6TiSCH Operation Sublayer (6top) Protocol (6P) - RFC8480

Qin Wang, Xavier Vilajosana, Thomas Watteyne

Internet Engineering Task Force (IETF)
Request for Comments: 8480
Category: Standards Track
ISSN: 2070-1721

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

November 2018

6TiSCH Operation Sublayer (6top) Protocol (6P)

Abstract

This document defines the "IPv6 over the TSCH mode of IEEE 802.15.4e" (6TiSCH) Operation Sublayer (6top) Protocol (6P), which enables distributed scheduling in 6TiSCH networks. 6P allows neighbor nodes to add/delete Time-Slotted Channel Hopping (TSCH) cells to/on one another. 6P is part of the 6TiSCH Operation Sublayer (6top), the layer just above the IEEE Std 802.15.4 TSCH Medium Access Control layer. 6top is composed of one or more Scheduling Functions (SFs) and the 6top Protocol defined in this document. A 6top SF decides when to add/delete cells, and it triggers 6P Transactions. The definition of SFs is out of scope for this document; however, this document provides the requirements for an SF.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8480.

Table of Contents

1. Introduction

All communication in an "IPv6 over the TSCH mode of IEEE 802.15.4e" (6TiSCH) network is orchestrated by a schedule [RFC7554]. The schedule is composed of cells, each identified by a [slotOffset,channelOffset] (Section 3.2.4). This specification defines the 6TiSCH Operation Sublayer (6top) Protocol (6P), which is terminated by 6top. 6P allows a node to communicate with a neighbor node to add/delete Time-Slotted Channel Hopping (TSCH) cells to/on one another. This results in distributed schedule management in a 6TiSCH network. 6top is composed of one or more Scheduling Functions (SFs) and the 6top Protocol defined in this document. The definition of SFs is out of scope for this document; however, this document provides the requirements for an SF.

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 (also referred to as "commands" in this document) to node "B". We use this example throughout this document: node A always represents the node that issues a 6P Request, and node B represents the node that receives this request.

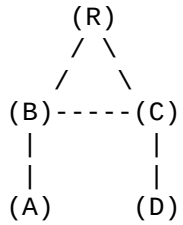

Figure 1: A Simple 6TiSCH Network

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 node B per unit of time exceeds the capacity offered by the TSCH cells it has scheduled to node 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 offered by the TSCH cells it has scheduled to node B, 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" this particular cell to a different [slotOffset,channelOffset] location in the TSCH schedule.

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

The 6top SF defines when to add/delete a cell to/on a neighbor. Different applications require different SFs; this topic is out of scope for 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-6P Transaction basis.

Section 2 describes 6top. Section 3 defines 6P. Section 4 provides guidelines on how to define an SF.

1.1. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. 6TiSCH Operation Sublayer (6top)

As depicted in Figure 2, 6top is the layer just above 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.

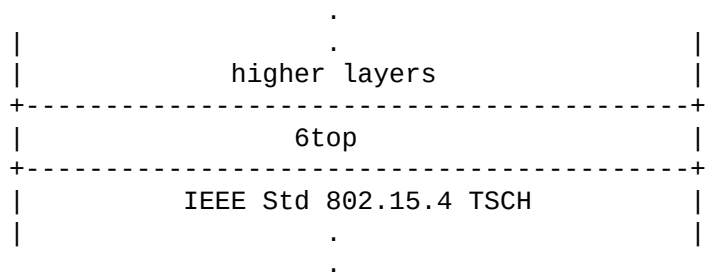

Figure 2: 6top in the Protocol Stack

The roles of 6top are to:

- Terminate 6P, which allows neighbor nodes to communicate to add/delete cells to/on one another.
- Run one or multiple 6top 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 two slotframes, as depicted in Figure 3:

- Slotframe 0 is used for traffic defined in the minimal 6TiSCH configuration. In Figure 3, Slotframe 0 is five slots long, but it can be shorter or longer.
- 6P allocates cells from Slotframe 1. In Figure 3, Slotframe 1 is 10 slots long, but it can be shorter or longer.

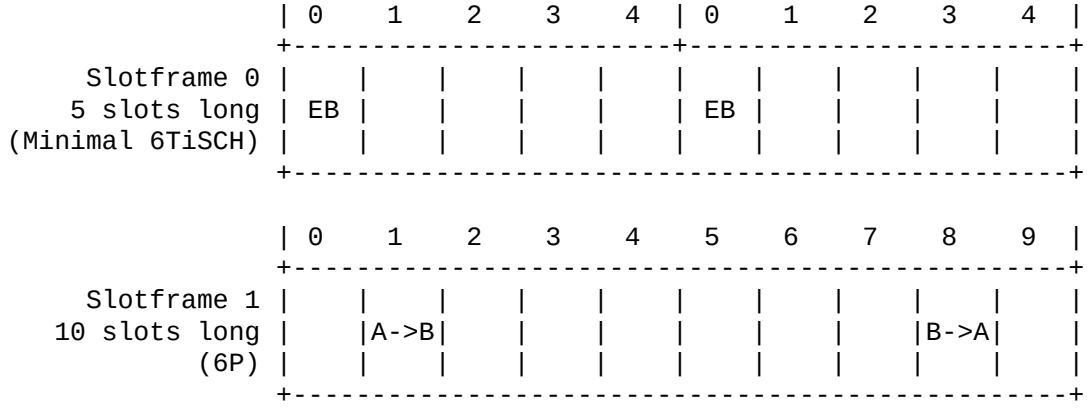

Figure 3: 2-Slotframe Structure when Using 6P alongside the Minimal 6TiSCH Configuration

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 scope for this document.

3. 6top Protocol (6P)

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 particular 6P Transaction is executed between two nodes as a result of an action triggered by one SF. For a 6P Transaction to succeed, both nodes must use the same SF to handle the particular transaction. 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 (1) the cell(s) has been added/deleted/relocated in the schedule of both nodes or (2) the 6P Transaction has failed.

6P messages exchanged between nodes A and B during a 6P Transaction SHOULD be exchanged on non-shared unicast cells ("dedicated" cells) between nodes 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 can cause loss of connectivity. One example is when node A has a transmit cell to node B but node B does not have the corresponding reception cell. To verify consistency, neighbor nodes maintain a sequence number (SeqNum). Neighbor nodes exchange the SeqNum as part of each 6P Transaction to detect a 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 is out of scope for this document.

A 6P Transaction can consist of two or three 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 so that the diagram is easier to understand.

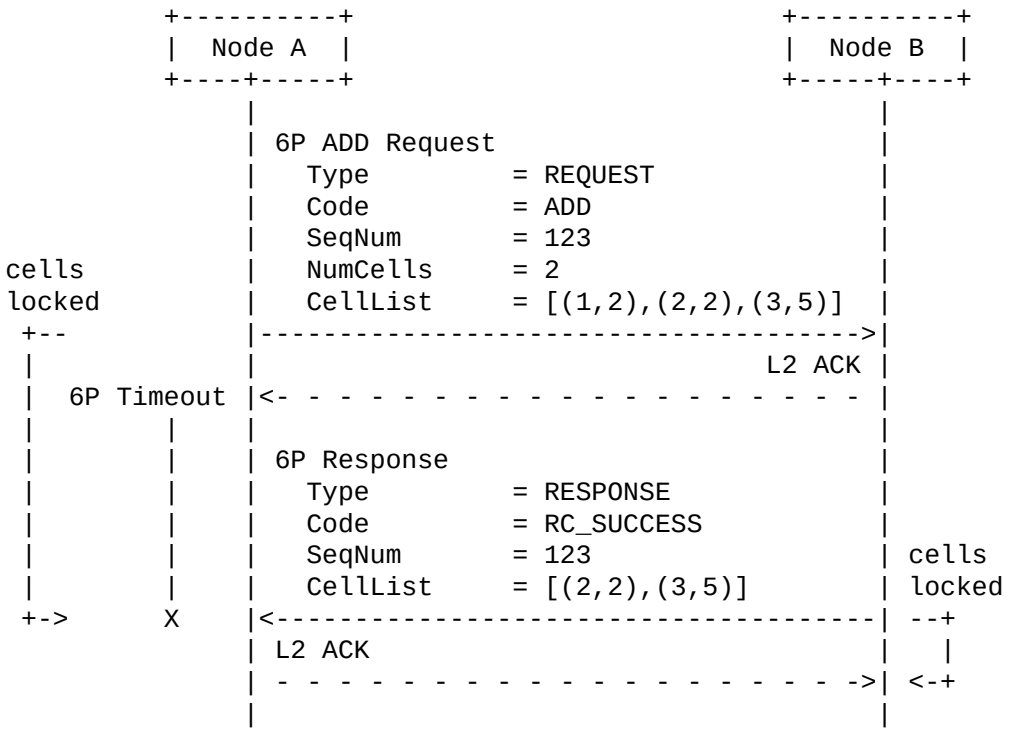

Figure 4: An Example 2-Step 6P Transaction

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

1. The SF running on node A determines that two extra cells need to be scheduled to node B.

2. The SF running on node A selects candidate cells for node B to choose from. Node A MUST select at least as many candidate cells as the number of cells to add. Here, node A selects three candidate cells. Node A locks those candidate cells in its schedule until it receives a 6P Response.

3. Node A sends a 6P ADD Request to node B, indicating that it wishes to add two cells (the "NumCells" value) and specifying the list of three 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 (i.e., receiving the link-layer acknowledgment), node A starts a 6P Timeout to abort the 6P Transaction in the event that no response is received from node B.

5. The SF running on node B selects two out of the three cells from the CellList of the 6P ADD Request. Node B locks those cells in its schedule until the transmission is successful (i.e., node B receives a link-layer ACK from node A). Node B sends back a 6P Response to node A, indicating the cells it has selected. The response is link-layer acknowledged by node A.

6. Upon completion of this 6P Transaction, two cells from node A to node B have been added to the TSCH schedule of both nodes A and B.

7. An inconsistency in the schedule can happen if the 6P Timeout expires when the 6P Response is in the air, if the last link-layer ACK for the 6P Response is lost, or if one of the nodes is power-cycled during the transaction. 6P provides an inconsistency detection mechanism to cope with such situations; see Section 3.4.6.2 for details.

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 so that the diagram is easier to understand.

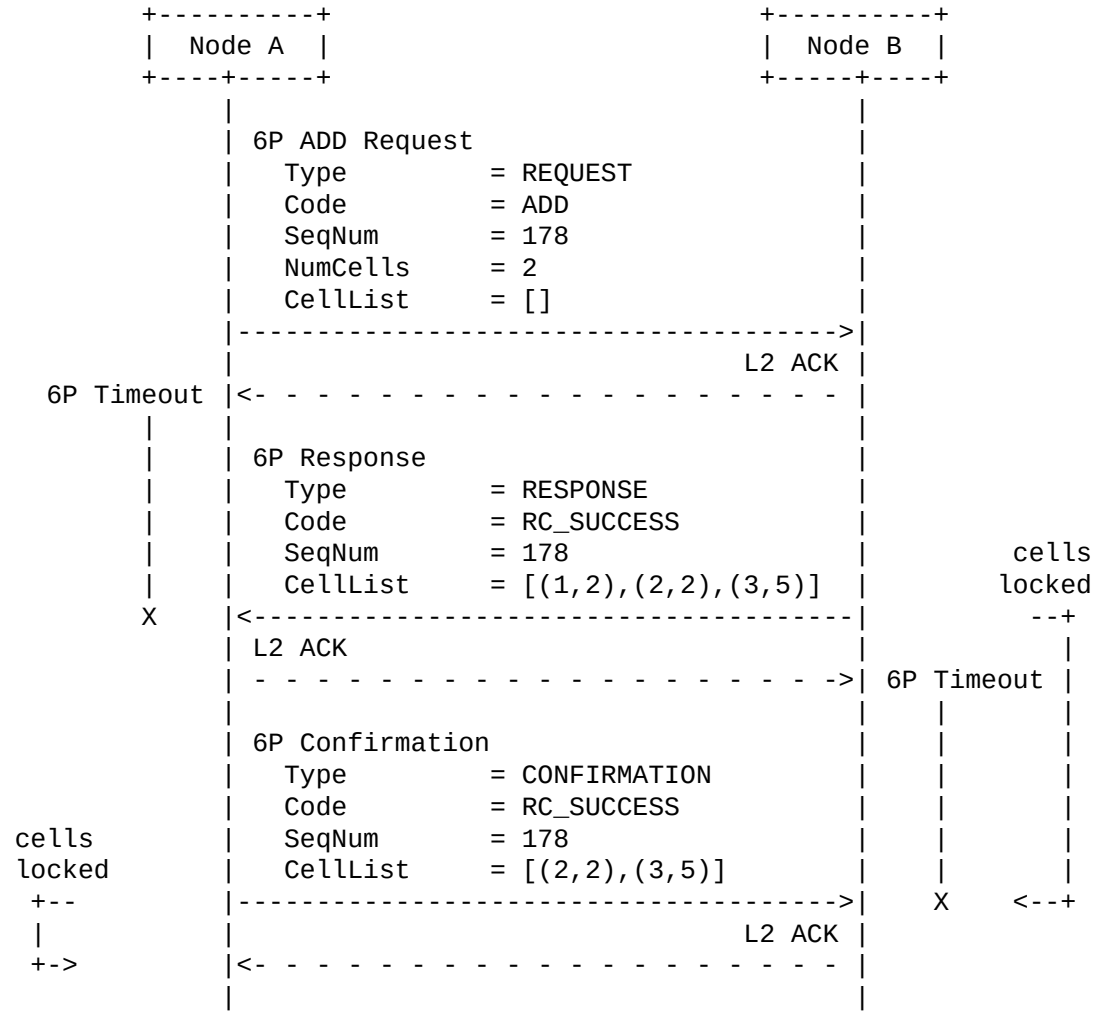

Figure 5: An Example 3-Step 6P Transaction

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

1. The SF running on node A determines that two extra cells need to be scheduled to node B. The SF uses a 3-step transaction, so it does not select candidate cells.

2. Node A sends a 6P ADD Request to node B, indicating that it wishes to add two 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 the event that no 6P Response is received from node B.

4. The SF running on node B selects three candidate cells and locks them. Node B sends back a 6P Response to node A, indicating the three cells it has 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 the event that no 6P Confirmation is received from node A.

6. The SF running on node A selects two cells from the CellList field in the 6P Response and locks them. 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 the 6P Transaction, two cells from node A to node B have been added to the TSCH schedule of both nodes A and B.

8. An inconsistency in the schedule can happen if the 6P Timeout expires when the 6P Confirmation is in the air, if the last link-layer ACK for the 6P Confirmation is lost, or if one of the nodes is power-cycled during the transaction. 6P provides an inconsistency detection mechanism to cope with such situations; see Section 3.4.6.2 for details.

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 802.15.4 Payload Information Element (IE) [IEEE802154]. The messages are encapsulated within 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 IEs, IEEE bit/byte ordering applies. Bits within each field in the "6top IE" subtype 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 SUBID_6TOP. 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.

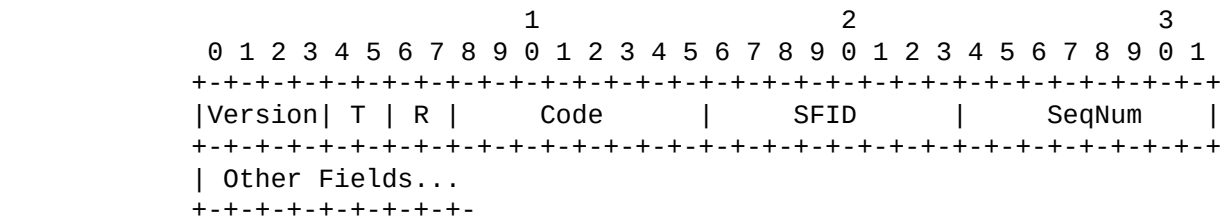

Figure 6: Generic 6P Message Format

6P Version (Version): The version of 6P. Only version 0 is defined in this document. Future specifications may define subsequent versions of 6P.

Type (T): The 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 has a Type value of REQUEST. Section 6.2.3 lists the 6P command identifiers. The Code field contains a 6P return code when the 6P message has a Type value of 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: The 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 for each new 6P Request issued to the same neighbor and using the same SF. 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 are detailed in Section 3.3.

6P Request, 6P Response, and 6P Confirmation messages for a given transaction MUST share the same Version, SFID, and SeqNum values.

Future versions of the 6P message SHOULD maintain the format of the 6P Version, Type, and Code fields for backward compatibility.

3.2.3. 6P CellOptions

An 8-bit 6P CellOptions bitmap is present in the following 6P Requests: ADD, DELETE, COUNT, LIST, and RELOCATE. The format and meaning of this field MAY be redefined by the SF; the routine that parses this field is therefore associated with a specific SF.

- 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 RELOCATE Request, the 6P CellOptions bitmap is used to specify what type of cell to relocate.
- In the 6P COUNT and LIST Requests, the 6P CellOptions bitmap is used as a selector of a particular type of cells.

The content of the 6P CellOptions bitmap applies to all elements in the CellList field. The possible values of the 6P CellOptions are as follows:

- TX = 1 (resp. 0) refers to macTxType = TRUE (resp. FALSE) in the macLinkTable of 802.15.4 [IEEE802154].
- RX = 1 (resp. 0) refers to macRxType = TRUE (resp. FALSE) in the macLinkTable of 802.15.4.
- S = 1 (resp. 0) refers to macSharedType = TRUE (resp. FALSE) in the macLinkTable of 802.15.4.

Section 6.2.6 provides the format of the 6P CellOptions bitmap; this format applies unless redefined by the SF. Figure 7 shows the meaning of the 6P CellOptions bitmap for the 6P ADD, DELETE, and RELOCATE Requests (unless redefined by the SF). Figure 8 shows the meaning of the 6P CellOptions bitmap for the 6P COUNT and LIST Requests (unless redefined by the SF).

Note: Here, we assume that node A issues the 6P command to node B.

CellOptions Value	The type of cells B adds/deletes/relocates to its schedule when receiving a 6P ADD/DELETE/RELOCATE Request from A
TX=0,RX=0,S=0	Invalid combination. RC_ERR is returned
TX=1,RX=0,S=0	Add/delete/relocate RX cells at B (TX cells at A)
TX=0,RX=1,S=0	Add/delete/relocate TX cells at B (RX cells at A)
TX=1,RX=1,S=0	Add/delete/relocate TX|RX cells at B (and at A)
TX=0,RX=0,S=1	Invalid combination. RC_ERR is returned
TX=1,RX=0,S=1	Add/delete/relocate RX|SHARED cells at B (TX|SHARED cells at A)
TX=0,RX=1,S=1	Add/delete/relocate TX|SHARED cells at B (RX|SHARED cells at A)
TX=1,RX=1,S=1	Add/delete/relocate TX|RX|SHARED cells at B (and at A)

Figure 7: Meaning of the 6P CellOptions Bitmap for the 6P ADD, DELETE, and RELOCATE Requests

Note: Here, we assume that node A issues the 6P command to node B.

CellOptions Value	The type of cells B selects from its schedule when receiving a 6P COUNT or LIST Request from A, from all the cells B has scheduled with A
TX=0,RX=0,S=0	All cells
TX=1,RX=0,S=0	All cells marked as RX only
TX=0,RX=1,S=0	All cells marked as TX only
TX=1,RX=1,S=0	All cells marked as TX and RX only
TX=0,RX=0,S=1	All cells marked as SHARED (regardless of TX, RX)
TX=1,RX=0,S=1	All cells marked as RX and SHARED only
TX=0,RX=1,S=1	All cells marked as TX and SHARED only
TX=1,RX=1,S=1	All cells marked as TX, RX, and SHARED

Figure 8: Meaning of the 6P CellOptions Bitmap for the 6P COUNT and LIST Requests

The CellOptions constitute an opaque set of bits, sent unmodified to the SF. The SF MAY redefine the format and meaning of the CellOptions field.

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 or more 6P Cells as defined in Figure 9. The content of the CellOptions field specifies the options associated with all cells in the CellList. This necessarily means that the same options are associated with all cells in the CellList.

A 6P Cell is a 4-byte field; its default format is:

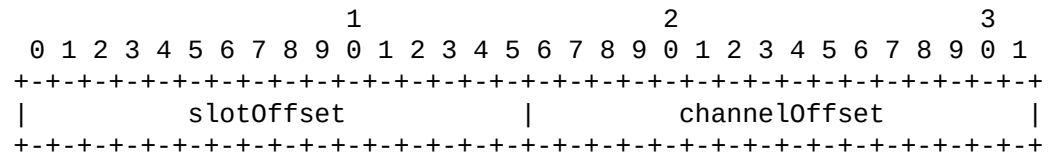

Figure 9: 6P Cell Format

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 length of the CellList field is implicit and is determined by the IE Length field of the Payload IE header as defined in 802.15.4. The SF MAY redefine the format of the CellList field; the routine that parses this field is therefore associated with a specific SF.

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 10 defines the format of a 6P ADD Request.

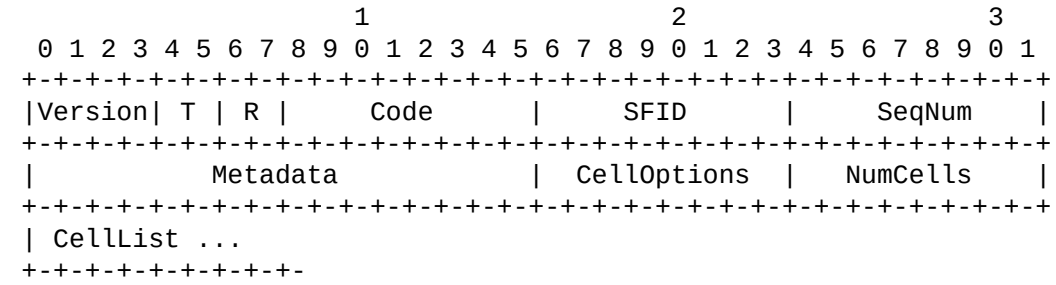

Figure 10: 6P ADD Request Format

Metadata: Used as extra signaling to the SF. The contents of the Metadata field are an opaque set of bytes passed unmodified to the SF. The meaning of this field depends on the SF and is out of scope for 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 node A wants to schedule to node B.

CellList: A list of zero or multiple candidate cells. Its length is implicit and is determined by the Length field of the Payload IE header.

Figure 11 defines the format of a 6P ADD Response and Confirmation.

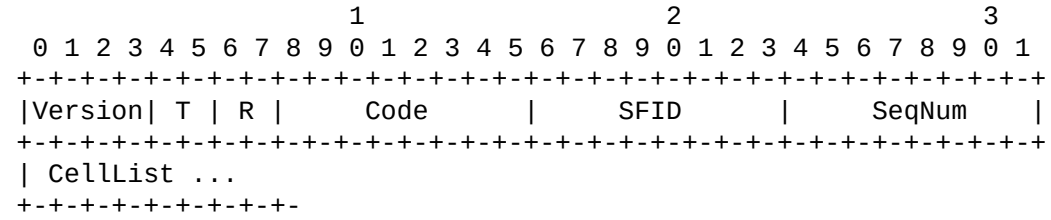

Figure 11: 6P ADD Response and Confirmation Format

CellList: A list of zero or more 6P Cells.

Consider the topology in Figure 1; in this case, 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 greater than or equal to NumCells. How many cells node A selects (NumCandidate) and how that selection is done are specified in the SF and are out of scope for this document. Node A sends a 6P ADD Request to node B that contains the CellOptions, the value of NumCells, and a selection of NumCandidate cells in the CellList. If 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. In the case of a 3-step transaction, the SF is responsible for ensuring that the returned Candidate CellList fits into the 6P Response.

Upon receiving the request, node B checks to see whether the CellOptions are set to a valid value as noted by Figure 7. If this is not the case, a Response with code RC_ERR is returned. If the number of cells in the received CellList in node B is smaller than NumCells, node B MUST return a 6P Response with the RC_ERR_CELLLIST code. Otherwise, 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 is out of scope for 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 (fewer than NumCells cells from the CellList can be used). In all cases, node B MUST send a 6P Response that includes a return code set to RC_SUCCESS and that specifies the list of cells that were scheduled following the CellOptions field. That list can contain NumCells elements (succeed), 0 elements (fail), or between 0 and NumCells elements (partially succeed).

Upon receiving the response, node A adds the cells specified in the CellList according to the 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 12 defines the format of a 6P DELETE Request.

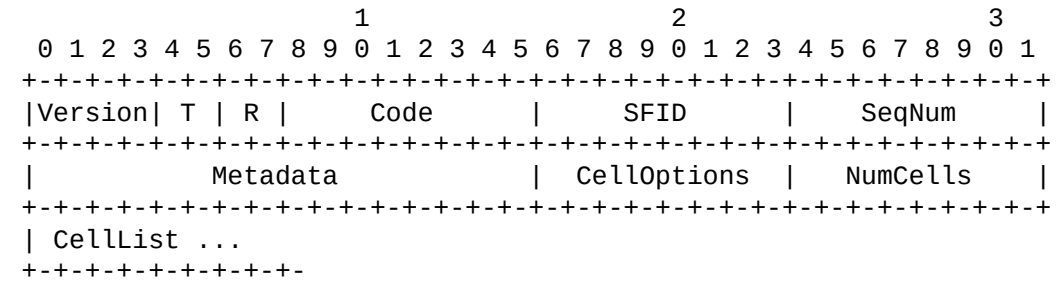

Figure 12: 6P DELETE Request Format

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

CellOptions: Indicates the options that need to be associated with the cells to delete. Only cells matching the CellOptions can be 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 zero or more 6P Cells. Its length is determined by the Length field of the Payload IE header.

Figure 13 defines the format of a 6P DELETE Response and Confirmation.

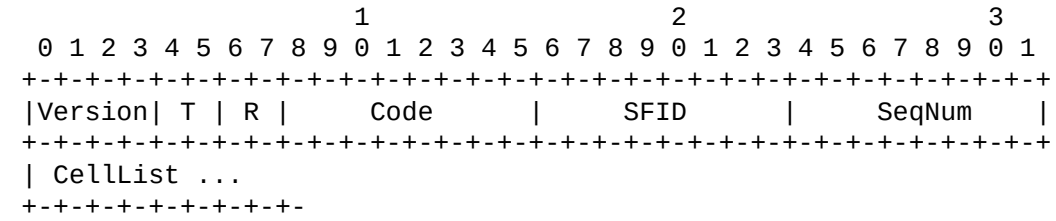

Figure 13: 6P DELETE Response and Confirmation Format

CellList: A list of zero or more 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_ERR_CELLLIST return code.
- The CellList in a 6P Request (2-step transaction) or 6P Response (3-step transaction) MUST be empty, contain exactly NumCells cells, or contain more than NumCells cells. The case where the CellList is not empty but contains fewer than NumCells cells is not supported; the RC_ERR_CELLLIST code MUST be returned when the CellList contains fewer than NumCells cells. If the CellList is empty, the SF on the receiving node MUST choose NumCells cells scheduled to the sender matching the CellOptions field and delete them. If the CellList contains more than NumCells cells, the SF on the receiving node chooses exactly NumCells cells from the CellList to delete.

3.3.3. Relocating Cells

Cell relocation consists of moving a cell to a different [slotOffset,channelOffset] location in the schedule. The Type field (T) is set to REQUEST. The Code field is set to RELOCATE. Figure 14 defines the format of a 6P RELOCATE Request.

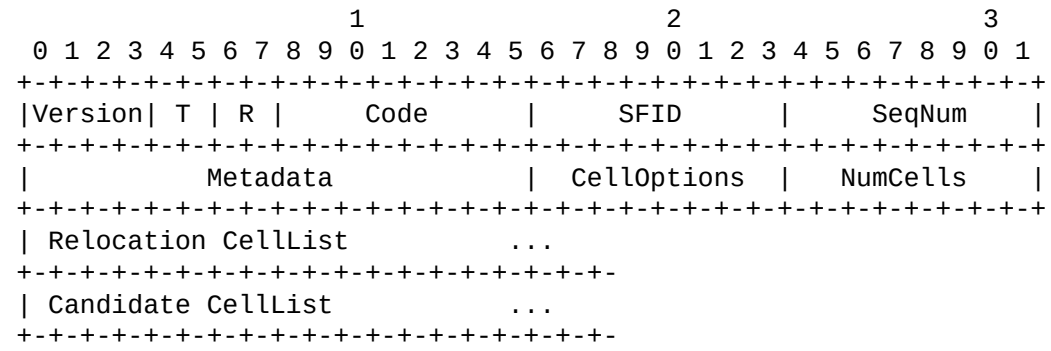

Figure 14: 6P RELOCATE Request Format

Metadata: Same usage as for the 6P ADD command; see Section 3.3.1.

CellOptions: Indicates the options that need to be associated with cells to be relocated.

NumCells: The number of cells to relocate. MUST be greater than or equal to 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. Its length is determined by the Length field of the Payload IE header.

In a 2-step 6P RELOCATE Transaction, node A specifies both (1) the cells it needs to relocate and (2) 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 (NumCandidate >= NumCells).

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

Figure 15 defines the format of a 6P RELOCATE Response and Confirmation.

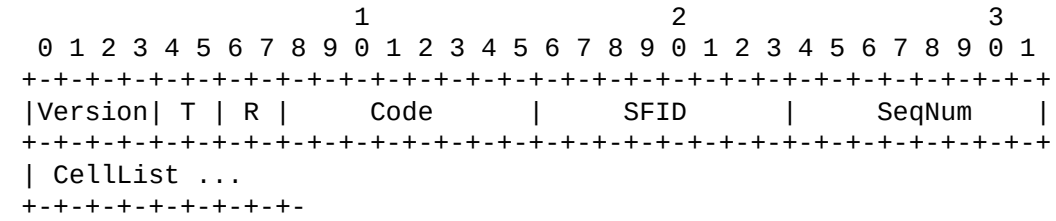

Figure 15: 6P RELOCATE Response and Confirmation Format

CellList: A list of zero or more 6P Cells.

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

Upon receiving the request, node B checks to see if the length of the Candidate CellList is greater than or equal to NumCells. Node B's SF verifies that all the cells in the Relocation CellList are scheduled with node A and are associated with the options specified in the CellOptions field. If either check fails, node B MUST send a 6P Response to node A with return code RC_ERR_CELLLIST. If both checks pass, 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 is out of scope for this document. That verification for the 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 (fewer than NumCells cells from the Candidate CellList can be used). In all cases, node B MUST send a 6P Response that includes a return code set to RC_SUCCESS and that specifies the list of cells that will be rescheduled following the CellOptions field. That list can contain NumCells elements (succeed), 0 elements (fail), or between 0 and NumCells elements (partially succeed). If N < NumCells cells appear in the CellList, this means that the first N cells in the Relocation CellList have been relocated and the remainder have not.

Upon receiving the response with code RC_SUCCESS, node A relocates the cells specified in the Relocation CellList of its RELOCATE Request to the new locations specified in the CellList of the 6P Response, in the same order. If the received return code is RC_ERR_CELLLIST, the transaction is aborted and no cell is relocated. In the case of a 2-step transaction, node B relocates the selected cells upon receiving the link-layer ACK for the 6P Response. In the case of a 3-step transaction, node B relocates the selected cells upon receiving the 6P Confirmation.

The SF SHOULD NOT relocate all cells between two nodes at the same time, as this might result in the schedules of both nodes diverging significantly.

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

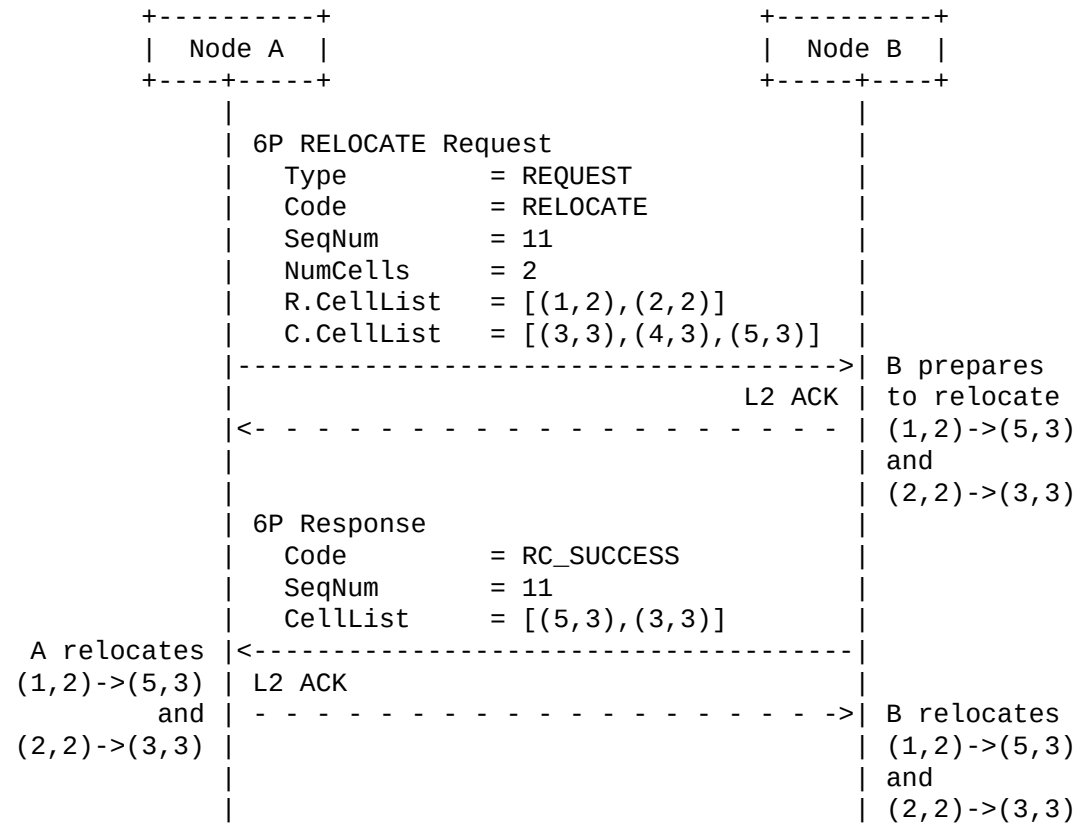

Figure 16: Example of a Successful 2-Step 6P RELOCATE Transaction

Figure 17 shows an example of a partially successful 2-step 6P RELOCATE Transaction.

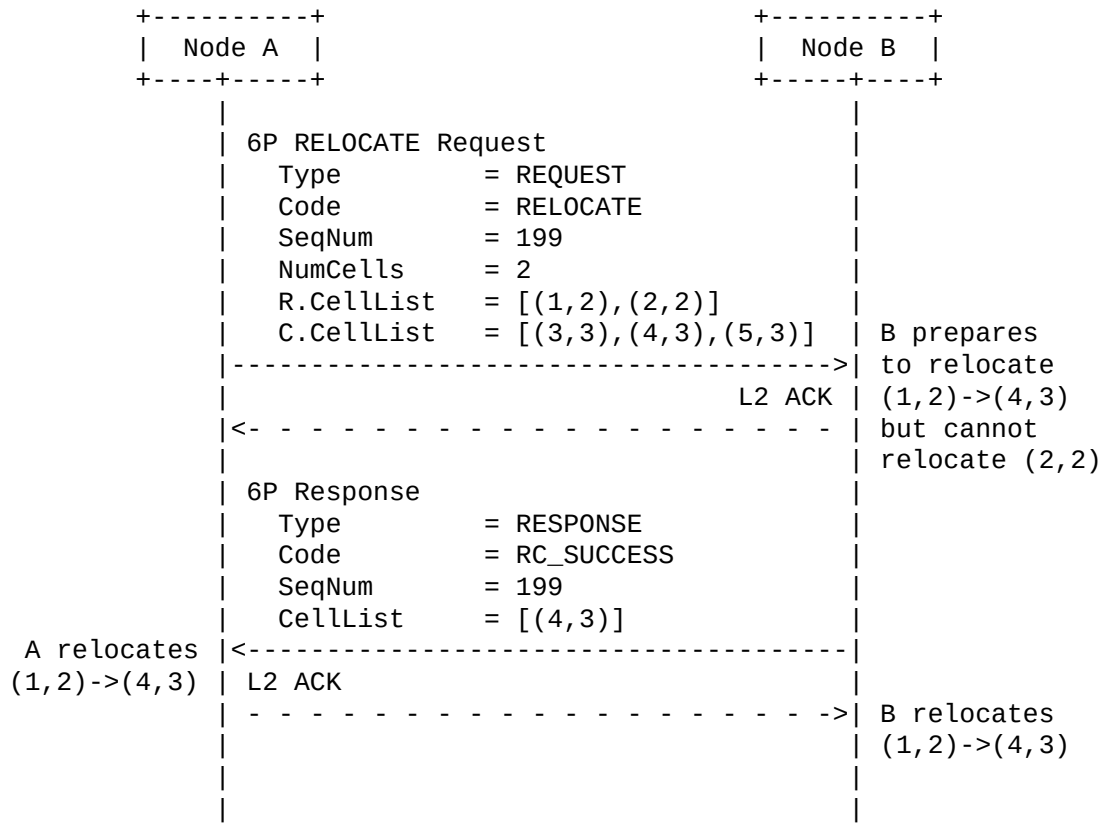

Figure 17: Example of a Partially Successful 2-Step 6P RELOCATE Transaction

Figure 18 shows an example of a failed 2-step 6P RELOCATE Transaction.

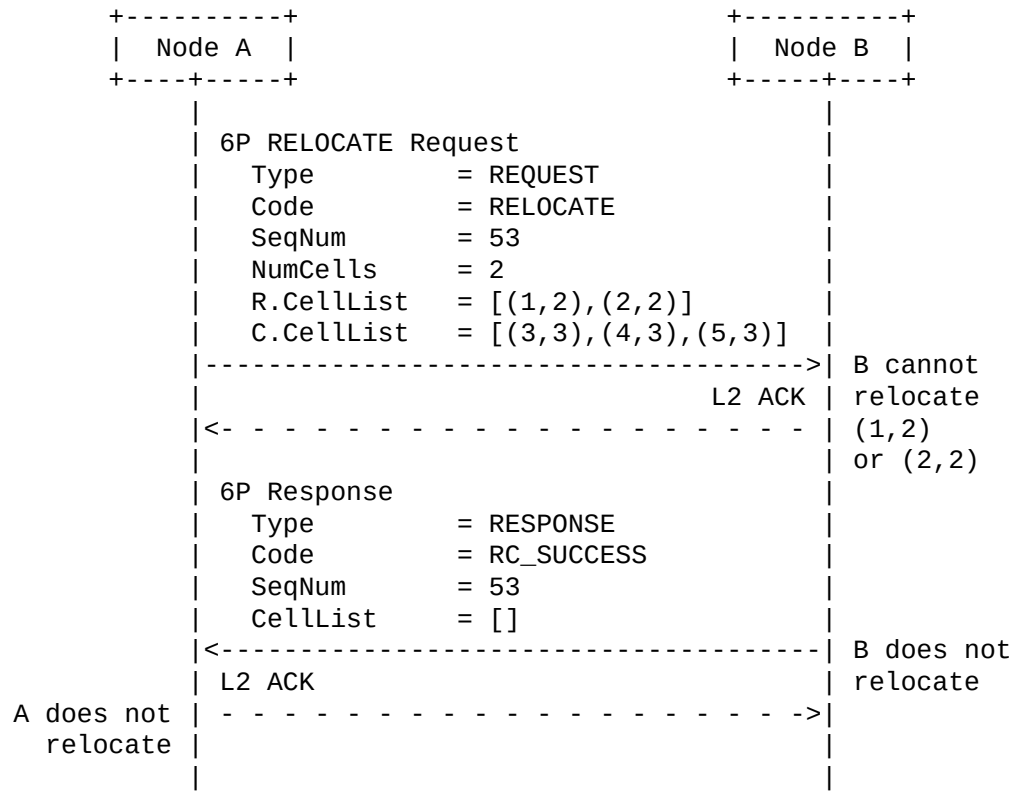

Figure 18: Failed 2-Step 6P RELOCATE Transaction Example

Figure 19 shows an example of a successful 3-step 6P RELOCATE Transaction.

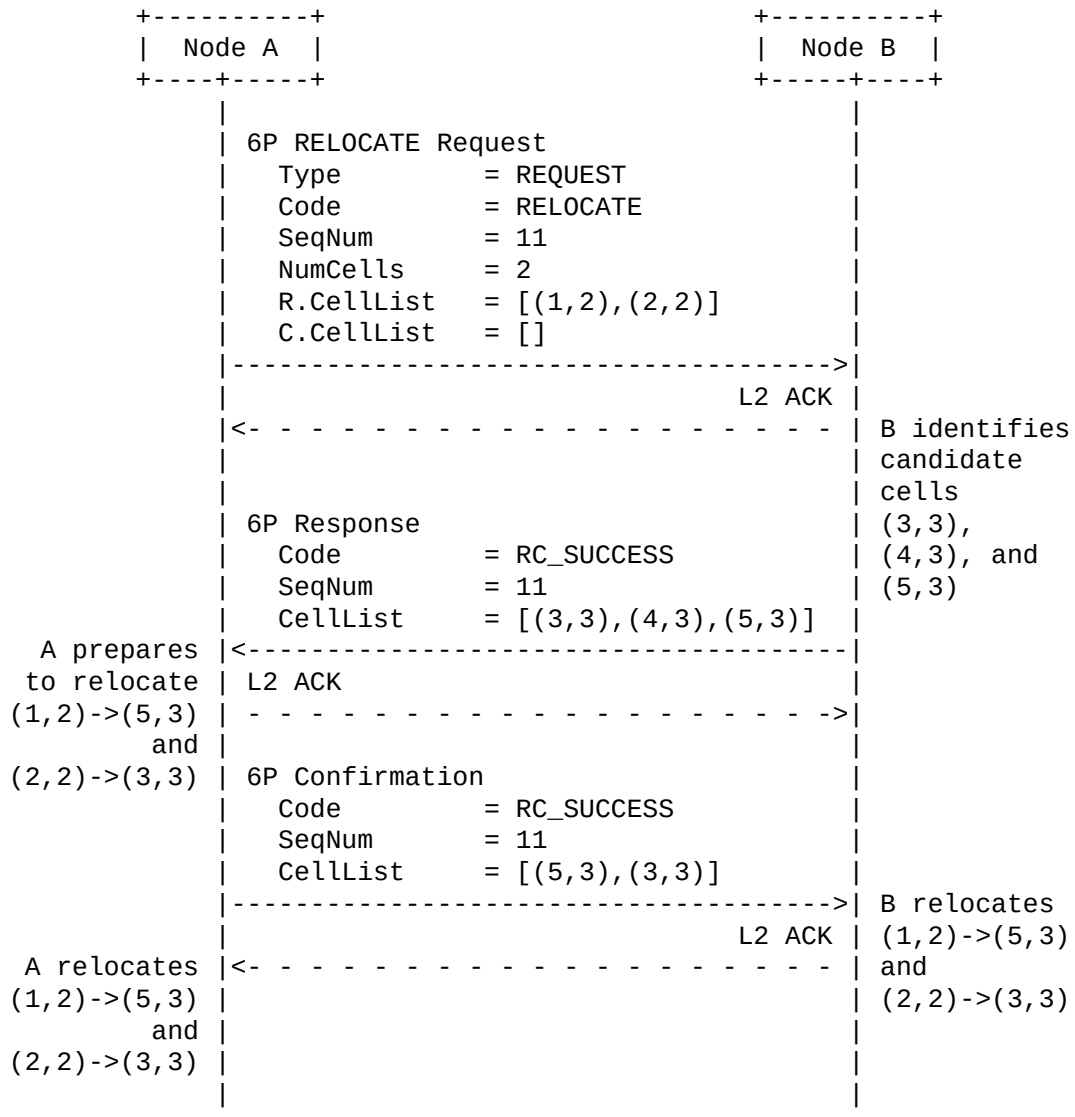

Figure 19: Example of a Successful 3-Step 6P RELOCATE Transaction

3.3.4. Counting Cells

To retrieve the number of scheduled cells node A has with B, node A issues a 6P COUNT command. The Type field (T) is set to REQUEST. The Code field is set to COUNT. Figure 20 defines the format of a 6P COUNT Request.

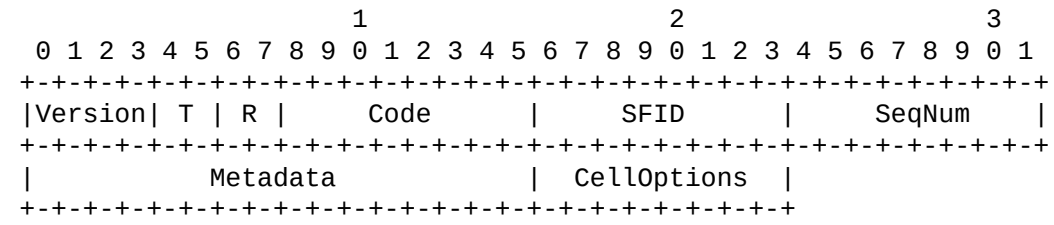

Figure 20: 6P COUNT Request Format

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

CellOptions: Specifies which type of cell to be counted.

Figure 21 defines the format of a 6P COUNT Response.

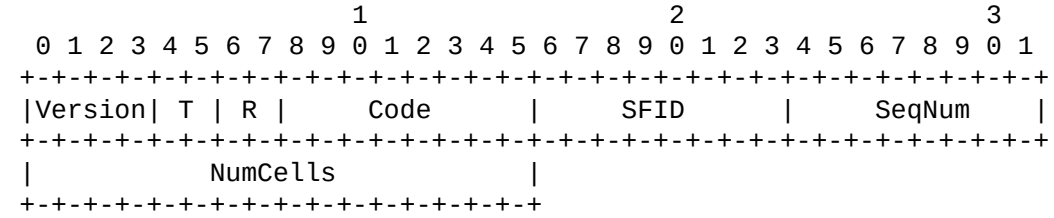

Figure 21: 6P COUNT Response Format

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

Node A issues a COUNT command to node B, specifying some 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 that 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 RC_SUCCESS and with NumCells containing the number of cells that match the request.

3.3.5. Listing Cells

To retrieve a list of scheduled cells node A has with node B, node A issues a 6P LIST command. The Type field (T) is set to REQUEST. The Code field is set to LIST. Figure 22 defines the format of a 6P LIST Request.

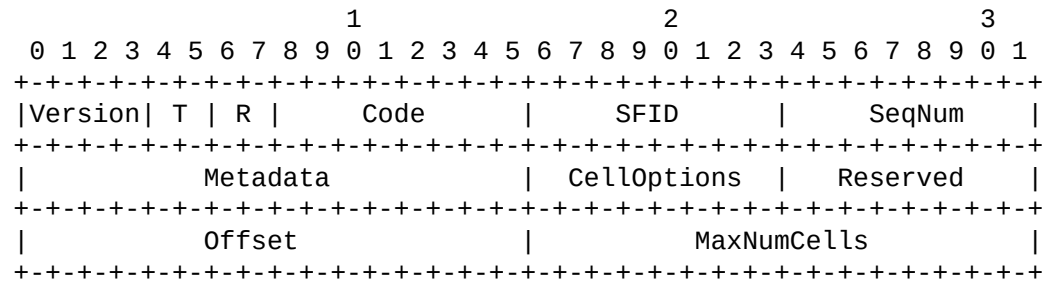

Figure 22: 6P LIST Request Format

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

CellOptions: Specifies which type of cell 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 that 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 return fewer than MaxNumCells cells -- for example, if MaxNumCells cells do not fit in the frame.

Figure 23 defines the format of a 6P LIST Response.

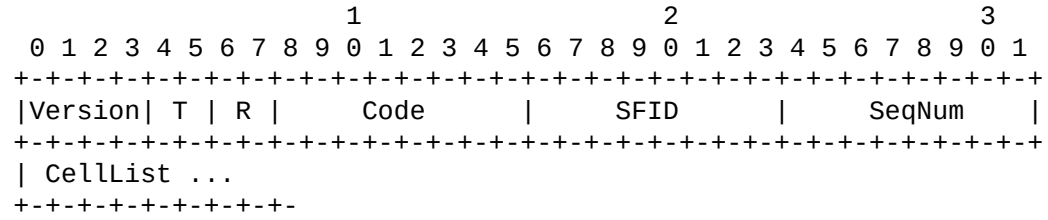

Figure 23: 6P LIST Response Format

CellList: A list of zero or more 6P Cells.

When receiving a LIST command, node B returns the cells scheduled with A 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 0 and MaxNumCells cells, starting from the specified offset. Node B SHOULD include as many cells as will fit in the frame. If the response contains the last cell, node B MUST set the Code field in the response to RC_EOL ("End of List", as per Figure 38 in Section 6.2.4), indicating to node A that there are 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 fewer 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 REQUEST. The Code field is set to CLEAR. Figure 24 defines the format of a 6P CLEAR Request.

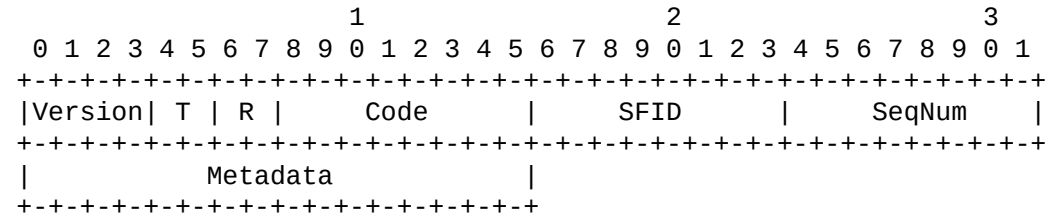

Figure 24: 6P CLEAR Request Format

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

Figure 25 defines the format of a 6P CLEAR Response.

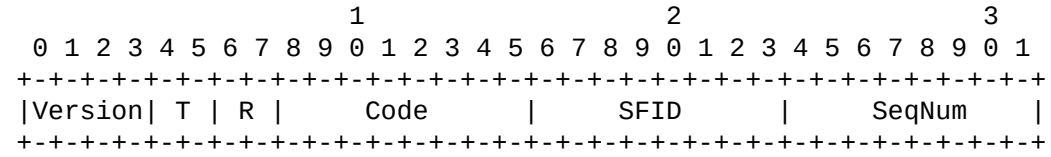

Figure 25: 6P CLEAR Response Format

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 inconsistency, the transaction MUST NOT be aborted. Upon 6P CLEAR completion, the value of SeqNum MUST be reset to 0.

The return code sent in response to a 6P CLEAR command SHOULD be RC_SUCCESS unless the operation cannot be executed. When the CLEAR operation cannot be executed, the return code MUST be set to RC_RESET.

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. The length of the payload is determined by the Length field of the Payload IE header. How the generic SIGNAL command is used is specified by the SF and is outside the scope of this document. The Type field (T) is set to REQUEST. The Code field is set to SIGNAL. Figure 26 defines the format of a 6P SIGNAL Request.

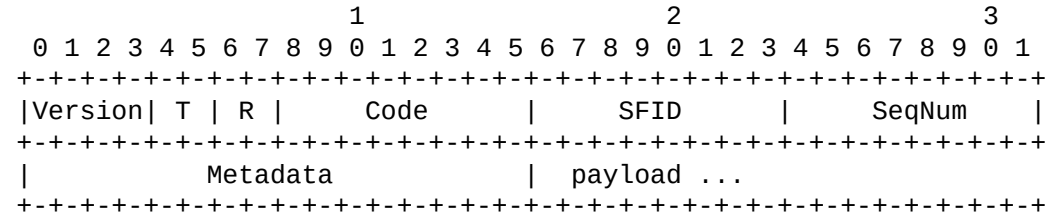

Figure 26: 6P SIGNAL Request Format

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

Figure 27 defines the format of a 6P SIGNAL Response.

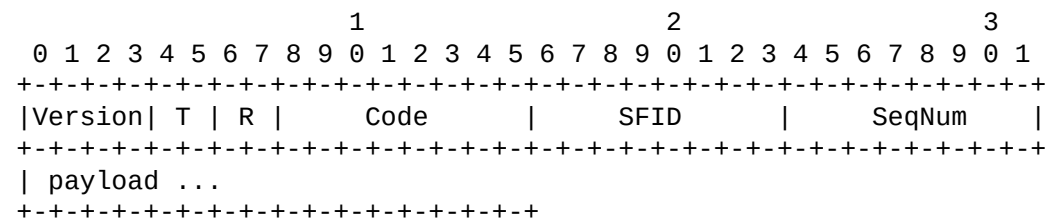

Figure 27: 6P SIGNAL Response Format

3.4. Protocol Functional Details

3.4.1. Version Checking

All messages contain a Version field. If multiple protocol versions of 6P 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_ERR_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 that it does not implement the 6P version in the 6P Request, node A can successfully parse that response.

When 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 of 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 with a return code of RC_ERR_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 the 6P Response of the same transaction.

3.4.3. Concurrent 6P Transactions

Only a single 6P Transaction at a time in a given direction can take place between two neighbors. That is, a node MUST NOT issue a new 6P Request to a given neighbor before the previous 6P Transaction it initiated has finished (or possibly 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 38 in Section 6.2.4) and discard this ongoing second transaction. A node receiving a RC_RESET code MUST abort the second transaction and treat it as though it never happened (i.e., reverting changes to the schedule or SeqNum done by this transaction).

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 with 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. If 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_ERR_BUSY (as per Figure 38 in Section 6.2.4). If the requested cells are locked, it MUST reply to that request with a 6P Response with return code RC_ERR_LOCKED (as per Figure 38). The node receiving RC_ERR_BUSY or RC_ERR_LOCKED MAY implement a retry mechanism as defined by the SF.

3.4.4. 6P Timeout

A timeout occurs when the node that successfully sent a 6P Request does not receive the corresponding 6P Response within an amount of time specified by the SF. In a 3-step transaction, a timeout also occurs when a node sending the 6P Response does not receive a 6P Confirmation. When a timeout occurs, the transaction MUST be canceled at the node where the timeout occurs. The value of the 6P Timeout should be greater than the longest possible time it takes to receive the 6P Response or Confirmation. 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 for this document.

3.4.5. Aborting a 6P Transaction

If the receiver of a 6P Request fails during a 6P Transaction and 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 having failed.

Similarly, in the case of a 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 having failed.

3.4.6. SeqNum Management

The SeqNum is the field in the 6top IE header used to match Request, Response, and Confirmation messages for a given transaction. The SeqNum is used to detect and handle duplicate commands (Section 3.4.6.1) and inconsistent schedules (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 case of supporting multiple SFs at a time, a SeqNum value is maintained per SF and per neighbor. 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 will increment the SeqNum by exactly 1 after the 6P Transaction ends. This ensures that, for the next 6P Transaction where it sends a 6P Request, the 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).

When node B receives a 6P Request from node A with SeqNum equal to 0, it checks the stored SeqNum for A. If A is a new neighbor, the stored SeqNum in B will be 0. The transaction can continue. If the stored SeqNum for A in B is different than 0, a potential inconsistency is detected. In this case, B MUST return RC_ERR_SEQNUM with SeqNum=0. The SF of node A MAY decide what to do next, as described in Section 3.4.6.2.

The SeqNum MUST be implemented as a lollipop counter: it rolls over from 0xFF to 0x01 (not to 0x00). This is used to detect a neighbor reset. Figure 28 lists the possible values of the SeqNum.

Value	Meaning
0x00	Clear, or after device reset
0x01-0xFF	Lollipop counter values

Figure 28: Possible Values of the 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 29 shows an example 2-step transaction in which node A receives a duplicate 6P Response.

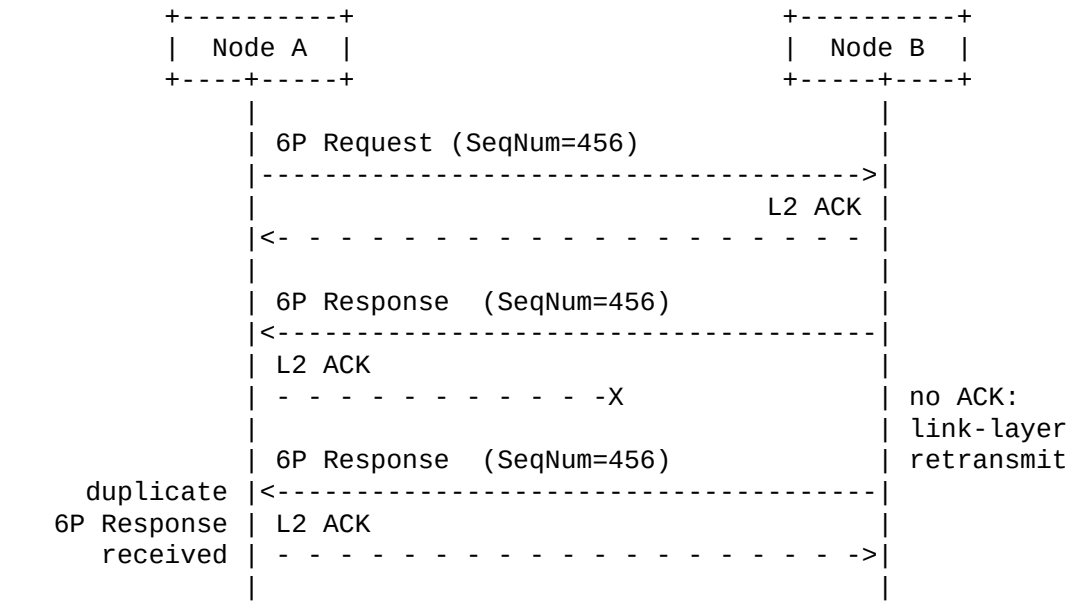

Figure 29: Example Duplicate 6P Message

Figure 30 shows an example 3-step transaction in which node A receives an out-of-order duplicate 6P Response after having sent a 6P Confirmation.

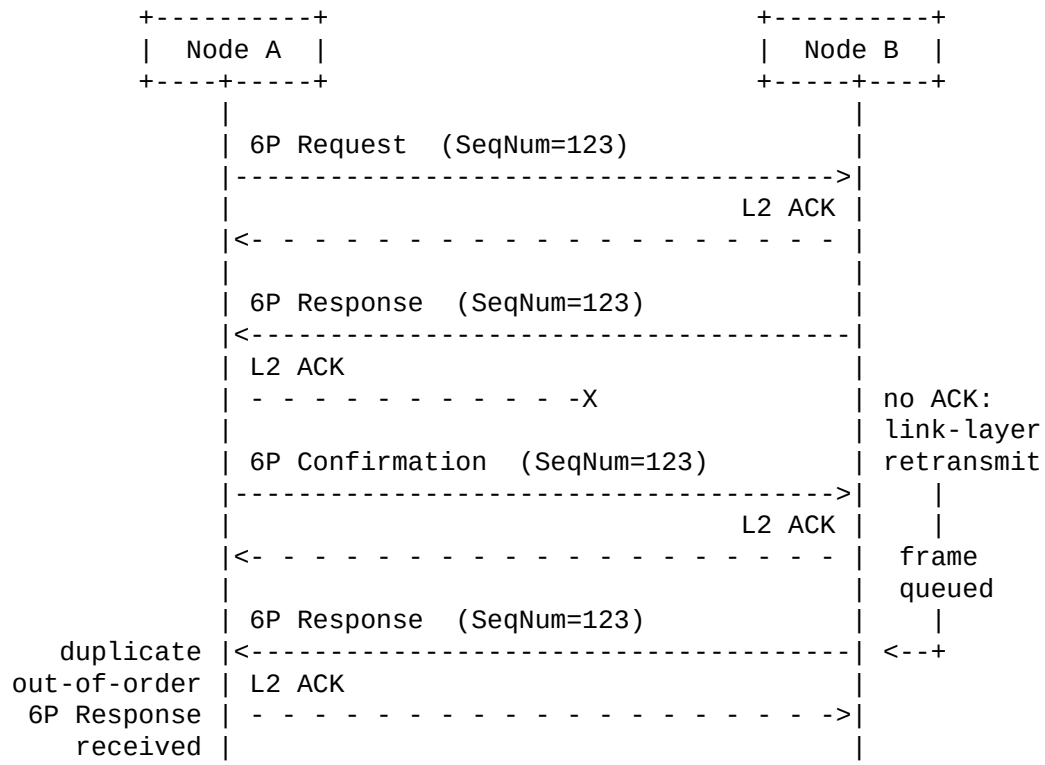

Figure 30: Example Out-of-Order Duplicate 6P Message

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 the 6P message at 6top.

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 does not have the corresponding receive cell and therefore 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. The SeqNum field increments by 1 in each message, as detailed in Section 3.4.6. 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 value, it has detected an inconsistency.

There are two cases in which a schedule inconsistency happens.

The first case is when a node loses state -- for example, when it is power-cycled (turned off, then on). In that case, its SeqNum value is reset to 0. Since the SeqNum is a lollipop counter, its neighbor detects an inconsistency in the next 6P Transaction. This is illustrated in Figures 31 and 32.

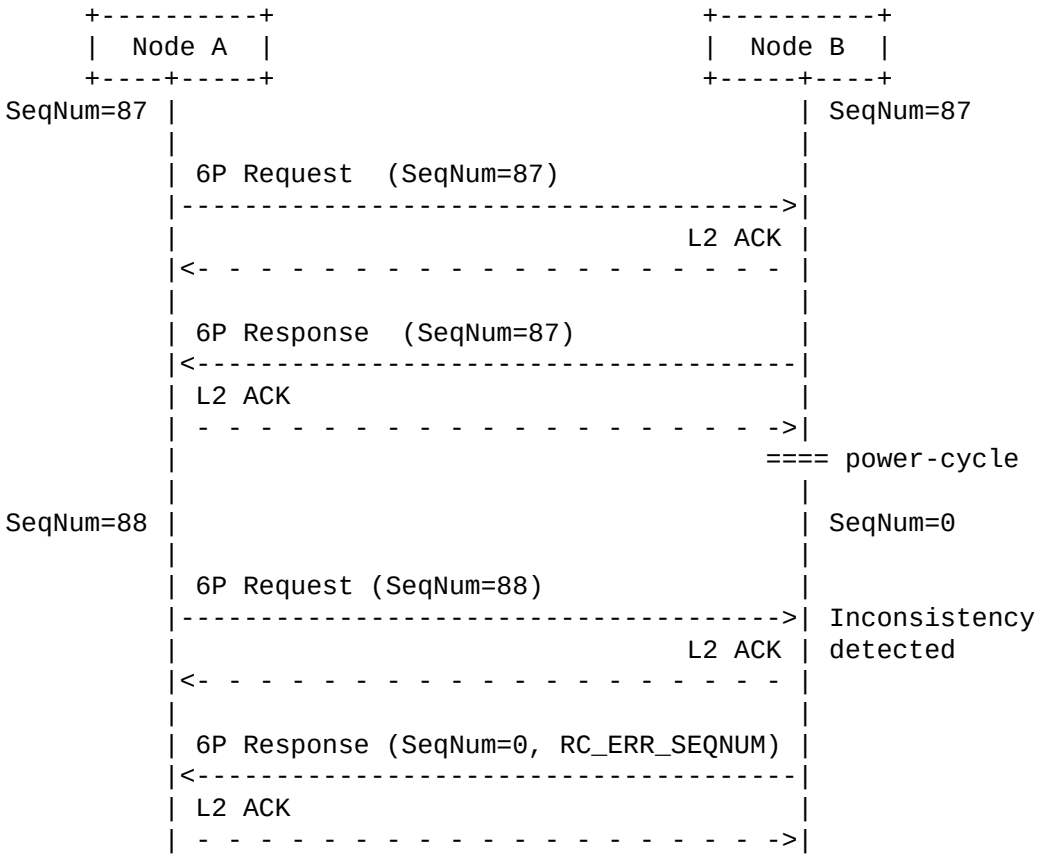

Figure 31: Example of Inconsistency Because Node B Resets (Detected by Node B)

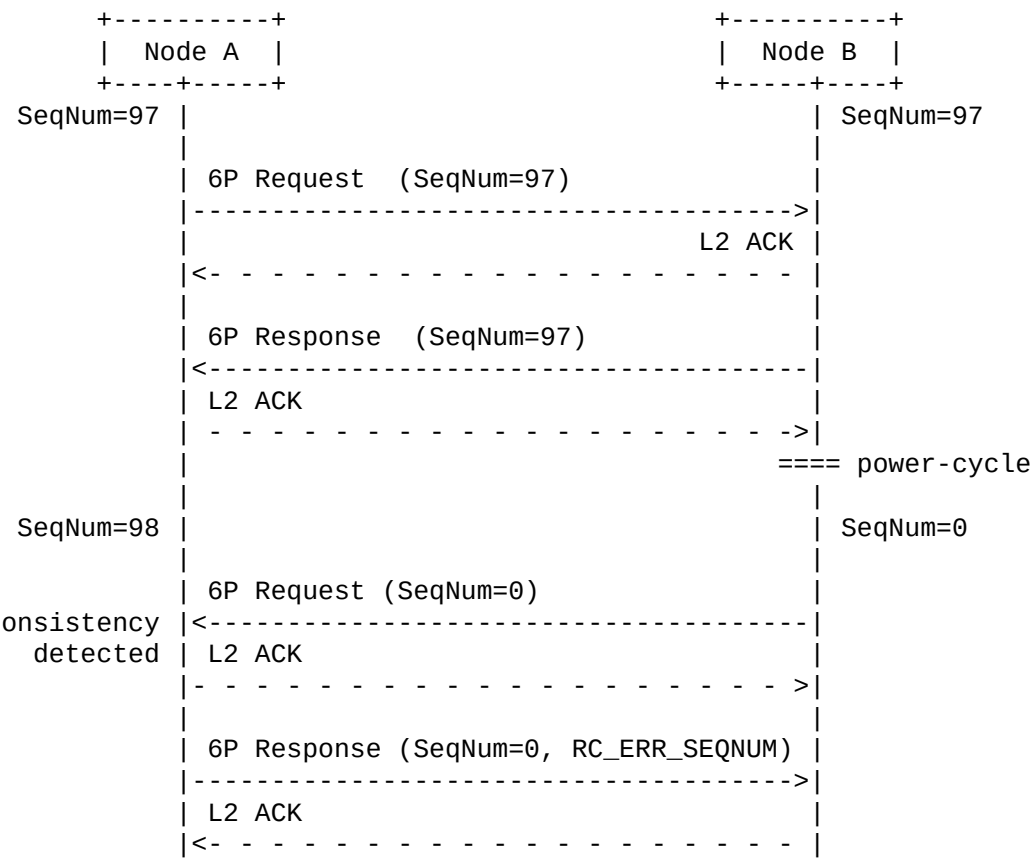

Figure 32: Example of Inconsistency Because Node B Resets (Detected by Node A)

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 33.

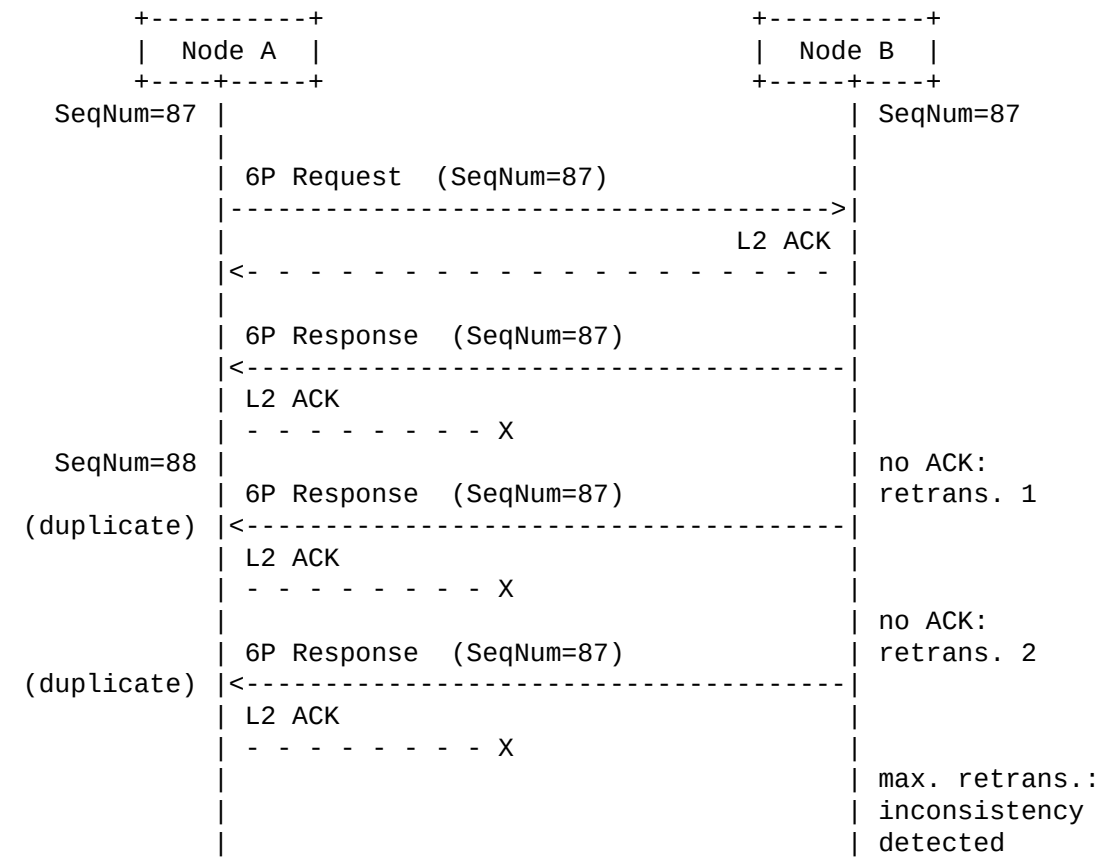

Figure 33: Example Inconsistency Because of Maximum Link-Layer Retransmissions (where Maximum = 2)

In both cases, node B detects the inconsistency.

If the inconsistency is detected during a 6P Transaction (Figure 31), the node that has detected it MUST send back a 6P Response or 6P Confirmation with an error code of RC_ERR_SEQNUM. In this 6P Response or 6P Confirmation, the SeqNum field MUST be set to the value of the sender of the message (0 in the example in Figure 31).

The SF of the node that has detected the inconsistency MUST define how to handle the inconsistency. Three possible ways to do this are as follows:

- Issue a 6P CLEAR Request to clear the schedule, and then rebuild.
- Issue a 6P LIST Request to retrieve the schedule.
- Internally "roll back" the schedule.

How to handle an inconsistency is out of scope for 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 38 (Section 6.2.4) indicates an error. When a node receives a 6P Response or 6P Confirmation with an error, it MUST consider the 6P Transaction as having failed. In particular, if this was a response to a 6P ADD, DELETE, or RELOCATE Request, the node MUST NOT add, delete, or 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, or relocate any cells as part of that 6P Transaction. If a node receives an unrecognized return code, the 6P Transaction MUST be considered as having failed. In particular, in a 3-step 6P Transaction, when receiving a 6P Response with a return code that it does not recognize, the requester (node A) MUST send a 6P Confirmation to the responder (node B) with return code RC_ERR and consider the transaction failed. Upon reception of a 6P Confirmation with return code RC_ERR, the responder MUST consider the transaction failed as well. Defining what to do after an error has occurred is out of scope for this document. The SF defines what to do after an error has occurred.

3.5. Security

6P messages MUST be secured through link-layer security. This is possible because 6P messages are carried as Payload IEs.

4. Requirements for 6top Scheduling Function (SF) Specifications

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 Specification

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/on 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 the 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 (the node either sent a 6P Response with an error code or received one).
- MUST specify the list of statistics to gather. Example statistics include the number of transmitted frames to each neighbor. If the SF does not require that statistics be gathered, the SF specification MUST explicitly say so.
- SHOULD clearly state the application domain the SF is created for.
- SHOULD contain examples that highlight normal and error scenarios.
- SHOULD contain a list of current implementations, at least during the Internet-Draft (I-D) state of the document, per [RFC7942].

- 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] ("CCM" stands for "Cipher block Chaining -- Message authentication code"). 6P benefits from the same level of security as any other Payload IE. 6P does not define its own security mechanisms. In particular, although a key management solution is out of scope for this document, 6P will benefit from the key management solution used in the network. This is relevant, as security attacks such as forgery and misattribution attacks become more damaging when a single key is shared amongst a group of more than two participants.

6P does not provide protection against DoS attacks. Example attacks include not sending confirmation messages in 3-step transactions and sending incorrectly formatted requests. These cases SHOULD be handled by an appropriate policy, such as rate-limiting or time-limited blacklisting of the attacker after several attempts. The effect on the overall network is mostly localized to the two nodes in question, as communication happens in dedicated cells.

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]:

Value	Subtype ID	Reference
1	SUBID_6TOP	RFC 8480

Figure 34: IETF IE Subtype SUBID_6TOP

6.2. 6TiSCH Parameters Subregistries

This section defines subregistries within the "IPv6 Over the TSCH Mode of IEEE 802.15.4e (6TiSCH)" parameters registry, hereafter referred to as the "6TiSCH parameters" registry. Each subregistry is described in a subsection.

6.2.1. 6P Version Numbers

The name of the subregistry is "6P Version Numbers".

The following note is included in this registry: "In the 6top Protocol (6P) [RFC8480], there is a field to identify the version of the protocol. This field is 4 bits in size."

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

The initial entry in this subregistry is as follows:

Version	Reference
0	RFC 8480

Figure 35: 6P Version Number Entry

All other version numbers are Unassigned.

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

6.2.2. 6P Message Types

The name of the subregistry is "6P Message Types".

The following note is included in this registry: "In version 0 of the 6top Protocol (6P) [RFC8480], there is a field to identify the type of message. This field is 2 bits in size."

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

Initial entries in this subregistry are as follows:

Type	Name	Reference
b00	REQUEST	RFC 8480
b01	RESPONSE	RFC 8480
b10	CONFIRMATION	RFC 8480

Figure 36: 6P Message Types

All other message types are Unassigned.

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

6.2.3. 6P Command Identifiers

The name of the subregistry is "6P Command Identifiers".

The following note is included in this registry: "In version 0 of the 6top Protocol (6P) [RFC8480], there is a Code field that 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 subregistry must include an identifier in the range 0-255, the corresponding name, and a reference to the 6P command identifier's documentation.

Initial entries in this subregistry are as follows:

Identifier	Name	Reference
0	Reserved	RFC 8480
1	ADD	RFC 8480
2	DELETE	RFC 8480
3	RELOCATE	RFC 8480
4	COUNT	RFC 8480
5	LIST	RFC 8480
6	SIGNAL	RFC 8480
7	CLEAR	RFC 8480
8-254	Unassigned	
255	Reserved	RFC 8480

Figure 37: 6P Command Identifiers

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

6.2.4. 6P Return Codes

The name of the subregistry is "6P Return Codes".

The following note is included in this registry: "In version 0 of the 6top Protocol (6P) [RFC8480], there is a Code field that 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 subregistry must include a return code in the range 0-255, the corresponding name, the corresponding description, and a reference to the 6P return code’s documentation. If the return code corresponds to a Response error, the "Is Error?" entry must indicate "Yes". Otherwise, "No" must be used.

Initial entries in this subregistry are as follows:

Code	Name	Description	Is Error?
0	RC_SUCCESS	operation succeeded	No
1	RC_EOL	end of list	No
2	RC_ERR	generic error	Yes
3	RC_RESET	critical error, reset	Yes
4	RC_ERR_VERSION	unsupported 6P version	Yes
5	RC_ERR_SFID	unsupported SFID	Yes
6	RC_ERR_SEQNUM	schedule inconsistency	Yes
7	RC_ERR_CELLLIST	cellList error	Yes
8	RC_ERR_BUSY	busy	Yes
9	RC_ERR_LOCKED	cells are locked	Yes

Figure 38: 6P Return Codes

All other message types are Unassigned.

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

6.2.5. 6P Scheduling Function Identifiers

The name of the subregistry is "6P Scheduling Function Identifiers".

The following note is included in this registry: "In version 0 of the 6top Protocol (6P) [RFC8480], there is a field to identify the Scheduling Function to handle the message. This field is 8 bits in size."

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

There are currently no entries in this subregistry.

SFID	Name	Reference
0-255	Unassigned	

Figure 39: SF Identifier (SFID) Entry

All message types are Unassigned.

The IANA policy for future additions to this subregistry depends on the value of the SFID, as shown in Figure 40. These specifications must follow the guidelines of Section 4.

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

Figure 40: SF Identifier (SFID): Registration Procedure

6.2.6. 6P CellOptions Bitmap

The name of the subregistry is "6P CellOptions Bitmap".

The following note is included in this registry: "In version 0 of the 6top Protocol (6P) [RFC8480], there is an optional CellOptions field that is 8 bits in size."

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

Initial entries in this subregistry are as follows:

bit	Name	Reference
0	TX (Transmit)	RFC 8480
1	RX (Receive)	RFC 8480
2	SHARED	RFC 8480
3-7	Reserved	

Figure 41: 6P CellOptions Bitmap

All other message types are Unassigned.

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

7. References

7.1. Normative References

[IEEE802154] IEEE, "IEEE Standard for Low-Rate Wireless Networks", IEEE 802.15.4, DOI 10.1109/IEEESTD.2016.7460875.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. Informative References

[CCM-Star] Struik, R., "Formal Specification of the CCM* Mode of Operation", IEEE P802.15-4/0537r2, September 2005.

[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>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[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 an SF document is RECOMMENDED:

- Introduction
- RFC 2119 Requirements Language (if applicable)
- 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
- Normative References (if applicable)
- Informative References (if applicable)

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
United States of America

Email: thomas.watteyne@analog.com