KTH Electrical Engineering

Modification of the IEEE 802.15.4 Implementation
Extended GTS implementation

AITOR HERNÁNDEZ

Stockholm July 22, 2011

TRITA-EE 2011:002
Version: 1.0

Contents

Acronyms

BI	Beacon Interval. 6, 8
CAP	Contention Access Period. 3, 5, 6, 17
CFP	Contention-Free Period. 3, 5, 6, 19, 21
GTS	Guaranteed Time Slot. 3–8, 11, 12, 17–20
LED	Light-Emitting Diode. 12
MAC	Medium Access Control. 5
PHY	Physical Layer. 6
SD	Superframe Duration. 6
SO	Superframe Order. 6
WSAN	Wireless Sensor Actuator Network. 3

Chapter 1

Introduction

In a network with multiple nodes transmitting packets to the same sink, there is a high probability of collision if we do not have a good protocol. With the IEEE 802.15.4 standard protocol [4], we have the possibility to transmit within two different periods. In one of them the nodes compete each other to get the channel, and in the other they have their own time slot, where the channel is only available for them, so they do not collide. These two periods are called Contention Access Period (CAP) and Contention-Free Period (CFP) respectively.

For TinyOS we use an existent implementation, the TKN15.4 [1]. Base on that, we implemented the Guaranteed Time Slot (GTS) mechanisms [2].

The IEEE 802.15.4 has been designed to allocate a maximum number of slots during the CFP. It is set to seven time slots, GTS slots. With this number in a Wireless Sensor Actuator Network (WSAN) scenario with multiple processes there is not enough. This is the motivation for the modification that we present in this document.

This technical report shows the modifications that are needed to have an increased number of GTS slots. In Chapter 2 the modifications are shown. Mainly, the modifications includes the superframe structure and the beacon frame.

The remainder of the documents is organized as follows. Chapter 3 shows an example scenario, where we use this modification. Chapter 5 explains step by steps the procedure to modify a network that is using the CAP to transmit during the CFP.

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

1.1 Installation

In this Chapter we explain the steps that are required to test the example application in the Chapter 3 or to create your new application in your project.

First of all, we download and configure the TinyOS Contribution tree where the GTS implementation is located.

1. Download the TinyOS Contribution tree from the CVS server. http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/kth/
2. After downloading the code, you should have the following folder in the `tinyos-2.x-contrib/kth/` folder: { `tkn154-gts/`, `tkn154-gts-mod/`, `tkn154-gts-mod-all/`}.
3. Add the environmental variable for TinyOS Contribution tree. The following command adds a line at the end of your ∼`/.bashrc` file, with the new variable:

   ```
   echo export TOSCONTRIB=/home/kthwsn/workspace/tinyos-2.x-contrib
   » ∼/.bashrc
   ```

   Check that the symbol » is wrong, you need to write two >.
4. Restart the terminal and compile the application `tinyos-2.x-contrib/kth/tkn154-gts/beacon-enabled/TestGts/coordinator`. It should not give any error.
5. Compile the application `tinyos-2.x-contrib/kth/tkn154-gts-mod/beacon-enabled/TestNetworkManager/coordinator`. It should not give any error.

If the applications compile, you could continue reading the next chapters.

Chapter 2

Modifications

The purpose of this GTS modification is to allocate a bigger number of slots for the CFP period. To achieve that goal, we increase the number of slots in the superframe and we modify the beacon frame structure in order to fit the new GTS descriptor on it.

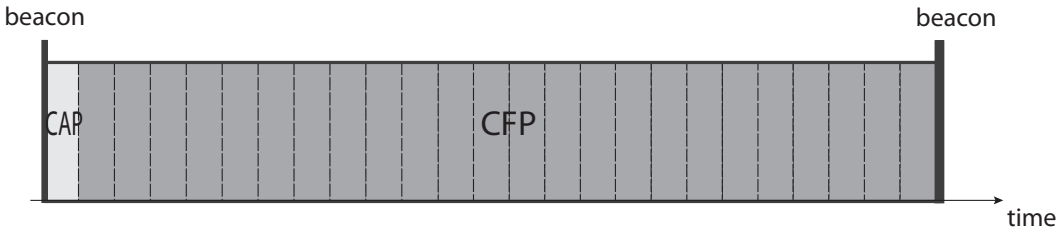

Figure 2.1: Superframe example with one slot in the CAP and twenty four slots in the CFP

Figure 3.2 shows an example of a superframe structure with twenty five slots, where one is for the CAP and twenty four for the CFP.

Moreover, the GTS mechanism are disabled in this modification and the slots are allocated manually by the coordinator, or, if any, the network manager connected to the coordinator.

2.1 Superframe Structure

In order to increase the number of slots in the superframe, we increase the `aNumSuperframeSlots` Medium Access Control (MAC) parameter. By increasing

this value, we modify the Beacon Interval (BI) and Superframe Duration (SD) values. The value is stored in the `TKN154_MAC.h` header file.

The following equations show these modifications:

$$\begin{aligned} SD &= 2^{SO} * aBaseSuperframeDuration \\ &= 2^{SO} * aBaseSlotDuration * aNumSuperframeSlots \qquad (2.1) \\ BI &= 2^{BO} * aBaseSuperframeDuration \\ &= 2^{BO} * aBaseSlotDuration * aNumSuperframeSlots \qquad (2.2) \\ & \qquad (2.3) \end{aligned}$$

where the slot duration keeps constant for a given Superframe Order (SO),

$$\begin{aligned} slotDuration &= 2^{SO} * aBaseSlotDuration \\ &= 2^{SO} * 60. \qquad (2.4) \end{aligned}$$

It is important to remark the modification of the BI and SD. The intervals are proportionals to the `aNumSuperframeSlots`. For example, Table 2.1 shows the BI for a different number of `aNumSuperframeSlots`. It is computed in ms with a theoretical $T_{symbol} = 16\mu s$, which is not the real value in our implementation.

In order to accommodate more slots in the CFP period, we need to modify the `CFP_NUMBER_SLOTS` constant. The constants is used to allocate memory for the GTS database in the coordinator (and the device). The value is stored in the `TKN154.h` header file.

2.2 Beacon Structure

The main contribution and modifications are done in the beacon structure to accommodate the increased GTS descriptor. We have considered a maximum of 32 GTS slots because the maximum number of bytes allowed in the Physical Layer (PHY) payload is limited by `aMaxBeaconPayloadLength`.

Figure 2.2 shows the beacon frame format with the modifications. If we compare the new beacon frame with the one in the standard, we see that only the *Pending Address filed* has been deleted. We see the changes on the specific fields. The Superframe Specification and GTS fields has changed.

In Figure 2.3, the format of the superframe specification field is shown. We increase the size of the *Final CAP Slot* to have enough bits to represent all the possible values of the final slot. The extreme case that we consider is with 32 slots,

aNumSuperframeSlots	16	20	32
BO	**BI** [ms]	**SI** [ms]	**SI** [ms]
1	30.7	38.4	61.4
2	61.4	76.8	122.9
3	122.9	153.6	245.8
4	245.8	307.2	491.5
5	491.5	614.4	983.0
6	983.0	1228.8	1966.1
7	1966.1	2457.6	3932.2
8	3932.2	4915.2	7864.3
9	7864.3	9830.4	15728.6
10	15728.6	19660.8	31457.3
11	31457.3	39321.6	62914.6
12	62914.6	78643.2	125829.1
13	125829.1	157286.4	251658.2
14	251658.2	314572.8	503316.5

Table 2.1: Table with the Beacon orders and its equivalent in time for different number of `aNumSuperframeSlots` and $T_{symbol} = 16\mu s$

Octets: 2	1	4/10	0/5/6/10/14	2	variable	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Secutiry Header	Superframe Specification	GTS fields	Beacon Payload	FCS
MHR				MAC Payload			MFR

Figure 2.2: Beacon frame format

Bits: 0 - 3	4-7	8-12	13	14	15
Beacon Order	Superframe Order	Final CAP Slot	Battery Life Extension (BLE)	PAN Coordinator	Association Permit

Figure 2.3: Format of the Superframe Specification field

so we need $log_2 32 = 5bits$. Then the *Battery Life Extension* fields is moved to the bit 13, which was not used.

The last modification regards the *GTS information* field or GTS descriptor, which is shown in Figure 2.4. In the *GTS Specification*, the *GTS Descriptor count*

has increased the number of bits because now we could have up to 32 GTS slots.

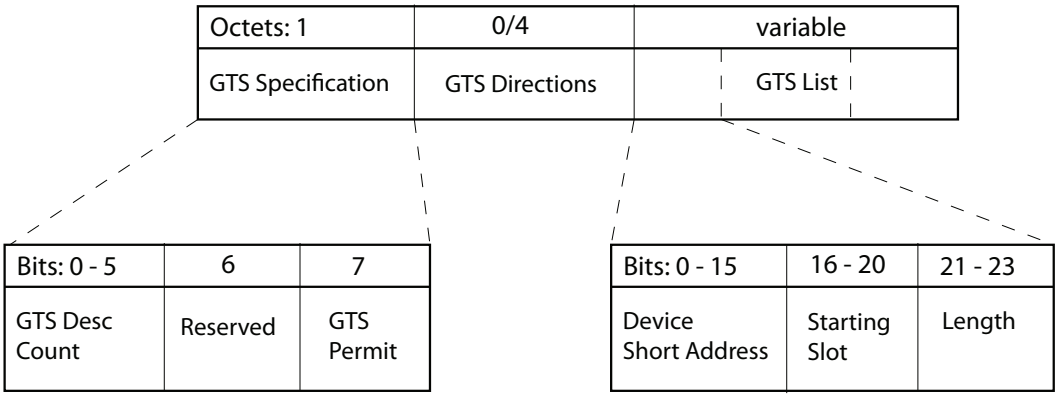

Figure 2.4: Format of the GTS information field

The *GTS Directions* fields is set with 4 bytes, to allocate the $4 * 8 = 32 slots$. Moreover for each GTS entry in the list, we need to increase it by one byte in order to increase the bits in the *Starting Slot* and *Length* fields.

2.3 GTS mechanisms

The GTS mechanism (allocation, deallocation and expiration)are disabled because these functionalities are not used in our experiments. If in your application are needed we need to modify the `DeviceCfpP.nc` and `CoordCfpP.nc`.

The IEEE 802.15.4 was designed to have one slot allocated for each mote that request it, and in case the coordinator receives two requests of the same mote, it discards. With the current modifications the coordinator adds the slots, and it could add slots with the same characteristics and it does not complains. In some scenarios it is also interesting to have more than one slot allocated for the same mote, with the same characteristics, in order to improve the reliability and reduce the periodicity. Because by using the default GTS, the period of our transmission is the BI. With this goal, we have another modification that could be found in the folder:

http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/kth/tkn154-gts-mod-all

Then, compiling with this new modification, the superframe structure could be as we have in Figure 2.5. It shows an example where the node with ID 1, has two pairs of GTS slots allocated, two slots for transmission and two for reception.

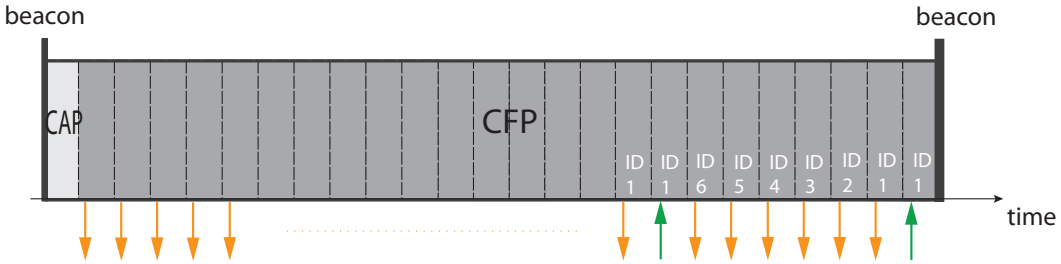

Figure 2.5: Superframe example with one slot in the CAP and twenty four slots in the CFP.

Chapter 3

Usage

In this Chapter we show an example of an application which uses the modification of the GTS. In this example we have the GTS descriptor alternating between a 18 slots allocation for the motes {1 .. 18 } and an empty GTS descriptor.

On the other hand, we provide a tool, the `TestPromiscuous`, which acts as a sniffer. We need this tool because the default CC2420 DK sniffer parses the standard IEEE 802.15.4 frame format. Thus, to analyze the new beacon frame we use the new `TestPromiscuous` mote.

3.1 TestNetworkManager

Figure 3.1 show the star topology that we use for the usage example. We need to compile at least one coordinator and one device in order to see the behavior and the functionality of this example.

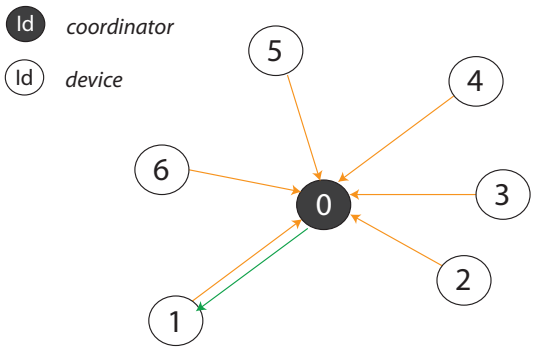

Figure 3.1: Star topology for the `TestNetworkManager`

In Figure 3.2 is shown the superframe structure with the slots allocated for the slots {1 .. 18 }. Only the mote with ID 1 has a slot for transmission and recepcion.

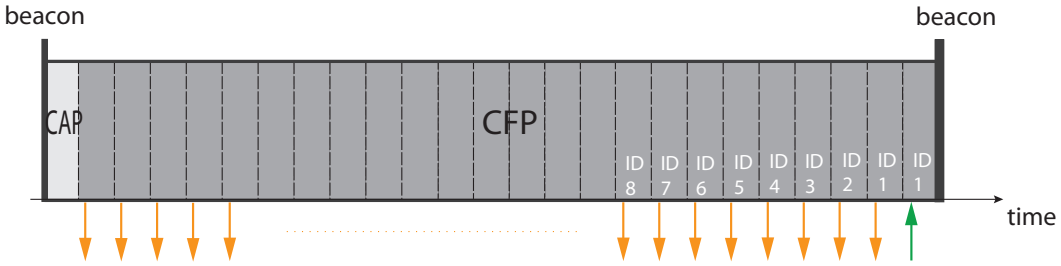

Figure 3.2: Superframe structure with the slots allocated in the CFP

3.1.1 Coordinator

As we know in the beacon-enabled mode of the IEEE 802.15.4 standard protocol, there is a coordinator that sends periodically beacons frames with the information of the network.

In this scenario, this mote manages the GTS descriptor. We have configure the coordinator to switch between a default GTS descriptor with all the slots allocated or an empty descriptor.

When the coordinator receives a message from the mote with ID 1, it replies to that mote with another message in the next slot. When it receives the message, the blue Light-Emitting Diode (LED) blinks.

3.1.2 Device

This application sends messages to the coordinator using the allocated slot for him in the superframe. Every time it receives a beacon, it checks if he has a slots allocated in order to send the packet.

When a beacon is received, the blue LED blinks. It shall blink at the same time as the coordinator. Moreover, if the green LED blinks it means that the message has been successfully sent to the coordinator.

3.2 TestPromiscuous

In Section 2.2 we see that the beacon frame format has change. This application is useful to analyze and check the messages that are transmitted in the network,

but mainly it provides the translation between the beacon in raw data and the description of the different fields.

Chapter 4

Results

Chapter 5

From CAP to CFP

In this Chapter we explain which are the modifications that we need to do in our application in order to transmit using the GTS mechanism. We assume that the application that we want to modify is using the standard IEEE 802.15.4 and transmitting during the CAP period.

All the steps that are explained here, are based on the default `TestData` application that exists in the TinyOS tree. They are placed in the `apps/test/tkn154/beacon-enabled/TestData`.

Before focusing on the different applications that we need to modify, we need to configure the compilation files to include the paths with the new implementation. In the latest version of TinyOS and TKN15.4 some compilation processes are modified. So if you still are using an old one, it is time to update.

5.1 Device

The next step is to modify the `Makefile.include` and add the new paths.

1. Go to the TinyOS tree and navigate to the `tos/lib/mac/tkn154/` folder.
2. Edit the `Makefile.include` and add the following lines at the beginning of the file:

```
TKN154_GTS_MOD_ALL_TOS=$(TOSCONTRIB)/kth/tkn154-gts-mod-all/tos
CFLAGS += -I$(TKN154_GTS_MOD_ALL_TOS)/lib/mac/tkn154 \
                    -I$(TKN154_GTS_MOD_ALL_TOS)/lib/mac/tkn154/dummies \
                    -I$(TKN154_GTS_MOD_ALL_TOS)/lib/mac/tkn154/interfaces/public \

TKN154_GTS_MOD_TOS=$(TOSCONTRIB)/kth/tkn154-gts-mod/tos
CFLAGS += -I$(TKN154_GTS_MOD_TOS)/lib/mac/tkn154 \
                    -I$(TKN154_GTS_MOD_TOS)/lib/mac/tkn154/dummies \
                    -I$(TKN154_GTS_MOD_TOS)/lib/mac/tkn154/interfaces/public \
```

```
include $(TKN154_GTS_MOD_PLATFORM_INCLUDE)

TKN154_GTS_ROOT=$(TOSCONTRIB)/kth/tkn154-gts
TKN154_GTS_TOS=$(TKN154_GTS_ROOT)/tos
TKN154_GTS_PLATFORM_INCLUDE?=$(TKN154_GTS_TOS)/platforms/$(PLATFORM)/
mac/tkn154/Makefile.include

CFLAGS += -I$(TKN154_GTS_TOS)/lib/mac/tkn154 \
          -I$(TKN154_GTS_TOS)/lib/mac/tkn154/dummies \
          -I$(TKN154_GTS_TOS)/lib/mac/tkn154/interfaces/MCPS \
          -I$(TKN154_GTS_TOS)/lib/mac/tkn154/interfaces/MLME \
          -I$(TKN154_GTS_TOS)/lib/mac/tkn154/interfaces/private \
          -I$(TKN154_GTS_TOS)/lib/mac/tkn154/interfaces/public \
          -I$(TKN154_GTS_TOS)/lib/debug \
          -I$(TKN154_GTS_TOS)/lib/interfaces
TKN154_GTS_MOD_PLATFORM_INCLUDE?=$(TKN154_GTS_MOD_TOS)/platforms/
$(PLATFORM)/mac/tkn154/Makefile.include
include $(TKN154_GTS_PLATFORM_INCLUDE)
```

3. To reduce the program size we disable some functionalities of the protocol by adding the following lines to the application `Makefile`:

```
CFLAGS += -DIEEE154_ASSOCIATION_DISABLED \
          -DIEEE154_DISASSOCIATION_DISABLED \
          -DIEEE154_PROMISCUOUS_MODE_DISABLED \
          -DIEEE154_COORD_REALIGNMENT_DISABLED \
          -DIEEE154_COORD_BROADCAST_DISABLED
```

In the Sections below we describe the steps for the applications. The modifications are divided in two applications, the Coordinator which needs to include the GTS database management, and the Device.

5.2 Coordinator

With the modifications that we explain in this documents, the allocation method disappears. It is the coordinator, who assign the slots manually. The assignment could be done by a computer [3, Self-Triggered Controller], or automatically by the coordinator based in a certain policy.

After the compilation files are properly set, we need to modify the source files of the applications. Add the `MLME_GTS` interface to the declarations and the other interfaces. In the `TestCoordReceiverC.nc` file add the following line to the *module* section:

```
uses interface MLME_GTS;

uses interface Get<ieee154_GTSdb_t*> as SetGtsCoordinatorDb;
uses interface GtsUtility;
uses interface Notify<bool> as IsEndSuperframe;
```

```
provides interface Notify<bool> as GtsSpecUpdated;
```

And in the *implementation* section we add the callbacks for the events of the interfaces and the required functions:

```
event void MLME_GTS.confirm (
                uint8_t GtsCharacteristics,
                ieee154_status_t status) {}

event void MLME_GTS.indication (
                uint16_t DeviceAddress,
                uint8_t GtsCharacteristics,
                ieee154_security_t *security) {}

event void IsEndSuperframe.notify( bool val ) {}

command error_t GtsSpecUpdated.enable() {return FAIL;}
command error_t GtsSpecUpdated.disable() {return FAIL;}
default event void GtsSpecUpdated.notify( bool val ) {return;}
```

And in the `TestDataApp.nc` we add the wiring with the following line in the *implementation* section:

```
App.MLME_GTS -> MAC;

MAC.SubGtsSpecUpdated -> App.GtsSpecUpdated;
App.GtsUtility -> MAC;
App.SetGtsCoordinatorDb -> MAC;
App.IsEndSuperframe -> MAC;
```

At this point we have the controller ready to be configure the GTS descriptor and transmit or receive data in the CFP period. In this example we fix the GTS descriptor to have two slots allocated to send an receive data between the mote 1, and the rest 15 slots for transmissions for the motes {2 .. 16}

```
void setDefaultGtsDescriptor() {
        uint8_t i;
        ieee154_GTSdb_t* GTSdb;

        GTSdb = call SetGtsCoordinatorDb.get();

        GTSdb->numGtsSlots = 0;
```

```
        i=0;
        call GtsUtility.addGtsEntry(GTSdb, i+1,
                IEEE154_aNumSuperframeSlots - (i+1),
                1, GTS_TX_ONLY_REQUEST);
        call GtsUtility.addGtsEntry(GTSdb, i+1,
                IEEE154_aNumSuperframeSlots - (i+1 + 1),
                1, GTS_RX_ONLY_REQUEST);

        for (i=2; i < 17; i++)
                call GtsUtility.addGtsEntry(GTSdb, i+1,
                        IEEE154_aNumSuperframeSlots - (i+1+1),
                        1, GTS_TX_ONLY_REQUEST);

        signal GtsSpecUpdated.notify(TRUE);
}
```

At this point, the beacon shall contain the information for the 16 GTS slots. Use the application `TestPromiscuous` that you could find in the folder `tkn154-gts-mod/apps/TestPromiscuous` to check if the GTS is correct.

5.3 Device

For the device, we need add similar lines to the code. Add the `MLME_GTS` interface to the declarations and the other interfaces. In the `TestDeviceSenderC.nc` file add the following line to the *module* section:

```
uses interface MLME_GTS;

uses interface GtsUtility;
uses interface Notify<bool> as IsEndSuperframe;

uses interface GetNow<bool> as IsGtsOngoing;
```

And in the *implementation* section we add the callbacks for the events of the interfaces and the required functions:

```
event void MLME_GTS.confirm (
                uint8_t GtsCharacteristics,
                ieee154_status_t status) {}

event void MLME_GTS.indication (
                uint16_t DeviceAddress,
                uint8_t GtsCharacteristics,
```

```
                ieee154_security_t *security) {}

event void IsEndSuperframe.notify( bool val ) {}
```

And in the `TestDataApp.nc` we add the wiring with the following line in the *implementation* section:

```
App.MLME_GTS -> MAC;

App.GtsUtility -> MAC;
App.IsEndSuperframe -> MAC;
App.IsGtsOngoing -> MAC.IsGtsOngoing;
```

Then, everything is ready to transmit and received using the CFP period. To transmit a packet, we need to add the `TX_OPTIONS_GTS` in the `MCPS_DATA.request` primitive. Below we have an example where we check if we have an slot available before trying to send.

```
if (call IsGtsOngoing.getNow()) {
        if (call MCPS_DATA.request (
                        &m_frame, // frame,
                        m_payloadLen, //payloadLength,
                        0, // msduHandle,
                        TX_OPTIONS_ACK // TxOptions,
           ) != IEEE154_SUCCESS)
        call Leds.led0On();
}
```

References

[1] 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.

[2] Aitor Hernandez. IEEE 802.15.4 implementation for TinyOs based on TKN15.4. GTS implementation. Technical report, TinyOS Contribution, January 2011. URL http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/kth/tkn154-gts/doc/pdf/gts_implementation.pdf.

[3] Aitor Hernandez, Joao Faria, and Jose Araujo. Wireless water tanks. diferent scenarios and controllers. Technical report, Royal Institute of Technology (KTH), July 2011.

[4] 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.