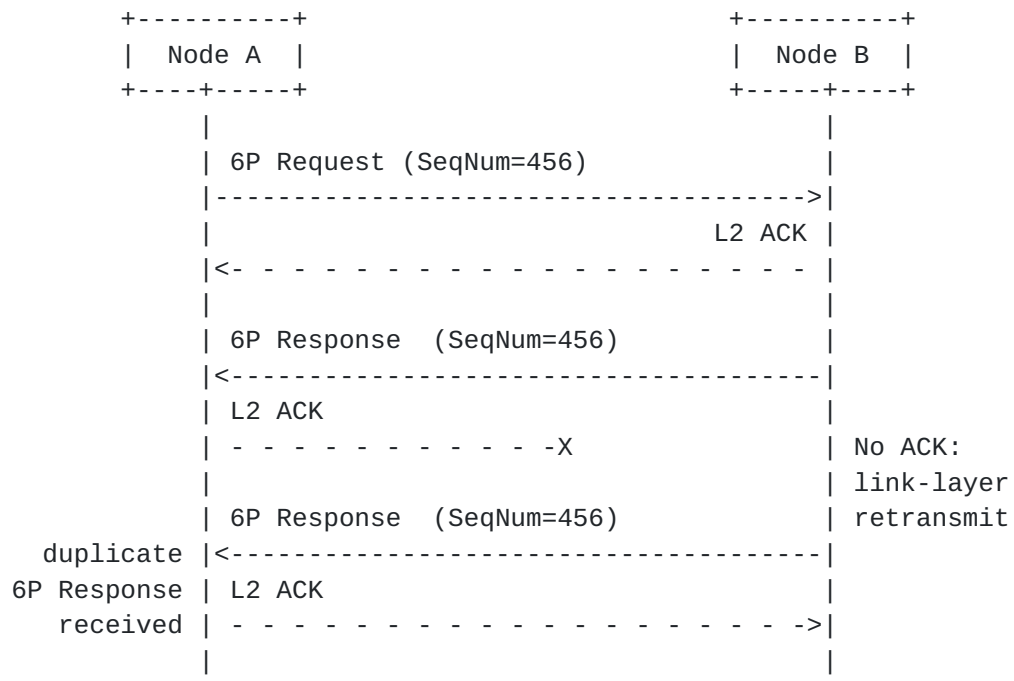


Figure 28: Example duplicate 6P message.

Figure 29 shows example 3-step transaction in which Node A receives a out-of-order duplicate 6P Response after having sent a 6P Confirmation.

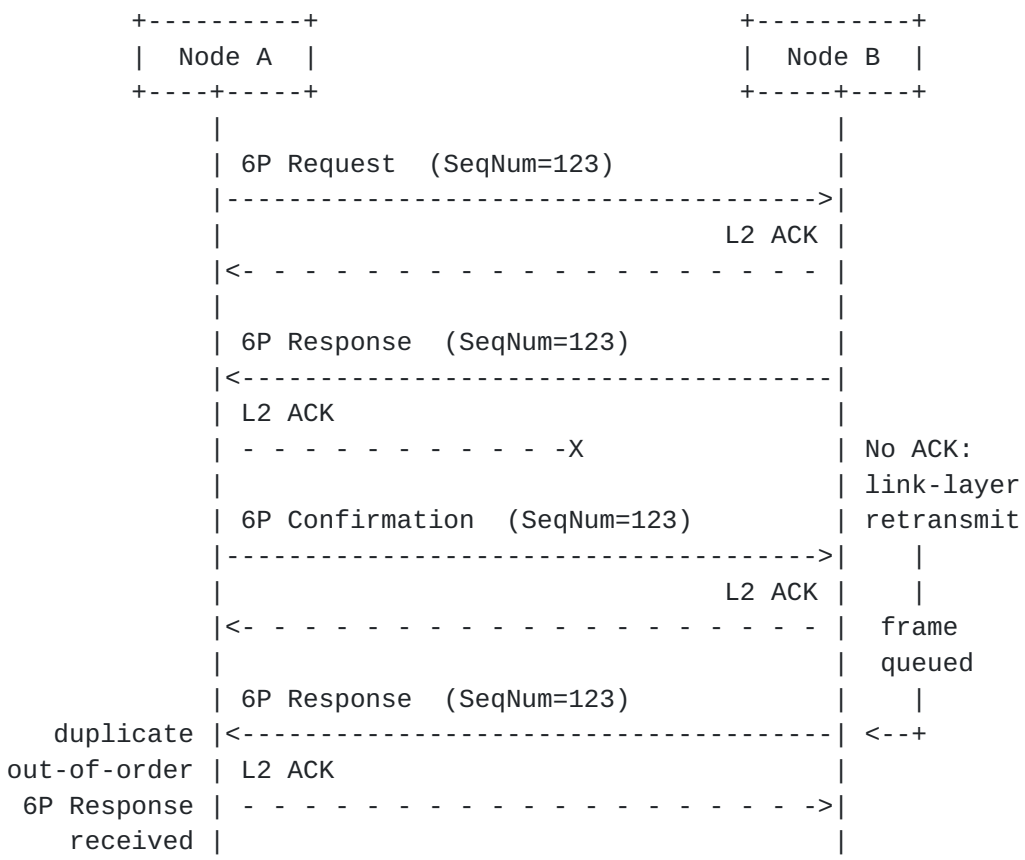


Figure 29: Example out-of-order duplicate 6P message.

A node detects a duplicate 6P message when it has the same SeqNum and type as the last frame received from the same neighbor. When receiving a duplicate 6P message, a node MUST send a link-layer acknowledgment, but MUST silently ignore it at the 6top sublayer.

#### 3.4.6.2. Detecting and Handling a Schedule Inconsistency

A schedule inconsistency happens when the schedules of nodes A and B are inconsistent. For example, when node A has a transmit cell to node B, but node B isn't listening to node A on that cell. A schedule inconsistency results in loss of connectivity.

The SeqNum field, which is present in each 6P message, is used to detect an inconsistency. Given that the SeqNum field increments by 1 at each message. A node computes the expected SeqNum field for the next 6P Transaction. If a node receives a 6P Request with a SeqNum value that is not the expected on, it has detected an inconsistency.

There are at least 2 cases in which a schedule inconsistency happens.

The first case is when a node loses state, for example when power cycled. In that case, its SeqNum value is reset to 0. Since the SeqNum is a lollipop counter, its neighbor detects an inconsistency at the next 6P transaction. This is illustrated in Figure 30.

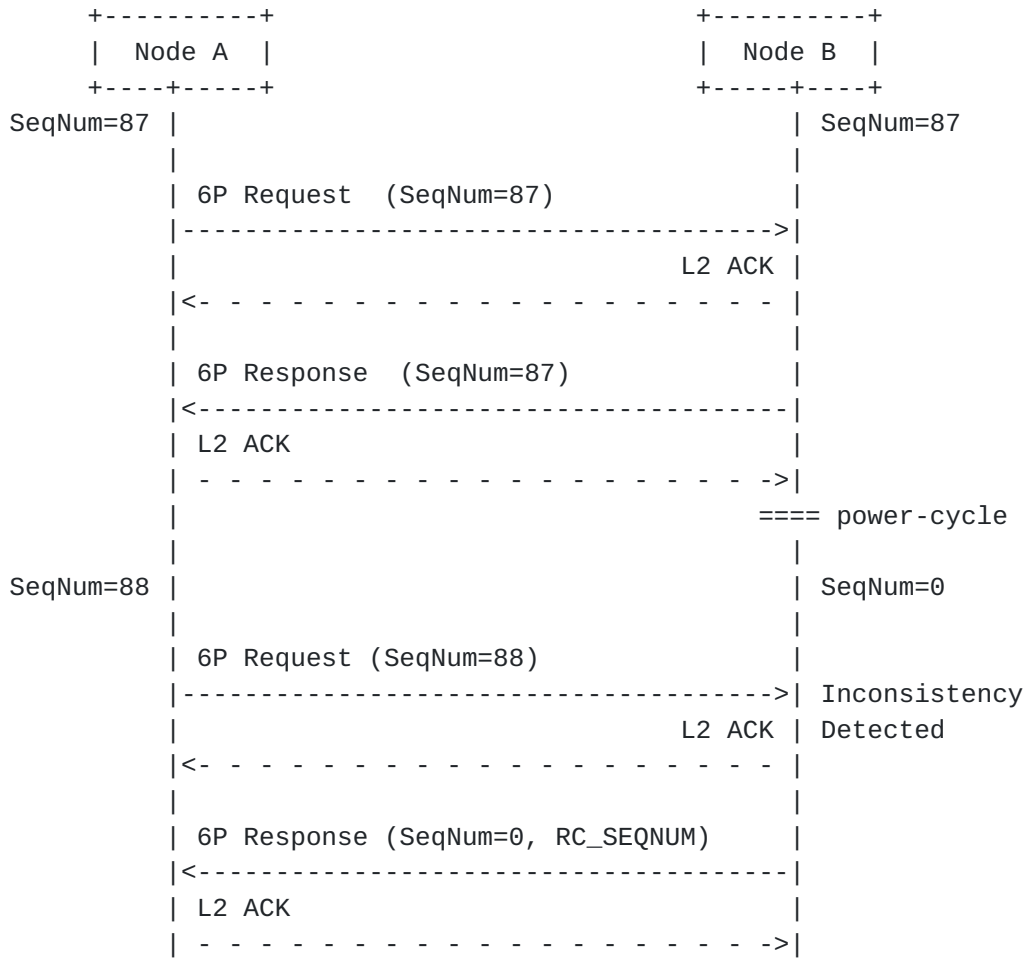


Figure 30: Example of inconsistency because of node reset.

The second case is when the maximum number of link-layer retransmissions is reached on the 6P Response of a 2-step transaction (or equivalently on a 6P Confirmation of a 3-step transaction). This is illustrated in Figure 31.

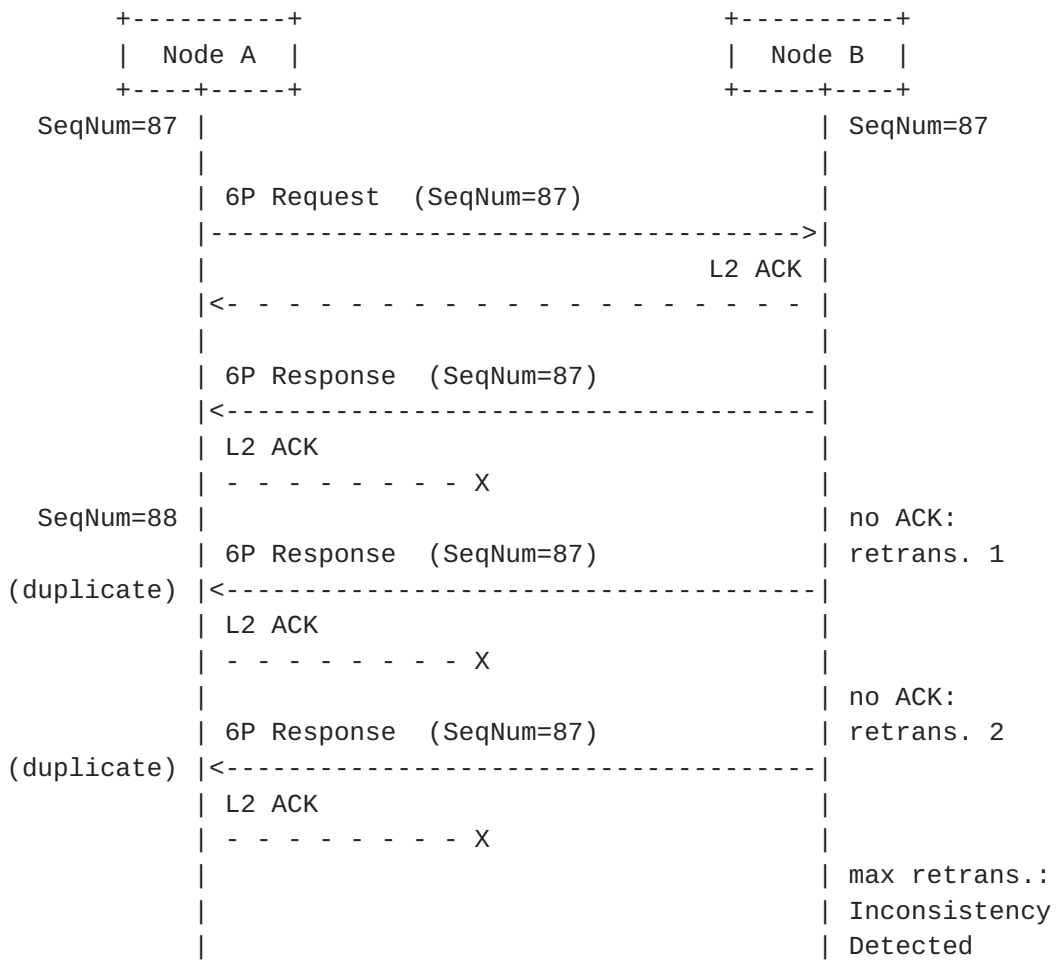


Figure 31: Example of inconsistency because of maximum link-layer retransmissions (here 2).

In both cases, node B detects the inconsistency.

If the inconsistency is detected during a 6P Transaction (Figure 30), the node that has detected it MUST send back a 6P Response or 6P Confirmation with an error code of RC_SEQNUM. In this 6P Response or 6P Confirmation, the SeqNum field MUST be set to the value of the sender of the message (to 0 in Figure 30).

The SF of the node which has detected the inconsistency MUST define how to handle the inconsistency. A first possibility is to issue a 6P CLEAR request to clear the schedule, and rebuild. A second possibility is to issue a 6P LIST request to retrieve the schedule. A third possibility is to internally "roll-back" the schedule. How to handle an inconsistency is out of scope of this document. The SF defines how to handle an inconsistency.

### 3.4.7. Handling Error Responses

A return code marked as Yes in the "Is Error" column in Figure 36 indicates an error. When a node receives a 6P Response or 6P Confirmation with such an error, it MUST consider the 6P Transaction as failed. In particular, if this was a response to a 6P ADD/DELETE/RELOCATE Request, the node MUST NOT add/delete/relocate any of the cells involved in this 6P Transaction. Similarly, a node sending a 6P Response or a 6P Confirmation with an error code MUST NOT add/delete/relocate any cells as part of that 6P Transaction. Defining what to do after an error has occurred is out of scope of this document. The SF defines what to do after an error has occurred.

## 3.5. Security

6P messages are secured through link-layer security. When link-layer security is enabled, the 6P messages MUST be secured. This is possible because 6P messages are carried as Payload IE.

# 4. Requirements for 6top Scheduling Functions (SF)

## 4.1. SF Identifier (SFID)

Each SF has a 1-byte identifier. Section 6.2.5 defines the rules for applying for an SFID.

## 4.2. Requirements for an SF

The specification for an SF

- MUST specify an identifier for that SF.
- MUST specify the rule for a node to decide when to add/delete one or more cells to a neighbor.
- MUST specify the rule for a Transaction source to select cells to add to the CellList field in the 6P ADD Request.
- MUST specify the rule for a Transaction destination to select cells from CellList to add to its schedule.
- MUST specify a value for the 6P Timeout, or a rule/equation to calculate it.
- MUST specify the rule for ordering cells.
- MUST specify a meaning for the "Metadata" field in the 6P ADD Request.
- MUST specify the SF behavior of a node when it boots.
- MUST specify how to handle a schedule inconsistency.
- MUST specify what to do after an error has occurred (either the node sent a 6P Response with an error code, or received one).

- MUST specify the list of statistics to gather. An example statistic is the number of transmitted frames to each neighbor. In case the SF requires no statistics to be gathered, the specific of the SF MUST explicitly state so.

- SHOULD clearly state the application domain the SF is created for.
- SHOULD contain examples which highlight normal and error scenarios.
- SHOULD contain a list of current implementations, at least during the I-D state of the document, per [RFC6982].
- SHOULD contain a performance evaluation of the scheme, possibly through references to external documents.
- SHOULD define the format of the SIGNAL command payload and its use.

- MAY redefine the format of the CellList field.
- MAY redefine the format of the CellOptions field.
- MAY redefine the meaning of the CellOptions field.

## 5. Security Considerations

6P messages are carried inside 802.15.4 Payload Information Elements (IEs). Those Payload IEs are encrypted and authenticated at the link layer through CCM* [CCM-Star] 6P benefits from the same level of security as any other Payload IE. The 6P protocol does not define its own security mechanisms. A key management solution is out of scope for this document. The 6P protocol will benefit for the key management solution used in the network.

## 6. IANA Considerations

### 6.1. IETF IE Subtype '6P'

This document adds the following number to the "IEEE Std 802.15.4 IETF IE subtype IDs" registry defined by [RFC8137]:

| Subtype | Name | Reference |
|---|---|---|
| IANA_6TOP_SUBIE_ID | 6P | RFCXXXX |

Figure 32: IETF IE Subtype '6P'.

### 6.2. 6TiSCH parameters sub-registries

This section defines sub-registries within the "IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) parameters" registry, hereafter referred to as the "6TiSCH parameters" registry. Each sub-registry is described in a subsection.

#### 6.2.1. 6P Version Numbers

The name of the sub-registry is "6P Version Numbers".

A Note included in this registry should say: "In the 6top Protocol (6P) [RFCXXXX] there is a field to identify the version of the protocol. This field is 4 bits in size."

Each entry in the sub-registry must include the Version in the range 0-15, and a reference to the 6P version's documentation.

The initial entry in this sub-registry is as follows:

| Version | Reference |
|---|---|
| 0 | RFCXXXX |

Figure 33: 6P Version Numbers.

All other Version Numbers are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC8126].

#### 6.2.2. 6P Message Types

The name of the sub-registry is "6P Message Types".

A Note included in this registry should say: "In the 6top Protocol (6P) version 0 [RFCXXXX], there is a field to identify the type of message. This field is 2 bits in size."

Each entry in the sub-registry must include the Type in the range b00-b11, the corresponding Name, and a reference to the 6P message type's documentation.

Initial entries in this sub-registry are as follows:

| Type | Name | Reference |
|---|---|---|
| b00 | REQUEST | RFCXXXX |
| b01 | RESPONSE | RFCXXXX |
| b10 | CONFIRMATION | RFCXXXX |

Figure 34: 6P Message Types.

All other Message Types are Reserved.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC8126].

### 6.2.3. 6P Command Identifiers

The name of the sub-registry is "6P Command Identifiers".

A Note included in this registry should say: "In the 6top Protocol (6P) version 0 [RFCXXXX], there is a Code field which is 8 bits in size. In a 6P Request, the value of this Code field is used to identify the command."

Each entry in the sub-registry must include the Identifier in the range 0-255, the corresponding Name, and a reference to the 6P command identifier's documentation.

Initial entries in this sub-registry are as follows:

| Identifier | Name | Reference |
|---|---|---|
| 0 | Reserved | |
| 1 | ADD | RFCXXXX |
| 2 | DELETE | RFCXXXX |
| 3 | RELOCATE | RFCXXXX |
| 4 | COUNT | RFCXXXX |
| 5 | LIST | RFCXXXX |
| 6 | SIGNAL | RFCXXXX |
| 7 | CLEAR | RFCXXXX |
| 8-254 | Unassigned | |
| 255 | Reserved | |

Figure 35: 6P Command Identifiers.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC8126].

### 6.2.4. 6P Return Codes

The name of the sub-registry is "6P Return Codes".

A Note included in this registry should say: "In the 6top Protocol (6P) version 0 [RFCXXXX], there is a Code field which is 8 bits in size. In a 6P Response or 6P Confirmation, the value of this Code field is used to identify the return code."

Each entry in the sub-registry must include the Code in the range 0-255, the corresponding Name, the corresponding Description, and a reference to the 6P return code's documentation.

Initial entries in this sub-registry are as follows:

| Code | Name | Description | Is Error? |
|---|---|---|---|
| 0 | RC_SUCCESS | operation succeeded | No |
| 1 | RC_EOL | end of list | No |
| 2 | RC_ERROR | generic error | Yes |
| 3 | RC_RESET | critical error, reset | Yes |
| 4 | RC_VERSION | unsupported 6P version | Yes |
| 5 | RC_SFID | unsupported SFID | Yes |
| 6 | RC_SEQNUM | schedule inconsistency | Yes |
| 7 | RC_CELLLIST | cellList error | Yes |
| 8 | RC_BUSY | busy | Yes |
| 9 | RC_LOCKED | cells are locked | Yes |

Figure 36: 6P Return Codes.

All other Message Types are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC8126].

### 6.2.5. 6P Scheduling Function Identifiers

6P Scheduling Function Identifiers.

A Note included in this registry should say: "In the 6top Protocol (6P) version 0 [RFCXXXX], there is a field to identify the scheduling function to handle the message. This field is 8 bits in size."

Each entry in the sub-registry must include the SFID in the range 0-255, the corresponding Name, and a reference to the 6P Scheduling Function's documentation.

The initial entry in this sub-registry is as follows:

| SFID | Name | Reference |
|---|---|---|
| 0 | Scheduling Function Zero | draft-ietf-6tisch-6top-sf0 |

Figure 37: SF Identifiers (SFID).

All other Message Types are Unassigned.

The IANA policy for future additions to this sub-registry depends on the value of the SFID, as defined in Figure 38. These specifications must follow the guidelines of Section 4.

| Range | Registration Procedures |
|---|---|
| 0-127 | IETF Review or IESG Approval |
| 128-255 | Expert Review |

Figure 38: SF Identifier (SFID): Registration Procedures.

### 6.2.6. 6P CellOptions bitmap

The name of the sub-registry is "6P CellOptions bitmap".

A Note included in this registry should say: "In the 6top Protocol (6P) version 0 [RFCXXXX], there is an optional CellOptions field which is 8 bits in size."

Each entry in the sub-registry must include the bit position in the range 0-7, the corresponding Name, and a reference to the bit's documentation.

Initial entries in this sub-registry are as follows:

| bit | Name | Reference |
|---|---|---|
| 0 | TX (Transmit) | RFCXXXX |
| 1 | RX (Receive) | RFCXXXX |
| 2 | SHARED | RFCXXXX |
| 3-7 | Reserved | |

Figure 39: 6P CellOptions bitmap.

All other Message Types are Reserved.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC8126].

# 7. References

## 7.1. Normative References

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

[RFC8137] Kivinen, T. and P. Kinney, "IEEE 802.15.4 Information Element for the IETF", RFC 8137, DOI 10.17487/RFC8137, May 2017, <https://www.rfc-editor.org/info/rfc8137>.

## 7.2. Informative References

[CCM-Star] Struik, R., "Formal Specification of the CCM* Mode of Operation, IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs).", September 2005.

[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>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

## Appendix A. Recommended Structure of an SF Specification

The following section structure for a SF document is RECOMMENDED:

- Introduction
- Scheduling Function Identifier
- Rules for Adding/Deleting Cells
- Rules for CellList
- 6P Timeout Value
- Rule for Ordering Cells
- Meaning of the Metadata Field
- Node Behavior at Boot
- Schedule Inconsistency Handling
- 6P Error Handling
- Examples
- Implementation Status
- Security Considerations
- IANA Considerations

## 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".

First F-Interop ETSI 6TiSCH plugtests: 6P is one of the protocols addressed during the First F-Interop ETSI 6TiSCH plugtests organized on 14-15 July 2017 in Prague, Czech Republic. It was attended by 14 entities, which 4-5 independent implementation bases.

ETSI 6TiSCH/6lo plugtests: 6P was one of the protocols addressed during the ETSI 6TiSCH #3 plugtests organized on 15-17 July 2016 in Berlin, Germany. 15 entities participated in this event, verifying the compliance and interoperability of their implementation of 6P. This event happened under NDA, so neither the name of the entities nor the test results are public. This event is, however, a clear indication of the maturity of 6P, and the interest it generates. More information about the event at http://www.etsi.org/news-events/events/1077-6tisch-6lo-plugtests.

ETSI 6TiSCH #2 plugtests: 6P was one of two protocols addressed during the ETSI 6TiSCH #2 plugtests organized on 2-4 February 2016 in Paris, France. 14 entities participated in this event, verifying the compliance and interoperability of their implementation of 6P. This event happened under NDA, so neither the name of the entities nor the test results are public. This event is, however, a clear indication of the maturity of 6P, and the interest it generates. More information about the event at http://www.etsi.org/news-events/events/1022-6TiSCH-2-plugtests.

OpenWSN: 6P is 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/.

F-Interop Interoperability/Conformance Testing tool The F-Interop project is putting together an online tool to conduct online and remote interoperability/conformance tests. 6P is one of the supported protocols.

6TiSCH simulator The 6TiSCH simulator is a Python-based high-level simulator which implements 6P and is built to evaluate the performance of differents SFs. More information at https://bitbucket.org/6tisch/simulator/.

Wireshark Dissector: A Wireshark dissector for 6P is implemented under a BSD open-source license. It is developed and maintained at https://github.com/openwsn-berkeley/dissectors/, and regularly merged into the main Wireshark repository. Please see the Wireshark documentation to see what version of 6P it supports.

## Appendix C. [TEMPORARY] Changelog

- draft-ietf-6tisch-6top-protocol-09
  - Requiring version 0 in RC_VERSION response.
  - Adding L2 ACK in figures.
  - Inconsistency management update.
  - Moving SF requirements to another section.
  - Moving implementation status to appendix.
  - Fixing typos.
- draft-ietf-6tisch-6top-protocol-08
  - Replacing GEN counter by SeqNum and timeout.
  - Adding SIGNAL command.
  - Adding RC_SEQNUM return code.
  - Clarifying IETF IE usage.
  - Cleaning up error codes.
  - Fixing typos.
- draft-ietf-6tisch-6top-protocol-07
  - Inverting RC_LOCKED and RC_BUSY error codes for concurrent transactions.
  - Adding missing implementations.
  - Fixing references.
  - Fixing typos.
- draft-ietf-6tisch-6top-protocol-06
  - Changing error code from RC_RESET to RC_CELLLIST when deleting unscheduled cells.
  - Fixing typos.
- draft-ietf-6tisch-6top-protocol-05
  - complete reorder of sections. Merged protocol behavior and command description
  - STATUS to COUNT
  - written-out IANA section
  - complete proof-read
- draft-ietf-6tisch-6top-protocol-04

    *  recommendation on which cells to use for 6P traffic
    *  relocation format: added numberofCells field
    *  created separate section about "cell suggestion"
    *  Added RC_ERR_CELLLIST and RC_ERR_EOL error codes
    *  Added example for two step with the failure
    *  Recommended numbers in IANA section
    *  single generation number
    *  IEEE802.15.4 -> IEEE Std 802.15.4 or 802.15.4
    *  complete proof-read

- draft-ietf-6tisch-6top-protocol-03

    *  Added a reference to [RFC8137].
    *  Added the Type field.
    *  Editorial changes (figs, typos, ...)

- draft-ietf-6tisch-6top-protocol-02

    *  Rename COUNT to STATUS
    *  Split LIST to LIST AB and LIST BA
    *  Added generation counters and describing generation tracking of the schedule
    *  Editorial changes (figs, typos, ...)

- draft-ietf-6tisch-6top-protocol-01

    *  Clarifying locking of resources in concurrent transactions
    *  Clarifying return of RC_ERR_BUSY in case of concurrent transactions without enough resources

- draft-ietf-6tisch-6top-protocol-00

    *  Informational to Std track

- draft-wang-6tisch-6top-protocol-00

    *  Editorial overhaul: fixing typos, increasing readability, clarifying figures.
    *  https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/47
    *  https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/54
    *  https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/55
    *  https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/49
    *  https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/53
    *  https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/44
    *  https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/48

    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/43
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/52
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/45
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/51
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/50
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/46
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/41
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/42
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/39
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/40

- draft-wang-6tisch-6top-sublayer-05

    * Specifies format of IE
    * Adds token in messages to match request and response

- draft-wang-6tisch-6top-sublayer-04

    * Renames IANA_6TOP_IE_GROUP_ID to IANA_IETF_IE_GROUP_ID.
    * Renames IANA_CMD and IANA_RC to IANA_6TOP_CMD and IANA_6TOP_RC.
    * Proposes IANA_6TOP_SUBIE_ID with value 0x00 for the 6top sub-IE.

- draft-wang-6tisch-6top-sublayer-03

    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/32/missing-command-list
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/31/missing-command-count
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/30/missing-command-clear
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/37/6top-atomic-transaction-6p-transaction
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/35/separate-opcode-from-rc
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/36/add-length-field-in-ie
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/27/differentiate-rc_err_busy-and
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/29/missing-rc-rc_reset

    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/28/the-sf-must-specify-the-behavior-of-a-mote
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/26/remove-including-their-number
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/34/6of-sf
    * https://bitbucket.org/6tisch/draft-wang-6tisch-6top-protocol/issues/33/add-a-figure-showing-the-negociation

- draft-wang-6tisch-6top-sublayer-02

    * introduces the 6P protocol and the notion of 6top Transaction.
    * introduces the concept of 6OF and its 6OFID.

## Authors' Addresses

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

Email: wangqin@ies.ustb.edu.cn

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

Email: xvilajosana@uoc.edu

Thomas Watteyne
Analog Devices
32990 Alvarado-Niles Road, Suite 910
Union City, CA 94587
USA

Email: thomas.watteyne@analog.com