KTH VETENSKAP OCH KONST

KTH Electrical Engineering

IEEE 802.15.4 Implementation based on TKN15.4 using TinyOS

GTS implementation

AITOR HERNÁNDEZ, PANGUN PARK
{aitorhh, pgpark}@kth.se

Stockholm, January 14, 2011

TRITA-EE 2011:002
Version: 1.0

Contents

Acronyms

ACK	ACKnowledge. 17, 25, 41
BI	Beacon Interval. 15
BO	Beacon Order. 13, 43, 54, 59
CAP	Contention Access Period. 13, 21, 23, 27, 40, 41, 44, 46
CFP	Contention Free Period. 8, 12, 13, 20, 27, 31, 39–42, 44–46, 50
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance. 4
FCFS	First-Come-First-Serve. 27, 50
FFD	Full-Function Device. 8
GTS	Guaranteed Time Slot. 3–8, 10–13, 15, 17–23, 25–28, 32, 39–45, 47, 49–51
MAC	Medium Access Control. 5, 6, 19, 20, 59
MLME	MAC subLayer Management Entity. 17, 21, 25
PAN	Personal Area Network. 4, 17, 19–21, 25, 27, 32, 39, 50, 51
PIB	PAN Information Database. 39, 41, 44
RFD	Reduced-Function Device. 8
SO	Superframe Order. 13, 15, 59, 60
SPI	Serial Peripheral Interface. 15, 59

Chapter 1

Overview

1.1 Introduction

When starting with the IEEE 802.15.4 standard protocol [9] we realized on the existence of different implementations for our platforms (TelosB [11]/ Tmote Sky [1]) in TinyOS [1]. After analysing the current available alternatives done in [5, Sec. 3.3] we select the TKN15.4 [4], which does not provide the Guaranteed Time Slot (GTS) mechanism.[2]

The purpose of this technical report is to provide a reference guide to the GTS implementation described in the IEEE 802.15.4 protocol specification based on the TKN15.4 implementation. The GTS implementation is available for the TelosB/TmoteSky platform. Hence, this technical report in conjunction with the TKN15.4 documentation, provides the overview and scheme of the current and the most reliable IEEE 802.15.4 implementation in TinyOS. Note that since we give a brief overview of the GTS mechanism of the IEEE 802.15.4, if the reader is not familiar with current standard, then we recommend the reading materials, [5, Chap. 2] and [3].

The remainder of this technical report is organized as follows. In this chapter we show and overview of the current state of the implementation. In Chapter 2 we describe the distribution of the main components of the GTS implementation focusing on the components description and the architecture structure. In Chapter 3 the different mechanisms that provides the GTS functionality are described, with a brief introduction of the standard and their equivalent implementation example.

[1]Running in Zolertia Z1 [8] but not fully validated

[2]It is interesting to check the analysis done in [5] which give us an idea of how the current implementations (IPP Hurray [2] and TKN15.4 [4]) work. Furthermore, we describe more details of several reasons to choose the TKN15.4.

The code is available on the following URL: `http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/kth/tkn154-gts`

1.2 Interoperability

The interoperability between devices, is one of the most important characteristics that a device needs.

One of the utilities that we use to analyse and debug our implementation is the IEEE 802.15.4/Zigbee protocol analyser from Texas Instruments [6] but although this board only sniffs IEEE 802.15.4 packets, it does not mean the implementation is fully standard compliant.

Another important test is the interoperability between motes with the TKN15.4 implementation and other IEEE 802.15.4 standard compliant devices. Devices need to communicate because the previous TKN15.4 and the GTS addition are done with a very strict and careful standard document tracking.

1.3 Missing functionalities

In this section, we summarize the number of missing functionalities of TKN15.4 implementation.

- **Security services** The security services have not been implemented. Although the CC2420 chip itself provides AES-128 encryption, the use of the AES-128, in our platform, may be inefficient and will certainly decrease throughput. [12, TEP 126]
- **Indirect transmissions** Frames are not kept in transaction queue in case Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) algorithm fails
- **Transmissions modes** On a beacon-enabled Personal Area Network (PAN), if the device cannot find the beacon, the data frame is not transmitted (but it should be transmitted using unslotted CSMA-CA [9, Sec. 7.5.6.1]. As far as the beacon-enabled mode and non beacon-enabled mode are implemented as different components, we select at compiling time.

Moreover remark the following point:

- **MicaZ platforms** The TKN15.4 has been designed to be a multi platform implementation, and provides MicaZ platform support. However, this implementation has not been tested on MicaZ platforms.

Chapter 2

GTS implementation

To understand and analyse how the implementation works, it is important to show which are the components that concerns. Furthermore, a reference model with the relationship between the components is shown. It shows an architectural overview of TKN15.4 with the new components for the GTS. A radio arbitration diagram is shown to illustrate how the components share the radio resource. In addition, we show the timing issues related to the beacon interval and the time slots durations, which are the critical timing.

2.1 Directory structure

Within the TinyOS 2 module the TKN15.4 Medium Access Control (MAC) implementation is located in the `tinyos-2.x/tos/lib/mac/tkn154` directory. The specific files for our GTS implementation are place in TinyOS Contribution tree and located in the `tinyos-2.x-contrib/kth/tkn154-gts` directory. The directory includes additional subdirectories for the interface definitions and the placeholder components. The location of platform specific code is in another directory. Table 2.1 shows an overview of all the directories involved.

As we can see, the directory structure is equal to the default TKN15.4. The GTS implementation is placed where the rest of the components, interfaces and dummies are. In order to run the example application please refer to the README file placed on `tinyos-2.x-contrib/kth/tkn154-gts`.

The code is available on the following URL: `http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/kth/tkn154-gts`

Content	Directory in the TinyOS Contribution Tree
TKN15.4 MAC Implementation	
components	`kth/tkn154-gts/tos/lib/mac/tkn154`
interfaces	`kth/tkn154-gts/tos/lib/mac/tkn154/interfaces`
placeholder components	`kth/tkn154-gts/tos/lib/mac/tkn154/dummies`
test applications	`kth/tkn154-gts/apps`
Platform specific code for TelosB	
configuration files	`kth/tkn154-gts/tos/platforms/telosb/mac/tkn154`

Table 2.1: TKN15.4 directories in the TinyOS-2 tree [4]

2.2 Components decomposition

2.2.1 Reference model

Figure 2.1 shows an architectural overview of TKN15.4, its main components and the interfaces that are used to exchange MAC frames between components. While this figure abstracts from the majority of interfaces and some configuration components, it illustrates one important aspect, namely, how access to the platform specific radio driver in the MAC components.

The `RadioControlP` component manages the access to the radio: with the help of an extended TinyOS 2.x arbiter component it controls which of the components on the level above is allowed to access the radio at what point in time.

We include the new components for the GTS implementation: `CfpTransmitP`, `DeviceCfpP`, `CoordCfpP` and `DispatchCfpQueueP`, they are in bold. The components in italic are the modified components.

2.2.2 Component description

The GTS functionality is implemented with the same structure and the same organization of the TKN15.4 of the MAC layer. Table 2.2 shows the components that provides the GTS functionality.

The `MLME-GTS` interface is provided by two components. But in practice, both components are complementary, and the dummy version of them does not provide the interface. It is reasonable to think that the best place to provide this interface would be in the `CfpTransmitP` components, but the coordinators and devices require different implementation due to different functionalities.

The interconnection between components, called *wiring* in `nesC`, are very important to know which components are related to each other. In our case, due to

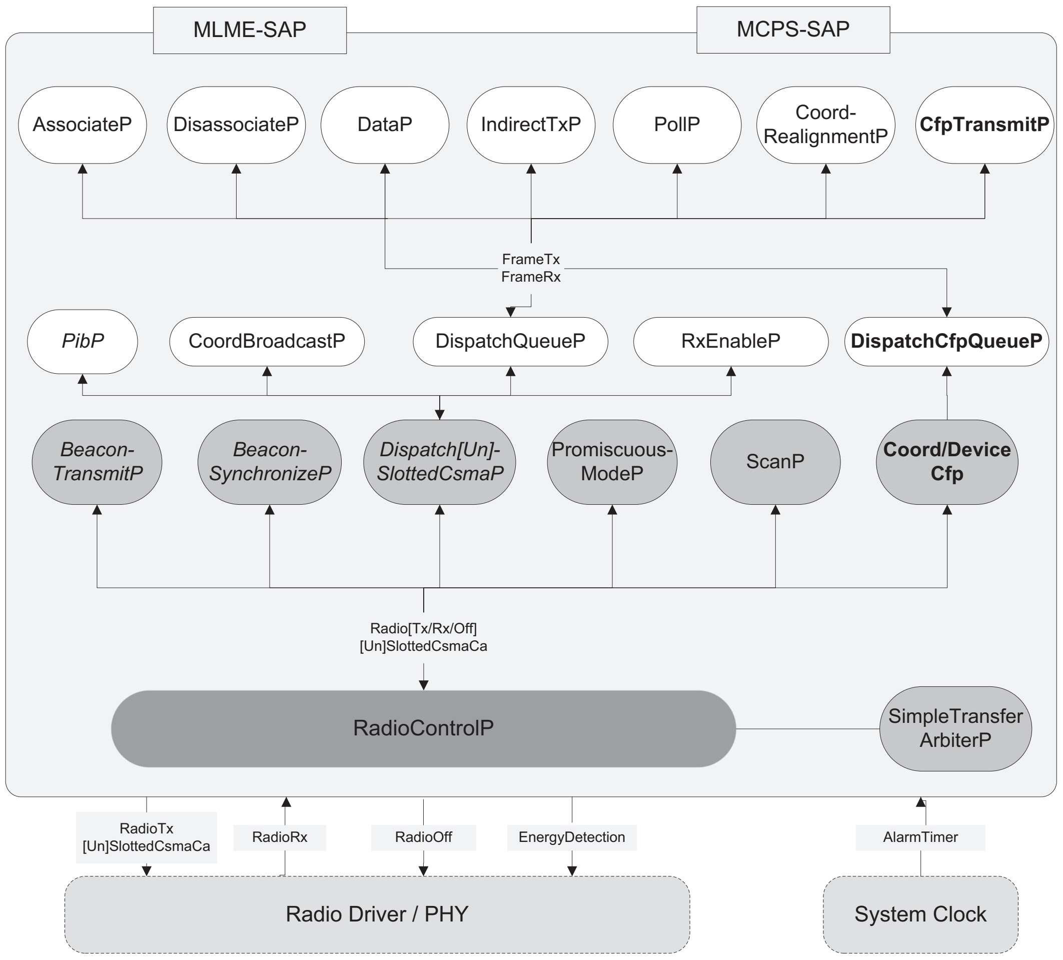


Figure 2.1: TKN15.4 architecture: components are represented by rounded boxes, interfaces by connection lines. The radio driver and symbol clock components are external to TKN15.4

the modularity of the TKN15.4 implementation, it is impossible to show the wiring graph generated by `nesdoc`. For this reason, Figure 2.2 and Figure 2.3 show a graph with the main wiring related to our implementation. [1]

Figure 2.2 shows a brief snapshot of the wiring for the GTS implementation in a device. Boxes with light grey are the main components involving the GTS mechanism.

Figure 2.3 show a brief snapshot of the wiring for the GTS implementation in

[1]To generate the complete graph: 1. compilation with the GTS enabled, `make tmote docs`; 2. open `se.kth.tinyos2x.mac.tkn154/docs/nesdoc/index.html`; 3. navigate to the component `TKN154BeaconEnabledP`

Component Name	Function[Provided IEEE 802.15.4 Interface]
`CfpTransmitP`	managing GTS transmission
`DeviceCfpP`	functions specific for a device [MLME-GTS]
`DispatchCfpQueueP`	frame transmission during Contention Free Period (CFP)
`CoordCfp`	functions specific for a coordinator [MLME-GTS]

Table 2.2: GTS components

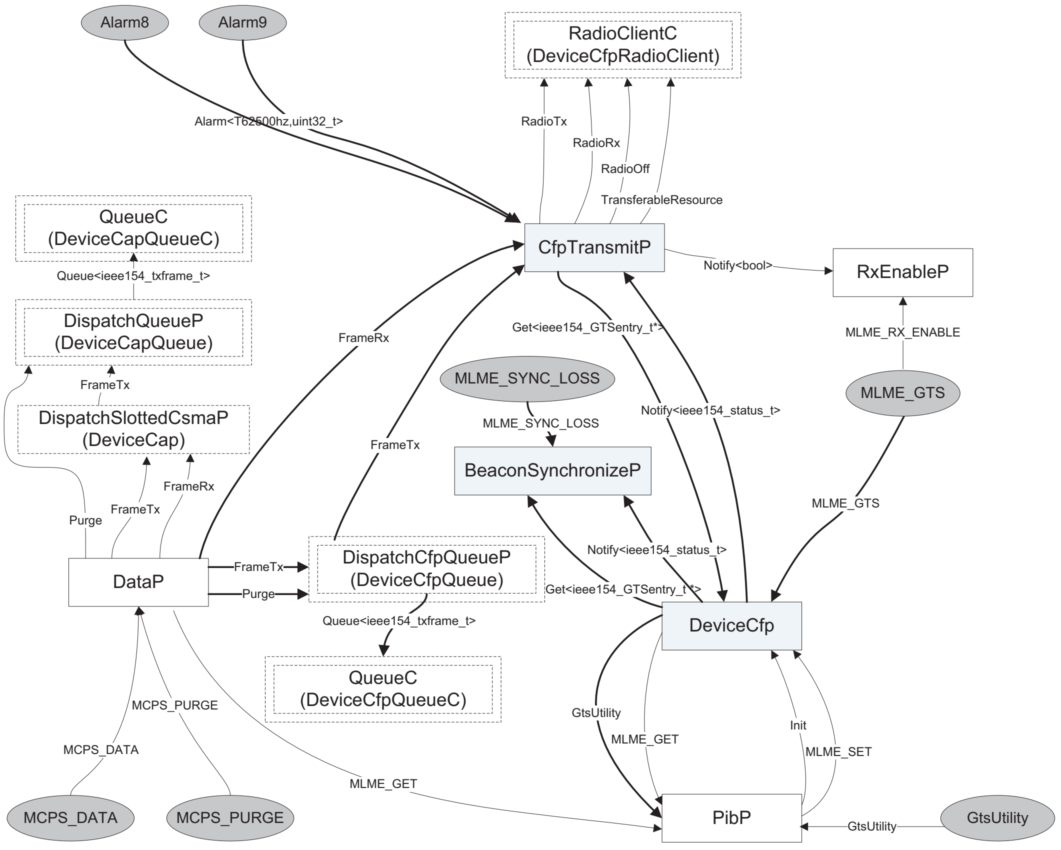


Figure 2.2: Wiring of the components related to the GTS implementation compiling for a device

a coordinator. Boxes with light grey are the main components involving the GTS mechanism.

In following sections, we describe the components related to the *Device* and the *Coordinator*. It is common to called *Device* the Reduced-Function Device (RFD) and *Coordinator* the Full-Function Device (FFD) or PAN coordinator. Hence, we use this nomenclature. Please, see more details of these components with interfaces

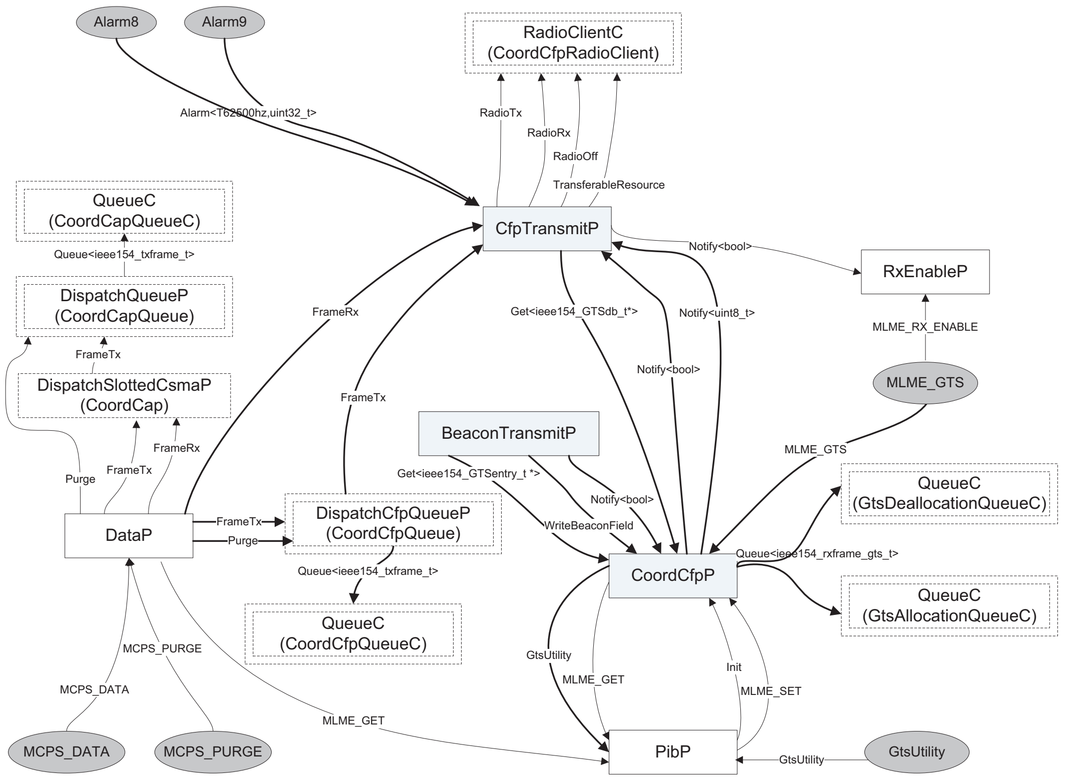


Figure 2.3: Wiring of the components related to the GTS implementation compiling for a coordinator

and variables in Appendix A or *nesC* documentation.

CfpTransmitP

The `CfpTransmitP` component, includes the functions to transmit and receive the packets. This component uses different interfaces that `DeviceCfp` or `CoordCfp` have. For this reason, we have different pre-compiler directives in order to reduce the program memory for device/coordinator. Their functionalities are:

- Control the slots and use them for reception or transmission. We can see that coordinator needs to manage the database of the seven entries and device has database of two entries. This is the reason of using C pre-compiler directives.
- Manage the radio state as function of the current slots
- Transmit/receive the packets

DeviceCfp

The `DeviceCfp` component contains the functions and provides the components needed exclusively for the device node. It gives the functionalities to be able to use the GTS in a device node, in conjunction with the `CfpTransmitP`, which has the common functions that coordinator and device need to share. Namely their functionalities are:

- `MLME_GTS` primitives for the device
- GTS management, allocations and deallocations

CoordCfp

The `CoordCfp` component contains the functions and provides the components needed exclusively for the coordinator node. It gives the functionalities to be able to use the GTS with a coordinator node, in conjunction with the `CfpTransmitP`, which has the common functions that coordinator and device need to share. Namely their functionalities are:

- `MLME_GTS` primitives for the coordinator
- Management GTS allocations request from devices
- GTS deallocations request from device
- GTS expiration if a slot has not been used for a certain time

DispatchCfpQueueP

The `DispatchCfpQueueP` component provides the functions to control the Queue for `FrameTx` and `FrameRx`.

GtsUtility

The utility component provides tools to manage and configure the GTSs.

Table 2.3 explains the functions and commands implemented. The first group is focused on the frames, and the second group is focused on the beacon frame, they provides the functions to read, parse and get the GTS descriptor and the manage the GTS characteristics from the GTS request. Furthermore it provides utilities to manage the GTS database that needs the coordinator.

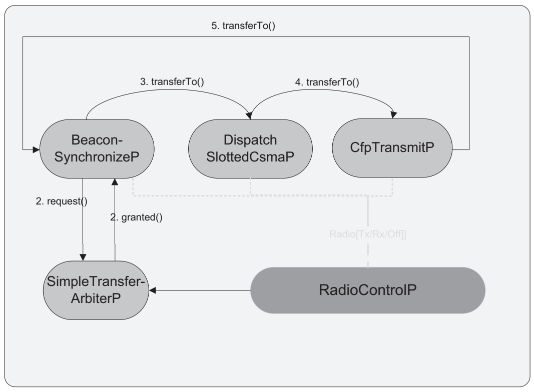
Function Name	Description
Frame functions	
`uint8_t getGtsCharacteristics(...)`	Get the GTS Characteristics from a frame payload
`uint8_t setGtsCharacteristics(...)`	Return the GtsCharacteristics to create a frame
`ieee154_status_t parseGtsCharacteristicsFromFrame(...)`	Parse a frame to get the GtsCharacteristics
`ieee154_status_t parseGtsCharacteristicsFromPayload(...)`	Parse a payload to get the GtsCharacteristics
BeaconFrame functions	
`uint8_t getGtsEntryIndex(...)`	Get the GTS entry index, in case we have it, otherwise return the new one
`error_t addGtsEntry(...)`	Add the new entry to the coordinator database
`error_t purgeGtsEntry(...)`	Purge the entry in the coordinator database. It reallocates the rest of the slots, to keep it in order
`error_t getGtsFields(...)`	Reads the GTS Fields of a beacon frame (except GTS List)
`error_t getSlotsById(...)`	Get the slots from the beacon for the selected ID. It will store the two GTSentry for the current `TOS_NODE_ID`.
`void setEmptyGtsEntry(...)`	Empty the slot. Set the length and starting slot to zero; direction to two (invalid) [2] and expiration to `FALSE`

Table 2.3: GtsUtility functions

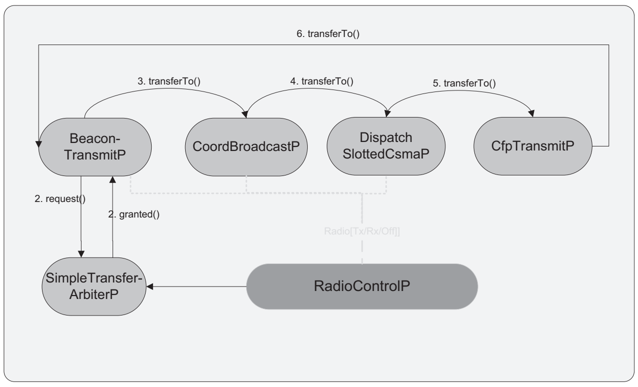
2.3 Radio arbitration

The radio arbitration mechanism is used to coordinate the activities of the components so that they do not overlap in time: typically a component is active only while it has exclusive access to the radio resource. It then performs a certain task and afterwards either releases the resource or passes it on to some other component. Figure 2.4 describes how the radio resource is transferred and which components can manage it.

The use of a TinyOS resource arbiter avoids inconsistencies in the radio driver state machine and is in line with the standard TinyOS 2.x resource usage model: before a component may access a resource it must first issue a request; once the `granted()` event is signalled by the arbiter the component can use the resource exclusively; and after usage, the resource must be released. TKN15.4 extends this model by allowing a component that owns the radio resource to dynamically transfer the ownership to another specific component.



(a) Device



(b) Coordinator

Figure 2.4: Transferring the radio token between the components responsible for an incoming/outgoing superframe. The commands `request()`, `transferTo()` and the `granted()` event are part of the `TransferableResource` interface.

2.4 Timers and alarms

In this section, we show the timers and alarms used for the GTS implementation. In Table 2.4 we see these alarms and the components where they are used.

Figure 2.5 shows the timers and alarms involved in the GTS implementation. In Figure 2.6, we present all the alarms involved on the CFP. Notice that since we are using $BO = SO$, `CfpEndAlarm` expires *guarTime* symbols before the end of the superframe. Once the `CfpEndAlarm` has fired, the `CfpSlotAlarm` stops as well.

2.4.1 Guard time

According to [10] there is *guard time* needed to compensate for clock drifts between nodes, and assure that the beacon is received on

Name	Description
`CfpSlotAlarm`	It indicates the end of the current time slots
`CfpEndAlarm`	It indicates the end of the CFP period and the superframe
`TrackAlarm`	It is fired when we expect the beacon frame and we enable the radio reception
`GTSDescPersistenceTimeout`	If the GTS request has not been served after *aGTSDescPersistence*, in the device, the GTS is discarded

Table 2.4: Timers and alarms involved in the GTS implementation

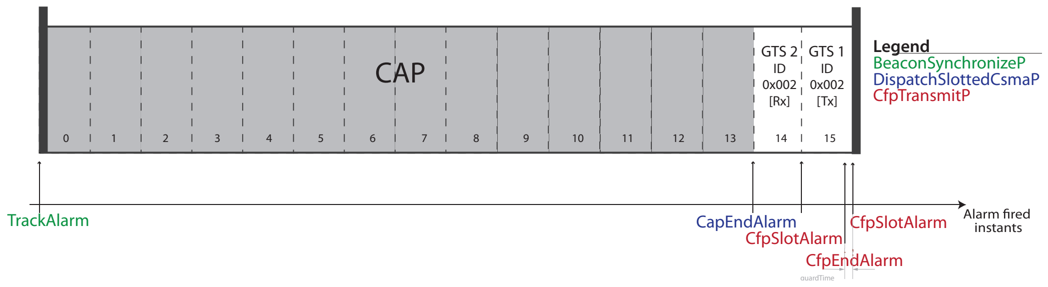


Figure 2.5: Fired instants of the timers and alarms involved in the GTS implementation

time. It is a given constant set in the platform specific components, `tos/platforms/telosb/mac/tkn154/TKN154_platform.h`.

Figure 2.7 shows the guard time in the superframe by having CAP and CFP periods. In Figure 2.7a, the guard time is needed before the end of the CFP period, otherwise the beacon could be lost. In Figure 2.7b with BO$\neq$ SO, we do not need the guard time, because we transfer the `RadioToken` to the `BeaconSynchronize` or `BeaconTransmit`, so we assure that the beacon reception will be served.

2.5 Timing analysis

In this section, we analyse some timing issues that the GTS implementation involves. It is specially important to have the acknowledge of the timing limitations because the requirements to transmit in the backoff boundaries, and to have the synchronization between motes for the time slots are critical.

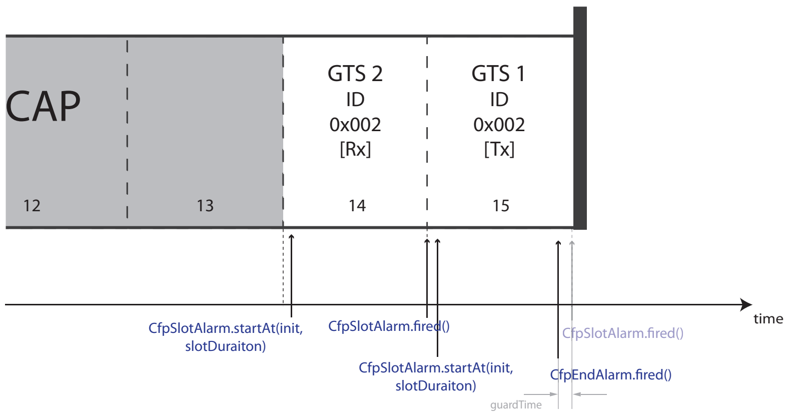
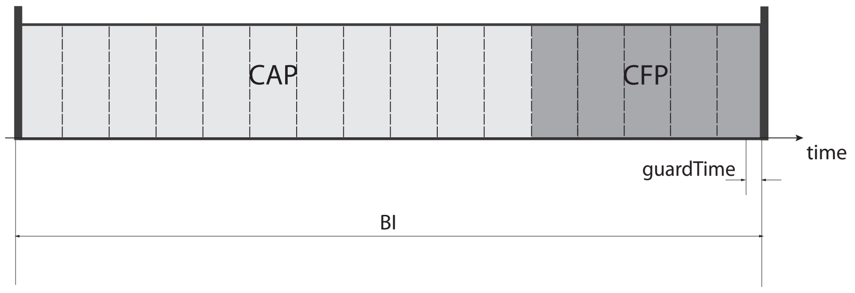
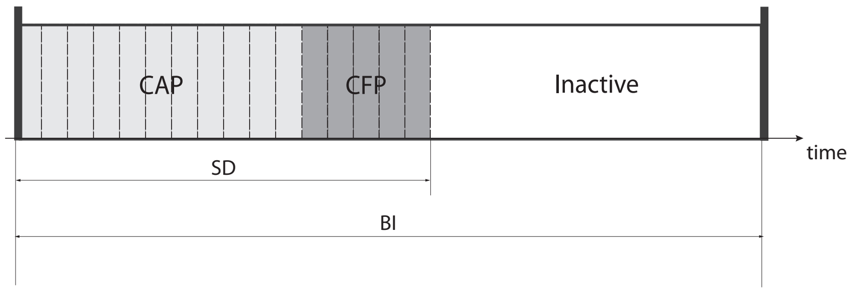


Figure 2.6: Detail of alarms involved in the GTS implementation



(a) $BO = SO$



(b) $BO \neq SO$

Figure 2.7: Influence of guard time

Our platforms do not have a clock that satisfies the precision/accuracy requirements of the IEEE 802.15.4 standard (e.g. 62.500 Hz, +-40 ppm in the 2.4 GHz band). We have a 32.768 kHz which can achieve a virtual[3] $T_{\text{symbol}} = \frac{1}{2*32768} = 15.259\mu s$. In the following sections, we discuss the hardware limitations.

[3]See `Alarm32khzTo62500hzTransformC` component

2.5.1 Beacon interval

Table 2.5 shows the values of the beacon interval computing the theoretical values as $BI_{theoretical} = aBaseSlotDuration \cdot aNumSuperframeSlots \cdot 2^{BO} = 60 \cdot 16 \cdot 2^{BO}$ and the experimental values as $BI_{experimental} = \frac{Duration[seconds]}{T_{\text{symbol}}|_{impl.}}$. In the table, we can see the difference between the real and experimental values of the beacon intervals.

	BI (Beacon Interval)			
	Theoretical		Experimental	
BO	[symbols]	[ms]	[symbols]	[ms]
1	1920	30.7	1829.1	29.3
2	3840	61.4	3660.1	58.6
3	7680	122.9	7322.1	117.2
4	15360	245.8	14646.0	234.3
5	30720	491.5	29293.3	468.7
6	61440	983.0	58589.8	937.4
7	122880	1966.1	117181.6	1874.9
8	245760	3932.2	234365.2	3749.8
9	491520	7864.3	468732.4	7499.7
10	983040	15728.6	937465.9	14999.5
11	1966080	31457.3	1874889.8	29998.2
12	3932160	62914.6	3749778.1	59996.4
13	7864320	125829.1	7499557.0	119992.9
14	15728640	251658.2	14999111.0	239985.8

Table 2.5: Beacon interval for $1 \leq BO \leq 14$. $T_{\text{symbol}} = 16\mu s$

In the Appendix C.1 there is an analysis of the accuracy/precision for the beacon interval time. We see which is the error that our virtual timer introduces and which is the standard deviation.[4]

2.5.2 Timeslot duration

As we have seen, the difference of the T_{symbol} in our motes makes that all the timing changes in comparison with the standard. Table 2.6 shows time slot duration as a function of the Superframe Order (SO) by having $BO > SO$.

For $SO \leq 2$, it is not possible to have a bi-directional GTS transmission, mainly due to a limitation on the real baud rate of the Serial Peripheral Interface (SPI) used for the communication between the microprocessor and the CC2420 [10]. For

[4]It is interesting to remark the difference between the Beacon Interval (BI) with `TKN15.4` and with the `OpenZB` implementation. [5]

$3 \leq SO \leq 12$, we have a good accuracy (low absolute error) and precision (standard deviation), and a standard deviation around one backoff period. For $SO \geq 13$, the synchronization with the beacon is not successfully achieved with the motes, so we are not able to use these SO.

	TS (Timeslot duration)			
	Theoretical		Experimental	
BO	[symbols]	[ms]	[symbols]	[ms]
1	120	1.9	-	-
2	240	3.8	-	-
3	480	7.7	453.2	7.3
4	960	15.4	912.8	14.6
5	1920	30.7	1827.3	29.2
6	3840	61.4	3653.5	58.5
7	7680	122.9	7327.1	117.2
8	15360	245.8	14642.1	234.3
9	30720	491.5	29289.4	468.6
10	61440	983.0	58582.9	937.3
11	122880	1966.1	117176.3	1874.8
12	245760	3932.2	234337.8	3749.4
13	491520	7864.3	-	-
14	983040	15728.6	-	-

Table 2.6: Table with the Time Slots as a function of the SO, assuming $BO > SO$. $T_{\text{symbol}} = 16\mu s$

In the Appendix C.3 we show different tables where you could find the difference between time slots in *seconds* and *symbols*. Moreover, in Appendix D we study the timeslot lower boundary for our motes.

Chapter 3

GTS mechanisms

In this section, after a brief description of the selected mechanism, some screenshots and figures are shown to verify how our implementation works. The current version of the GTS implementation supports all the following IEEE 802.15.4 functionalities: GTS allocation, GTS deallocation, GTS usage, GTS expiration and GTS reallocation. The IEEE 802.15.4 does not explicitly define the GTS request serving queue. Hence, we describe the implementation at the end of the chapter.

3.1 GTS allocation

A GTS allows a device to operate on the channel within a portion of the superframe that is dedicated exclusively to that device and it shall be used only for communications between the PAN coordinator and a device associated with the PAN. A single GTS may extend over one or more superframe slots. The PAN coordinator may allocate up to seven GTSs at the same time, if there is sufficient capacity in the superframe. [9, Sec. 7.5.7.2]

Figure 3.1 shows the mechanism to allocate a GTS.(1) A device is instructed to request the allocation of a new GTS through the `MLME-GTS.request` primitive with the GTS characteristics set according to the requirements of the intended application. (2) On reception of a GTS request command indicating a GTS allocation request, the PAN coordinator shall first check if there is available capacity in the current superframe. (3) On reception of the ACK to the GTS request command, the device shall continue to track beacons and wait for at most *aGTSDescPersistenceTime* superframes. (4) If the GTS descriptor is received, the MAC sub-Layer Management Entity (MLME) shall notify the next layer of the success with `MLME-GTS.indication` with *SUCCESS* status, if not, it shall indicates the failure with `MLME-GTS.indication`, indicating the *NO_DATA* status.

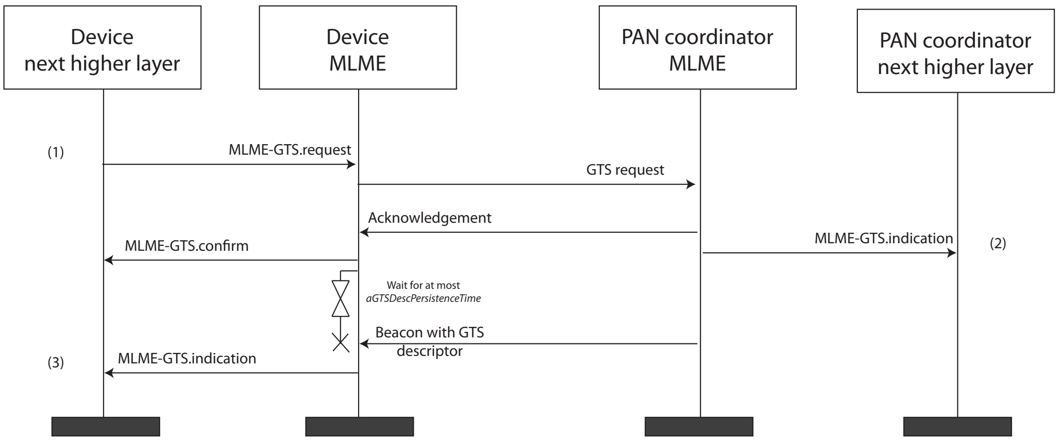
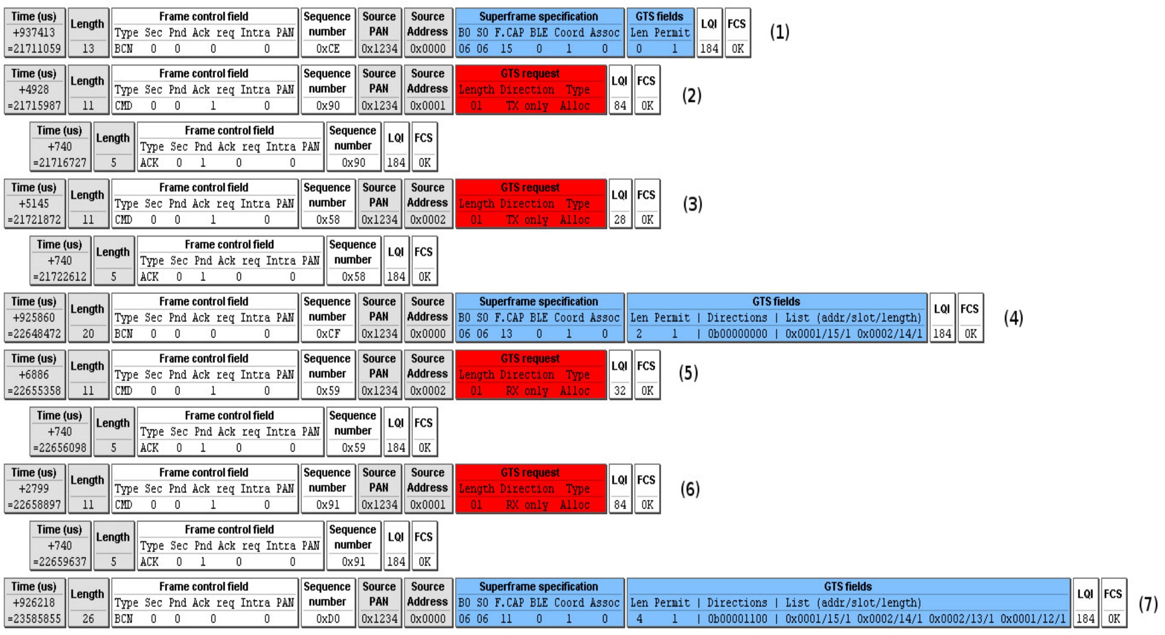


Figure 3.1: MLME-GTS allocation mechanism

3.1.1 Implementation example


```
Time (us) | Length | Frame control field               | Sequence | Source | Source  | Superframe specification        | GTS fields | LQI | FCS
+937413   |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | BO SO F.CAP BLE Coord Assoc     | Len Permit |     |
=21711059 | 13     | BCN  0   0       0         0      | 0xCE     | 0x1234 | 0x0000  | 06 06 15    0   1     0         | 0   1      | 184 | OK     (1)

Time (us) | Length | Frame control field               | Sequence | Source | Source  | GTS request                | LQI | FCS
+4928     |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | Length Direction Type      |     |
=21715987 | 11     | CMD  0   0       1         0      | 0x90     | 0x1234 | 0x0001  | 01     TX only   Alloc     | 84  | OK     (2)

    Time (us) | Length | Frame control field               | Sequence | LQI | FCS
    +740      |        | Type Sec Pnd Ack req Intra PAN    | number   |     |
    =21716727 | 5      | ACK  0   1       0         0      | 0x90     | 184 | OK

Time (us) | Length | Frame control field               | Sequence | Source | Source  | GTS request                | LQI | FCS
+5145     |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | Length Direction Type      |     |
=21721872 | 11     | CMD  0   0       1         0      | 0x58     | 0x1234 | 0x0002  | 01     TX only   Alloc     | 28  | OK     (3)

    Time (us) | Length | Frame control field               | Sequence | LQI | FCS
    +740      |        | Type Sec Pnd Ack req Intra PAN    | number   |     |
    =21722612 | 5      | ACK  0   1       0         0      | 0x58     | 184 | OK

Time (us) | Length | Frame control field               | Sequence | Source | Source  | Superframe specification        | GTS fields                                                    | LQI | FCS
+925860   |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | BO SO F.CAP BLE Coord Assoc     | Len Permit | Directions | List (addr/slot/length)          |     |
=22648472 | 20     | BCN  0   0       0         0      | 0xCF     | 0x1234 | 0x0000  | 06 06 13    0   1     0         | 2   1      | 0b00000000 | 0x0001/15/1 0x0002/14/1          | 184 | OK     (4)

Time (us) | Length | Frame control field               | Sequence | Source | Source  | GTS request                | LQI | FCS
+6886     |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | Length Direction Type      |     |
=22655358 | 11     | CMD  0   0       1         0      | 0x59     | 0x1234 | 0x0002  | 01     RX only   Alloc     | 32  | OK     (5)

    Time (us) | Length | Frame control field               | Sequence | LQI | FCS
    +740      |        | Type Sec Pnd Ack req Intra PAN    | number   |     |
    =22656098 | 5      | ACK  0   1       0         0      | 0x59     | 184 | OK

Time (us) | Length | Frame control field               | Sequence | Source | Source  | GTS request                | LQI | FCS
+2799     |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | Length Direction Type      |     |
=22658897 | 11     | CMD  0   0       1         0      | 0x91     | 0x1234 | 0x0001  | 01     RX only   Alloc     | 84  | OK     (6)

    Time (us) | Length | Frame control field               | Sequence | LQI | FCS
    +740      |        | Type Sec Pnd Ack req Intra PAN    | number   |     |
    =22659637 | 5      | ACK  0   1       0         0      | 0x91     | 184 | OK

Time (us) | Length | Frame control field               | Sequence | Source | Source  | Superframe specification        | GTS fields                                                                          | LQI | FCS
+926218   |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | BO SO F.CAP BLE Coord Assoc     | Len Permit | Directions | List (addr/slot/length)                                | |
=23585855 | 26     | BCN  0   0       0         0      | 0xD0     | 0x1234 | 0x0000  | 06 06 11    0   1     0         | 4   1      | 0b00001100 | 0x0001/15/1 0x0002/14/1 0x0002/13/1 0x0001/12/1        | 184 | OK     (7)
```

Figure 3.2: GTS allocation mechanism

Figure 3.2 shows the process of GTS allocation and Figure 3.3 shows the state of the superframe. The steps are described here:

1. GTS is enabled but there is no slot allocated.

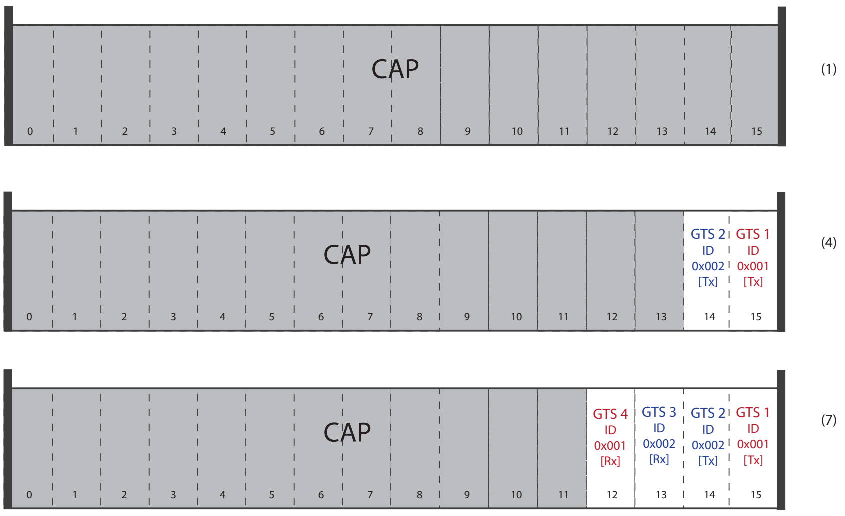


Figure 3.3: Beacon during GTS allocation mechanism

2. The coordinator 0x0000 has received a GTS allocation from the device 0x0001. The request has length = 1, and direction = 0 for transmission. The direction is related to the device.

3. The coordinator 0x0000 has received a GTS allocation from the device 0x0002. The request has length = 1 and direction = 0 for transmission.

4. Slot 15 and 14 have been allocated for transmission to the devices 0x0001 and 0x0002. In the list we have the description of the slots indicating short address, starting slot and length.

5. The coordinator 0x0000 has received a GTS allocation from the device 0x0002. The request has length = 1 and direction = 1 for reception.

6. The coordinator 0x0000 has received a GTS allocation from the device 0x0001. The request has length = 1 and direction = 1 for reception.

7. Slots 15, 14, 13 and 12 have been allocated for transmission and reception to the devices 0x0001 and 0x0002.

3.2 GTS usage

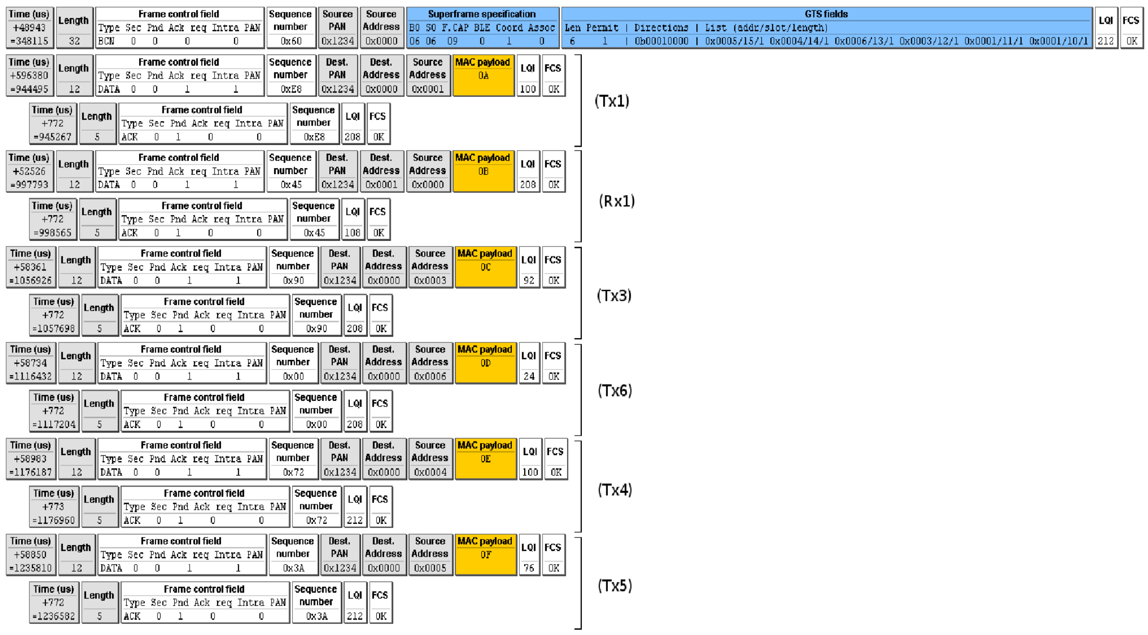
When the MAC sublayer of a device that is not the PAN coordinator receives an `MCPS-DATA.request` primitive with TxOptions parameter indicating a GTS transmission, it shall determine whether it has a valid transmit GTS. If a valid GTS is found, the MAC sublayer shall transmit the data during the GTS.

If the device has any received GTSs, the MAC sublayer of the device shall ensure that the receiver is enabled at a time prior to the start of the GTS and for the duration of the GTS.

When the MAC sublayer of the PAN coordinator receives an `MCPS.DATA.request` primitive with TxOptions parameter indicating a GTS transmission, it shall determine whether it has a valid received GTS corresponding to the device with the requested destination address. If a valid GTS is found, the PAN coordinator shall defer the transmission until the start of the receive GTS.

For all allocated transmit GTSs (relative to the device), the MAC sublayer of the PAN coordinator shall ensure that its receiver is enabled at a time prior to the start and for the duration of each GTS. [9, Sec. 7.5.7.3]

3.2.1 Implementation example



```
| Time (us) | Length | Frame control field               | Sequence | Source | Source  | Superframe specification      | GTS fields                                                                                                        | LQI | FCS |
|           |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | BO SO F.CAP BLE Coord Assoc   | Len Permit | Directions | List (addr/slot/length)                                                       |     |     |
| +48943    |        |                                   |          |        |         |                               |                                                                                                                   |     |     |
| =348115   | 32     | BCN  0   0       0         0      | 0x60     | 0x1234 | 0x0000  | 06 06 09    0   1     0       | 6   1      | 0b00010000 | 0x0005/15/1 0x0004/14/1 0x0006/13/1 0x0003/12/1 0x0001/11/1 0x0001/10/1     | 212 | OK  |

| Time (us) | Length | Frame control field               | Sequence | Dest.  | Dest.   | Source  | MAC payload | LQI | FCS |
|           |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | Address |             |     |     |
| +596380   |        |                                   |          |        |         |         | 0A          |     |     |
| =944495   | 12     | DATA 0   0       1         1      | 0xE8     | 0x1234 | 0x0000  | 0x0001  |             | 100 | OK  |
    | Time (us) | Length | Frame control field               | Sequence | LQI | FCS |
    |           |        | Type Sec Pnd Ack req Intra PAN    | number   |     |     |
    | +772      |        |                                   |          |     |     |
    | =945267   | 5      | ACK  0   1       0         0      | 0xE8     | 208 | OK  |     (Tx1)

| +52526    |        |                                   |          |        |         |         | 0B          |     |     |
| =997793   | 12     | DATA 0   0       1         1      | 0x45     | 0x1234 | 0x0001  | 0x0000  |             | 208 | OK  |
    | +772      |        |                                   |          |     |     |
    | =998565   | 5      | ACK  0   1       0         0      | 0x45     | 108 | OK  |     (Rx1)

| +58361    |        |                                   |          |        |         |         | 0C          |     |     |
| =1056926  | 12     | DATA 0   0       1         1      | 0x90     | 0x1234 | 0x0000  | 0x0003  |             | 92  | OK  |
    | +772      |        |                                   |          |     |     |
    | =1057698  | 5      | ACK  0   1       0         0      | 0x90     | 208 | OK  |     (Tx3)

| +58734    |        |                                   |          |        |         |         | 0D          |     |     |
| =1116432  | 12     | DATA 0   0       1         1      | 0x00     | 0x1234 | 0x0000  | 0x0006  |             | 24  | OK  |
    | +772      |        |                                   |          |     |     |
    | =1117204  | 5      | ACK  0   1       0         0      | 0x00     | 208 | OK  |     (Tx6)

| +58983    |        |                                   |          |        |         |         | 0E          |     |     |
| =1176187  | 12     | DATA 0   0       1         1      | 0x72     | 0x1234 | 0x0000  | 0x0004  |             | 100 | OK  |
    | +773      |        |                                   |          |     |     |
    | =1176960  | 5      | ACK  0   1       0         0      | 0x72     | 212 | OK  |     (Tx4)

| +58850    |        |                                   |          |        |         |         | 0F          |     |     |
| =1235810  | 12     | DATA 0   0       1         1      | 0x3A     | 0x1234 | 0x0000  | 0x0005  |             | 76  | OK  |
    | +772      |        |                                   |          |     |     |
    | =1236582  | 5      | ACK  0   1       0         0      | 0x3A     | 212 | OK  |     (Tx5)
```

Figure 3.4: GTS data transmission

Figure 3.4 shows a sniffer dump where five motes are running at the same time, and transmitting during the CFP with their own slot. The coordinator is configured to send packets to the device 0x0001. The devices 0x0001, 0x0003, 0x0004, 0x0005, 0x0006 have a slot allocation request for both directions, but if they do not use it, then it will be deallocated by the expiration mechanism.

In this example, all the devices and the coordinator write the slot number in their transmission packet payload in order to assure that they are sending the packet in the correct slot. So, we can see the relation between the payload value for each node and the GTS descriptor.

3.3 GTS deallocation

The GTS deallocation could be initiated either by the coordinator or the device.

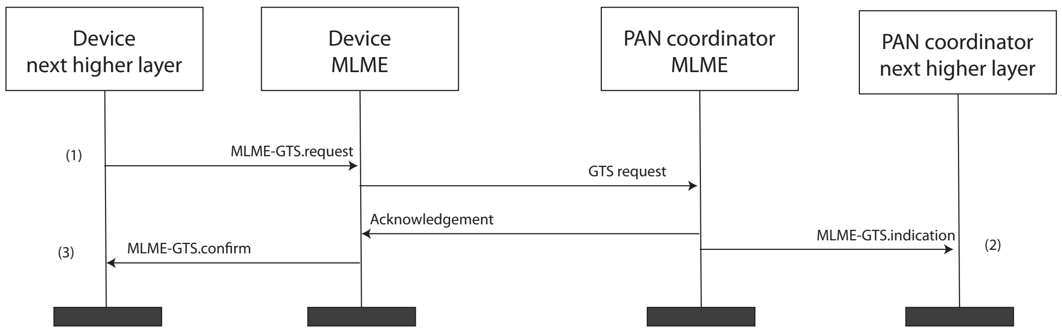


Figure 3.5: MLME-GTS deallocation mechanism initiated by the device

Figure 3.5 show the mechanism to deallocate a GTS initiated by the device. (1) A device is instructed to request the deallocation of an existing GTS through the `MLME-GTS.request` primitive using the characteristics of the GTS it wishes to deallocate. From this point onward the GTS to be deallocated shall not be used by the device, and its stored characteristics shall be reset. (2) On the reception of a GTS request command with the Characteristics Type subfield of the GTS Characteristics field set to zero (GTS deallocation), the PAN coordinator shall attempt to deallocate the GTS. If the GTS characteristics contained in the GTS request command match the characteristics of a know GTS, the MLME of the PAN coordinator shall deallocate the specified GTS and notify the higher layer.

GTS deallocation may be initiated by PAN coordinator due to a deallocation request from the next higher layer, the expiration of the GTSs, or maintenance of the minimum Contention Access Period (CAP) length.

Figure 3.6 shows the GTS deallocation mechanism initiated by the next higher layer of the PAN coordinator. (1) The MLME shall receive the `MLME-GTS.request` primitive with the GTS Characteristics set to zero and the length and direction subfields set according to the characteristics of the GTS to deallocate. [1] (2) The notification is achieved when the MLME issues the `MLME-GTS.indication` primitive. (3) In case of any deallocation initiated by PAN coordinator, it shall deallocate

[1]By default this has been modified. See Section 3.6

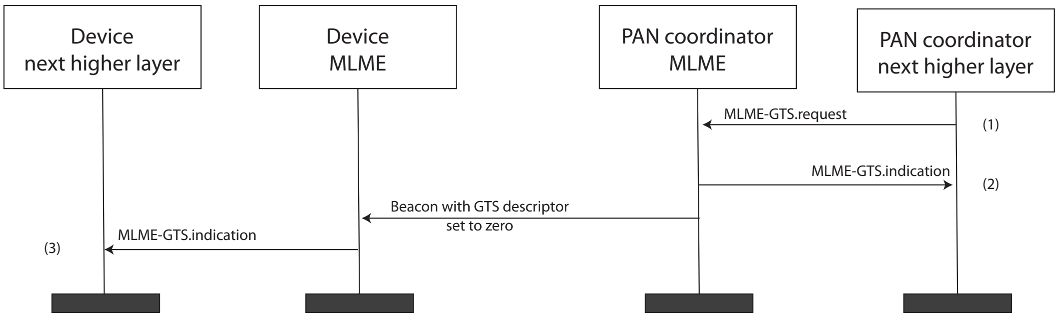
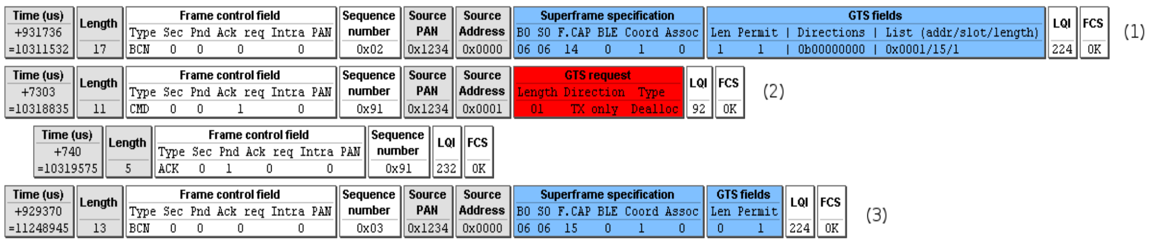


Figure 3.6: MLME-GTS deallocation mechanism initiated by PAN coordinator

the GTS and add a GTS descriptor into its beacon frame corresponding to the deallocated GTS, but with its starting slot set to zero. [9, Sec. 7.5.7.4]

3.3.1 Implementation example



```
Time (us)  | Length | Frame control field               | Sequence | Source | Source  | Superframe specification        | GTS fields                                                  | LQI | FCS
+931736    |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | BO SO F.CAP BLE Coord Assoc     | Len Permit | Directions | List (addr/slot/length)    |     |      (1)
=10311532  | 17     | BCN   0   0       0         0     | 0x02     | 0x1234 | 0x0000  | 06 06  14    0    1     0       | 1      1   | 0b00000000 | 0x0001/15/1                | 224 | OK

Time (us)  | Length | Frame control field               | Sequence | Source | Source  | GTS request                 | LQI | FCS
+7303      |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | Length Direction  Type      |     |          (2)
=10318835  | 11     | CMD   0   0       1         0     | 0x91     | 0x1234 | 0x0001  |  01     TX only  Dealloc    | 92  | OK

Time (us)  | Length | Frame control field               | Sequence | LQI | FCS
+740       |        | Type Sec Pnd Ack req Intra PAN    | number   |     |
=10319575  | 5      | ACK   0   1       0         0     | 0x91     | 232 | OK

Time (us)  | Length | Frame control field               | Sequence | Source | Source  | Superframe specification        | GTS fields  | LQI | FCS
+929370    |        | Type Sec Pnd Ack req Intra PAN    | number   | PAN    | Address | BO SO F.CAP BLE Coord Assoc     | Len Permit  |     |          (3)
=11248945  | 13     | BCN   0   0       0         0     | 0x03     | 0x1234 | 0x0000  | 06 06  15    0    1     0       | 0      1    | 224 | OK
```

Figure 3.7: GTS deallocation mechanism

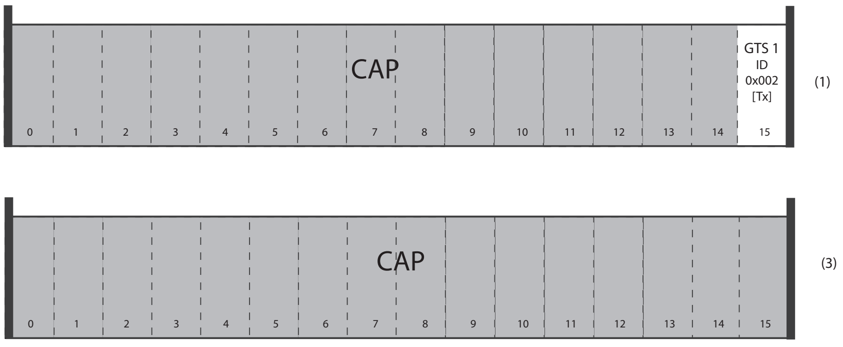


Figure 3.8: Beacon during GTS deallocation mechanism

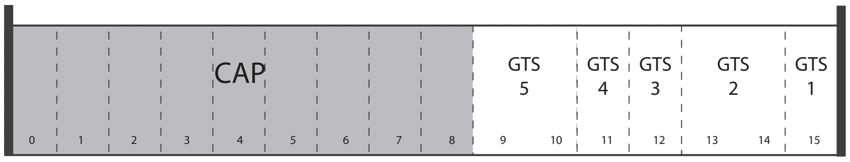
Figure 3.7 shows the process of GTS deallocation initiated by the device and Figure 3.8 shows the state of the superframe after receiving the actual packet. The steps are described:

1. Slot 15 is allocated for the device 0x0001. See state in Figure 3.8.

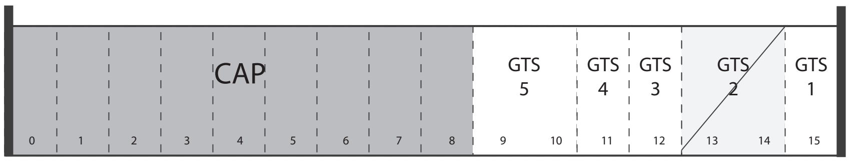
2. The coordinator 0x0000 has received a GTS deallocation from the device 0x0001.

3. Any slot is allocated.

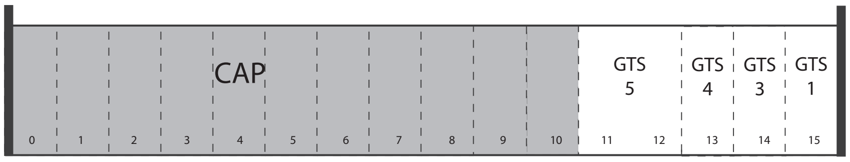
3.4 GTS reallocation



(a)



(b)



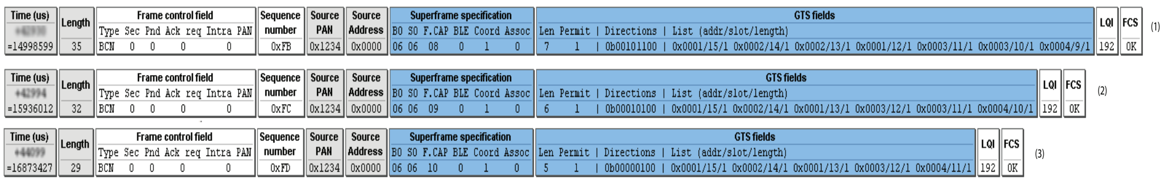
(c)

Figure 3.9: CFP defragmentation on GTS reallocations

The deallocation of a GTS may result in the superframe becoming fragmented. For example, Figure 3.9 shows three stages of a superframe with allocated GTSs. In Figure 3.9a, five GTSs are allocated starting at slots, 15, 13, 12, 11 and 9, respectively. If GTS 2 is now deallocated (Figure 3.9b, there will be a gap in the superframe during which nothing can happen. To solve this, GTS 3 to 5 will have to be shifted to fill the gap, thus increasing the size of the CAP, Figure 3.9c. [9, Sec. 7.5.7.5]

3.4.1 Implementation example

When a slot has been deallocated, it could do a gap in the superframe as Figure 3.11. To solve this, the slots will realign to avoid the existence of those gaps.



Time (us)	Length	Frame control field: Type Sec Pnd Ack req Intra PAN	Sequence number	Source PAN	Source Address	Superframe specification: BO SO F.CAP BLE Coord Assoc	GTS fields: Len Permit | Directions | List (addr/slot/length)	LQI	FCS	
[redacted] =14998599	35	BCN 0 0 0 0	0xFB	0x1234	0x0000	06 06 08 0 1 0	7 1 | 0b00101100 | 0x0001/15/1 0x0002/14/1 0x0002/13/1 0x0001/12/1 0x0003/11/1 0x0003/10/1 0x0004/9/1	192	OK	(1)
[redacted] =15936012	32	BCN 0 0 0 0	0xFC	0x1234	0x0000	06 06 09 0 1 0	6 1 | 0b00010100 | 0x0001/15/1 0x0002/14/1 0x0001/13/1 0x0003/12/1 0x0003/11/1 0x0004/10/1	192	OK	(2)
[redacted] =16873427	29	BCN 0 0 0 0	0xFD	0x1234	0x0000	06 06 10 0 1 0	5 1 | 0b00000100 | 0x0001/15/1 0x0002/14/1 0x0001/13/1 0x0003/12/1 0x0004/11/1	192	OK	(3)

Figure 3.10: GTS reallocation mechanism

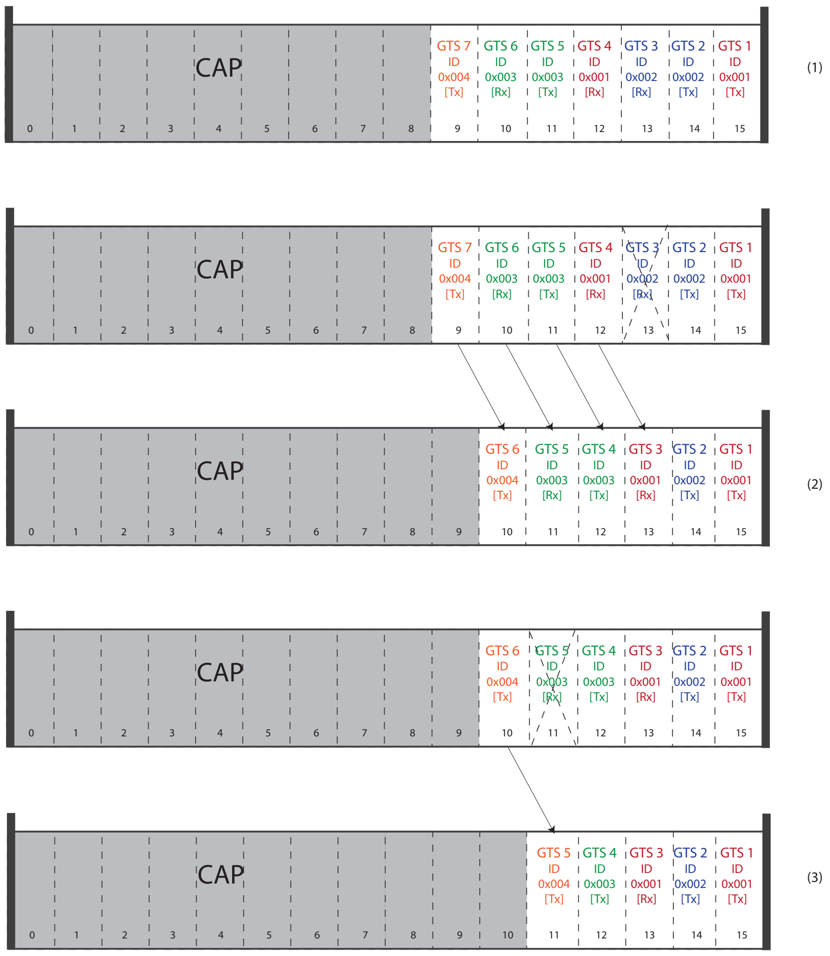


Figure 3.11: Beacon during GTS reallocation mechanism

To illustrate this mechanism we have removed the data packets from a sniffer dump because we just want to focus on what happens with the slots and how the coordinator re-organize them in the superframe.

Figure 3.10 shows the beacons from the sniffer, and Figure 3.11 shows the evolution of the superframe structure. The steps are:

1. All the slots available are allocated for transmission or reception.
2. Slot 13 has been reallocated for reception to the device 0x0002. Slots 9, 10, 11 and 12 have been shifted.
3. Slot 11 has been reallocated for reception to the device 0x0003,. Slot 10 has been shifted.

3.5 GTS expiration

The MLME of the PAN coordinator shall attempt to detect when a device has stopped using a GTS using the following rules:

- For a transmit GTS, it shall assume that a device is no longer using its GTS if a data frame is not received from the device in the GTS at least every $2 * n$ superframes, where n is defined below.
- For receive GTSs, it shall assume that a device is no longer using its GTS if an ACKnowledge (ACK) is not received from the device at least every $2 * n$ superframes, where n is defined below. If the data frames sent in the GTS do not require ACK, the MLME of the PAN coordinator will not able to detect whether a device is using its receive GTS.

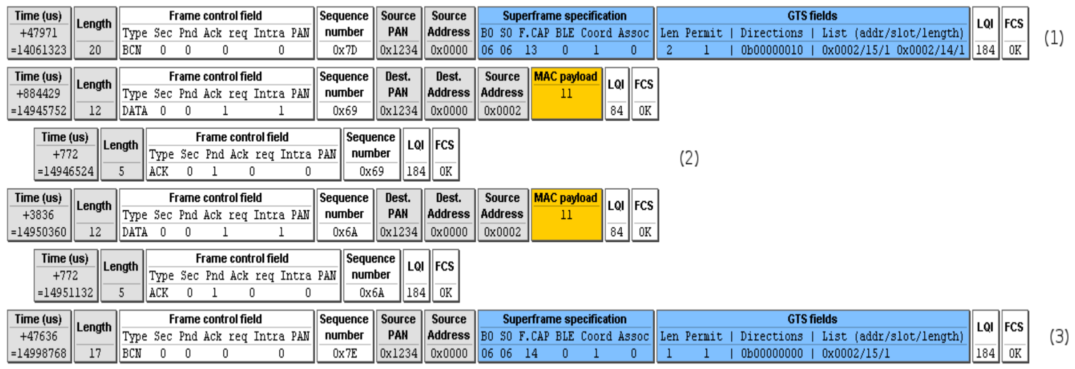
  The value of n is defined as follows:

  $$n = \begin{cases} 2^{8-macBeaconOrder} & 0 \leq macBeaconOrder \leq 8 \\ 1 & 9 \leq macBeaconOrder \leq 14 \end{cases}$$

3.5.1 Implementation example

Figure 3.12 shows a deallocation mechanism due to an expiration. The steps are described:

1. Slots 15 and 14 are allocated for transmission and reception respectively.



```
(1)
Time (us) | Length | Frame control field                    | Sequence | Source  | Source   | Superframe specification          | GTS fields                                                               | LQI | FCS
+47971    |        | Type Sec Pnd Ack req Intra PAN          | number   | PAN     | Address  | BO SO F.CAP BLE Coord Assoc       | Len Permit | Directions | List (addr/slot/length)                    |     |
=14061323 | 20     | BCN  0   0       0         0            | 0x7D     | 0x1234  | 0x0000   | 06 06 13    0   1     0           | 2   1      | 0b00000010 | 0x0002/15/1 0x0002/14/1                    | 184 | OK

(2)
Time (us) | Length | Frame control field                    | Sequence | Dest.   | Dest.    | Source   | MAC payload | LQI | FCS
+884429   |        | Type Sec Pnd Ack req Intra PAN          | number   | PAN     | Address  | Address  | 11          |     |
=14945752 | 12     | DATA 0   0       1         1            | 0x69     | 0x1234  | 0x0000   | 0x0002   |             | 84  | OK

    Time (us) | Length | Frame control field                | Sequence | LQI | FCS
    +772      |        | Type Sec Pnd Ack req Intra PAN      | number   |     |
    =14946524 | 5      | ACK  0   1       0         0        | 0x69     | 184 | OK

Time (us) | Length | Frame control field                    | Sequence | Dest.   | Dest.    | Source   | MAC payload | LQI | FCS
+3836     |        | Type Sec Pnd Ack req Intra PAN          | number   | PAN     | Address  | Address  | 11          |     |
=14950360 | 12     | DATA 0   0       1         1            | 0x6A     | 0x1234  | 0x0000   | 0x0002   |             | 84  | OK

    Time (us) | Length | Frame control field                | Sequence | LQI | FCS
    +772      |        | Type Sec Pnd Ack req Intra PAN      | number   |     |
    =14951132 | 5      | ACK  0   1       0         0        | 0x6A     | 184 | OK

(3)
Time (us) | Length | Frame control field                    | Sequence | Source  | Source   | Superframe specification          | GTS fields                                                               | LQI | FCS
+47636    |        | Type Sec Pnd Ack req Intra PAN          | number   | PAN     | Address  | BO SO F.CAP BLE Coord Assoc       | Len Permit | Directions | List (addr/slot/length)                    |     |
=14998768 | 17     | BCN  0   0       0         0            | 0x7E     | 0x1234  | 0x0000   | 06 06 14    0   1     0           | 1   1      | 0b00000000 | 0x0002/15/1                                | 184 | OK
```

Figure 3.12: GTS expiration mechanism

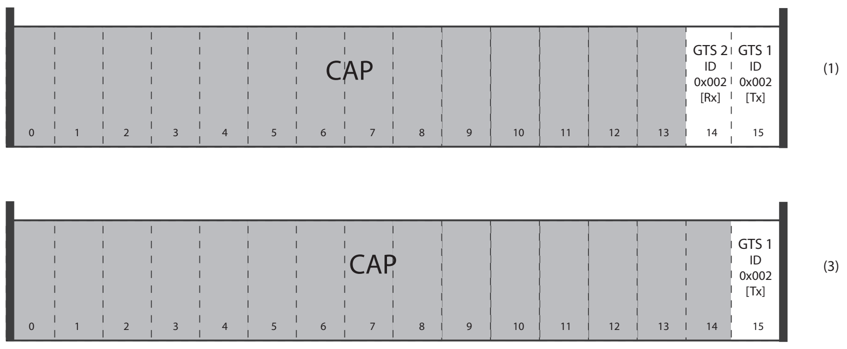


Figure 3.13: Beacon during GTS expiration mechanism

2. The device 0x0002 is sending packets to the coordinator 0x0000. This pattern has been repeated for $n = 2 * 2^{8-macBeaconOrder} = 8$ superframes (macBeaconOrder = 6). After that, the coordinator detects that the slot allocated for reception has not been used for $n = 8$ superframes.

3. Slot 14, the reception slot, has been deallocated.

3.6 Standard vs Implementation

3.6.1 GTS allocation

There are two issues that are important to mention, the GTS request send by the device and the GTS serving queue that the coordinator uses.

GTSs request sent by the device

We assume that the device sends the GTS allocation request once and until it does not receive the confirmation, it is not able to send another GTS request. So, one the device has sent a GTS, it has to wait until the next beacon or, if the GTS descriptor is full, until *aGTSDescPersistenceTime*.

GTSs request served by the coordinator

The current standard does not explicitly defined how the coordinator shall manage the GTS request received. The condition is that the device should have a chance to allocate its slot during *aGTSDescPersistenceTime*. Moreover, the GTSs shall be allocated on a First-Come-First-Serve (FCFS) basis.

The way we have implemented it, is with two queues: one for GTS allocation requests and another for GTS deallocation requests. Figure 3.14 shows the diagram where the implementation of the queues is described. With these queues we are able to serve the GTS within the *aGTSDescPersistenceTime*. The FCFS mechanism is used for allocation and deallocation independently.

On the left of Figure 3.14 is shown the allocation serving mechanism. The coordinator process the allocation request queue in three cases: (1) when a GTS allocation request is received (2) when a GTS deallocation has been served.(3) and when the CFP period has expired, this is important to delete the expired request from the allocation queue. But, as far as we are using a queue, we just check the first element. In case it would be implemented as a buffer it have increased a lot the computational load and unexpected behaviour would appear.

In Appendix B.1 we find another mechanism implemented with just one queue for allocations and deallocations.

3.6.2 CAP maintenance

In IEEE 802.15.4 [9, Sec. 7.5.7.1], the PAN coordinator shall preserve the minimum CAP length of *aMinCAPLength* and take preventive action if the minimum CAP is not satisfied. In our implementation, we assure that the *aMinCAPLength* will be preserve.

During the GTS allocation, the PAN coordinator shall check if there is available capacity in the current superframe and the desired GTS length would not reduce the *aMinCAPLength*. If the *aMinCAPLength* is not preserved, the coordinator shall not allocated it and no empty slot is sent. Then we do not reduce the CAP with the temporary empty GTS slot descriptor.

3.6.3 GTS deallocation

For the GTS deallocation mechanism we do not used the deallocation specified on the standard. When a coordinator starts a deallocation, it shall set the starting slot to zero, and keep this GTS descriptor for *aGTSDescPersistenceTime* beacons. We assume that the device will notify his deallocation with the `MLME_GTS.indication()` as soon as it does not see his GTS descriptor. Hence, there is no necessity to keep the empty GTS descriptor for *aGTSDescPersistenceTime* beacons, and then we serve the GTS allocations requests on the queue.

On the other hand, the default deallocation mechanism is implemented and ready to used, but not activated by default. See Appendix B.2 for more information.

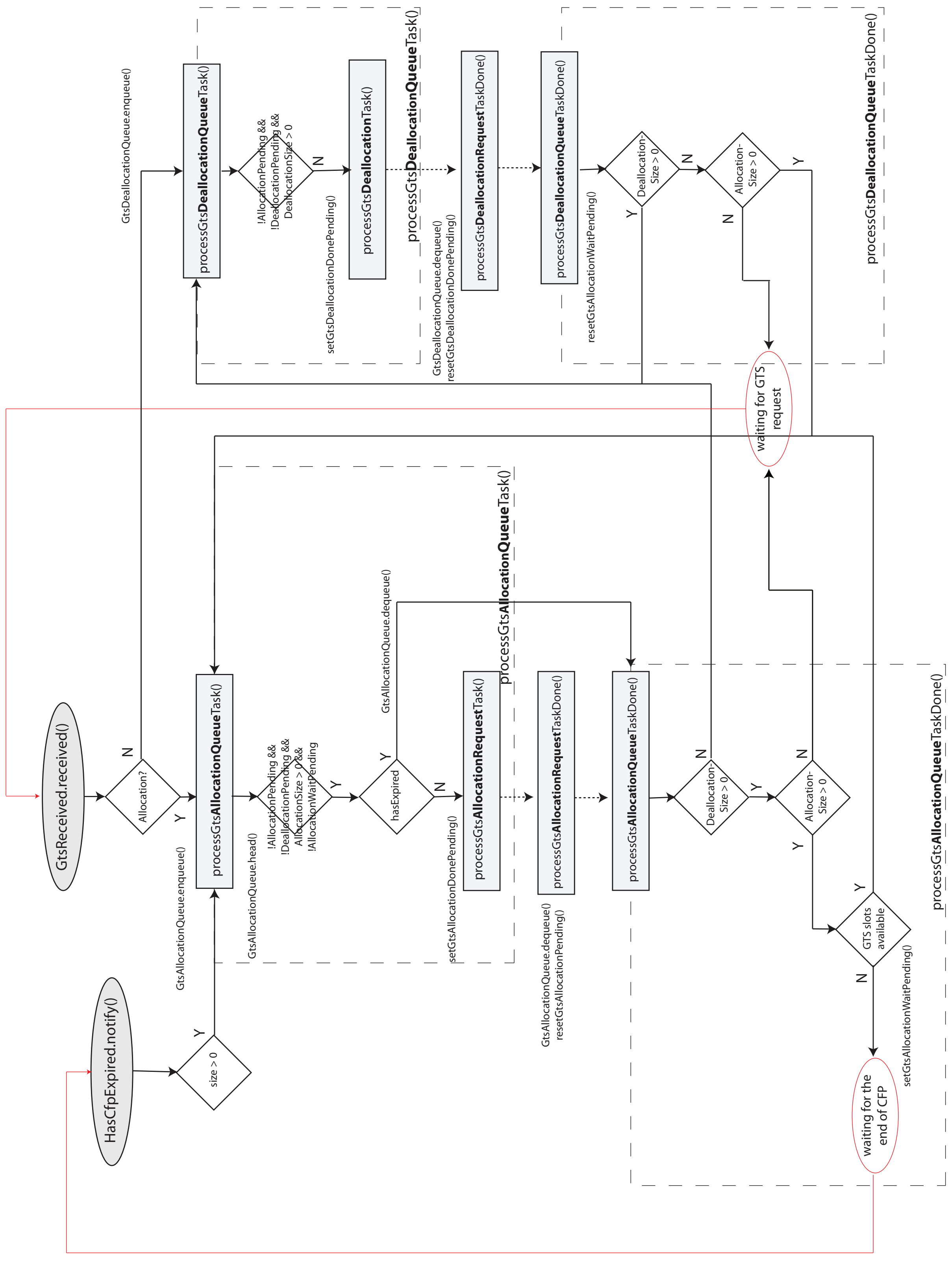

GtsReceived.received()
HasCfpExpired.notify()
Allocation?
N
Y
size > 0
Y
GtsDeallocationQueue.enqueue()
GtsAllocationQueue.enqueue()
processGtsDeallocationQueueTask()
!AllocationPending &&
!DeallocationPending &&
DeallocationSize > 0
N
setGtsDeallocationDonePending()
processGtsDeallocationTask()
processGtsDeallocationQueueTask()
GtsDeallocationQueue.dequeue()
resetGtsDeallocationDonePending()
processGtsDeallocationRequestTaskDone()
processGtsDeallocationQueueTaskDone()
resetGtsAllocationWaitPending()
Deallocation-
Size > 0
Y
N
Allocation-
Size > 0
N
Y
processGtsDeallocationQueueTaskDone()
waiting for GTS
request
processGtsAllocationQueueTask()
GtsAllocationQueue.head()
!AllocationPending &&
!DeallocationPending &&
AllocationSize > 0 &&
!AllocationWaitPending
Y
hasExpired
Y
N
GtsAllocationQueue.dequeue()
setGtsAllocationDonePending()
processGtsAllocationRequestTask()
processGtsAllocationQueueTask()
processGtsAllocationRequestTaskDone()
GtsAllocationQueue.dequeue()
resetGtsAllocationPending()
processGtsAllocationQueueTaskDone()
Deallocation-
Size > 0
N
Y
Allocation-
Size > 0
N
Y
GTS slots
available
Y
N
waiting for the
end of CFP
setGtsAllocationWaitPending()
processGtsAllocationQueueTaskDone()

Figure 3.14: Requests queues management

Chapter **4**

Examples applications

In this chapter, we show an example of how we can use the CFP period for communications between the device node and the coordinator and vice-versa.

To summarize, given an application where we are transmitting data, there are two steps needed in order to be able to transmit/receive data during the CFP:

1. Send the `MLME_GTS.request()` with the desired parameters:

   ```
   call MLME_GTS.request(call GtsUtility.setGtsCharacteristics(slotLength,
   direction, allocation) ,NULL);
   ```

   where,
   ```
   slotLength = {1..7}
   direction = {GTS_TX_ONLY_REQUEST, GTS_RX_ONLY_REQUEST}
   allocation = {GTS_ALLOCATE_REQUEST, GTS_DEALLOCATE_REQUEST}
   ```

2. Once, we receive the `MLME_GTS.confirm()` with status `IEEE154_SUCCESS`, we can send packets through the CFP by using:

   ```
   call MCPS_DATA.request( &m_frame, m_payloadLen, 0, options)
   ```

   where,
   ```
   options = { TX_OPTIONS_GTS | TX_OPTIONS_ACK }
   ```

4.1 TestGts

The *TestGts* example shows how to communicate using the CFP period. This application has been used for all the analysis of the previous chapter.

4.1.1 Functionality

In this application one node takes the PAN coordinator role in a beacon-enabled 802.15.4 PAN, it transmits periodic beacons and waits for incoming data frames. A second node acts as a device, it first scans the pre-defined channel for beacons from the coordinator and once it finds a beacon it tries to synchronize to and track all future beacons. Then it configures the GTS slots that it needs, sending the `MLME_GTS.request()`. The devices send a GTS allocation request for transmission, and if TOS_NODE_ID==0x02 the node tries to send a GTS allocation request for reception as well.

Once, the devices has the slots allocated, they start sending data packets to the coordinator using their slots. The NODE_ID=0x02 will receive packet in its reception slot.

4.1.2 Indicators

LED2 Coordinator and device should both toggle LED2 in unison, indicating that they are sending/receiving beacons.

LED1 They should also toggle LED1 when coordinator receives a packet (`MCPS_DATA.indication()`) or, in the device side, they toggle LED1 when they send a packet (`MCPS_DATA.confirm()`)

LED0 This led is blinking in case of failure, or an error in the transmission. We fail if we do not have any allocated slot.

4.1.3 Options

We summarize the options available for this applications. See Appendix B for more detailed explanation.

Deallocations We have implemented a deallocation mechanism to end the simulation test. See more details for the deallocations initiated by the device are working. (See `MLME_GTS.confirm()` and `EndAlarm`)

Standard deallocation mechanism We have the standard deallocation mechanism initiated by the coordinator disabled. In the standard GTS deallocation initiated by the coordinator, it sets the starting slot to zero and delete it after aGTSDescPersistenceTime. To enable it compile with CFLAGS += -DTKN154_STD_DEALLOCATION. By default we delete the GTS descriptor directly after the deallocation initiate by the deallocation.

GTS request with one queue CFLAGS += DTKN154_ONE_REQUEST_QUEUE By adding this parameter, we switch the two queues of the GTS request, to one queue. In this case, we do not distinguish between allocation and deallocations request, so it will be inefficient depending on the amount of request on the network.

For more information, see Appendix B.

References

[1] Moteiv Corporation. Tmote sky data sheet. Technical report, San Francisco, June 2006. URL `http://www.bandwavetech.com/download/tmote-sky-datasheet.pdf`.

[2] André Cunha, Mário Alves, and Anis Koubâ. IPP Hurray!: An IEEE 802.15.4 protocol implementation (in nesc/tinyos): Reference guide v1.2. Technical Report TR-061106, ISEP-IPP, May 2007.

[3] Sinem Coleri Ergen. ZigBee/IEEE 802.15.4 Summary, September 2004.

[4] Jan-Hinrich Hauer and Adam Wolisz. TKN15.4: An IEEE 802.15.4 MAC Implementation for TinyOS 2. Technical report, Technical University Berlin - Telecommunication Networks Group, March 2009. URL `http://www.tkn.tu-berlin.de/publications/papers/TKN154.pdf`.

[5] Aitor Hernandez. Wireless Process Control using IEEE 802.15.4. Master's thesis, Royal Institute of Technology (KTH), November 2010. URL `https://eeweb01.ee.kth.se/upload/publications/reports/2010/XR-EE-RT_2010_020.pdf`.

[6] Texas Instruments. Smartrf packet sniffer - user manual. Technical Report Rev. 1.10. URL `http://focus.ti.com/docs/toolsw/folders/print/smartrftm-studio.html`.

[7] Philip Levis. Tinyos programming. Technical Report 1.3, October 2006. URL `http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf`.

[8] Zolertia | Z1. URL `http://www.zolertia.com/products/Z1`.

[9] IEEE Std 802.15.4, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) , September 2006. URL `http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf`.

[10] P. Suriyachai, U. Roedig, and A. Scott. Implementation of a MAC protocol for QoS support in wireless sensor networks. *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1–6, mar. 2009.

[11] Crossbow Technology. Crossbow telosb. Technical report, San Jose, California. URL `http://www.willow.co.uk/TelosB_Datasheet.pdf`.

[12] TinyOS. TEPs. Technical report. URL `http://docs.tinyos.net/index.php/TEPs`.

Appendices

Appendix A

Components

In this chapter we will describe each interface and variables used for the components related to the GTS implementation: DeviceCfp , CoordCfp, CfpTransmit.

A.1 DeviceCfp

A.1.1 Uses

`interface MLME_SYNC_LOSS;` Interface that implements the actions in case the device loses synchronization with the PAN coordinator. When it occurs, all its GTS allocations shall be lost [9, Sec. 7.5.7 p. 192], and if the device wants to transmit/receive during the CFP again, it has to request a new slot.

`interface MLME-GET;` Interface that is used to get the configuration variables from the PAN Information Database (PIB). Use: `call MLME_GET.variableName()`.

`interface GtsUtility;` See Section 2.2.2.

`interface FrameUtility;` Interface that provides utilities to access the content of a frame.

`interface Notify<ieee154_status_t> as HasCfpTimeSlots;` Interface that `BeaconSynchronize` signals when we has received a non-empty GTS descriptor. Once the notify is received in the `DeviceCfp`, it checks if they have or not their slots, and signal the `MLME_GTS.confirm()` or `MLME_GTS.indication()`.

`interface Get<ieee154_GTSentry_t*> as GetCfpTimeSlots;` Interface to share the GtsEntry from the `BeaconSynchronize` component and the `DeviceCfp`. Once the `HasCfpTimeSlots` notifies the device, we copy the received entries to the database in the `DeviceCfp`.

`interface Pool<ieee154_txframe_t> as TxFramePool;` Interface that provides the pool that has the packets in the same queue and control it.

`interface Pool<ieee154_txcontrol_t> as TxControlPool;` Interface that controls the previous pool, and prevents it to overflow.

`interface Timer<TSymbolIEEE802154> as GTSDescPersistenceTimeout;` Once it receives the acknowledgement to the GTS request command, the devices shall continue to track beacons and wait for at most *aGTSDescPersistenceTime* superframe. If the GTS descriptor is received and the `MLME_GTS.confirm()` is signalled this timer stops running. If the timer fired the `MLME-GTS.confirm` primitive is sent with status of *NO_DATA*.See [9, Sec 7.5.7.2 p. 193].

`interface FrameTx as GtsRequestTx;` Interface that contains the function to send packet through the radio. It sends packets in the CAP.

A.1.2 Provides

`interface Init;` The basic synchronous initialization interface.

`interface MLME_GTS;` Interface that provides all the primitives necessary to use the GTS. The `request()`, `confirm()` and `indication()` are provided.

`interface Get<ieee154_GTSentry_t*> as GetGtsDeviceDb;` Interface that is provided to share the device database and use it in the `CfpTransmitP` component. The pointer to the vector with the GTS, is shared but is not modified in other components.

A.1.3 Variables

`ieee154_GTSentry_t deviceDb[2]` Variable that stores the database for the device. There is the configuration for the transmission and reception slots (starting slot, length and direction). This databse is read from the `CfpTransmitP` component using the `GetGtsDeviceDb`, which manages the transmissions and receptions during the CFP.

`/* ----------- Vars to GTS request ------------ */`

`bool m_gtsOngoingRequest` Indicates that a GTS request is been processed.

`bool m_gtsOngoing` Indicates that we have a GTS slot allocated.

`uint8_t m_gtsCharacteristicsType` Indicates which is the characteristics type of the ongoing GTS request.

`ieee154_GTSentry_t GtsEntryRequested` Variable that stores the requested GTS entry.

A.2 CoordCfp

A.2.1 Uses

`interface MLME-GET;` Interface that is used to get the configuration variables from the PIB. Use: `call MLME_GET.variableName()`.

`interface TimeCalc;` Interface that provides time utilities.

`interface IEEE154Frame as Frame;` Interface that allows access to the content of a IEEE 802.15.4 frame. It is not the same as `FrameUtility`.

`interface SuperframeStructure as SF;` Superframe utilities to get parameters and configuration of the beacon. Superframe start time, slot duration, number of CAP and CFP slots, GTS fields are some of the parameters we can get.

`interface Pool<ieee154_txframe_t> as TxFramePool;` Interface that provides the pool that has the packets in the same queue and control it.

`interface Pool<ieee154_txcontrol_t> as TxControlPool;` Interface that controls the previous pool, and prevents it to overflow.

`interface GtsUtility;` See Section 2.2.2.

`interface FrameRx as GtsRequestRx;` Interface that is used to transmit the `MLME_GTS.request` and detect the ACK.

`interface Notify<uint8_t> as HasGtsSlotExpired;` Interface that is signalled by the `CfpTransmitP` components each time a GTS slot has finished

`interface Notify<bool> as HasCfpExpired;` Interface that is signalled by the `CfpTransmitP` when the CFP has finished. It updates the GTS database with the new slots.

`interface Queue<ieee154_rxframe_gts_t> as GtsAllocationQueue;` Queue that manages the GTS allocations requests.

`interface Queue<ieee154_rxframe_gts_t> as GtsDeallocationQueue;` Queue that manages the GTS deallocation requests.

A.2.2 Provides

`interface Init;` The basic synchronous initialization interface.

`interface WriteBeaconField as GtsInfoWrite;` Interface that is used by the `BeaconTransmitP` to write the GTS descriptor in the beacon, in case it was needed. In that interface we analyse all the database that the coordinator has looking for a GTS slot. It is done, just once if the GTS slots don't change.

`interface MLME_GTS;` Interface that provides all the primitives necessary to use the GTS. The `request()`, `confirm()` and `indication()` are provided.

`interface Get<ieee154_GTSdb_t*> as GetGtsCoordinatorDb;` Interface that provides to share the coordinator database, with the data of all the slots. The pointer is shared and the rest of the components that use that interface will be able to modify that variable.

`interface Notify<bool> as GtsSpecUpdated;` Interface that notifies to other components that the GTS descriptor in the beacon has to be rewritten.

A.2.3 Variables

`uint8_t m_state, m_stateDe;` Variables to manage the allocations/deallocations queues. PROCESS_DONE_PENDING, REQUEST_PENDING.

`uint8_t m_gtsSpecField [GTS_LIST_MULTIPLY * CFP_NUMBER_SLOTS + 2];` Buffer to write the coordinator database in the beacon

`/* ------------ Vars to store GTS entries ------------ */`

`ieee154_GTSentry_t db[CFP_NUMBER_SLOTS];` This is the vector where all the slots informations are stored. This database is used by the `CfpTransmit` by using the `GetGtsCoordinatorDb` interface. The `ieee154_GTSentry_t` contains the following information: starting slot, length, direction, associate device address (short address) and a field indicating the expiration. It is not used simultaneously by the `CoordCfp` and `CfpTransmit`, because `CoordCfp` uses the database for the next superframe and the `CfpTransmit` reads the current one.

`ieee154_GTSdb_t GTSdb;` This is the database shared with the `GetGtsCoordinatorDb` interface. It contains the vector of each `ieee154_GTSentry_t` and the number of actual GTS slots that has been used.

`ieee154_GTSentry_t dbNext[CFP_NUMBER_SLOTS];` This vector, is only used by the `CoordCfp` to set the new GTS descriptor for the next superframe.

`ieee154_GTSdb_t GTSdbNext;` Database that points to the dbNext vector.

`uint8_t indexGtsSlot;` Index of the current GTS slot.

`/* ------------ Vars to GTS request ------------ */`

`ieee154_GTSentry_t* currentEntry` Pointer to the current slot entry which the mote is using in the current network.

`uint16_t m_expiredTimeout` Variable that stores the expiration timeout for the selected Beacon Order (BO).

`#ifdef TKN154_STD_DEALLOCATION`

`/* ------------ Vars to GTS deallocation ------------ */`

`ieee154_GTSentry_t GtsEntryDeallocate[CFP_NUMBER_SLOTS]` Variable that stores entries to be deallocated.

`uint8_t m_gtsDeallocateIn` Pointer to the new position.

`uint8_t m_gtsDeallocateOut` Pointer to the out position.

`#endif`

A.3 CfpTransmit

In the `CfpTransmitP` component, we have some pieces of code with conditional inclusions (*preprocessor directives*) to be able to differentiate device and coordinator without the use of `nesC` conditional sentences, and reduce the size of the program.

A.3.1 Uses

`interface Leds;` Interface that provides commands for controlling three LEDs.

`interface TransferableResource as RadioToken;` As it has been shown in Section 2.3 we need to share the radio resource with all the components to use it properly.

`interface GetNow<token_requested_t> as IsRadioTokenRequested;` Interface that checks if any component has request the `RadioToken`, and immediately it will be released. In a normal operation, the `RadioToken` is not request/release but in some cases like SCAN or RESET is needed.

`interface Alarm<TSymbolIEEE802154,uint32_t> as CfpSlotAlarm;` This is the alarm that triggers every instant one slot has started. When it fires we check in which slot we are at the moment, and if we have to change the radio state to transmission, reception or inactive. Figure 2.5 and Figure 2.6

`interface Alarm<TSymbolIEEE802154,uint32_t> as CfpEndAlarm;` This alarm indicates that the CFP has finished and the token has to be transfer to the `BeaconTransmitP` or `BeaconSynchronizeP` to continue in the inactive period or transmit, receive the beacon. Figure 2.5

`interface TimeCalc;` Interface that provides time utilities.

`interface GetNow<bool> as IsRxEnableActive;` Interface that indicates if the reception is active.

`interface Notify<bool> as RxEnableStateChange;` Interface that notifies when the state has changed.

`interface Notify<const void*> as PIBUpdateMacRxOnWhenIdle;` Interface that notifies when the `RxOnWhenIdle` has changed.

`interface SuperframeStructure as SF;` Superframe utilities to get information and configuration of the beacon. Superframe start time, slot duration, number of CAP and CFP slots, GTS fields are some of the parameters we can get.

`interface RadioTx;` Interface that provides the functions to be able to transmit using the radio.

`interface RadioRx;` Interface that provides the functions to be able to receive using the radio.

`interface RadioOff;` Interface that provides the functions to turn off the radio.

`interface MLME-GET;` Interface that is used to get the configuration variables from the PIB.

`interface MLME-SET;` Interface that is used to set the configuration variables to the PIB.

`interface IEEE154Frame as Frame;` Interface that allows to access the content of a IEEE 802.15.4 frame. It is not the same as `FrameUtility`.

`#ifndef IEEE154_BEACON_TX_DISABLED`

`interface Get<ieee154_GTSdb_t*> as GetGtsCoordinatorDb;` Interface that is used just when we compile the program for the coordinator. See on page 40)

`#else`

`interface Get<ieee154_GTSentry_t*> as GetGtsDeviceDb;` Interface that is used just when we compile the program for the coordinator. See on page 42)

`#endif`

A.3.2 Provides

`interface Init;`

`interface FrameTx as CfpTx;` Interface that is used to transmit data in the contention free period instead of contention access period.

`interface FrameRx as CfpRx;` Interface that is used when a packet is received in the contention free period.

`interface Notify<bool> as WasRxEnabled;` Interface that indicates if the radio was enabled to receive packets.

`#ifndef IEEE154_BEACON_TX_DISABLED`

`nterface Notify<uint8_t> as HasGtsSlotExpired;` Interface that is signalled by the `CfpTransmitP` components each time a GTS slot has finished

`nterface Notify<bool> as HasCfpExpired;` Interface that is signalled by the `CfpTransmitP` when the CFP has finished. It updates the GTS database with the new slots.

`#endif`

A.3.3 Variables

`/* ------------ Vars to CFP tx ------------ */`

`norace ieee154_status_t m_txStatus;` Indicates the transmission status.

`norace uint32_t m_transactionTime;` To store the transaction time needed for the packets.

`norace uint16_t m_slotDuration;` To store the slot duration of the beacon. It is equal to: $timeSlotDuration = aBaseSlotDuration * 2^{SO}$

`norace uint16_t m_guardTime;` That guard time, it is the time that the radio needs to change its state to inactive. See [?].

`norace uint32_t m_capDuration;` It is the CAP duration. When the radio resource is transferred to the `CfpTransmitP` component, that time is recalculated because it is possible that the CAP has changed.

$capDuration = numCapSlots * timeSlotDuration = numCapSlots * aBaseSlotDuration * 2^{SO}$

`norace uint32_t m_gtsDuration;` It is the CFP duration. When the radio resource is transfer to the `CfpTransmitP` component, that time is recalculated because it is possible that the CFP has changed.

$gtsDuration = cfpDuration = numCfpSlots * timeSlotDuration = numCfpSlots * aBaseSlotDuration * 2^{SO}$

`norace uint32_t m_cfpInit;` It is the time when the contention free period has to start. It is used by the TrackAlarm to align the device and the coordinator.

$cfpInit = capDuration + superframeStartTime$

`norace ieee154_macMaxFrameRetries_t m_macMaxFrameRetries;` Variable that is used to control the number of retransmission that are available.

`norace ieee154_macMaxFrameTotalWaitTime_t m_macMaxFrameTotalWaitTime;` Variable that stores the maximum time that it waits until the acknowledge is received and we have to retransmit the packet, if the `m_macMaxFrameRetries` is not achieved.

`norace ieee154_macRxOnWhenIdle_t macRxOnWhenIdle;` Variable that indicates in which state of the radio, during the idle state. If it `TRUE` the radio has to be in reception state, to wait for new packets.

`norace bool m_lock;` To prevent non-atomic access to a variable, instead of using atomic block, because although it will assure that the block is accessed with atomic access, it will take some time.[7, Sec. 4.5]

`norace slot_mode_t slot_mode` State to distinguish between the slot modes for the current slot. {TX_MODE, RX_MODE, IDLE_MODE }

`norace uint8_t m_slotNumber;` Indicates the slot number, where it is at the moment. Variable that is used to load the entry that belongs to the actual slot number.

`norace uint8_t m_numGtsSlots;` Indicates the index for the coordinator database.

`norace ieee154_txframe_t *m_currentFrame;` Variable that points to the actual frame, that has been prepared to send.

`norace ieee154_txframe_t *m_lastFrame;` Variable that points to the last frame that has been sent.

`/* ------------ Vars to expiration GTS ------------ */`

`norace ieee154_GTSentry_t* currentEntry;` Variable that points to the current slot entry depending on the slot number.

`norace uint16_t m_expiredTimeout;` Variable that stores the number of superframes that the coordinator has to wait if the slot it is not used. Afterwards the coordinator starts the GTS expiration mechanism.[9, Sec. 7.5.7.6]

`#ifndef IEEE154_BEACON_TX_DISABLED`

`ieee154_GTSentry_t* deviceDb` Device's slots database to read and check the currect direction.

`#endif`

Appendix **B**

Optional functionalities

We provide some useful functions that are not enabled by default. Table B.1 show the *preprocessor directives* used with their descriptions.

Preprocessor directives	**Description**
`IEEE154_BEACON_TX_DISABLED`	Distinguish between the device and the coordinator in a network
`IEEE154_GTS_DISABLED`	To disable the GTS implementation
`TKN154_STD_DEALLOCATION`	The deallocation activated by default does not set the length of the slot to zero, and keep the descriptor in the slot for `aGTSDescPersistenceTime`. Section B.2
`TKN154_ONE_REQUEST_QUEUE`	Use a unique queue for allocation and deallocations. Section B.1

Table B.1: *preprocessor directives* and the functionalities they enable

To enable them, we need to add in the *Makefile* or *Makefile.include* (depending on necessities) the following line:

```
CFLAGS += -DPREPROCESSOR_DIRECTIVE_NAME
```

B.1 Single GTS request queue

In Section 3.6.1, by default we use a double queue to manage the GTSs requests, one for allocations and another for deallocations.

On the other hand, to have a FCFS basis for allocations and deallocations, we use a single queue.

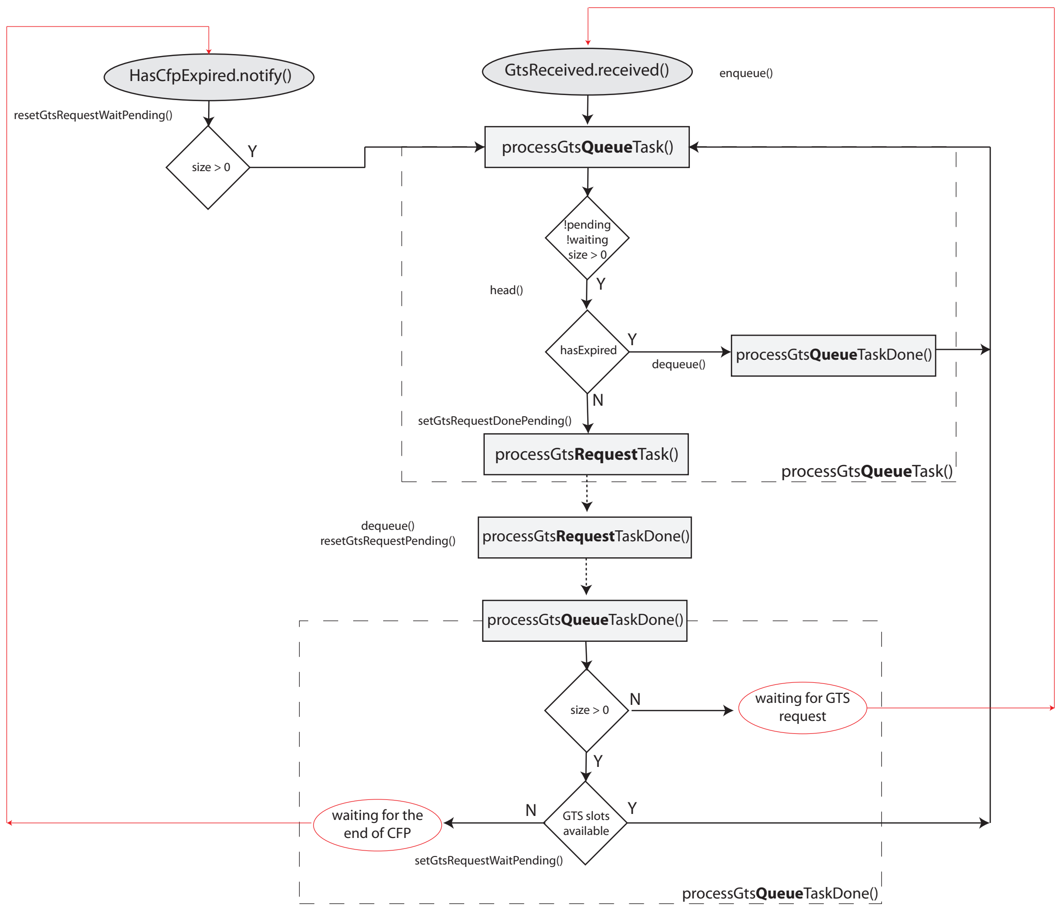


Figure B.1: Requests queues management

Figure B.1, show the diagram for the GTS queue. If all the slots were allocated, the GTS request could not be serve because there were not space available, the PAN coordinator stop serving requests until the end of the CFP, where we start again serving request, to check if within this time, some deallocations were done.

To enable this feature: `CFLAGS += -DTKN154_ONE_REQUEST_QUEUE`

B.2 Deallocation mechanism

In Section 3.6.3 the standard does not explicitly define the deallocation mechanism. But we provide it, as an optional feature, in case it is necessary for your application.

With this mechanism, when the PAN coordinator, initiate a deallocation mechanism, before remove it from the GTS descriptor, we set the starting slot to zero. The descriptor shall remain in the beacon frame for *aGTSDescPersistenceTime* superframe. [9, Sec. 7.5.7.4]

To enable this feature: `CFLAGS += -DTKN154_STD_DEALLOCATION`

Note, that this feature has not been completely tested.

APPENDIX **C**

Timing validations

In this Appendix we show the experimental values that we get and compare them with the standard to see the error and standard deviation that we have. In addition, we have a comparison between TKN15.4 and IPP Hurray in [5].

C.1 Beacon intervals

The implementation is not completely standard compliant due to hardware limitations. The limitation is coming from our oscillator that runs at 32.768 kHz.

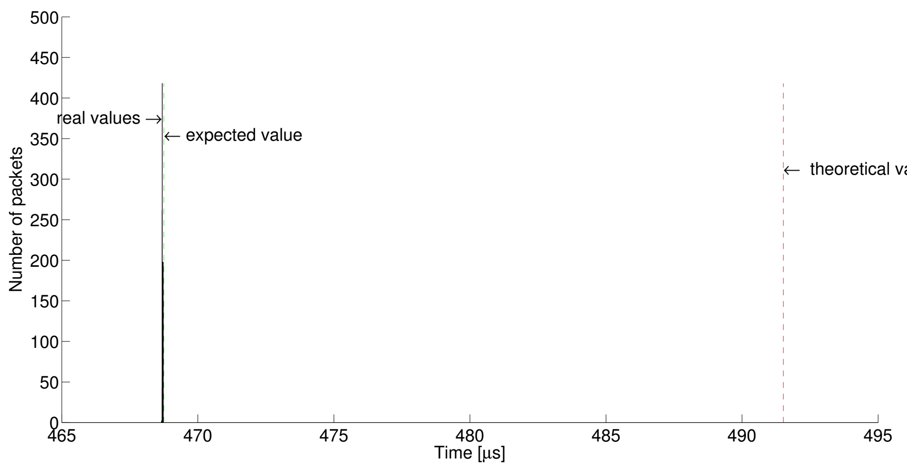


Figure C.1: Histogram of the beacon time interval for BO=5

Figure C.1 shows the histogram of time interval between two consecutive beacons for BO=5. The green dashed line on the left represents the expected value taking into account that the $T_{\text{symbol}} = 15.259\mu s$. The red dashed line represents the theoretical value of the beacon interval. We see that the histogram is quite tight and close to the expected value.

Table C.1 shows the influence of this difference. As we can see on the fifth column (Error). The difference between the value that we get from the mote and the theoretical value is increasing with the BO, but the relative error shown in column six remains constant. Moreover, on the seventh column, we see that the standard deviation is lower than $25ns$.

BO	**BI**			**Error**		**Precision**
	$T_{symb} = 16\mu s$	$T_{symb} = 15.26\mu s$	experimental	|BI(16μs)-BI(exp.)|		Std. deviation
	[ms]	[ms]	[ms]	[ms]	[%]	[μs]
1	30.7	29.3	29.3	1.5	4.7	7.5
2	61.4	58.6	58.6	2.9	4.7	8.4
3	122.9	117.2	117.2	5.7	4.7	9.8
4	245.8	234.4	234.3	11.4	4.6	10.0
5	491.5	468.8	468.7	22.8	4.6	14.4
6	983.0	937.5	937.4	45.6	4.6	13.7
7	1966.1	1875.0	1874.9	91.2	4.6	5.0
8	3932.2	3750.0	3749.8	182.3	4.6	2.6
9	7864.3	7500.0	7499.7	364.6	4.6	7.5
10	15728.6	15000.0	14999.5	729.2	4.6	13.8
11	31457.3	30000.0	29998.2	1459.0	4.6	20.3
12	62914.6	60000.0	59996.4	2918.1	4.6	6.5
13	125829.1	120000.0	119992.9	5836.2	4.6	10.2
14	251658.2	240000.0	239985.8	11672.5	4.6	22.1
				Total	4.6 %	10.8 μs

Table C.1: Beacon interval on TKN154.4 implementation

C.2 Beacon intervals and time slots

Table C.2 shows the comparison between the theoretical values, calculated as

$$\begin{aligned} BI &= aBaseSlotDuration * aNumSuperframeSlots * 2^{BO} \\ TimeSlotDuration &= aBaseSlotDuration * 2^{BO}, \end{aligned}$$

for the theoretical case. And for the implementation we convert the timings that we obtain to symbols of the real implementation,

$$BI_{impl} = \frac{t_{BI_{impl}}}{t_{symbol}} = t_{BI_{impl}}[s] * 2 * 32.768kHz$$
$$TimeSlotDuration_{impl} = \frac{t_{timeSlot}}{t_{symbol}} = t_{timeSlot}[s] * 2 * 32.768kHz$$

	Theoretical		Experimental	
BO [symbol]	**BI** [symbol]	**Time Slot** [symbol]	**BI** [symbol]	**Time Slot** [symbol]
1	1920	120	1917.9	-
2	3840	240	3837.8	-
3	7680	480	7677.7	475.2
4	15360	960	15357.4	957.1
5	30720	1920	30716.2	1916.0
6	61440	3840	61435.9	3831.0
7	122880	7680	122873.8	7683.0
8	245760	15360	245749.7	15353.3
9	491520	30720	491501.5	30712.2
10	983040	61440	9830004.3	61429.6
11	1966080	122880	1965964.4	122868.3
12	3932160	245760	3931927.3	245721.0
13	7864320	491520	7863855.5	-
14	15728640	983040	15727707.8	-

Table C.2: Beacon interval and Time slot duration on TKN154.4 implementation $T_{\text{symbol}} = 16\mu s$ for theoretical values. $T_{\text{symbol}} = 15.259\mu s$ for experimental values

Table C.3 shows the comparison between the theoretical values with $T_{\text{symbol}} = 16\mu s$. We just collect the data from the report, to show them together.

C.3 Timeslot durations

In Table C.4 we can see the difference between the expected value from the standard and the value that we obtain from the experimental results.

In Table C.5 we can see the difference between the expected value from the standard and the value that we obtain from the experimental results. These values are in symbols considering $T_{\text{symbol}} = 16\mu s$.

	Theoretical		Implementation	
BO [symbol]	**BI** [symbol]	**Time Slot** [symbol]	**BI** [symbol]	**Time Slot** [symbol]
1	1920	120	1829.1	-
2	3840	240	3660.1	-
3	7680	480	7322.1	453.2
4	15360	960	14646.0	912.8
5	30720	1920	29293.3	1827.3
6	61440	3840	58589.8	3653.5
7	122880	7680	117181.6	7327.1
8	245760	15360	234365.2	14642.1
9	491520	30720	468732.4	29289.4
10	983040	61440	937465.9	58582.9
11	1966080	122880	1874889.8	117176.3
12	3932160	245760	3749778.1	234337.8
13	7864320	491520	7499557.0	-
14	15728640	983040	14999111.0	-

Table C.3: Beacon interval and Time slot duration on TKN154.4 implementation. $T_{\text{symbol}} = 16\mu s$

In the previous tables, we have compared the timeslot duration in *symbols* and *seconds*. But with them, we can not assure that the implementation is correct, to check it we show a table with the values in symbols, considering for the theoretical case $T_{\text{symbol}} = 16\mu s$, and for the experimental values $T_{\text{symbol}} = 15.259\mu s$ which is the real T_{symbol} we have in our motes. Then in Table C.6

	Theoretical	Experimental		
SO	**TS**	**TS**	**Error**	**Std deviation**
	[ms]	[ms]	[ms]	[ms]
1	1.92	-	-	-
2	3.84	-	-	-
3	7.68	7.25	0.43	0.36
4	15.36	14.50	0.86	0.35
5	30.72	29.22	1.50	0.35
6	61.44	58.46	2.98	0.37
7	122.88	117.03	5.85	0.51
8	245.76	234.27	11.49	0.32
9	491.52	468.63	22.89	0.28
10	983.04	937.33	45.71	0.46
11	1966.08	1874.82	91.26	0.53
12	3932.16	3749.41	182.75	1.49
13	7864.32	-	-	-
14	15728.64	-	-	-

Table C.4: Table with the Time Slots as a function of the SO, assuming $BO > SO$

	Theoretical	Experimental		
SO	**TS**	**TS**	**Error**	**Std deviation**
	[symbols]	[symbols]	[symbols]	[symbols]
1	120.00	-	-	-
2	240.00	-	-	-
3	480.00	453.22	26.78	22.53
4	960.00	912.80	47.20	24.55
5	1920.00	1827.26	92.74	23.96
6	3840.00	3653.52	186.48	23.37
7	7680.00	7327.07	352.93	22.93
8	15360.00	14642.05	717.95	19.96
9	30720.00	29289.42	1430.58	17.26
10	61440.00	58582.90	2857.10	28.97
11	122880.00	117176.32	5703.68	33.36
12	245760.00	234337.85	11422.15	93.38
13	491520.00	-	-	-
14	15728.64	-	-	-

Table C.5: Table with the Time Slots as a function of the SO, assuming $BO > SO$. $T_{\text{symbol}} = 16\mu s$.

	Theoretical	Implementation		
SO	**TS**	**TS**	**Error**	**Std deviation**
	[symbols]	[symbols]	[symbols]	[symbols]
1	120	-	-	-
2	240	-	-	-
3	480	475.2	4.8	23.6
4	960	957.1	2.9	25.7
5	1920	1916.0	4.0	25.1
6	3840	3831.0	9.0	24.5
7	7680	7683.0	3.0	24.0
8	15360	15353.3	6.7	20.9
9	30720	30712.2	7.8	18.1
10	61440	61428.6	11.4	30.4
11	122880	122868.3	11.7	35.0
12	245760	245721.0	39.0	97.9
13	491520	-	-	-
14	983040	-	-	-

Table C.6: Table with the Time Slots as a function of the SO, assuming $BO > SO$. $T_{\text{symbol}} = 16\mu s$ for theoretical values. $T_{\text{symbol}} = 15.259\mu s$ for experimental values

Appendix **D**

Timeslot boundary

D.1 GTS mechanism

The slot size T_s should be as small as possible in order to reduce the energy consumption to use large BO and small SO.

To determine the lower bound for T_s we consider the analysis made in [10]. We summarize the slot size analysis that suits for us. The required slot size has been determined by aligning T_{st}, time slot used for transmission, and T_{sr}, time slot used for reception. Thus, using the acknowledge mechanism, they obtained

$$T_s = t_g + t_{ts} + t_{xm} + t_{pm} + t_{xa} + t_{tr} + t_g, \tag{D.1}$$

where t_g is the guard time required at the beginning and end of the slot to compensate for clock drifts between nodes; t_{ts} is the transfer time from the MAC layers data FIFO buffer into the buffer of the CC2420 transceiver and t_{xm} is the time the message needs to be transmitted. t_{pm} is the time the receiver needs to process the message and initiate the transmission of an acknowledgement message and t_{xa} is the transmission time for the acknowledge. Thereafter, the message is transferred from the CC2420 buffer to the micro controller where it is either enqueued in the MAC layer application FIFO buffer for forwarding or in the MAC layer application FIFO buffer (transfer time t_{tr}). Finally, we have again a small guard time.

A lower bound for T_s can be given by determining the best theoretical possible execution time for all steps that have to be carried out in a slot. Assuming that the message processing time is negligibly small ($t_{pm} = 0$) and that the message transfers times t_{ts} and t_{tr} between CPU and CC2420 are determined by the speed of the SPI interface interconnecting them but taking into account the large processing overhead that reduced the theoretical SPI bus transfer rates to 173.91 kbits/s instead of the theoretical maximum rate of 500 kbits/s. Thus, a practical lower bound for the slot size on the TelosB mote can be given by

$$
\begin{aligned}
T_s &= t_g + t_{ts} + t_{xm} + t_{pm} + t_{xa} + t_{tr} \\
&= 0.15ms + 0.31ms + (x+11) \cdot 46\mu s + 0.40ms + \\
&+ (x+13) \cdot 32\mu s + 0.21ms + 0.35ms + 1.28ms + \\
&+ (x+5) \cdot 46\mu s + 0.15ms,
\end{aligned} \tag{D.2}
$$

where x is the payload in bytes. As t_{pa} on sender side and t_{tr} on receiver side are executed in parallel and $t_{tr} > t_{pa}$.

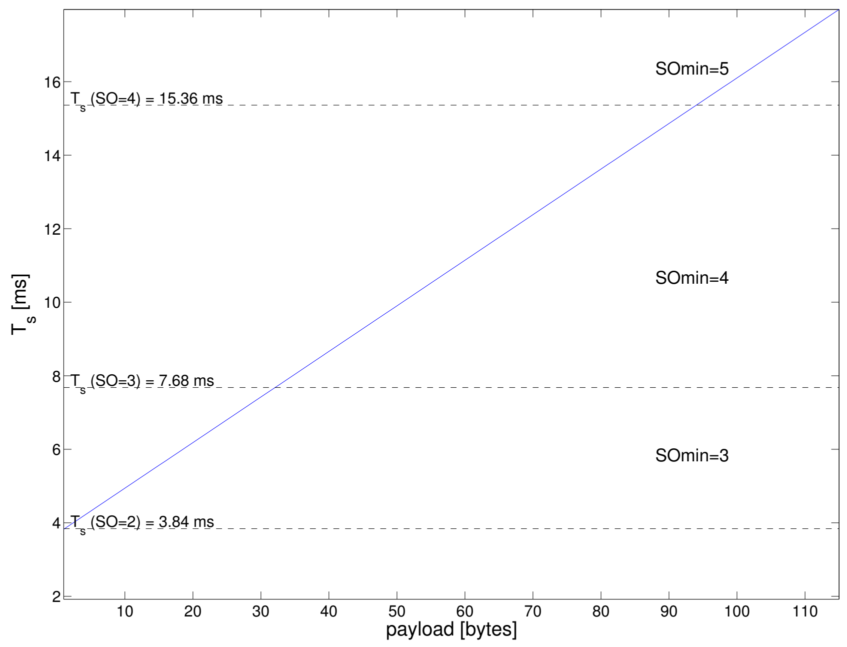


Figure D.1: Lower bound for T_s as a function of payload

Figure D.1 shows represents the timeslot size (T_s) as a function of the payload length given by the Equation (D.2). As far as the timeslot length depends on the payload length we can determine the lower bound for T_s. The dashed lines are the lines refereed to the timeslot length given a certain SO. The value of SO is show above the dashed line.

The region between two dashed lines correspond to the range where we need to use $SO \geq SO_{min}$. To summarize we have:

$$
SO_{min} = \begin{cases} 3 & 1 \leq payload \leq 32 \\ 4 & 32 < payload \leq 94 \\ 5 & 94 < payload \leq 117 \end{cases}
$$