Università di Pisa · Scuola Superiore Sant'Anna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Master in Computer Science and Networking

Master Thesis

Design and implementation of an energy-aware protocol for multi-hop real-time communications in wireless networks

Supervisors

Prof. Giorgio Buttazzo
Dr. Gianluca Franchino

Candidate

Davide La Rosa

Academic Year 2011/2012

Contents

Summary

The technological evolution in the last decade made possible the widespread use of the sensor networks in many fields and with it, also the interest has grown due to the possible vaste range of applications. The design of wireless embedded systems for real-time applications requires a careful management of timing and energy requirements. The delay introduced in the network has a significant impact on the system performance and should be closely evaluated together with the ability of the protocol to delivery the data in time, avoiding its expiration.

This work presents an extension of the WBuST real-time protocol for multi-hop wireless networks composed of embedded devices. These devices, usually battery powered, also requires careful managing of the power consumption in order to extend the battery life at most. This protocol organizes the network as a cluster tree and provides the routing capabilities to transfer the packets among the devices, guaranteeing the packet delivery within their deadlines. At the end of the document, an extensive analysis of the protocol performance, through experimental tests, are presented.

List of Figures

Acronyms

WSN	Wireless Sensor Network
LR-WPAN	Low Rate-Wireless Personal Area Network
FFD	Full-Function Device
RFD	Reduced-Function Device
ZC	Zigbee Coordinator
ZR	Zigbee Router
ZED	Zigbee End Device
ADC	Analog to Digital Converter
TTRT	Target Token Rotation Time
TTHT	Target Token Holding Time
CSMA	Carrier Sense Multiple Access
CSMA\CA	Carrier Sense Multiple Access\Collision Avoidance
CSMA\CD	Carrier Sense Multiple Access\Collision Detection
CCA	Clear Channel Assessment
LQI	Link Quality Indication
RSSI	Received Signal Strength Indicator
BER	Bit Error Rate
MCU	Micro-Controller Unit
DSC	Digital Signal Controller
ISR	Interrupt Service Routine
CW	Communication Window
BAS	Budget Allocation Schemes
PA	Proportional Allocation
NPA	Normalized Proportional Allocation
LA	Local Allocation
MLA	Modified Local Allocation
WCAU	Worst Case Achievable Utilization
DMR	Deadline Miss Ratio
ADMR	Average Deadline Miss Ratio

1 Introduction

Nowadays, thanks to the continuously increasing technological evolution in such fields like computer science, electronics and telecommunications, and the miniaturizing capabilities reached by the current production standards, sensors have spread everywhere. Starting from simple tasks like acquiring and elaborating physical signals, they evolved to the point of being able to cooperate among them, building up networks, exchanging data and providing high computational capabilities systems. That is where the definition *Smart sensors* comes from.

1.1 Wireless Sensor Networks

A sensor network (Wireless Sensor Network or WSN) is a group of smart sensors connected each other through appropriate interfaces capable of exchanging control and monitoring information to reach a given objective [1]. Each sensor is commonly a transducer that is tightly bound to the measured system. Each unit could be connected to the others by two main ways: cables or wireless communications. Cables connections are generally used when there are no particular limitations about the power consumed by the devices or the communication infrastructure is partially existing. Besides, this system enforce the security of the transmitted data since, to be able to read the information exchanged, a physical access to the communications line is required. On the other side this system entails potentially critical issues such as the difficulty of installation and maintenance in harsh environments, the placement cost of the devices and the correspondent cables, and the difficulty of adding or moving around the sensors due to the intrinsic static nature of the infrastructure.

Due to this latter problems, wireless networks seem to be a much better solution, even if additional issues like signal propagation, interference, noise, security and also legislations have be faced. All these aspects concur to the project design complexity that immediately reflects on the final realization costs. That is why is important to carefully assess the needs that the designed system has to satisfy, in order to provide the best solution minimizing costs, potential problems and anticipate future enhancements.

The main objective of a sensor network is to collect, verify, elaborate and distribute the data gathered by every single sensor in such a way that, after some time, the network, as a whole, got a wide and uniform information on the monitored system. Since generally, every basic unit, does not have big power and computational capability, it is important to reduce unnecessary data transmission or whatever other kind of energy waste at the most.

Even if it is possible to use the algorithms adopted in the *ad-hoc* wireless networks, usually they are incompatible with the requirements needed by the sensor networks. The main reasons are:

- the number of nodes belonging to a sensor network could be orders of magnitude bigger than those in an ad-hoc network
- spatial density of the sensors could be really high
- sensors have tight power budget to observe
- the network topology could vary frequently in time
- sensors could experience failures or malfunctionings and these should not affect the network efficiency
- sensors generally adopt a broadcast communication scheme

The relative small distance among the sensors allows to use multi-hop communication strategies to connect sensors within the same network or between neighbouring ones. This permits lower transmission power consumption, improving the node energy requirements. This is a fundamental aspect since sensors operate with power sources that generally cannot be replaced or, at least, replaced frequently.

The needs asked by the sensor networks, pushed the industry to the creation of a new infrastructure called *Low Rate- Wireless Personal Area Network*

(LR-WPAN), characterized by small range communications, low energy consumption and low transmission speed. The LR-WPANs are easy to install, provide reliable data transfer, are cheap and they maintain a simple and flexible protocol. In the last years, companies are getting even more interested to the applications that can be built with the WSN, basically for three reasons: the reduction of the costs due the installation as well as the testing and verification phases, the reduction of the number of movable parts (like the connectors) potentially subject to failures and the increment of the sensors density that could provide more accurate data about the industrial processes leading to an improvement of the production quality.

1.2 Smart Sensors

Smart Sensors are much more than simple transducers of physical quantities in that they have measuring, storage, computing and communication capabilities to provide the correct and unbiased representation of the monitored data. The heart of these integrated devices is the microcontroller that coordinates in a seamless way all the other modules. A general scheme of a smart sensor is shown in figure 1.1.

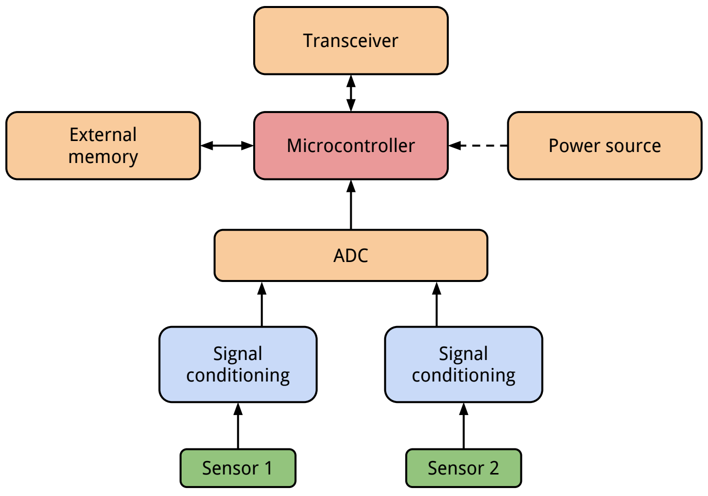

Figure 1.1. *Block diagram of a smart sensor device.*

The basic blocks a smart sensor consists of are the following:

- one or more sensors to transform the real physical quantity into an analog voltage

- one or more signal conditioning units that appropriately manipulate the analog signal, through operations like amplification, filtering, isolation, etc. to make the sensor output suitable for the next processing stage
- an Analog to Digital Converter (ADC) unit that is in charge of converting a continuous analog signal into a sequence of discrete values
- an external memory to store data, that could be both permanent (FLASH) or temporary (RAM)
- a communication interface to exchange data with other devices (cable, infrared, wireless, etc.)
- a microcontroller that is in charge of interfacing with the attached components elaborating the received digital signals, synchronizing and communicating with the other nodes in the network
- a power source (mains, battery, solar panels, etc.) that feeds all the active electronic components

The employment of smart sensors allow to shift the elaboration of the information towards the physical phenomenon, avoiding a central processor in charge of handling all the data coming from the whole monitored environment. Smart sensors are able to take local decisions, based upon the assessment of the events, preventing the overload of the network as well as the control center of unnecessary, redundant or useless data. The current technological evolution made this kind of devices extremely cheap, expanding the base of the potential users. Besides that, thanks to their flexibility, they also allow fast systems design and implementation shortening the time required to reach the market.

1.3 Network Classifications

Networks are usually classified by the maximum range they can cover or, with a more technical view, by the interconnection structure between the nodes. The first classification, shown in figure 1.2, identifies networks by their reach. Generally the range covered by a network is proportional to the maximum bandwidth the network itself can sustain and inversely proportional to the power consumption.
Starting from the smallest one, we have the *Body Area Network* (BAN), the *Personal Area Network* (PAN), the *Local Area Network* (LAN), the *Metropolitan Area Network* (MAN) and at last the *Wide Area Network* (WAN). In the

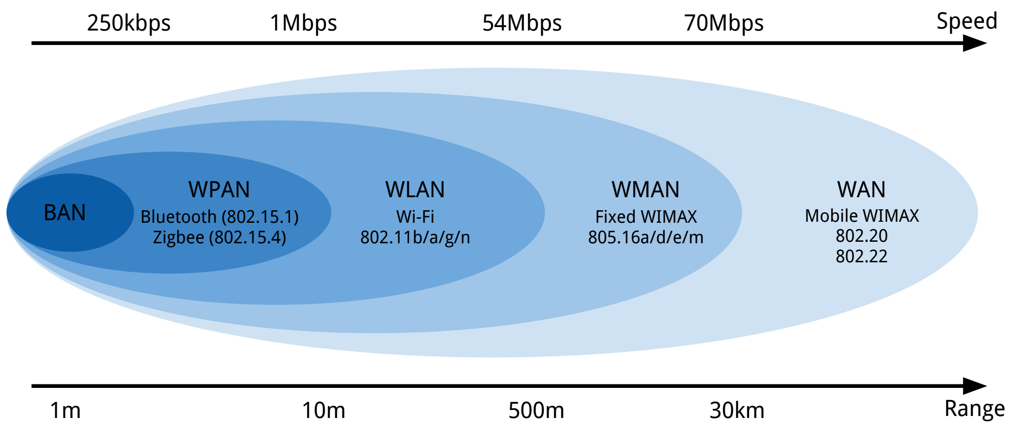

Figure 1.2. *Classification of the wireless networks based on the covered area and the transmission speed.*

Embedded System field, the first two kinds of networks are the most significative. A (Wireless) PAN has a quite small range and it is used to interconnect devices around an individual's person workspace. The most famous standards for these networks are Bluetooth and Infrared Data Association (IrDA). A BAN, also called wearable network, covers an extremely small range, around as big as a human body volume and it is extensively used for health monitoring systems. These kind of networks allow, for instance, to attach several sensors, detecting different medical parameters, to the human body and periodically communicate them to a base station in charge of their elaboration to provide a complete overview of the patient conditions to the personnel.

For what concerns the network topologies, there exist several models, the most important ones are shown in figure 1.3.
The first three topologies, *ring*, *mesh* and *bus* don't strictly need a node that coordinates all the others to allow the communications. All the nodes can, in a distributed way, exchange messages with the others, possibly following alternative paths and increasing the robustness of the network. These kind of networks are scalable and cost-effective, even if the bus topology has a limit on the maximum number of nodes that can be attached, after which the shared medium becomes too crowded and the efficiency starts to drop.

Then we have the *tree* and the *star* topologies, characterized by an hierarchical organization. The *tree* topology has a root node that acts as the main coordinator and all the other nodes. Actually the *star* topology can be seen as a tree with only two hierarchical levels. In these kinds of network, the communications are managed by one or more nodes called *coordinator*s,

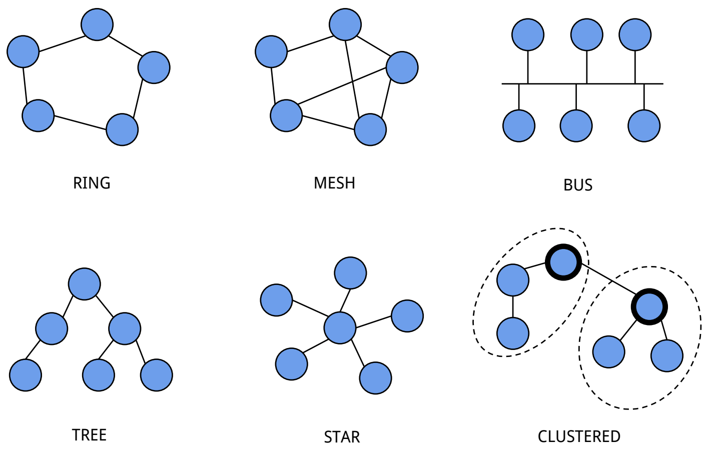

Figure 1.3. *Networks classification based on the interconnection structure.*

in charge of synchronizing the transmission to and from all the other nodes.

There exist also hybrid approaches like the clustered networks, in which, nodes are splitted and grouped together forming clusters. Inside each cluster the nodes are connected together with a mesh or a star for instance, and then the clusters are connected each other through a hierarchical structure. Every cluster has an elected coordinator node, representing the access point to the subnetwork, that is connected with the coordinators of the other clusters to exchange inter-cluster traffic. This kind of approach drastically reduces the number of connections required with respect to a mesh topology without worsening too much the latency between the nodes and allowing a better organization of the network.

1.4 Embedded Real-Time Systems

An Embedded Real-Time system is a special-purpose computing system dedicated to handle a particular task within a larger system for which, the functional correctness doesn't depend only on the validity of the results but also on the time taken to produce those results [2]. Since the embedded systems are dedicated to specific tasks, completely known in the design phase, the hardware can be tailored to satisfy the required needs reducing the size and the costs to the minimum, enforcing both reliability and performance. The platforms upon which an embedded system may be build are various, depending on the required computing capabilities, complexity and power consumption. Microcontrollers range from the most basic components like the Programmable Logic Controller (PLC), to more complex ones like ARM, PIC and Atmel architectures. They can be equipped with internal permanent memory storage (EEPROM memory), oscillators, timers, Analog-to-Digital and Digital-to-Analog converters, Pulse-Width Modulation (PWM) generators, built-in USB and Ethernet interfaces. Nowadays embedded systems are everywhere: telecommunication, automotive industry, medical equipment, consumer electronics, household appliances, transportation and military are just a few examples of fields in which these systems are deeply rooted. Because of that, these systems have to face events that happens in the real world and they have to handle them with an appropriate reaction time. Therefore, quite often, embedded systems are real-time systems as well.

A real-time system, as the name suggests, is a system that has to provide not only correct results, but also within a given time constraint and the speed at which the system time flows has to be the same as the speed of the time in the real world. In this way a real-time system can promptly react to events happening in the physical world. On these kind of systems, a particular class of operating systems are used, the Real-Time Operating Systems (RTOS). A general overview of a RTOS is shown in figure 1.4. A RTOS doesn't

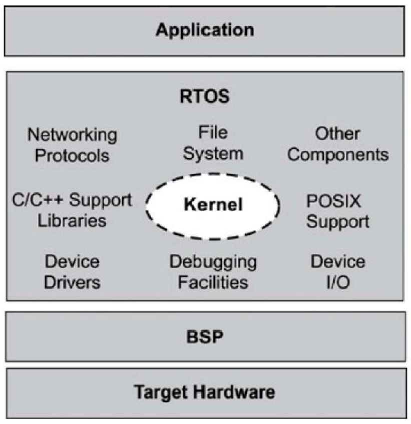

Figure 1.4. *Overview of a Real-Time Operating System architecture.*

necessarily be fast, but it should guarantee that a task execution ends within a given predetermined time, called *deadline*. Indeed, the most important aspect for a RTOS is the predictability. To guarantee in advance whether a task will end within its deadline or not, a feasibility test has to be performed. The feasibility test tells if a set of tasks can be scheduled in such a way that all of them will respect their own deadlines.

There are basically five kinds of tasks that can be handled by a RTOS (listed basing on the increasing criticality): non real-time, soft real-time, firm real-time, on-time and hard real-time [3]. To evaluate the performance of a scheduling algorithm on a task set, an utility function is defined for all the five classes of tasks (fig. 1.5). Each utility function reflects the value that a

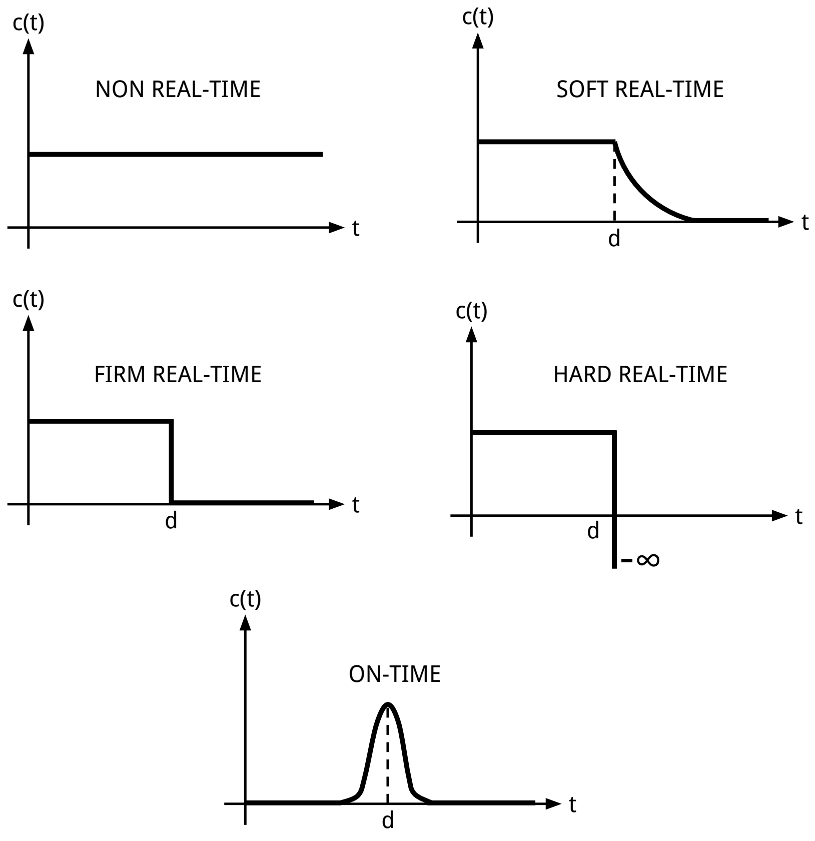

Figure 1.5. *Utility function for all the five different kinds of task.*

task has depending on its completion time:

- **Non real-time tasks**: they don't have any time constraints, served with a best-effort policy, their value is proportional to their importance and independent of the completion time.
- **Soft real-time tasks**: they have a deadline, and their value is constant till the deadline, after which it starts to decrease. It means that they can tolerate a deadline miss, up to a given amount of time.
- **Firm real-time tasks**: they have a deadline, after which the execution of the tasks becomes useless but not harmful and they can actually be discarded.

- **On-Time real-time tasks**: they have a deadline and their value is high only if the task is completed around the deadline, not too early and not too late.
- **Hard real-time tasks**: they have a rigid deadline to abide by. Missing the deadline might cause unpredictable catastrophic consequences on the whole system, in fact invalidating it.

Very often, real-time systems need to exchange data among them. During data communications between these systems, three kinds of error could occur:

- **Information error**: when the system is not able to interpret the data or the data is interpreted in a wrong way
- **Timing error**: when the system is not able to provide data before its expiration
- **Transmission error**: when the data is corrupted along the way through the communication channel

The last kind of error is most likely to occur with wireless communications since the medium is shared and noise as well as interferences could impair the signal. Therefore, two Quality of Service (QoS) parameters are defined to control the network behaviour: the *Deadline for delivery* (DL) that states the latest time at which the receiver must receive the information being sent and the *Probability of correct Delivery within Deadline* (PDL), strictly correlated to the channel quality, that controls how reliable the transfer must be [4].

In the non real-time applications the correctness of the transmitted data is more important than its timing, indeed algorithms such as error detection and correction or packets retransmission are used to ensure the data integrity despite the potential delays they could introduce.

2

Communication Protocols for Sensor Networks

While in the general-purpose systems, all the seven layers of the Open Systems Interconnection (OSI) model are usually implemented, in the real-time systems field this becomes unfeasible due to the inevitable and unacceptable delay that all these layers would introduce. The main goal is to reduce at most the number of layers and the processing time required for each of them in order to have fast communication stack percolations. Therefore, for real-time sensor networks, only three layers have been kept: *Application*, *Data Link* and *Physical*. In case additional layers are required by an application, they could be partially added and integrated with the ones already present.

2.1 IEEE 802.15.4 LR-WPAN

The main features of the IEEE 802.15.4 standard are network flexibility, low cost, very low power consumption, and low data rate in an adhoc self-organizing network among inexpensive fixed, portable and moving devices.

It is developed for applications with relaxed throughput requirements which cannot handle the power consumption of heavy protocol stacks [5].

The two basic kinds of devices defined in this standard are the *Full-Function Device* (FFD) and the *Reduced-Function Device* (RFD). The first kind, is a device that has the full functionalities at the MAC layer, while the second has a reduced set of functionalities. This allow an RFD to save power thanks to the less computational capabilities required. *ZigBee*, a protocol for wireless low-power communications based on the IEEE 802.15.4, defines the same three kinds of devices as: *ZigBee Coordinator* (ZC) in place of PAN Coordinator, *ZigBee Router* (ZR) in place of Full-Function Device and *ZigBee End Device* (ZED) in place of Reduced-Function Device.

An LR-WPAN has to contain at least one Full-Function Device operating as the PAN coordinator, that is a node in charge of managing the WPAN network. These devices can be combined together forming different network topologies, as the ones shown in figure 2.1.

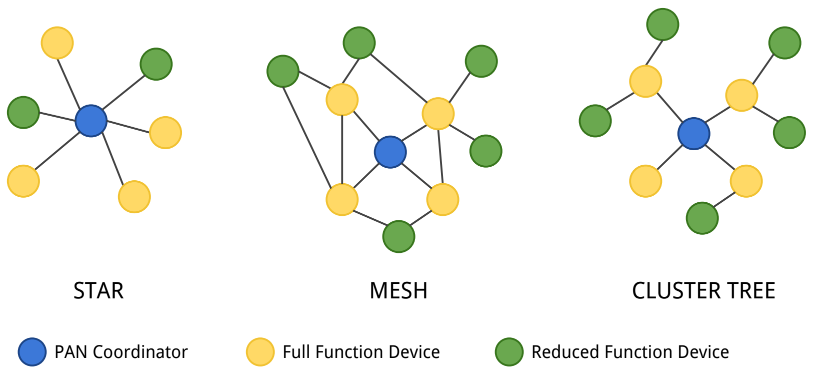

Figure 2.1. *IEEE 802.15.4 Network topologies.*

The *star* topology relies on a central PAN coordinator device, to which the others node can establish connections. The PAN Coordinator is in charge of accepting the other devices join or leave requests and handling as well as routing the communication between the devices. The PAN coordinator may be mains powered while the devices will most likely be battery powered. Every FFD, when turned on for the first time, may create its own network, choosing a PAN identifier not used by other coordinators in its range.

The *mesh* or *peer-to-peer* topology, requires just one PAN Coordinator as in the previous case, but the other nodes are free to communicate among them, provided that they are within each other range. This kind of network is much more robust than the previous one since the devices can use distributed algorithms to self-organize themselves and self-heal the network in case of failures. Moreover, messages sent by nodes, can benefit of multipath routing, increasing the delivery reliability.

At last, the *cluster-tree* topology adds the concept of *clusters* to the network. The first FFD device that becomes a PAN Coordinator, also creates a cluster, becoming the *Cluster Head* (CLH) of it. The subsequent devices, may ask to join that cluster and, if the coordinator allows them, they become its childs. The newly joined devices start to send beacons that could be received by other interested devices. After some time, the PAN Coordinator may instruct a device to become the *Cluster head* of another cluster, connected to the first one. In this way it is possible to increase the reach of the network, with the unavoidable drawback of increasing the message latency.

The structure of a LR-WPAN device is composed of a PHY layer, which contains the radio frequency (RF) transceiver along with its low-level control mechanism, and a MAC sublayer that provides access to the physical channel for all types of transfer. Above them, there are the layers that provide the desired function of the device.

2.1 Physical layer

The PHY layer offers three data transmission rates based on the frequency band used. Low frequencies allow longer reach with lower propagation losses at a cost of reduced transmission speed, while high frequency is used for faster low-ranged communications. The details about the available frequencies and the correspondent modulation formats are shown in table 2.1.

Frequency band	Chip rate	Modulation	Bit rate	Symbol rate	Symbols	Channels
[Mhz]	[kchips/s]		[kbit/s]	[ksymbols/s]		
868-868.6	300	BPSK	20	20	Binary	1
902-928	600	BPSK	40	40	Binary	10
2400-2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal	16

Table 2.1. *IEEE 802.15.4 PHY layer frequency bands and transmission speeds.*

The receiver sensitivities of -85dBm at 2.4Ghz and -92dBm at 868/915Mhz, together with the transmit power, determine the maximum achievable range. The PHY, in addition to generating and modulating the carrier and (de)-coding data, performs also three evaluations during its functioning: Receiver Energy Detection (ED), Link Quality Indication (LQI) and Clear Channel Assessment (CCA). These parameters are used to monitor the quality of the communications and to take actions whenever it drops under a fixed threshold.

2.1 Medium Access Control

IEEE 802.15.4 provides two ways to access the medium, one called *Beacon Enabled* and the other one *Non Beacon Enabled*. The *Beacon Enabled* mode uses a periodic packet called *superframe* that allow the nodes to synchronize with the coordinator. In this way the coordinator can schedule the transmission periods of all the nodes in its network. Instead, in the *Non Beacon Enabled* mode, the access to the channel is completely unregulated and therefore totally asynchronous. Independently on which of the two modes is being used, all the nodes have to register to the coordinator, with a procedure called *Association*. It is the job of the coordinator to keep a list of all the associated devices, to send the periodic beacon (if required), and to exchange packets with the neighboring nodes.

In the *Beacon Enabled* mode, the communication window contained between the transmission of two consecutive beacons, can be divided into 3 parts (fig. 2.2): a *Contention Access Period* (CAP) during which the channel is accesses with the CSMA/CA technique and used to exchange control information, a *Contention Free Period* (CFP) in which each node transmits only during the time slot it has been assigned to by the coordinator (called *Guaranteed Time Slot* or GTS) and, at last, an *Inactive* period used to save power, for instance turning off the radio transceiver [7].

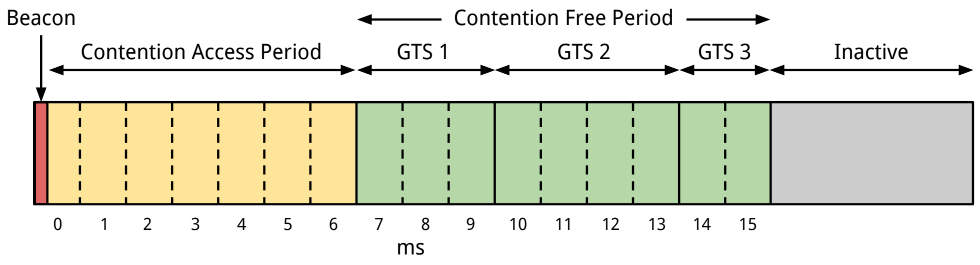

Figure 2.2. *IEEE 802.15.4 Superframe structure.*

During the Contention Access Period, the devices use a CSMA/CA protocol, while during the Contention Free Period they access the channel with a Time-Division strategy. Each device, during its GTS, should guarantee that its transmission stops before the beginning of another device GTS. A coordinator can allocate up to 7 GTS for each *superframe* and each GTS can occupy more than one time slot. In any case, the *superframe* has to contain a minimum CAP. Another solution uses the whole *superframe* as a CAP, becoming in fact a CSMA protocol.

2.2 Medium Access Control

The Medium Access Control (MAC) layer of a protocol, manages how a device accesses the transmission medium to send its data. Two big families of MAC protocol can be distinguished:

- **Controlled Access**: this is a collision-free class of protocols since each node knows when it is allowed to transmit, therefore avoiding collisions with other transmitting nodes. The protocols like Master-Slave, Token-Passing and TDMA belong to this family.
- **Uncontrolled Access**: this is a class of protocols in which transmissions take place as soon as the node has some data to transmit. This imply no cost at all to organize the devices, but with the probability that a transmission could overlap with someone else. In case, the devices should detect the collision and provide mechanisms to retransmit the packets. Protocols like CSMA, CSMA/CA, CSMA/CD belong to this family.

2.2 Master-Slave

Master-Slave protocols define a single node acting as a coordinator, called *master*, while all the other nodes are called *slave*. All the *slaves*, prior to transmit, have to wait to be authorized by the *master*. The *master* could apply any scheduling algorithm without the need of synchronizing the nodes, but with two drawbacks: the need of sending a control message for every *slave* activation and the existence of a single point of failure (fig. 2.3).

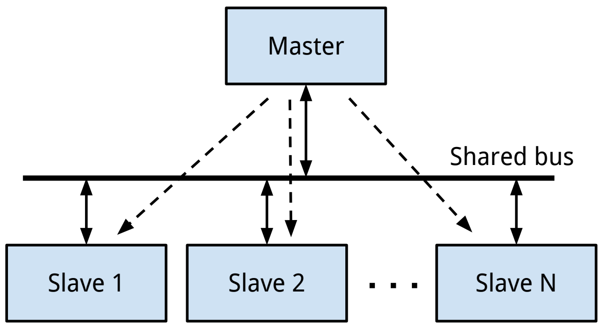

Figure 2.3. *Master-Slave protocol functioning in a bus topology.*

2.2 Token Passing

In the Token Passing protocols, the access to the medium is allowed only when a node has a special packet called *token*. This packet is exchanged among the nodes with an order dependent on the utilized scheduling algorithm. The most common scheduling strategy used is called *Round Robin*. With this scheduling, the devices are enabled to transmit in a circular way (fig. 2.4). The advantage is that collisions are eliminated, and that the channel bandwidth can be fully utilized without idle time when demand is heavy. The disadvantage is that even when demand is light, a station having data to transmit must wait for the token, increasing latency.

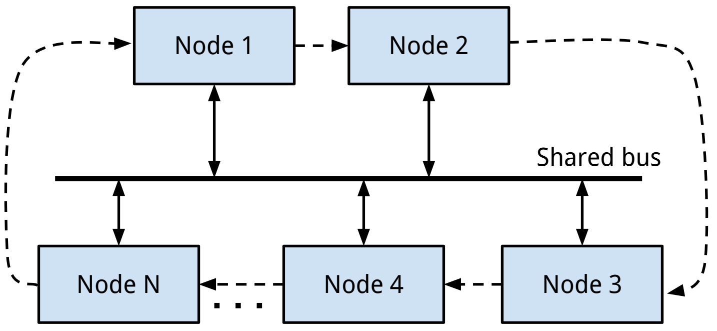

Figure 2.4. *Token Passing protocol with a Round Robin scheduling in a bus topology.*

Few parameters are introduced when Token Passing protocols are used:

- **Target Token Rotation Time (TTRT)**: is the time needed to perform a round trip along the ring
- **Real Token Rotation Time (RTRT)**: is the time effectively taken by the *token* to percolate the ring the last time
- **Token Holding Time (THT)**: is the time span during which the node owns the *token*

This protocol should also provide mechanisms to recover the *token* in case it is lost, otherwise the whole network would result as blocked. When the Token Passing protocols are used in real-time systems, it is necessary to bound the TTRT to guarantee that every node has a transmission window suitable with their transmission deadlines [8].

2.2 TDMA

With Time Division Multiple Access (TDMA), the data stream is divided into frames and, in turn, the frames are divided into time slots. Each devices is assigned with one or more of these time slots in such a way that the communication does not overlap. Each time slot is separated from the next by a guard interval, necessary to keep into account small synchronization error between the devices (fig. 2.5). TDMA allows multiple devices to share the same transmission medium, like a frequency channel for instance, guaranteeing collision-free communications. The drawback is that a device has a fixed available bandwidth, even in case it has a lot of data to send and the other devices are in idle. For this reason, a Dynamic TDMA has been designed providing on demand assignment of the time slots depending on the device traffic load.

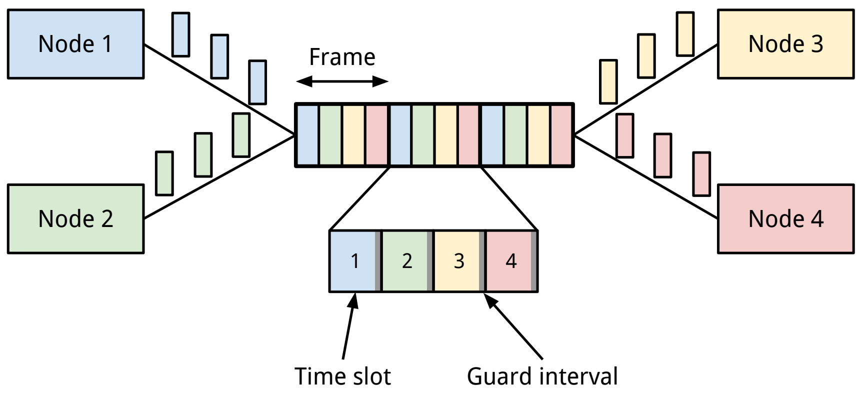

Figure 2.5. *TDMA protocol in case of 4 nodes sharing the same channel.*

2.2 FDMA

The Frequency Division Multiple Access (FDMA) is similar to the TDMA approach, but instead of dividing the time, it partitions the frequencies. This scheme is immune to the timing synchronizations required for the TDMA but, unless a device is equipped with a full-duplex transceiver, it cannot receive on a frequency and simultaneously transmit on another. A disadvantage that this kind of protocol has is the crosstalk between adjacent frequency that could impair the communication. In wireless sensor networks, in case of cluster-tree topology, FDMA can be used to assign each cluster a frequency in such a way that local cluster communications can take place at the same time, while for inter-cluster communications the coordinators periodically switch frequency to exchange traffic with the neighboring coordinator devices. FDMA and TDMA can also be combined together to obtain a full partitions of both frequencies and time, increasing the number of available slots as well as the number of supported devices.

2.2 CSMA

The Carrier Sense Multiple Access (CSMA) is a probabilistic medium access control in which a device is able to detect if there is another communication ongoing on the same transmission medium. This is achieved by detecting both the carrier wave and the presence of an encoded signal of some other devices. There exist two versions of this protocol, one is the CSMA with Collision Detection (CSMA/CD), while the other is CSMA with Collision Avoidance (CSMA/CA).

CSMA/CD sends the data as soon as it is ready, detecting later if a collision happened and, in case, stopping immediately the current transmission. In this way the channel is freed just whenever a collision is detected, shortening the time required before another attempt to send. In wireless networks, the Collision Detection approach is not feasible due mainly to the hidden node problem. The other approach, CSMA/CA, is more cautious than the first one in that, before transmitting, it tries to detect an ongoing transmission. In case a node is already transmitting, it simply delays the communication by a random time interval called *backoff time*, decreasing the probability of collision.

CSMA/CA protocols can also be further subdivided into *Non Priority CSMA/CA* and *Priority CSMA/CA*. As the names suggest, the first version treats all the packets in the same way, while the second one differentiates them using different rules to distinguish different classes of packets. In real-time networks, of course the priority could be based on the deadline that a packet or a stream of packets have. The policies adopted to distinguish among different priorities are based on:

- ***inter-frame time* differentiation**: higher priority packets have a lower *inter-frame time*, that is the time a packet has to wait, and during which the channel has to remain free, before starting to transmit.
- **outgoing queue differentiation**: each node has as many outgoing queues as the number of priority classes. Each packet ready to be sent is put in the correspondent queue and waits to be transmitted. The node serves the queues starting from the one with the highest priority and going down till it finds one that is not empty and then sends the first packet of that queue.
- ***backoff* growth factor**: this strategy assigns a *backoff* time increment

inversely proportional to the priority. This allow higher priority packets to wait less in case the channel is already busy when the node transmits.

2.2 Real-time system protocols

MAC protocols for wireless real-time networks have been mainly designed to guarantee an high throughput keeping into account both the bandwidth required and the power consumption, but generally not considering the real-time requirements or the jitter control. Those protocols that keep into account also these factors can be classified following two different systems. The first classification, presented in [9], divides the protocols in *synchronous scheme* and *asynchronous scheme*. The synchronous protocols are those that require some sort of synchronization among the nodes of the network such as *Cluster TDMA* [11], *Cluster Token*, *Implicit Earliest Deadline First (I-EDF)* [12], *Robust Implicit EDF (RI-EDF)* [14], *TDMA-Based MAC (TB-MAC)* [13] and WBuST [18].

The protocols using a *synchronous* scheme generally implements a Time Division Multiplex approach subdividing the time into slots assigned to each nodes. This implies the presence of a coordinator node to synchronize all the other devices in the network and the possibility of wasting bandwidth in case a node doesn't use the whole reserved time slot due to less traffic to transmit or an hardware/software failure.

The *asynchronous* scheme protocols, usually based on Black-Burst (a broadcast protocol that addresses hidden node and reliability problems in multi-hop vehicular networks) [15] and added with a real-time functionalities, require a fully connected networks and are not easy to extend to multi-hop environments.

Another classification proposed in [10] divides the protocols into three categories: *scheduling based*, *contention based* and *scheduling/contention based*. The first class access the medium in a TDMA manner, since each node exactly knows when it is allowed to transmit. In case a node doesn't have data to transmit or it is less than expected, it could give the amount of time saved to the next node waiting to transmit. The *contention based* approach, instead, is based on the CSMA with Collision Avoidance scheme. They are less performant than the *scheduling* protocols since the probability of a collision can be reduced but not eliminated. The last category is a mix between the first two. WBuST can be classified as a *scheduling/contention based* protocol since its time frame is partitioned in a contention period used to exchange control information and a scheduling period used to transmit the node traffic.

3 WBuST Protocol

The *Wireless Budget Sharing Token* (WBuST) protocol is a MAC layer protocol designed for real-time wireless networks of embedded devices [18]. WBuST can handle real-time and best-effort traffic while applying power saving strategies to guarantee the maximum device lifetimes. The are four main sources of energy waste that can be identified at the MAC level:

- **Collision:** every time a packet experience a collision and became corrupted, it has to be sent again, wasting power
- **Overhearing:** energy consumption due to listening to the channel and receiving packets addressed to other nodes
- **Control overhead:** energy consumed by exchanging control packets
- **Idle listening:** energy consumed in receiving mode while waiting for packets

Contention-based protocols like CSMA/CA are mainly subject to *collisions* and *idle listening* issues, while scheduling-based protocols are mainly affected by *control overhead*. WBuST is an hybrid approach that mitigates both problems, providing an efficient power saving mechanism while still guaranteeing the real-time stream packet deadlines. WBuST has been designed to operate in both single-hop and multi-hop networks. In this chapter the main concepts of the basic single-hop version are introduced, while in the next chapter the extension for multi-hop support is presented.

3.1 Network model

The WBuST protocol is an implementation of the BuST protocol [19] for wireless environments. It belongs to the category of *token-passing* protocols and it has been introduced to improve the performance of the *timed-token* protocols like FDDI and FDDI-M in cabled networks [20]. In WBuST, the time slot in which a node is allowed to transmit is shared between the synchronous traffic (*real-time*) and the asynchronous traffic (*non real-time*).

A WBuST network is composed of a set of nodes. In every network there is one *coordinator* node and one or more *normal* nodes (fig. 3.1). The coordinator is in charge of scheduling the streams transmission, managing the network and synchronizing the nodes together sending periodic *beacon* messages.

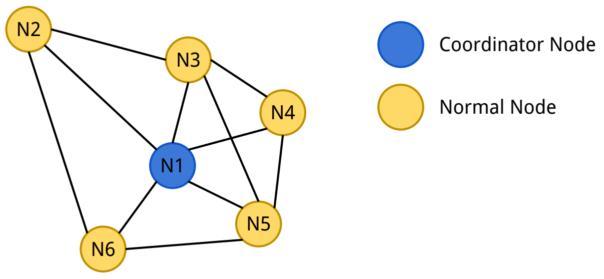

Figure 3.1. *WBuST single-hop network topology.*

Each node i has one or more synchronous message streams S_i associated to it which are described by three parameters (fig. 3.2):

- C_i, the maximum amount of time required to transmit one message of the stream
- T_i, the interarrival time between two consecutive stream messages
- D_i, the relative deadline associated to the stream S_i, that is, the maximum amount of time that can elapse between a message arrival and the completion of its transmission

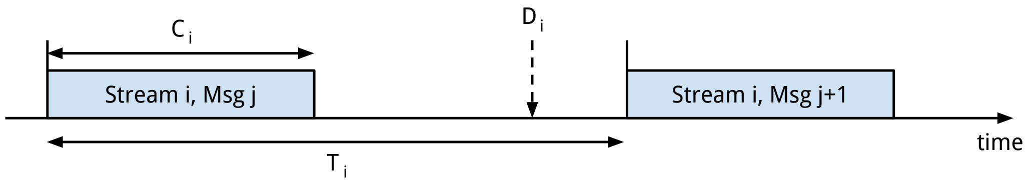

Figure 3.2. *Parameters describing a stream.*

At this point we can define the concept of channel utilization of a stream S_i, that is

$$U_i = \frac{C_i}{min(T_i, D_i)}$$

while the total channel utilization is given by the sum of all the stream utilizations

$$U = \sum_{i=1}^{n} U_i$$

where n is the number of streams in the network.
Let's define now few more parameters of the protocol:

- τ is the time needed to transmit the token between nodes, included the overhead introduced by the protocol
- T_b is the beacon period which defines the dimension of each Communication Window
- T_{BT} is the greatest value of T_b that guarantees the correct operation of WBuST

The parameter T_{BT} has the same meaning as the Target Token Rotation Time ($TTRT$) for the timed token protocols. To guarantee the correct functioning of the protocol, T_{BT} has to be not greater than the minimum relative deadline $D_{min} = min_i(D_i)$. This is a necessary and sufficient condition to guarantee at least one packet transmission for each node i, between the time t_i^r a new message in S_i is produced for transmission and its absolute deadline $d_i = t_i^r + D_i$. An example showing the maximum delay a message may experience, is shown in figure 3.3.
The maximum delay between t_3^r a new message is ready in the stream S_3 and the end of the budget B_3 in the next CW. The delay is equal to $T_b \leq T_{BT}$ and has to be no greater than D_3. Given $D_{min} = D_3$ and $T_b = T_{BT}$, the condition that guarantees at least one packet transmission becomes: $t_3^r + T_b \leq t_3^r + D_3$.
A message experiences the worst-case transmission delay when it is produced just after the end of the budget assigned to its node. For any choice of the

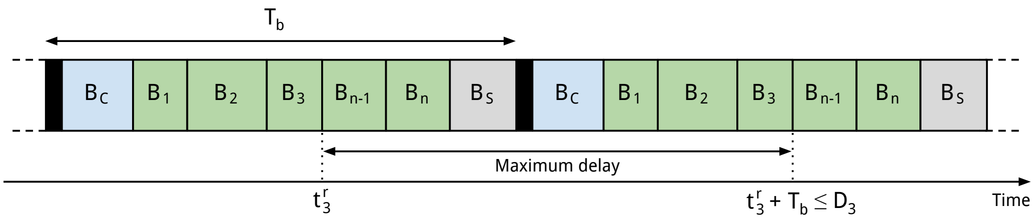

Figure 3.3. *Maximum delay a message may experience.*

protocol parameters, two constraints have always to be satisfied in order to allow a correct communication among the nodes:

- **Protocol constraint:** the total bandwidth allocated to the nodes must be less than the available network bandwidth, formally written as
$$\sum_{i=1}^{n} \frac{B_i}{T_{BT}} \leq 1 - \frac{\tau}{T_{BT}}$$
- **Deadline constraint:** if $s_{i,j}$ is the time at which the transmission of the j-th message in stream S_i is completed, the deadline constraint requires that for $i = 1, ..., n$ and $j = 1, 2, ...$
$$s_{i,j} \leq t_{i,j} + D_i$$
where $t_{i,j}$ is the message arrival time and D_i is its relative deadline.

It's worth to notice that in the previous two formulas, while $t_{i,j}$ and D_i are defined by the application, $s_{i,j}$ depends on the synchronous bandwidth allocation and on the T_{BT} value.

3.1 Communication Window Structure

WBuST assigns, to each stream S_i, a budget B_i for real-time traffic transmission. From now on we will consider only nodes with one stream, since if a node has more than one stream, we can put together the budgets and consider them as a one big stream. Whenever a node receives the token, it is allowed to transmit up to an amount of time equal to B_i. The asynchronous traffic can be sent by a node, every time it holds the token and if there is no real-time traffic to transmit.

The T_{BT} is chosen as the smaller D_i among all the streams in the networks. After the T_{BT} is set, the coordinator computes, for each stream, the relative budget B_i and the starting transmission time I_i. The time is divided into

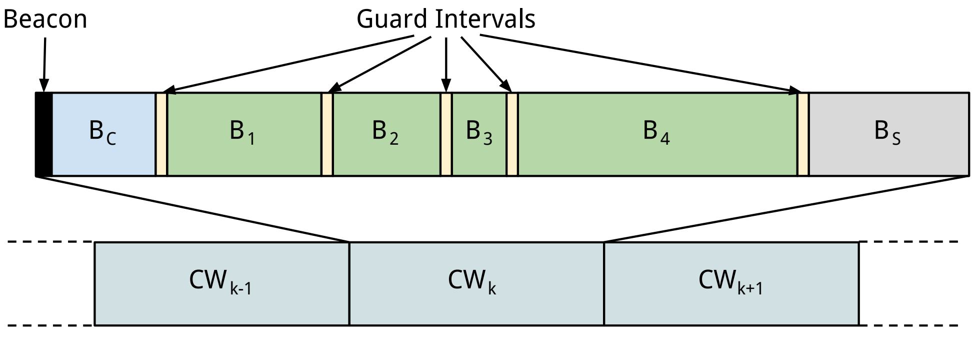

Figure 3.4. *WBuST Communication window structure.*

periodic intervals called Communication Window (CW), whose structure is illustrated in figure 3.4.
Each CW starts with a special packet sent by the coordinator node called *beacon*. The beacon is used to communicate the CW length, synchronize the node and communicate the CW schedule. Each CW is divided into several kind of slots:

- B_C is the contention slot and it immediately follows the beacon. It is accessed with the CMSA/CA scheme and it is used to exchange control information with the coordinator like joining or leaving the network, reserving slots, etc.
- B_i is the contention-free slot reserved for the node i. In this period, node i can transmit its synchronous/asynchronous messages.
- B_S is the slot collecting all the unused time in the CW and exploited to put in sleep mode all the nodes to save energy.

Each node has a timer used to count the elapsed time since the *beacon* reception with a resolution of 1ms. Each time slot is separated from the next one by a *guard interval*, necessary to tolerate the small but unavoidable node synchronization misalignments due to both the limited timer resolution and the not perfectly constant transmission delays.

The *contention period* can also be removed if the network topology is static and no node joins/departures are expected.

3.1 Protocol properties

The properties that the WBuST protocol exhibits, holds when the constraints are satisfied. There are two properties that bound the maximum delay for a message, called WC_i, in case of real-time streams only and real-time streams together with asynchronous traffic.

Maximum delay with real-time streams only. Under the WBuST protocol, if $T_i \geq T_{BT}$ and the network traffic is only generated by real-time streams, for $i = 1, ..., n$:

$$WC_i = \left\lceil \frac{C_i}{B_i} \right\rceil (T_{BT} - B_i) + C_i$$

Maximum delay with both real-time and asynchronous streams. Under the WBuST protocol, if $T_i \geq T_{BT}$ and the network traffic is generated by real-time and best-effort streams, for $i = 1, ..., n$:

$$WC_i = \left\lceil \frac{C_i}{B_i} \right\rceil T_{BT}$$

Since WBuST uses the same channel access strategy of the BuST protocol, the proof of these two properties can be found in [19].

3.2 Budget Allocation Schemes

In order to assign to each stream a budget for transmission, several allocation schemes can be used. The schemes for the *timed-token* protocols, as in [21] are classified as *global* or *local*, depending on the quantity of information they need to provide the scheduling. Local information can be, for instance, the stream set assigned to a node, while a global information is the number of nodes in the network.

Another classification proposed in [24], divides the allocation schemes among the *T_{BT}-partitioning* and the *C_i-partitioning* ones. The class of the *T_{BT}-partitioning* schemes, computes the stream budget as a fraction of the maximum value of the beacon period (T_{BT}). The allocation schemes belonging to this category are the *Proportional Allocation* (PA) and the *Normalized Proportional Allocation* (NPA) and are shown in table 3.1. The value α represents the bandwidth wasted due to the protocol overhead and is calculated as

$$\alpha = \frac{\tau}{T_{BT}}$$

while

$$\beta_{min} = \min_i \frac{T_i}{T_{BT}}$$

The *C_i-partitioning* schemes, instead, assign the stream budget as a fraction of the maximum time C_i needed to send a message. These schemes are

Scheme	Allocation rule	U*	Schedulability test
PA	$B_i = U_i(T_{BT} - \tau)$	$\frac{1-3\alpha}{2(1-\alpha)}$	$U \leq \frac{\beta_{min}}{(1-\alpha)\left\lceil \frac{\beta_{min}}{1-\alpha} \right\rceil} - \frac{\alpha}{1-\alpha}$
NPA	$B_i = \frac{U_i}{U}(T_{BT} - \tau)$	$\frac{\lfloor \beta_{min} \rfloor}{\lfloor \beta_{min} + 1 \rfloor}(1-\alpha)$	$T_i \geq T_{BT}\left(\left\lceil \frac{1}{1-\frac{U}{1-\alpha}} \right\rceil - 1\right)$

Table 3.1. *T_{BT}-partitioning schemes.*

Scheme	Allocation rule	U*	Schedulability test
LA	$B_i = \frac{C_i}{\left\lfloor \frac{T_i}{T_{BT}} - 1 \right\rfloor}$	$\frac{\lfloor \beta_{min} \rfloor}{\lfloor \beta_{min} + 1 \rfloor}(1-\alpha)$	$U \leq \frac{\lfloor \beta_{min} \rfloor}{\lfloor \beta_{min} + 1 \rfloor}(1-\alpha)$
MLA	$B_i = \frac{C_i}{\left\lfloor \frac{T_i}{T_{BT}} \right\rfloor}$	$\frac{\lfloor \beta_{min} \rfloor}{\lfloor \beta_{min} + 1 \rfloor}(1-\alpha)$	$U \leq \frac{\lfloor \beta_{min} \rfloor}{\lfloor \beta_{min} + 1 \rfloor}(1-\alpha)$

Table 3.2. *C_i-partitioning schemes.*

the *Local Allocation* (LA) and the *Modified Local Allocation* (MLA) and are shown in table 3.2.
The parameter U^* represents the *Worst Case Achievable Utilization (WCAU)*, that is the maximum network utilization for which every real-time message, if $U \leq U^*$, is guaranteed to be sent within the deadline [22, 23]. The value of U^*, calculated for BuST in [25], is still valid also for WBuST since it employs the same scheduling policies [26, 27]. From now on and for all the streams in the network, we will consider $D_i = T_i$. This does not affect the results since $U_i = \frac{C_i}{min(T_i, D_i)}$ and therefore the results are valid also the case $D_i \leq T_i$, simply by replacing T_i with D_i. The WCAU is widely utilized to guarantee the schedulability of a stream set when only an estimate of the real-time traffic is available, without knowing the characteristics of every real-time message.

It is possible to provide also the maximum transmission time for real-time messages with WBuST protocol. Indeed, for all the budget allocation schemes seen till now, if $\forall i = 1, ..., n : T_i \geq T_{BT}$, it holds:

$$\forall i, j : s_{i,j} \leq t_{i,j} + \left\lceil \frac{C_i}{B_i} \right\rceil \left(\sum_{r=1}^{n} B_r + \tau \right)$$

3.3 Bandwidth reclaiming

In WBuST, whenever a node has less traffic to transmit than expected and doesn't use its reserved slot completely, the left budget can be recycled to increase the transmission time of other nodes. Therefore, when a node saves some budget, the unused budget is added to the budget of the next node. The unused budget collected after the last node transmission is added to the sleep budget. In this way the sum of all the node budgets plus the sleep budget is kept constant and equal to T_b. This means that the saved budget cannot be used across different CWs, since it is depleted in the same CW in which it has been accumulated. This power saving scheme is called *Remainder Sleep Time* and allow a dynamic sleep slot adaptation, depending on the network utilization. Several ways can be used to communicate to a node that the previous budget finished before its expiration. WBuST uses a special message that is sent by node that has no more traffic to transmit to the node owning the next stream allowed to transmit. The following formula, which proof is shown in [18], gives the worst case transmission time for any message of a stream S_i, when the budget recycling mechanism is employed. For $i = 1, ..., n$, if $T_i \geq T_{BT} + \sum_{j=1}^{i} B_j$, then:

$$WC_i = \left\lceil \frac{C_i}{B_i} \right\rceil (T_{BT} - B_i) + C_i + \sum_{j=1}^{i} B_j$$

The situation causing the maximum transmission delay is shown in figure 3.5.

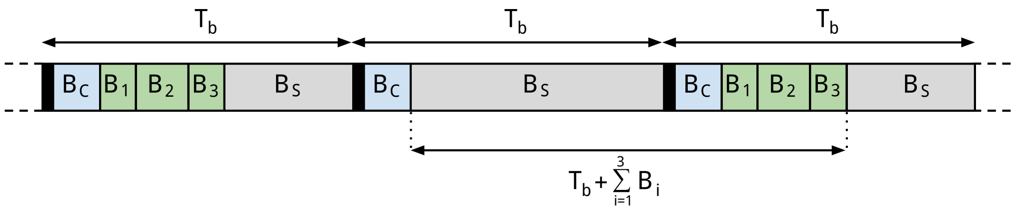

Figure 3.5. *Example of bandwidth reclaiming mechanism.*

Indeed, if at the end of B_C all the nodes have no traffic to transmit and if a massage becomes ready in every node, as soon as the sleep budget starts, the node 3 will experience the maximum delay. This because it has to wait its turn that actually is the last one. It is important to notice that the position in the ordering affects the maximum transmission delay between two consecutive channel accesses. This consideration can be used to order the node/stream slots from the one with a shorter deadline to the one with the longest deadline.

3.3 Non real-time traffic

One of the characteristics of the WBuST protocol is the ability to send both real-time and best-effort traffic. Up to now only the real-time traffic guarantees have been studied, hence the minimum bandwidth for best-effort traffic will be analized now. To study the worst-case service for non real-time traffic we suppose that every node has always best-effort messages to send, therefore having the full channel utilization. The minimum bandwidth that a node can reserve for this kind of traffic depends on the budget allocation scheme used. In [19] is shown that for a node i, the minimum bandwidth U_i^{BE} that can be guaranteed for best-effort traffic is:

- with PA scheme $$U_i^{BE} = U_i \left(\frac{1}{U + \frac{\alpha}{1-\alpha}} - 1 \right)$$
- with NPA scheme $$U_i^{BE} = U_i \left(\frac{1-\alpha}{U} - 1 \right)$$

The value U_i^{BE} can increase in case the nodes have no real-traffic to transmit. Even if each stream has a guaranteed bandwidth for sending best-effort traffic, it is clear that the allocation is not fair. Indeed, the time reserved for non real-time traffic transmission is proportional to the quantity of real-time traffic of the stream. A way to mitigate this situation is to put a limit, for each stream, on the maximum best-effort traffic that can be sent.

4 Multi-Hop Extension for WBuST

In this chapter, the multi-hop extension for WBuST is presented. This extension allows a more structured network as well as permitting communications among devices that are not able to directly communicate due to a limited operative range. The multi-hop capability increases also the available bandwidth, exploiting different channel frequencies at the same time.

4.1 Network topology

In a multi-hop network, the devices are divided into adjacent groups called *clusters*. Each cluster is assigned a different radio channel in such a way that, the transmissions within adjacent clusters can take place at the same time without interfering with each other. Clusters can be connected among them to form various network topologies allowing a great flexibility and adaptability to different scenarios.

A WBuST network is composed of n clusters, each named C_i. A node in the network can assume different roles:

- **Normal node:** a node that exchanges information with the nodes belonging to the same cluster and can send inter-cluster traffic
- **Coordinator node:** a node located in the central area of the cluster in charge of synchronizing the nodes within its area and scheduling their transmissions
- **Router node:** a node located in the central area of the cluster in charge of interacting with other router nodes

Very often, especially in the sensor networks field, the nodes performing coordination functions also perform the routing functions, therefore both functionalities are merged within the same device. In figure 4.1 an example of a clustered-tree structured WBuST network is shown.

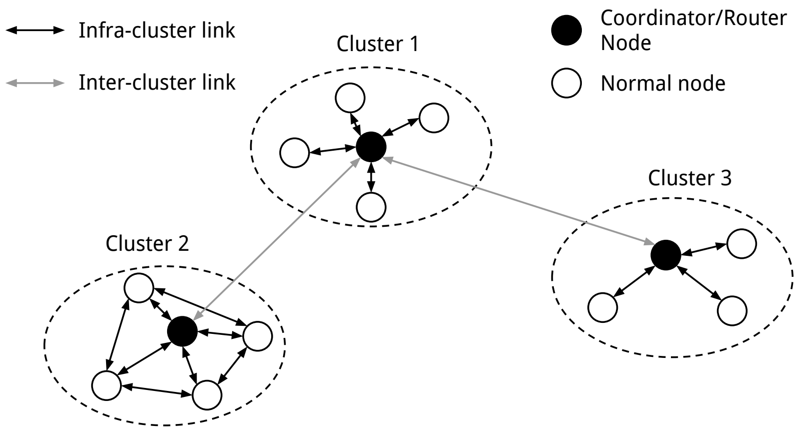

Figure 4.1. *Example of a WBuST Multi-Hop network topology.*

As we can see from the picture, each cluster can employ its own connection topology: the clusters 1 and 3 are organized in a *star* topology while the cluster 2 uses a *mesh* topology. The employed topology depends on whether the devices belonging to the same cluster are each other in their radio operational reach or not. If they are, they are able to directly communicate among them, otherwise the *coordinator/router* node has to act as a bridge between them. Of course, this requires that the *coordinator/router* node should always be directly reachable from every other node within the cluster.

4.2 Inter-cluster communication

While for the infra-cluster traffic the Communication window is the same as in the single-hop protocol, to allow inter-cluster traffic few slots have to be added at the beginning of every CW. To allow communications between two clusters, few rules have to be added. An inter-cluster communication between two clusters, as for instance those shown in figure 4.2 occurs with the procedure described in the next lines. Once the procedure is defined for the basic case, it can be extended for any kind of cluster topology. Let's identify the *coordinator/router* nodes as R_i.

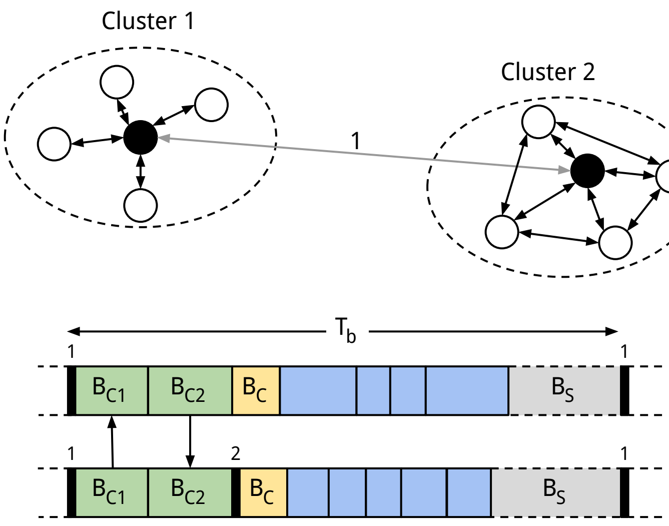

Figure 4.2. *Inter-cluster communication between a pair of cluster.*

The rules allowing a correct communication are:

- The link between the two clusters must be synchronized by a beacon transmitted by one of the two *coordinator*s, defined during the design phase to act as a master. In this example the master is R_1, which is in charge of managing the link
- Both clusters must use the same beacon period T_b
- To allow simultaneous communications within different clusters and avoiding packet collisions, each cluster must use different radio channels. Since the devices are usually equipped with low-cost radio modules, they are supposed to work with an half-duplex transceiver that can use only one frequency at a time and cannot receive and transmit simultaneously

- During the period in which the two routers exchange inter-cluster traffic, the radio channel used is the one of the master
- Each router can transmit both real-time and best-effort traffic using a budget B_{C_i} assigned at design time
- Both router budgets must be allocated in the communication windows of both clusters

The communication between the two clusters proceeds as follows:

1. At the beginning of each CW in C_1, R_2 switches to the C_1's channel and waits to receive a beacon from R_1
2. Once the beacon is received, both routers are synchronized and the communication can start. The slot B_{C_1} is used to transfer traffic from C_1 to C_2, while the slot B_{C_2} is used to transfer traffic the other way around. For the master coordinator, the budgets for inter-cluster communication are placed just after the beacon, while for the other routers they are placed in succession
3. At the end of the budget B_{C_2}, R_2 switches to C_2's channel and sends its beacon to synchronize the infra-cluster communication that starts with the contention period B_C
4. Instead, at the end of B_{C_2}, in C_1 the contention period starts immediately since it is not necessary to send another beacon

As can be noticed, the bandwidth lost due to the protocol overhead caused by beacon transmissions, is higher in the non-master cluster. This basic scenario can be extended with n clusters connected in several ways like chain, tree, ring, etc.

4.3 Clustered-tree network

This section presents the implementation of the WBuST protocol for clustered-tree multi-hop networks. This topology has been chosen since it is the most adaptable to the common wireless sensor network scenarios while, at the same time, being able to maintain the latency of the communication among the nodes under a reasonable limit even in case of big networks. In this topology, clusters are hierarchically organized as a tree. The root cluster is in charge of starting the first communication window, while the others, in cascade, will propagate the CWs from the parent to the childs clusters.

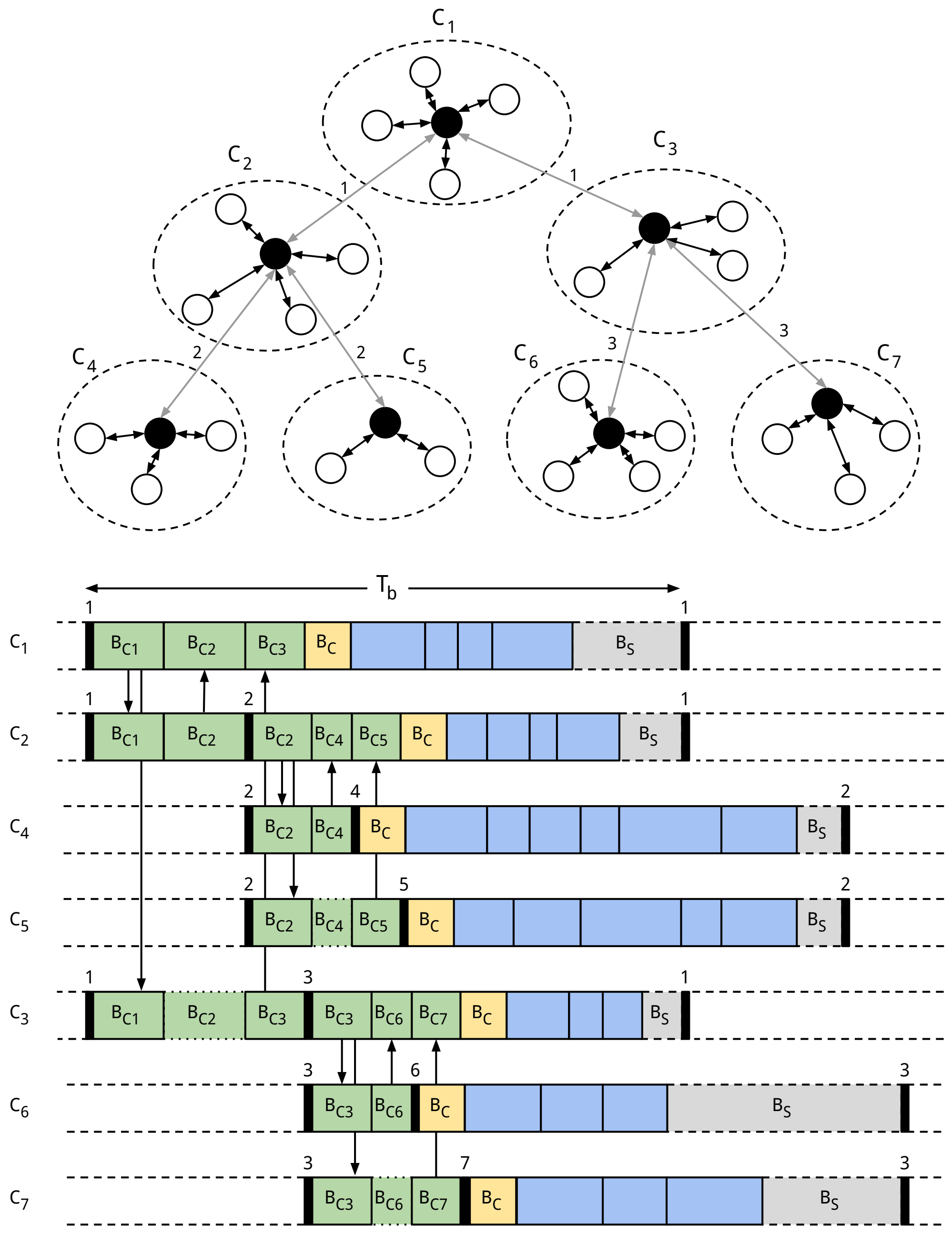

Figure 4.3. *Example of a WBuST clustered-tree binary network and the relative traffic schedulation. The number on top of the beacons represents the cluster that sends that beacon. The arrows among the slots indicate the direction in which the inter-cluster traffic flows.*

To explain the principles of the clustered-tree communications, let's take for instance the network shown in the first part of figure 4.3.
In this topology, the communication between two clusters are synchronized by the parent cluster. The child clusters should therefore wait the parent beacon on its channel. The number written next to the inter-cluster link represents the cluster that is in charge of managing the communication. The communication scheduling starts from the root cluster and propagates downwards till it reaches the leaf nodes. The schedule of the communication windows for this network are shown in the second part of figure 4.3. At the beginning of each CW of cluster C_1, R_1 sends its beacon to both R_2 and R_3 that are listening on its channel. Once the beacon is received, C_1, C_2 and C_3 are synchronized and they can start transmitting their inter-cluster traffic within their own respective slots B_{C_1}, B_{C_2} and B_{C_3}. After transmitting its messages in B_{C_2}, R_2 switches on its channel and transmits the beacon to coordinate the inter-cluster communication with C_4 and C_5 (that are listening on C_2's channel) and successively, its own infra-cluster communication. This procedure is recursively repeated, starting from the root cluster and for every branch, till the last level of the tree is reached. For each cluster, the infra-cluster communication window, as seen in the previous chapter for the single-hop version of WBuST, is composed by a contention period slot accessed with the CSMA/CA protocol, a series of time slots reserved for the nodes and a possible power saving period used to save energy.

As can be noticed by the schedulation graph, apart the first and second level of the tree, every other cluster experiences a delayed start of its CW as big as the bandwidth used by the parent to communicate, in turn, with its own parent. Moreover, the childs of a cluster, are subject to an additional delay due to the order in which they are served. Indeed C_3, before being able to send data to its parent C_1, it has to wait the amount of time needed by C_2 to communicate with the parent. The bandwidth wasted due to the waiting for the preceding brothers communications is shown in the graph as a dotted slot. In this example, the clusters experiencing this kind of delay are C_3, C_5 and C_7. These periods of inactivity can be used to turn off the radio module and save power. The implementation of this protocol requires a careful management of the synchronization times in order to limit the nodes clock misalignments as the beacons propagate down in the tree.

4.4 Traffic scheduling

4.4 Bandwidth constraints

With respect to the single-hop version and in order to route all the traffic generated by the clusters throughout the tree, a new bandwidth requirement should be met. Let's first define the parameters that will be used later on to formally write the requirement:

- U_k^{DOWN} is the channel utilization required by the cluster C_k to send downstream traffic to its childs
- U_j^{UP} is the channel utilization required by the cluster C_j to send upstream traffic to its parent
- $U_{i,h}^S$ is the channel utilization allocated for the stream S_h in the cluster C_i
- U^{CP} is the channel utilization required for the contention period slot, if any
- U^{PS} is the channel utilization allocated for power saving, if any
- α is, as already described in the previous chapter, the bandwidth wasted due to the overhead introduced by the protocol

and the following utility functions:

- $streams(i)$ is the set of all the streams scheduled in the cluster C_i
- $dest(h)$ is the destination node of the stream S_h
- $parent(i)$ is the parent cluster of the cluster C_i
- $childs(i)$ is the set of the child clusters of C_i
- $brothers(i)$ is the set of all the clusters having the same parent as the cluster C_i

For every cluster C_i composing the network, after having defined the following parameters:

$$\begin{aligned} U_i^{UP} &= \sum_{\substack{h \in streams(i) \\ dest(h) \in parent(i)}} U_{i,h}^S + \sum_{j \in childs(i)} \sum_{\substack{h \in streams(j) \\ dest(h) \in parent(i)}} U_{j,h}^S \\ U_i^{DOWN} &= \sum_{\substack{h \in streams(i) \\ dest(h) \in childs(i)}} U_{i,h}^S + \sum_{\substack{h \in streams(k) \\ dest(h) \in childs(i)}} U_{k,h}^S \end{aligned}$$

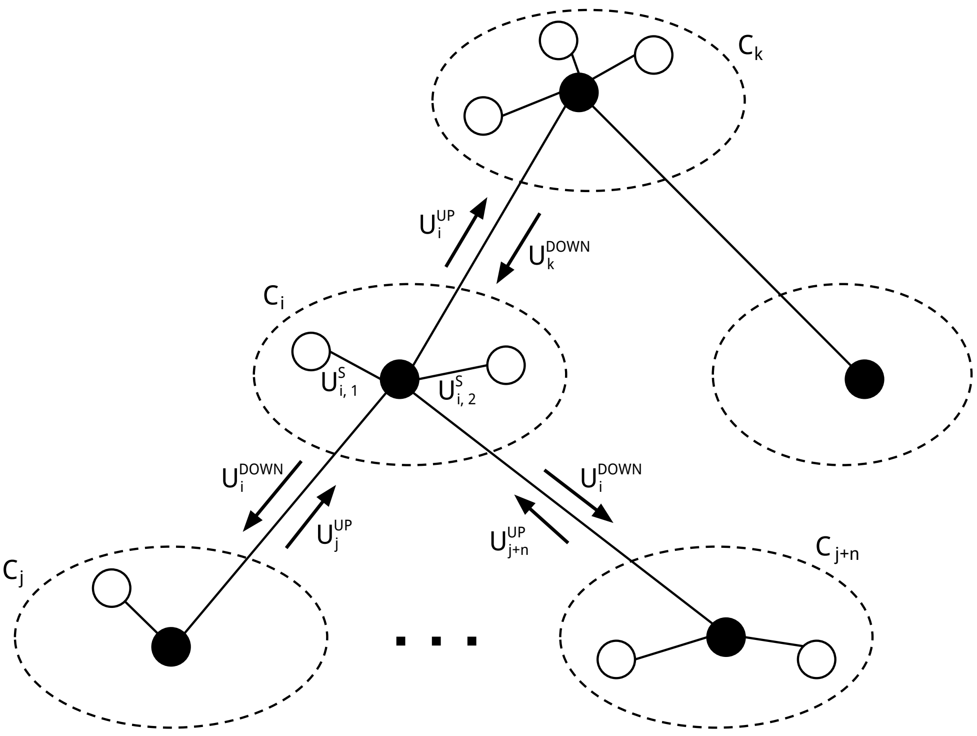

Figure 4.4. *Incoming and outgoing flows in a tree cluster.*

it is possible to provide the property that must be satisfied to have a correct operating protocol:

$$U^{DOWN}_{parent(i)} + \sum_{\substack{h \in brothers(i) \\ h<i}} U^{UP}_h + U^{UP}_i + U^{DOWN}_i +$$
$$\sum_{j \in childs(i)} U^{UP}_j + U^{CP} + \sum_{h \in streams(i)} U^S_{i,h} + U^{PS} \le 1 - \alpha$$

It is straightforward to notice that, for the root cluster C_1, the following simplifications hold:

$$U^{DOWN}_{parent(1)} = 0$$
$$\sum_{\substack{h \in brothers(1) \\ h<1}} U^{UP}_h = 0$$
$$U^{UP}_1 = 0$$

therefore the resulting equation for the root cluster is:

$$U_1^{DOWN} + \sum_{j \in childs(1)} U_j^{UP} + U^{CP} + \sum_{h \in streams(1)} U_{1,h}^{S} + U^{PS} \leq 1 - \alpha$$

4.4 Budget Allocation Scheme

Unlike the single-hop version of the protocol, in which several budget allocation schemes could have been employed, with the multi-hop capability only one of the previous schemes turns out to be suitable for the budget assignments. The reason is that, to guarantee the inter-cluster synchronization, all the CWs of all the clusters must have the same length, that is, the beacon period has to be the same for all the clusters. In other words, given a multi-hop network formed by n clusters C_j and a T_{BT}, the following property must hold:

$$T_b = \tau + B_C + \sum_{i=1}^{n} B_i + B_S = T_{BT}$$

To guarantee this requirement, it is necessary to select $T_{BT} \leq \min_i D_i$ and to select a scheme that satisfies the previous equation. The only scheme that satisfies such a requirement is the Normalized Proportional Allocation (NPA). For the other schemes, if $U < 1$, then $T_b < T_{BT}$.

4.5 Power saving

To analyse the power consumption during the WBuST protocol functioning, an extension of the model in presented in [18] has been developed. The radio module can be put in a sleep state during the power saving time slots. Moreover, we should distinguish between the receiving state and the transmitting one. Let's define the following parameters:

- P^{TX} is the power consumed by a node in a transmitting state
- P^{RX} is the power consumed by a node in a receiving state
- P^{SLP} is the power consumed by a node in sleep mode

Considering all the inter-cluster and infra-cluster slots, the average energy consumed by a node i during a CW of size T_{BT} is:

$$E_i = \Bigg[P^{RX} \Bigg(U^{DOWN}_{parent(i)} + \sum_{j \in childs(i)} U^{UP}_j + \sum_{h \in streams(i)} U^{S}_{i,h} + \sum_{\substack{h \in brothers(i) \\ h<i}} U^{UP}_h \Bigg) + P^{TX} \left(U^{UP}_i + U^{DOWN}_i + U^{S}_i \right) + P^{SLP} U^{PS} \Bigg] T_{BT}$$

Due to the adopted power saving scheme called *Remainder Sleep Time*, the unused time is accumulated at the end of the CW, when all the nodes can turn off their transceiver at the same time. This allow a flexible adaptation to the traffic load and gives the possibility to guarantee a desired lifetime for each network cluster forcing a fixed sleep slot at the end of every CW. To assign a fixed sleep budget and still be sure that the stream set is schedulable, we have to consider the *Worst Case Achievable Utilization* and therefore derive U^{PS} such that, if $U + U^{PS} \leq U^*$, then all the messages deadline will be met. For the NPA scheme, imposing $B_S = \frac{U^{PS}(T_{BT}-\tau)}{U+U^{PS}}$, it turns out:

$$U^{PS} = \frac{U B_S}{T_{BT} - \tau - B_S}$$

4.6 Implementation

The code of the WBuST protocol, prior to be extended, has been restructured to fit the Erika Enterprise *contrib* library. This library includes all the third party contributions which are distributed with Erika Enterprise. The organization of the source code is shown in figure 4.5.
Apart from the KAL and HAL sources, already present in the library, the modules implementing the WBuST protocol are:

- **wbust_buffer.c:** this module defines and implements all the function needed to manage the message buffers, both in transmission and in reception

- **wbust_message.c:** this module defines all the message types allowed to travel in the network as well as their structures and the functions needed to build them

- **wbust_coordinator.c:** this module defines all the coordinator-related functions, like the beacon creation, the schedulability check and the network management structures

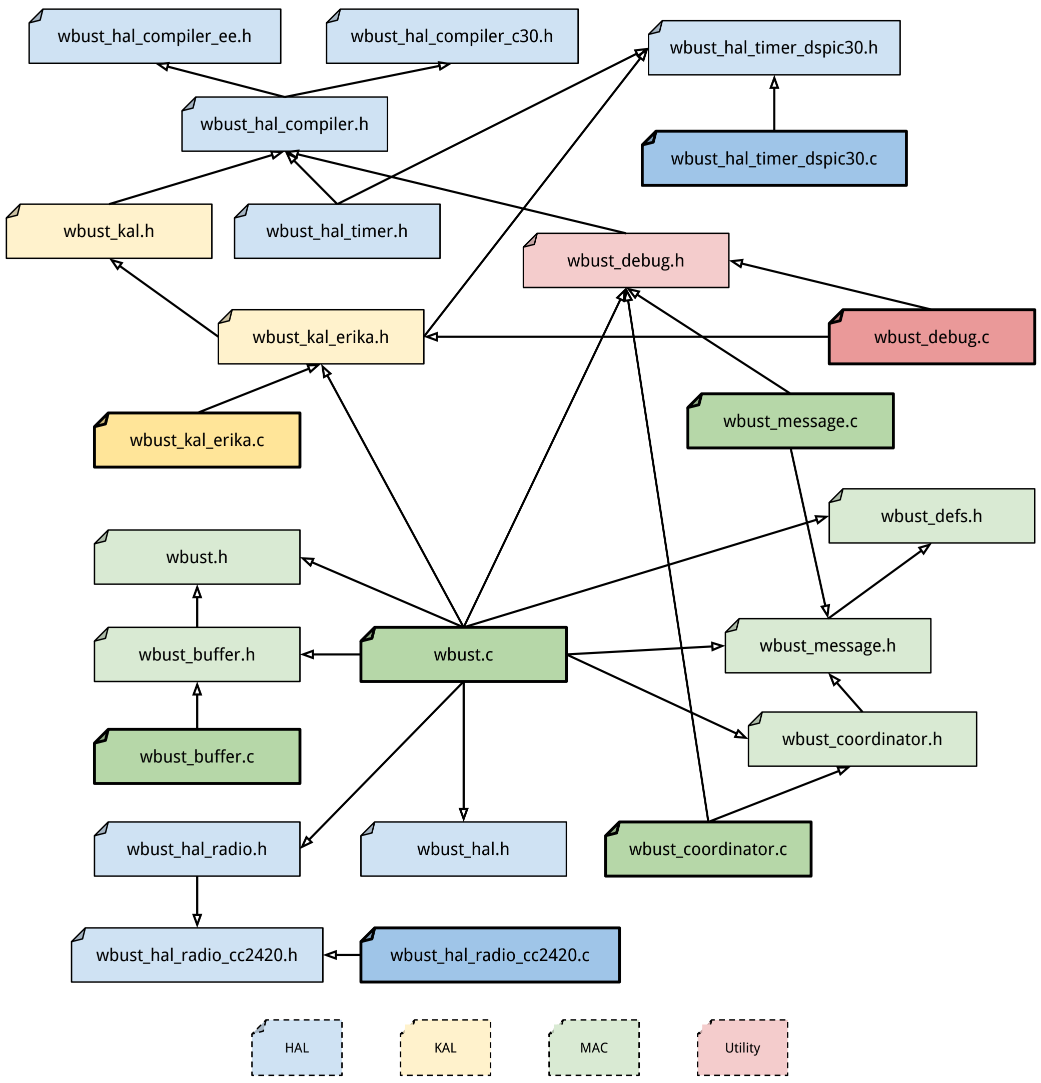

Figure 4.5. *WBuST source files dependency diagram.*

- **wbust.c:** is the main file containing all the logic of the protocol, the processes and the ISRs definitions

The API exposed by the WBuST library to the application layer is composed of the following functions:

- wbust_init (void) initializes the protocol clearing all the support structures and setting up the timers

- wbust_setup (WBUST_NODE_CONFIG* node_conf, WBUST_CLUSTER_CONFIG* clusters_conf, uint8 cluster_num, uint8 power_aware)

- wbust_is_root_coordinator (void) returns whether the node is the root coordinator of the network or not
- wbust_is_coordinator (void) returns whether the node has the coordinator functionalities or not
- wbust_start (void) starts the protocol
- wbust_stop (void) stops the protocol
- wbust_stream_allowed (uint8* stream_id, uint16 max_tx_time, uint16 period, uint16 deadline) tells if the stream is allowed to transmit in the network
- wbust_enqueue_aperiodic_message (uint8* msg, uint16 msg_length, uint8 dest_addr) stores an aperiodic message, if possible, in the queue ready to be sent
- wbust_enqueue_periodic_message (uint8 stream_id, uint8* msg, uint16 msg_length, uint16 msg_id, uint8 dest_addr, uint32 absolute_deadline) stores, if possible, a periodic message in the relative queue waiting to be transmitted
- wbust_set_msg_received_callback (void (*callback) (uint8* rxBuffer)) sets the callback for the message received notification
- wbust_get_network_time () returns the current protocol absolute time

4.7 Node structure

A simplified view of the WBuST protocol architecture is shown in figure 4.6. At the PHY level we have the RF transceiver, which has two 127-bytes length queues called `TX_FIFO` and `RX_FIFO` for transmitting and receiving respectively. The WBuST protocol logic is all contained in the MAC layer. The time is divided into periodic intervals called Communication Windows (CW). Each CW starts when the coordinator node sends a beacon message, similar to what happens in the 802.15.4 protocol. Each node has a timer used to keep track of the flow of time with a 1ms granularity. Each time a beacon is received, the timer is set to 0. This timer keeps track of the time elapsed since the beginning of the CW allowing the node to send when it is allowed to do so. Four tasks are executed on a normal node: *Send Message*, *Acquire Message*, *Elaborate Message* and *Send Budget Left*.

The *Send Message* task is activated, in every CW, for the first time when a reserved time slots starts or when the previous node gives the unused budget to the next one. Apart from the first activation, this task is subsequently activated every time the transceiver signals that the message in the buffer has

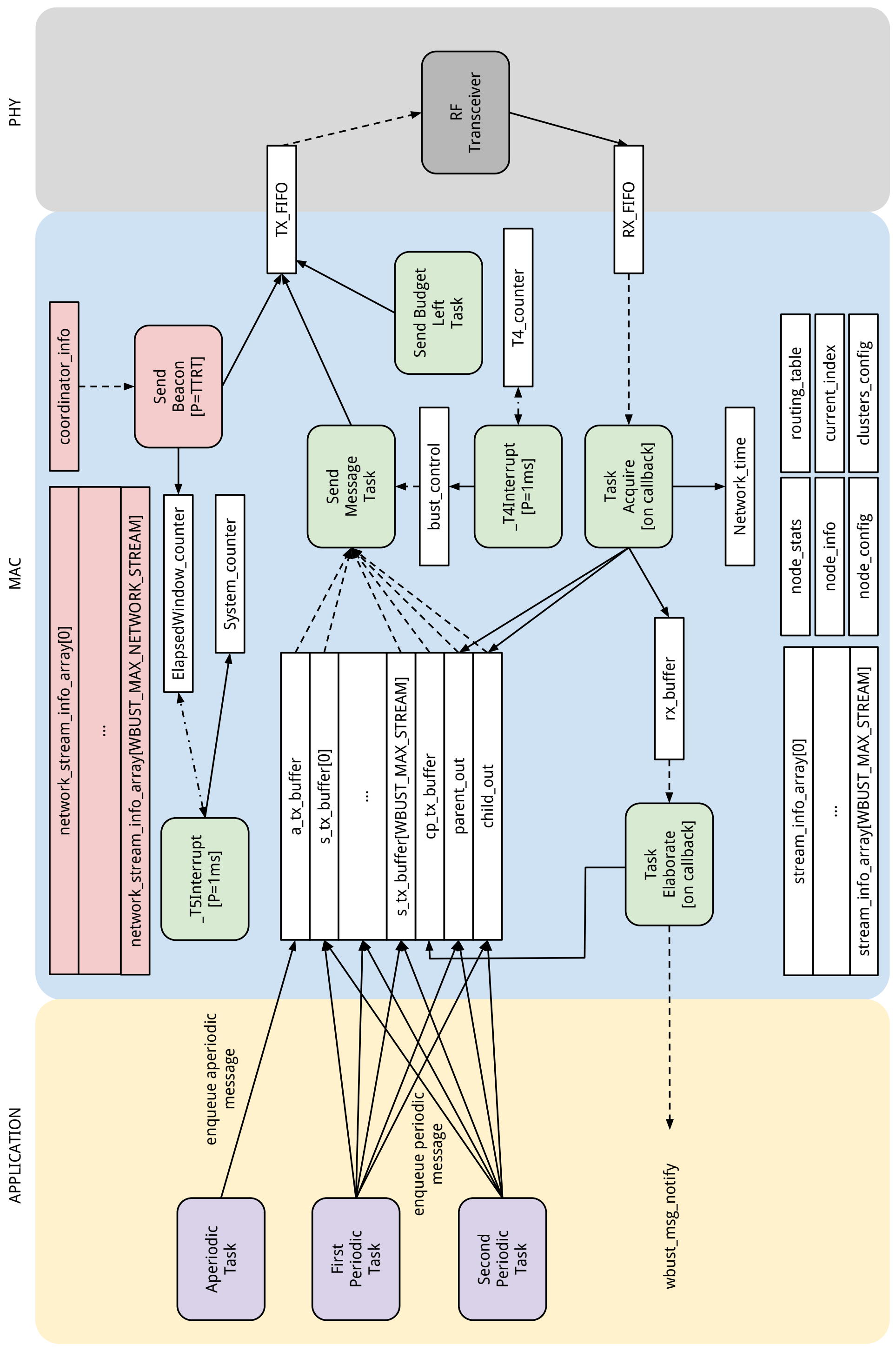

Figure 4.6. *WBuST Node structure.*

been transmitted. Whenever this task is activated, it checks in which time slot the node currently is, in the following order: upstream slot, downstream slot and local stream slot. Then, if there is a periodic message waiting in the correspondent queue and there is enough time to send that message, it sends it. Otherwise, if there is not enough time or there are no messages waiting, it checks whether an aperiodic message is present to be sent.

The *Acquire Message* task is activated every time the transceiver signals, through an interrupt, the presence of a received message in the buffer. This task, executed with the highest priority, immediately copies the message from the transceiver buffer to a local buffer and then checks its CRC, discarding it in case it is not valid. Further checks are performed to discard the packet in case it is a local packet not directed to the node or route the packet in case it is directed to other clusters (using a routing table built from the network topology). In case the packet has to be elaborated by the node, it activates the task *Elaborate Message*, that has a lower priority and is in charge of elaborating the packet. The only case in which the packet is immediately elaborated is when it is a beacon, since the start of the CW has to be delayed as less as possible from the moment in which the message has been sent or when the packet is a simulation start/stop.

The *Elaborate Message* task is in charge of selecting which type of message has been received and to perform the correspondent actions. Basically it calls the callback function to communicate to the application that a message has been received. In case the received message is hard real-time message, it also checks its latency to update the statistics.

Finally, the *Send Budget Left* task is called every time a node does not have anymore messages to send and the time slot is not finished yet. This task simply checks whether it is convenient to send a message to the other nodes (evaluating the remaining time of the current slot and the next CW start) or not. In the case the message can be sent, it is prepared, copied in the transmission queue of the transceiver and then sent.

The coordinator/router nodes, have one more task called *Send Beacon*, called every T_{BT} ms that simply sends the beacon to the other nodes in the cluster and to the coordinator nodes of all its child clusters. This task has also the job of initializing the first time slot of the CW, depending on the network schedulation.

Moreover, in order to store the messages, every node has several queues:

- `a_tx_buffer`: used to store the outgoing aperiodic messages
- `s_tx_buffer[0..MAX_STREAM]`: one queue for each stream, used to store the outgoing periodic messages

- `cp_tx_buffer`: used to store the outgoing contention period messages
- `parent_out`: used, by the router nodes only, to store the inter-cluster messages directed to the parent cluster
- `child_out`: used, by the router nodes only, to store the inter-cluster messages directed to the child cluster
- `rx_buffer`: used to store the incoming messages

4.8 Message definition

In this section, all the kinds of message exchanged by the WBuST protocol, are explained. Basically the following types have been defined:

- `WBUST_MSG_TYPE_BEACON`: used by a coordinator node to send the synchronization beacon to the other nodes
- `WBUST_MSG_TYPE_BUDGET_LEFT`: used to anticipate the time slot of a node when the previous one didn't use its slot completely or to communicate to all the nodes in a cluster to enter in the power saving state
- `WBUST_MSG_TYPE_PERIODIC_STREAM`: used to transmit a periodic (real-time) message
- `WBUST_MSG_TYPE_APERIODIC_STREAM`: used to transmit an aperiodic (best-effort) message
- `WBUST_MSG_TYPE_SIM_START`: used for simulation purposes to start the protocol execution
- `WBUST_MSG_TYPE_SIM_STOP`: used for simulation purposes to stop the protocol execution

All the messages are composed of a common header plus a payload that depends on the message type. The packet header, shown in figure 4.7 (a), contains the following fields:

- **Sequence number** (16 bits): represents the packet unique ID given by the source node
- **Source address** (8 bits): represents sender node of the packet
- **Destination address** (8 bits): represents the recipient node of the packet

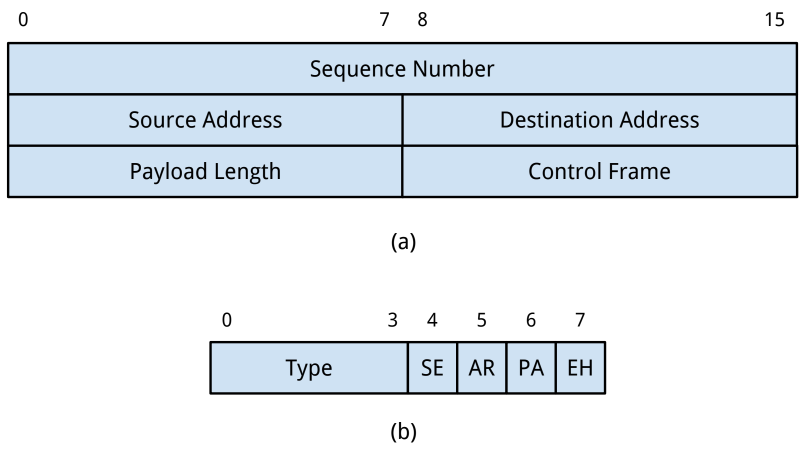

Figure 4.7. *WBuST Packet header structure.*

- **Payload length** (8 bits): represents the size, expressed in bytes, of the packet payload
- **Control frame** (8 bits): contains the control information of the packet, explained later

The *sequence number* field is currently used just for debugging purposes, but it could be used to implement some packet acknowledge, QoS or flow control mechanisms. For both the *source address* and the *destination address* fields, every value between 0x01 and 0xFE are admitted. The 0x00 value is reserved and used as a null-address, while the 0xFF represents the broadcast address. The address is splitted into two fields: the first 4 bits represents the cluster number, while the other 4 bits represent the node number. In this way WBuST can handle at most 16 cluster composed of a maximum of 16 nodes each (except of course the two reserved addresses). The maximum value the *payload length* field can assume is 122, since the maximum packet size that can be handled by the transceiver is 128 bytes and 6 of them are required for the header structure. An additional 8-bit CRC field is present but currently not used since the transceiver already provides its own automatic CRC computing and verification capability. Inside the header, the control frame field that is shown in figure 4.7 (b) is subdivided in:

- **Type** (4 bits): represents the packet type (maximum 16 types)
- **Security Enable** (1 bit): flag that, when set, enables the data encryption to provide transmission confidentiality

- **Acknowledge Request** (1 bit): flag that, when set, requires an acknowledge when received
- **Power Aware** (1 bit): flag that, when set, allow the nodes to use a power saving strategy
- **Extension Header** (1 bit): flag that, when set, tells the presence of an additional header to handle the real-time packets

In the following, all the message types will be described in detail.

4.8 Beacon message

The *beacon* messages (fig. 4.8) are sent by the coordinator nodes and they are composed of the common header plus a payload:

- **Beacon period** (16 bits): represents the communication window duration (ms) before the next beacon
- **Contention period** (16 bits): represents the contention period duration (ms), if any
- **Network time** (32 bits): is the global network time (ms), propagated from the root coordinator downwards in the tree

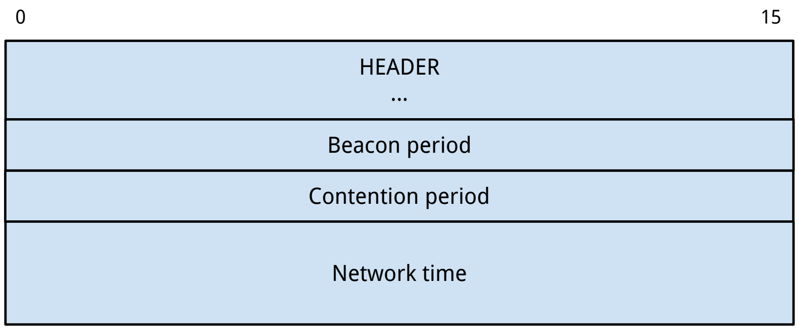

Figure 4.8. *Beacon packet structure.*

For this kind of messages, the field *type* in the header is set to 0x01 and neither acknowledge or extension header are required, therefore the correspondent flags are set to 0.

4.8 Budget left message

Whenever a stream does not have anymore messages to send and there is no best-effort traffic to transmit in the node, if the available budget is not finished yet, the node sends a *budget left* message (fig. 4.9). This message is sent in a broadcast mode setting the *destination address* to 0xFF. This allow

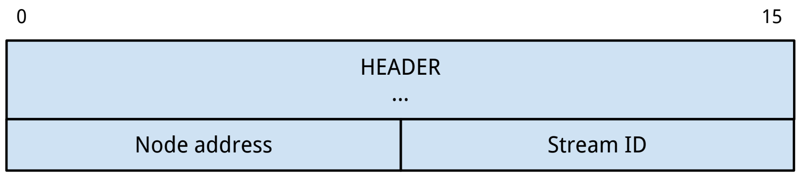

Figure 4.9. *Budget left packet structure.*

the next stream in the schedulation to anticipate its transmission period, avoiding the bandwidth waste that, otherwise, would occur. This kind of message has the *type* field set to 0x04 and carries the following payload:

- **Node address** (8 bits): address of the next node that owns the next scheduled stream
- **Stream ID** (8 bits): local ID in the next node of the stream allowed to anticipate the transmission

When the last stream of the schedulation ends the transmission before the allocated budget, the node sends the *budget left* message with the field *node address* set to 0x00 and *stream ID* set to 0xFF. When a node receives the message with this special combination of values, if the power saving option is enabled, it knows it can put the transceiver in a sleep state to save energy. The *budget left* messages sent to save energy are not sent if the remaining time before the next beacon arrival is less or equal than 4 ms, due to the turn on/turn off time required by the transceiver and the power consumed to switch between the states.

4.8 Periodic message

The periodic messages, identified by the *type* field set to 0x08, are sent at regular intervals. They can be hard real-time messages (fig. 4.10) and therefore have a deadline within which they have to be sent. To store this information an extension header is provided for this kind of messages. It is signaled in the header by the field *Extension Header* set to 1 and it contains the following fields:

- **Absolute deadline** (32 bits): maximum time from when the message is enqueued in the node to when it is sent in the network
- **Transmission time** (32 bits): time when the message leaves the source node. Used by the destination node to compute the latency of the message
- **Source stream ID** (8 bits): local ID of the stream that produced the message. This ID, together with the source address is used to uniquely identify the stream in the network

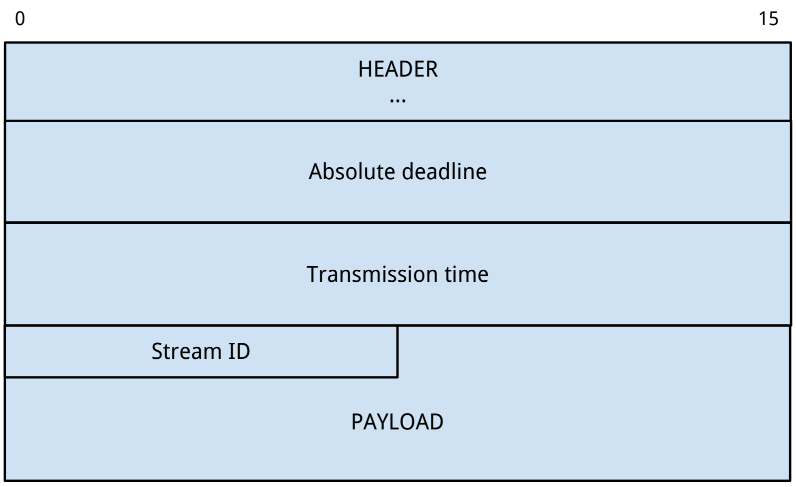

Figure 4.10. *Periodic packet structure.*

The remaining available space is reserved for the payload data that is appended just after the extended header block

4.8 Aperiodic message

The aperiodic messages, also called best-effort messages, are not sent on a constant rate and they do not have any guarantee on their delivery time. Basically they are sent whenever a node does not have periodic messages to send but the budget assigned to it is not finished yet. They are identified by the *type* field set to 0x09 and, apart the header, they do not carry additional control information.

Simulation messages

The simulation start and stop messages are used just in order to automatically start and stop the tests. They are not used in the real operating WBuST protocol. They are identified by the *type* field set to respectively 0x0E and 0x0F. The simulation stop message is also used to collect all the statistics from the nodes to the root node.

5

Development Tools and Devices

In the embedded systems field, the knowledge of the hardware the system is running on is fundamental to understand and to develop code on top of it. Therefore, in this section will be described all the hardware devices on which the protocol has been run as well as the tools used to develop it.

5.1 Hardware Devices

5.1 FLEX Boards

FLEX is an embedded board realized by Evidence, an Italian company, that allows a quick and easy development and testing of real-time applications. The board is based on the Microchip dsPIC DSC microcontroller and its key characteristics are: the modular architecture, a robust electronic design, the support for the Erika Enterprise real-time kernel and an increasingly number of available application notes [29].

The flexibility of this board is due to the modular configuration it has been designed with. There are two kinds of basic board containing the microcontroller that are called *Base Boards*. These can be expanded adding on top of them several other boards called *Daughter Boards*. The two available Base Boards are the FLEX Light (fig. 5.1) and the FLEX Full (fig. 5.2). Both versions are equipped with the Microchip dsPIC33FJ256MC710 microcontroller, extension connectors with 2.55mm step, power supply and ICD2 programmer plugs, and a set of leds. The FLEX Full has also an USB connector for data transfer and an additional onboard Microchip PIC18F2550 microcontroller for integrated programming.

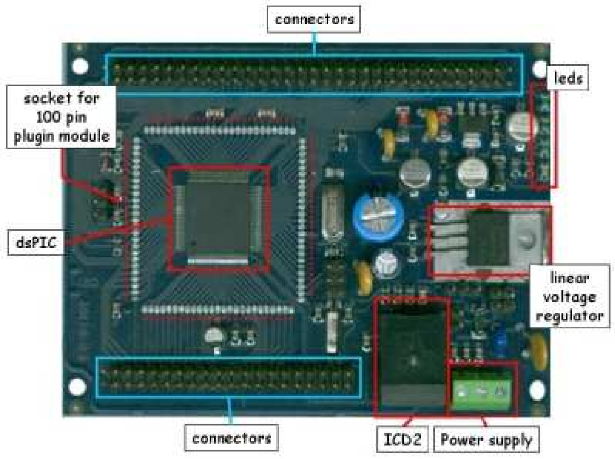

Figure 5.1. *FLEX Light Base board.*

The Daughter Boards comes in three versions: the Thru Hole, the Multibus Base and the Demo Daughter Boards. The first one is simply a plate filled with pinholes at different patterns (1.27, 2.54 and 5.28mm) targeted for the development of small, homemade, custom circuits that can be transparently interfaced with the FLEX Base Boards. The second one is used to extend the communication capabilities of the base board with several busses like UART, CAN, I2C, SPI and Ethernet. The third board, the one used for this work, is shown in figure 5.3.
This board adds several components like 2 Digital-to-Analog converters, a 2 line LCD, 8 LEDs, 4 buttons, a 3-axis accelerometer, a buzzer, a thermal sensor, a potentiometer, an IR transceiver, a light sensor and a connector for a IEEE 802.15.4 compatible transceiver. These additional elements allow prototyping of a great range of applications with a single device that can be configured based upon different needs.

The heart of the FLEX Boards is the Microchip dsPIC33FJ256MC710, an

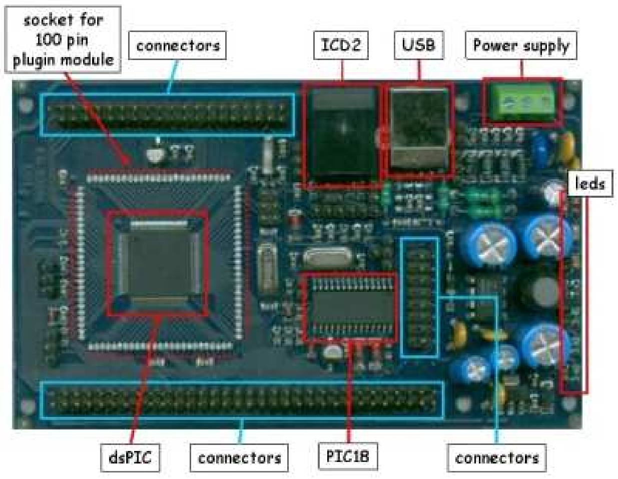

Figure 5.2. *FLEX Full Base board.*

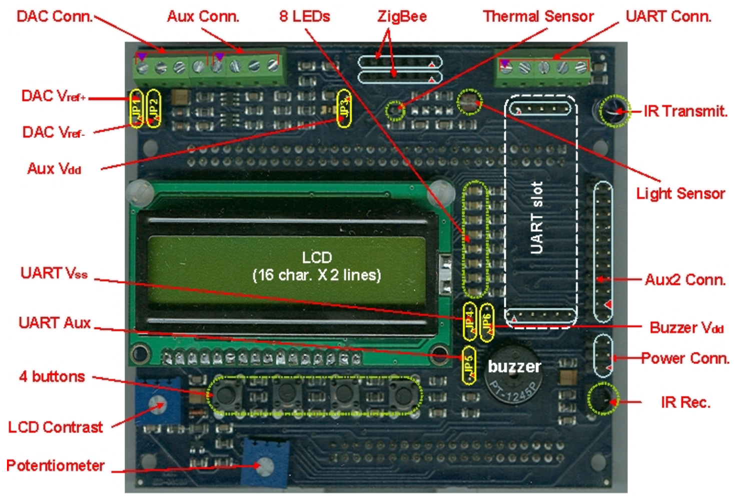

Figure 5.3. *FLEX Demo Daughter board.*

high-performance 16-bit microcontroller belonging to the Digital Signal Controller (DSC) family. It employs a powerful 16-bit architecture that seamlessly integrates the control features of a microcontroller (MCU) with the computational capabilities of a Digital Signal Processor (DSP). The resulting functionality is ideal for applications that rely on high-speed, repetitive computations, as well as control. The DSP engine, together with other features like dual 40-bit accumulators, hardware support for division operations, barrel shifter, 17 x 17 multiplier, a large array of 16-bit working registers and a wide variety of data addressing modes, provide the Central Processing Unit (CPU) with extensive mathematical processing capability [30]. A general block diagram of this microcontroller, taken from its manual, is shown in figure 5.4.

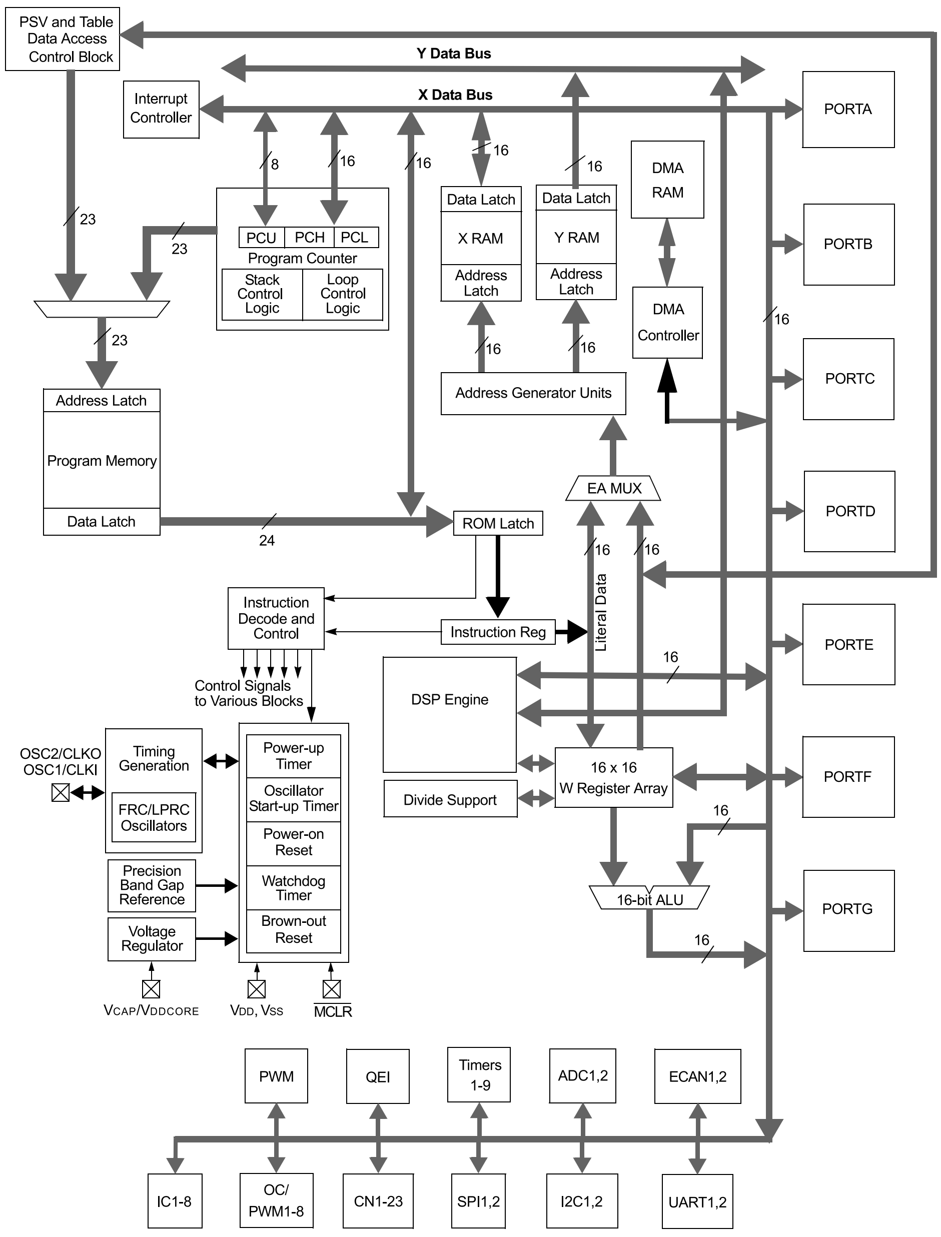

Figure 5.4. *General block diagram of the dsPIC33FJ256MC710 Microcontroller.*

This dsPIC is equipped with 256 Kbyte of Program Flash Memory and 30 Kbyte of Static RAM (SRAM) for runtime data storage. The SRAM also comprehends 2 Kbyte of Direct Memory Access (DMA) to allows data transfer between memory and a peripheral while CPU is executing code, avoiding the cycle stealing. For what concerns the time management functions we have nine 16-bit timers that, in case, can pair up to make four 32-bit timers. The maximum clock speed that can be reached, with the internal oscillator and the Phase-Locked Loop (PLL) is 40Mhz and since among the 83 instructions of the microcontroller set, most of them are executed in one cycle, the maximum instruction execution rate can be up to 40MIPS. Other features like SPI, I2C, UART and Enhanced CAN (ECAN) modules, 8-channels PWM motor controller and 24-channels double ADCs make this device a powerful and multifunctional solution.

5.1 Radio CC2420 Transceiver

The Chipcon CC2420, from Texas Instruments, is a single-chip 2.4 GHz IEEE 802.15.4 compliant RF transceiver designed for low power and low voltage wireless applications. It employs a digital Direct Sequence Spread Spectrum[1] baseband modem providing an effective data rate of 250 kbps [31].

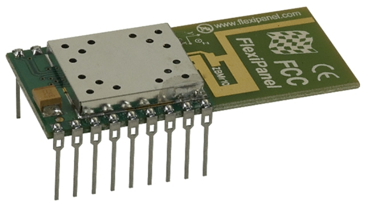

Figure 5.5. *Easybee module with the CC2420 transceiver.*

The transceiver CC2420 is mounted on the FlexiPanel Easybee module shown in figure 5.5 and is accessed by the microcontroller through an SPI interface. As the IEEE 802.15.4 specifies, this module provides 16 communication channels within the 2.4 GHz band, in 5 MHz steps, numbered 11 through 26. This transceiver offers several functionalities:

- **Packet handling:** it automatically detects the packets complying with the IEEE 802.15.4 standard
- **Data buffering:** it provides two queues, each of them 128-bytes long, one for transmitting (TXFIFO) and the other for receiving (RXFIFO). This allow a lower data rate link than the 250 kbps, between the microcontroller and the transceiver, reducing the workload and timing requirements

[1]The Direct Sequence Spread Spectrum (DSSS) is a modulation technique in which, using the full bandwidth available to the device, each bit is transmitted as a redundant sequence of values, called *chips*. It is used for transmission and reception of weak signals.

- **Burst transmissions:** it is possible to store more than one packet in the TXFIFO (up to 128 total bytes of course) and then transmit all of them in a single shot

- **Data encryption and authentication:** these security operations are based on AES encryption using 128 bit keys. Security operations are performed within the transmit and receive FIFOs on a frame basis. In case, this device could be used as a standalone encryption module as well

- **Clear Channel Assessment (CCA):** it is based on the measured RSSI value and a programmable threshold to sense the carrier presence. The CCA function is used to implement the CSMA-CA functionality.

- **Link Quality Indication (LQI):** a built-in RSSI (Received Signal Strength Indicator) giving an 8-bit digital value is used to detect the received signal energy. The RSSI value is always averaged over 8 symbol periods (128 μs). This RSSI value can be used to produce the LQI for a given packet that indicates its strength/quality

- **Packet timing information:** the module alerts the microcontroller through two pins (called FIFOP and SFD) whenever a new packet has been received or a packet transmission has been completed.

- **Output power adjustment:** the transmission power can be programmed from a minimum of -25 dBm (8.5 mA) up to 0 dBm (17.4 mA)

The CC2420 behaviour is modeled as a state machine that switches between different operation states. These states are:

- **Power Down (PD):** after the module is turned on and the voltage regulator is powered up or by manually sending a *turn-off* command

- **Idle:** from the Power Down state when the oscillator is stable or by manually issuing an *idle* command

- **Receive (RX):** from Idle state by issuing a *set-rx* command or automatically whenever a packet transmission has been completed

- **Transmit (TX):** from Idle or Receive states by issuing a *set-tx* command. The CCA functionality can be used to check the carrier presence before starting the transmission

The Idle state, thanks to its extremely low current consumption of 426 μA, should be used whenever power saving is necessary or advisable.

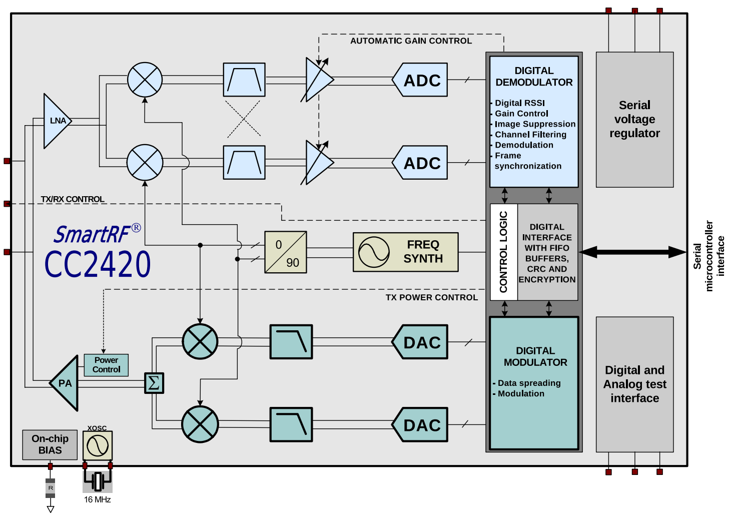

Figure 5.6. *Simplified block diagram of the CC2420 transceiver.*

5.2 Software tools

5.2 Erika Enterprise

Erika (Embedded Real-tIme Kernel Architecture) Enterprise, developed by Evidence, is a free and open source Real-Time Operating System that implements the OSEK/VDX API [32]. The latter is a consortium that has produced specifications for an embedded operating system (OSEK), a communications stack, and a network management protocol (VDX) for automotive embedded systems. Erika Enterprise provides a very small memory footprint real-time kernel of 1-4 Kb for both single and multicore embedded systems. Its features can be summarized in the following points:

- implementation of the 4 standards conformance classes[2] BCC1, BCC2, ECC1, ECC2 plus 3 custom conformance classes called FP (*Fixed Priority*) with Immediate Priority Ceiling, EDF (*Earliest Deadline First*) and FRSH

- support for both preemptive and non-preemptive multitasking

- support for multistack and stack sharing between application tasks and Interrupt Service Routines (ISRs) to reduce RAM usage

[2]The four OSEK standard conformance classes are:
BCC1: Basic tasks, only one task per priority and no multiple activations
BCC2: Basic tasks, more than one task per priority and multiple activations
ECC1: Basic and extended tasks, only one task per priority and no multiple activations
ECC2: Basic and extended tasks, more than one task per priority and multiple activations

- shared resources support
- periodic task activation using alarms

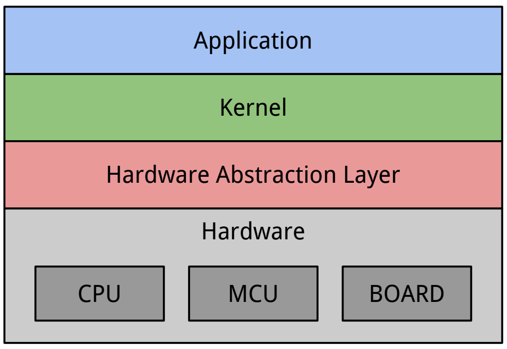

Figure 5.7. *Erika Enterprise layered architecture.*

The layered architecture of Erika Enterprise (fig. 5.7) is built on the real hardware that comprise all the physical devices. To be able to interface different hardware architectures, an Hardware Abstraction Layer (HAL) has been designed with the aim of hiding the physical differences between similar devices and provide a simple and uniform set of functions to the upper layer. The HAL is also in charge of managing all the operating system activities that depend on the hardware like the interruption handling or the context switches. The Kernel layer, in turn, provides an high level RTOS API to the applications for task, alarm, resource and semaphore utilization as well as providing the scheduling strategies for the tasks.

The OSEK/VDX consortium defined a specification language called OIL (OSEK Implementation Language) as a standard for application configuration. This language is used to statically define the components and the functionalities that have to be created to run an application, like CPU, OS, TASK, COUNTER, ALARM, etc. Erika Enterprise fully supports this language complying with both its structure and syntax.

5.2 RT-Druid

RT-Druid is a set of plugins for the Eclipse IDE, the default environment for Erika Enterprise application development. It is composed of a code generator and a schedulability analyser. The code generator is basically an OIL language compiler that, based upon the configuration instructions for the system, creates part of the RTOS source code (like makefiles and data structures) that will be compiled together with the application code. A scheme of a generic OSEK application development is shown in figure 5.8. As we can see, the user has to provide the application source code and the OIL configuration file. With the latter file, the code generator will produce both own code and OSEK OS (Erika Enterprise in this case) code. Compiling together these sources with the user's source code and linking the output with additional OS libraries, gives the final application executable code for the requested target architecture.

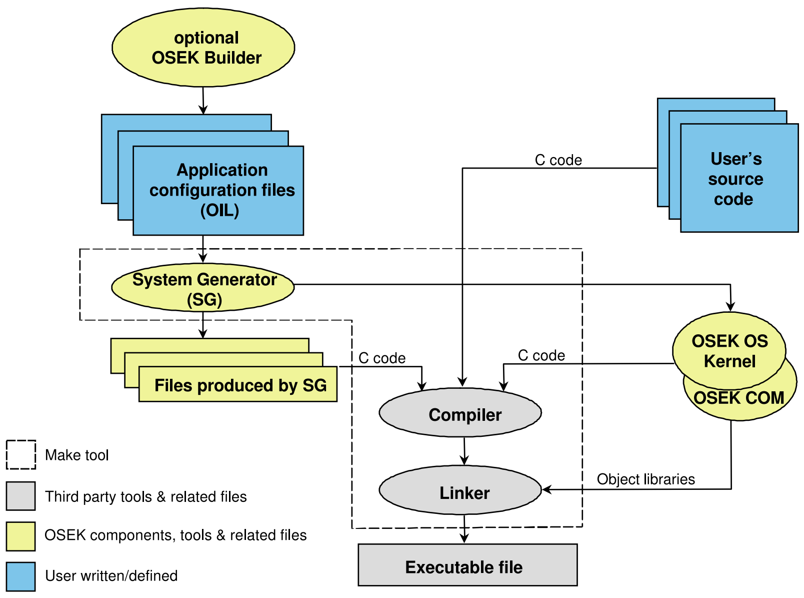

Figure 5.8. *Development process for OSEK applications.*

RT-Druid offers also a wide set of example applications (template projects) ready to be compiled and executed on the FLEX Boards for both showing the functionalities and testing purposes.

5.2 MPLAB IDE

MPLAB is an IDE made by Microchip, for the development of embedded applications, that provides several functionalities like project management, a source code editor, a compiler, a debugger and several microcontroller related functions like program, erase, read, turn-on and turn-off. It supports all the versions of Microchip MCUs (PIC and dsPIC) and, to interface with them, it can use several USB devices (ICD2, ICD3, PICKit2, etc.). These devices, called *programmers*, allow an MCU to be comfortably programmed and debugged directly from the boards they are placed within, simplifying the programmers' job.

The compiler used in this work is the MPLAB C30 Compiler, of which there exist a free version, fully working but with less code optimizations available. This compiler is a porting of *gcc*, ANSI-C compliant, optimized and adapted for the Microchip MCUs architecture.

5.2 TrafficGenerator and SimulationsManager Utilities

Besides the already existing software tools, two additional tools have been developed to allow a faster and more comfortable debugging and data collection. These tool have been written in C# and they could be extended for future uses.

The first tool is called *TrafficGenerator* and it is used to both generate the stream sets and to visualize their respective schedulation. It allows an easy visualization about how the time slots of each node are interleaved and the correspondent channel used to transmit. When the stream sets are ready to be tested, the tool creates a file called simulations.h containing all the structures needed to simulate the planned scenarios and that will be compiled along with the other source files. This program allows to create schedulations using both the PA and the NPA budget allocation scheme.

The second tool is called *SimulationManager* and it is used to speed up the statistics collection and elaboration. It basically connects to the serial port and receives the results from the root node every time a simulation ends. It afterwards elaborates the data depending on the simulation number and the scenario considered, computing the averages and the deviations of the studied parameters. At the end, it writes all the processed data to a spreadsheet file in order to graphically visualize the collected results, allowing a quick check about their validity.

6

Experimental Results

6.1 Testbed

The tests in the laboratory have been carried out employing 10 FLEX boards organized, as shown in figure 6.1, as a tree made by 5 clusters, each of them composed by two nodes: a coordinator/router node and a normal node. All the nodes of the tree are generating periodic traffic directed to the root node. The root node acts as a sink, to simulate the typical scenario we face when dealing with Wireless Sensor Networks: all the nodes are in charge of collecting some data produced by the sensors and send it to the root node, which has to elaborate it. To collect the statistics we are interested in, each node keeps a table of its own local statistics and, at the end of the simulation, exploiting the WBuST protocol itself, sends them to the sink.
The nodes have been physically placed on the workbench at an average distance of 0.5 meters each other, using the full transmission power of the CC2420 transceiver. Each node is associated with a stream having the same period and deadline, both within the interval [250ms, 550ms]. For each simulation, the set of all the stream of all the nodes is called *stream set*.

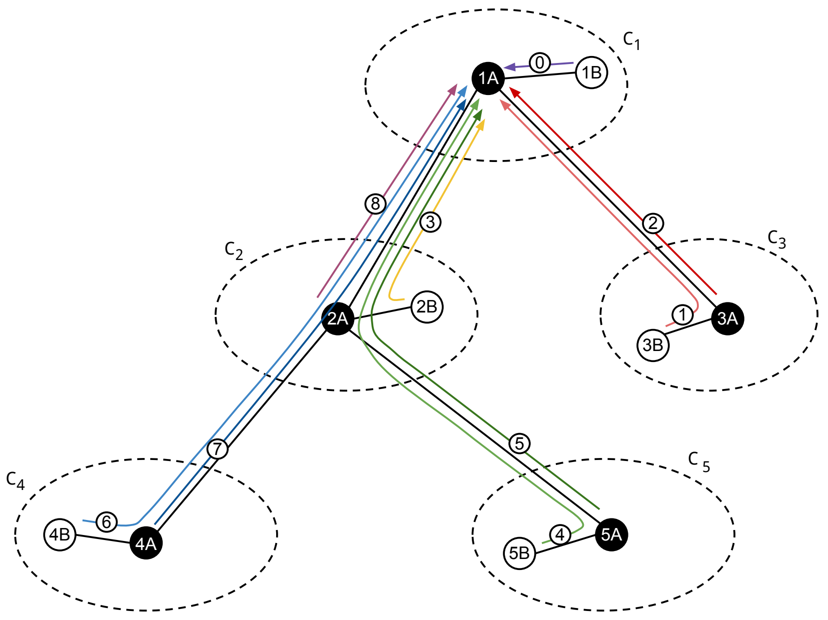

Figure 6.1. *Network topology used to test the WBuST protocol.*

To generate the stream set, the UUniFast algorithm (listing 6.1), presented in [28], has been used. This algorithm is capable of generating random stream sets with a predefined total utilization factor. The complexity of the UUniFast algorithm is $O(n)$ and the generated utilizations are characterized by an uniform distribution. All the informations about the stream sets, budget allocation scheme, network topology, routing tables are generated by the *TrafficGenerator* program and saved in a file called simulations.h. To monitor the network, the statistics have been collected after the initial transient regime, when the utilization of the network by the nodes is constant.
The simulations have been carried out with the following scheme. Four scenarios have been identified as interesting to be studied:

- Only periodic traffic with no guaranteed sleep budget
- Only periodic traffic with guaranteed sleep budget
- Both periodic and aperiodic traffic with no guaranteed sleep budget
- Both periodic and aperiodic traffic with guaranteed sleep budget

When present, the guaranteed sleep budget has been fixed to 10% of the total communication window length in order to study the statistics variation when forcing a given power saving period. For each scenario, 10 simulation

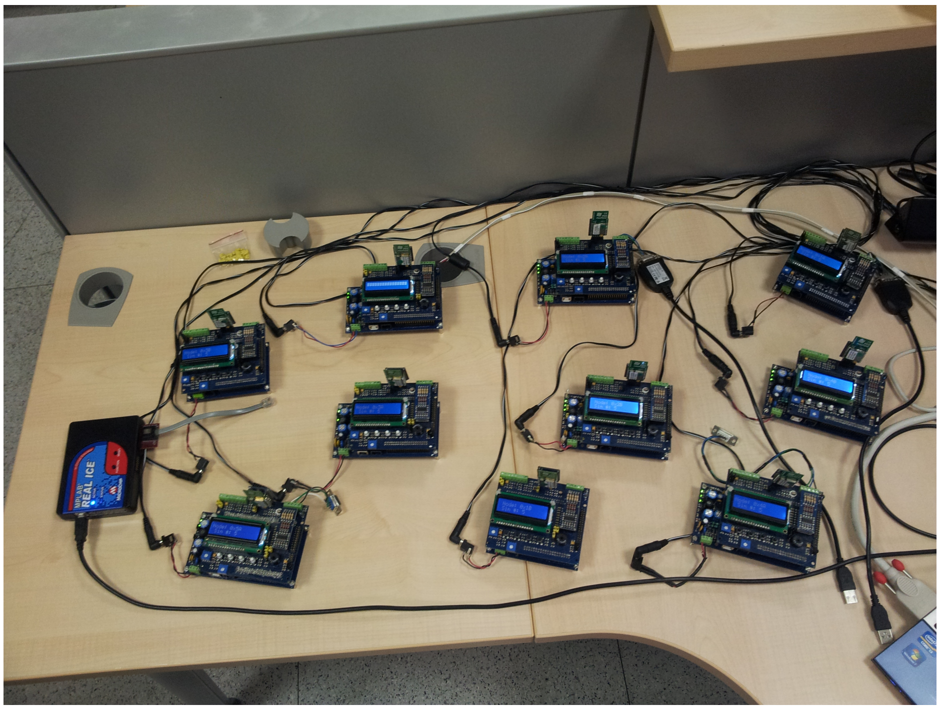

Figure 6.2. *Set of 10 FLEX boards employed to analyse the WBuST performance.*

```c
void UUniFast (int n, double maxU)
{
        double sumU = maxU;
        for (int i = 0; i < n - 1; i++)
        {
                double nextSumU;
                nextSumU = sumU * pow (rand () / RAND_MAX, 1.0 / (n - i));
                vectU[i] = sumU - nextSumU;
                sumU = nextSumU;
        }
        vectU[n - 1] = sumU;
}
```

Listing 6.1. *UUniFast algorithm.*

have been performed, varying the utilization of the root cluster from 0.1 to 1.0 at 0.1 steps. For each simulation, 3 stream sets have been created and for each stream set 10 runs have been executed in order to reduce the statistical fluctuation. The results have been then averaged and the minimum, maximum and variance computed.

For each of the previous run, the following statistics have been collected:

- Number of aperiodic messages received
- Total aperiodic payload received
- Number of periodic messages received
- Total periodic payload received
- Total time during which the node is in power saving mode
- Latency of the messages from all the other nodes in the network and the one taken into consideration
- Number of discarded messages by the router-nodes
- Received beacons
- Missed beacons
- Number of valid and expired periodic messages
- Number of transceiver errors (wrong CRC/wrong message length)

The two most important parameters to study are the Deadline Miss Ratio (DMR), that is the ratio between the hard real-time messages sent within the deadline and total messages produced by the stream, and the latency, that is the time taken by a message from when it leaves the source node to when it is received by the destination node. The latency is important to understand how much delay a message experiences depending on the path it follows in the tree and the utilization factor of the network.

6.2 Results

For each of the four scenarios, the following statistics will be discussed: latency, average deadline miss ratio, sleep time, total periodic traffic received and total aperiodic traffic received. At the end, a comparative analysis among the scenarios will be carried out. For all the simulations, the parameter $\alpha = 0.14$ and therefore, the correspondent value of the WCAU, derived from the formula shown in the previous chapter, is $U^* = 0.43$. The statistics have been collected during a 10s monitoring period, after the network had reached its top speed.

6.2 No aperiodic, no sleep budget

The first scenario considered consist of just real-time streams, with no best effort traffic and no guaranteed sleep budget. The graph in figure 6.3 shows the latencies from all the nodes to the root node.

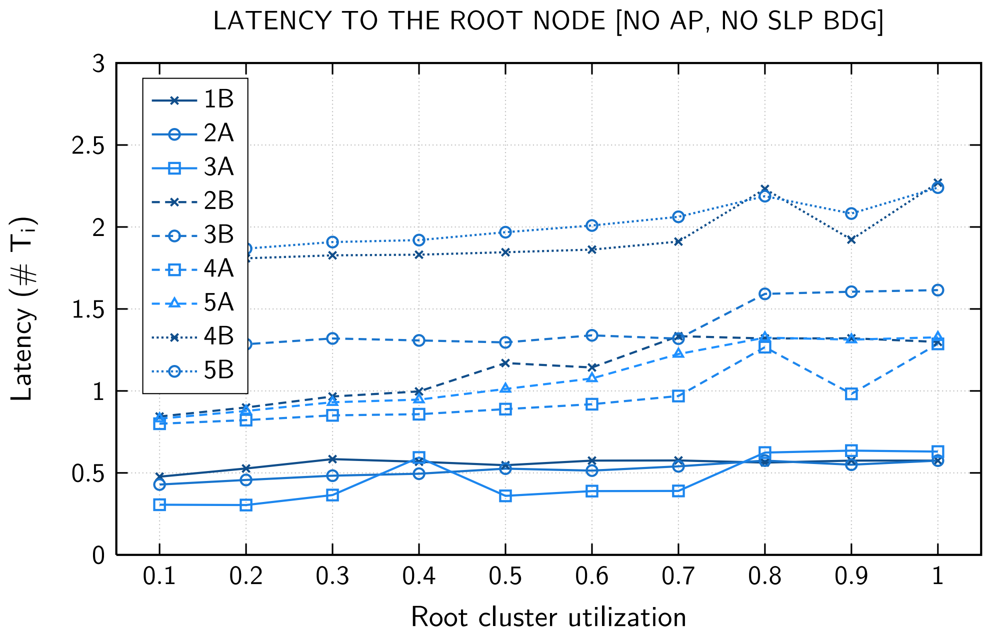

Figure 6.3. *Message latencies from the nodes to the root node with only periodic traffic and no guaranteed sleep budget.*

The latency is expressed, for each stream i, as the number of stream periods T_i taken by every message sent by that stream to reach the destination. In this way, the latencies are normalized and they can be compared each other in a consistent manner. It is clear that the stream latencies tightly depends on the source node that is generating the stream. More precisely they depend on the number of hops between the source node and the destination node (that is the root, in these experiments). Indeed, as we can see from figure 6.1, the nodes 1B, 2A and 3A have a distance of 1 hop to the root, the nodes 2B, 3B, 4A and 5A are 2 hops away and the nodes 4B and 5B are 3 hops distant from the root. The latencies are almost constant or slightly increasing and, in this second case, they stay within 20% of the starting point with the lowest utilization. The most important result shown in this graph is that, independently from the root cluster utilization and even when it is fully loaded, a message sent by any node, destinated to the root, takes an amount of time bounded to the number of hops between it and the root multiplied by the stream period.

The next graph, shown in figure 6.4, shows the average deadline miss ratio among all the nodes. As the theory guarantees, up to the value of $U^* = 0.43$, the ADMR is 0. The deadline miss ratio starts to rise at U = 0.6, assuming the value of 1.5% and keeps rising till it reaches the highest value at U = 1 when the ADMR = 40%.

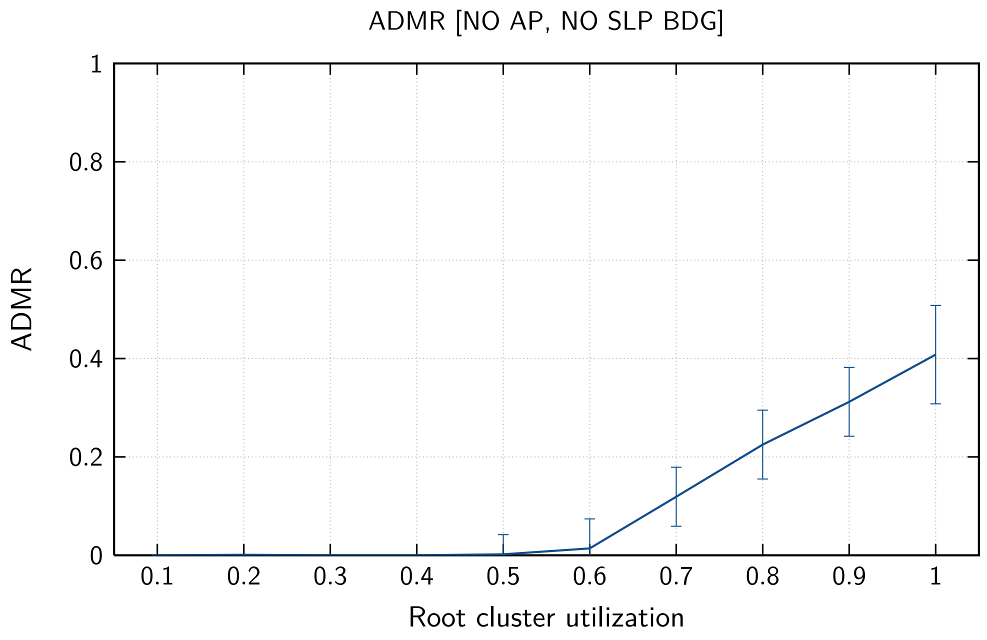

Figure 6.4. *Average Deadline Miss Ratio with only periodic traffic and no guaranteed sleep budget.*

The graph shown in figure 6.5 represents the cluster sleep times when varying the root cluster utilization. When analysing this results, we have to keep in mind that the root cluster traffic splits as we go downwards in the tree, therefore the more the nodes are far from the root, the less traffic they are handling. But we have also to keep into account that a cluster, before being able to communicate with the parent, it has to wait the inter-cluster communications of all its preceding brothers, as seen, for instance, in figure 4.3 for the clusters C_3, C_5 and C_7.
As we can see from the graph, the cluster C_4 is able to reach the highest power saving time while its brother, the cluster C_5, cannot reach the same amount since in every CW it has to wait the C_4 inter-cluster slot before starting its communication slot. This delays the C_5 local communication windows and therefore delays also the power saving state activation time of about 25% with respect to C_4. For the same reason, the sleep time of C_3 is higher that the one of its left brother C_2. We can also notice that starting from U = 0.7

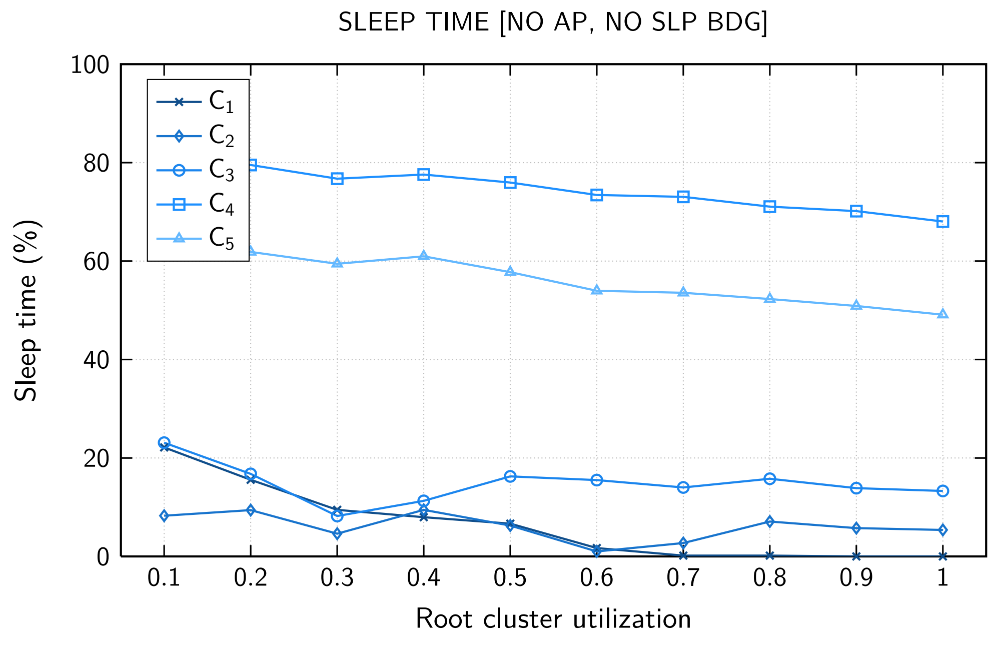

Figure 6.5. *Node sleep times with only periodic traffic and no guaranteed sleep budget.*

and going on, the sleep time of the root cluster C_1 is 0, meaning that the root cluster has not enough bandwidth to enter in the power saving state.

The graph in figure 6.6 shows instead the total periodic traffic received at the root node. The traffic includes also the WBuST control data but excludes the physical overhead introduced by the IEEE 802.15.4 protocol (11 bytes per PHY Protocol Data Unit).
As can be seen, the throughput of the protocol increases linearly till U = 0.6 and then drastically reduces the rising speed up to U = 0.9. When the utilization factor is 1, the protocol undergoes a performance degradation that even worsen the amount of periodic data transmitted.

6.2 No aperiodic, sleep budget

There could be situations in which the need of power saving pushes to accept an higher message ADMR. In this section we will discuss what happens when we force a guaranteed power saving budget, for every node, to 10% of the total functioning time. The first graph about the latencies, shown in figure 6.7, shows that there is almost no changes with respect to the case in which the power saving budget is not guaranteed. Only few nodes experience a small increase, of at most 10%, when the utilization is higher than 0.7.
The figure 6.8, instead, shows a considerable change of the ADMR when we force a minimum power saving budget. Until U = 0.5, the values are the

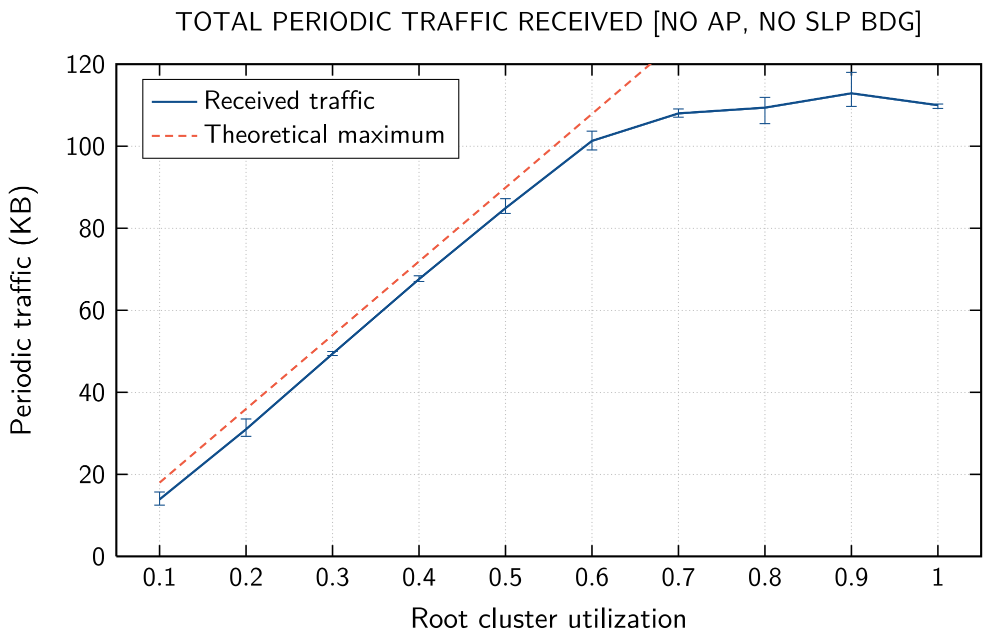

Figure 6.6. *Total periodic traffic received at the root node with only periodic traffic and no guaranteed sleep budget.*

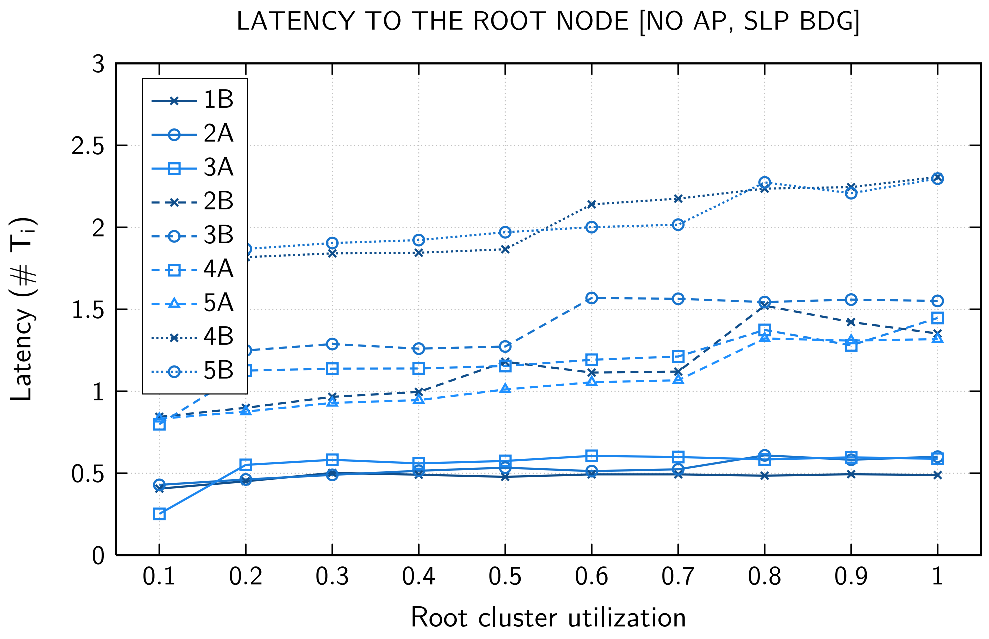

Figure 6.7. *Message latencies from the nodes to the root node with only periodic traffic and guaranteed sleep budget.*

same of those in the previous scenario, but after that value, they experience an increase of around 10% with all the other utilizations. This because, till U = 0.5, the power saving period of each node is higher than the minimum required and after that it comes into play reducing the budget allocated to the node streams therefore increasing the ADMR.

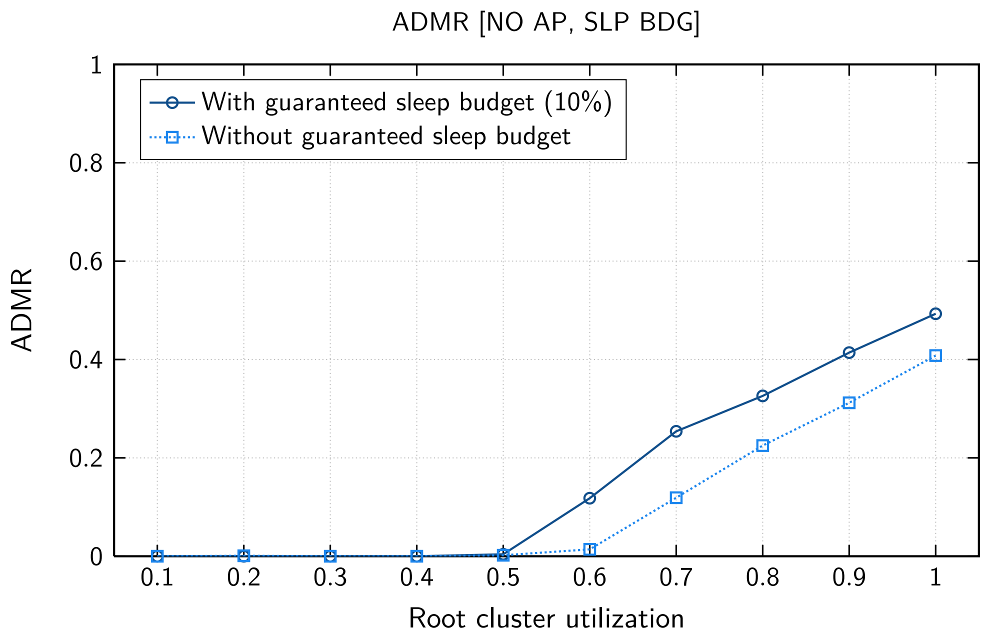

Figure 6.8. *Average Deadline Miss Ratio with only periodic traffic and guaranteed sleep budget.*

Looking now at the sleep times shown in figure 6.9, we notice that the cluster C_4 and C_5 are not influenced by the guaranteed sleep budget since their utilizations are not above 0.5. Instead, the other clusters show an increase in the sleep time and we can notice that the cluster C_1, that previously had no sleeping time with U > 0.6, now guarantees the 10% of power saving in all the load conditions.
Also the graph in figure 6.10, concerning the periodic traffic received at the root, confirms that, with U > 0.6, the curve is flattened and the peak of the traffic, when U = 0.9 is 10% less than the one in the previous scenario without the guaranteed sleep time.

6.2 With aperiodic, no sleep budget

Let's now introduce an aperiodic (best-effort) traffic generator on each node and let's see how this influences the statistics we have analysed so far. The

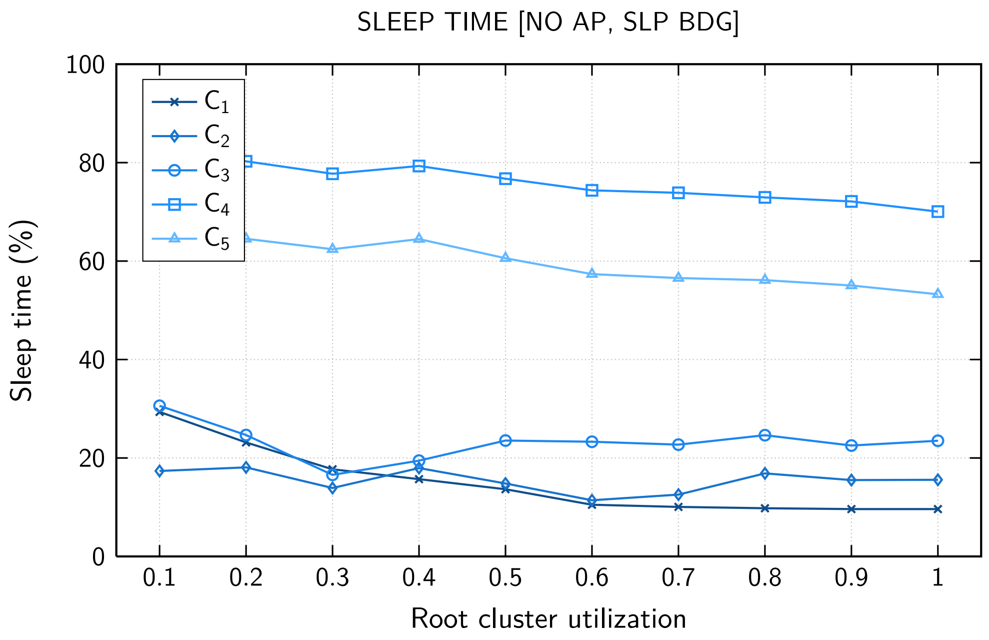

Figure 6.9. *Node sleep times with only periodic traffic and guaranteed sleep budget.*

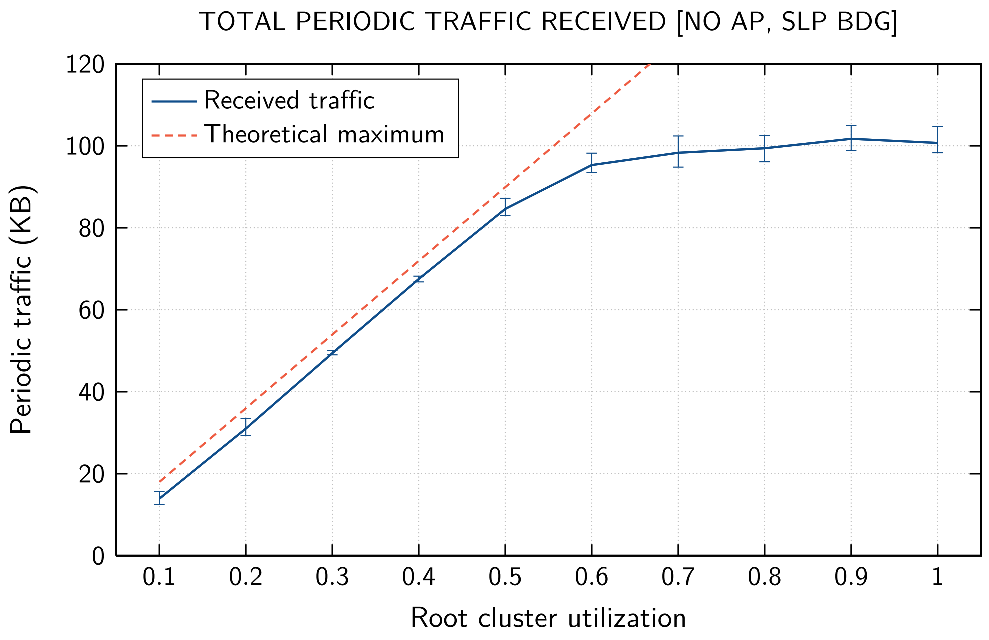

Figure 6.10. *Total periodic traffic received at the root node with only periodic traffic and guaranteed sleep budget.*

aperiodic traffic has an utilization $U^{BE} = 0.05$. The first case is the one without guaranteed sleep budget. As we can see from the latency graph in figure 6.11, there are basically no important differences with respect to the previous scenarios.

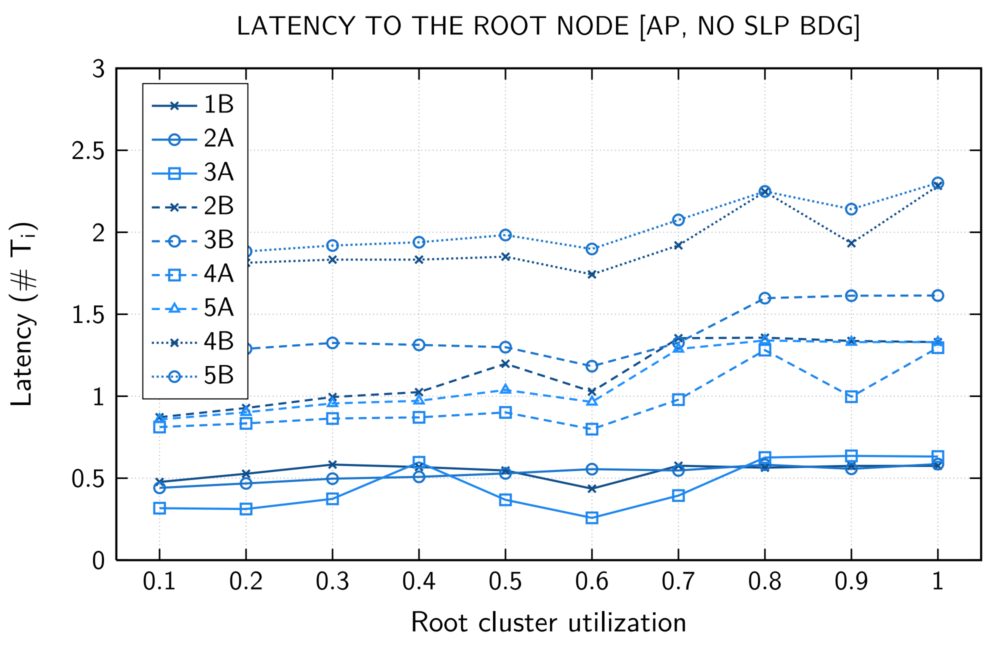

Figure 6.11. *Message latencies from the nodes to the root node with periodic and aperiodic traffic and no guaranteed sleep budget.*

Instead, the ADMR graph in figure 6.12 shows an increase, when U > 0.6, of about 5% in the deadline miss ratios with respect to the scenario without the aperiodic traffic. This because, the aperiodic messages use the unused time slot possibly left by a stream that would otherwise be accumulated by the time slot of the next stream and used to send periodic traffic. Although the values are increased a bit, the ADMR still complies with the limit U^* computed at the beginning of this section.
For what concerns the sleep time, shown in figure 6.13, all the plots are basically unchanged with respect to the scenario with no aperiodic traffic. This because the aperiodic traffic utilization is quite low and the sleep time is heavily influenced by the inter-cluster communication slot sizes that do not use the bandwidth reclaiming mechanism.
The last two graphs, in figure 6.14 and 6.15, show respectively the periodic and the aperiodic traffic received. While the periodic traffic does not display any big difference with respect to the scenario without aperiodic traffic, the aperiodic traffic plot is worth of comments. For U $\leq$ 0.5, all the aperiodic

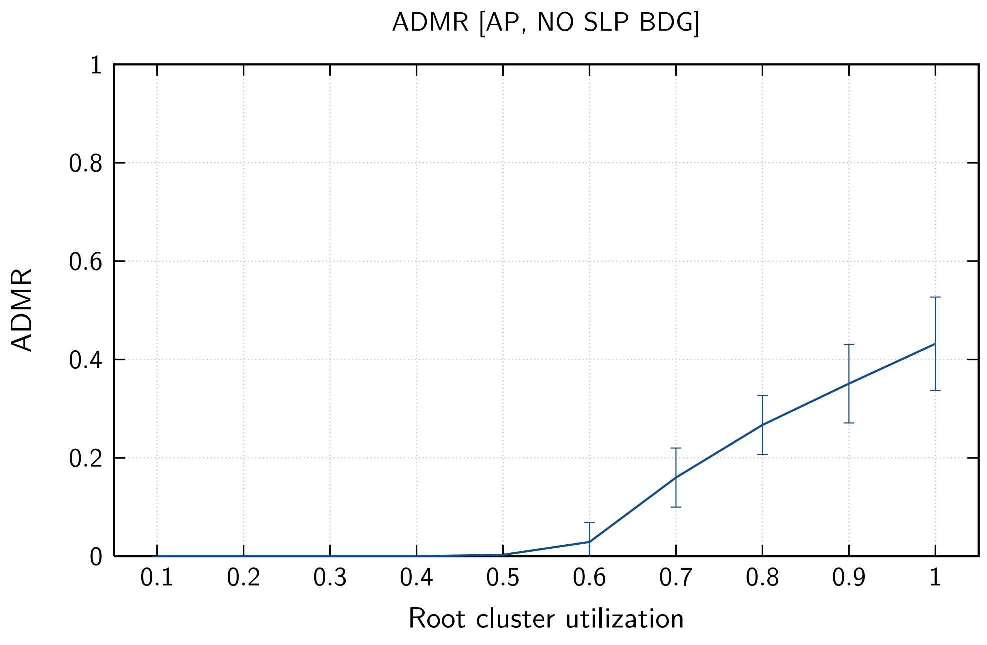

Figure 6.12. *Average Deadline Miss Ratio with periodic and aperiodic traffic and no guaranteed sleep budget.*

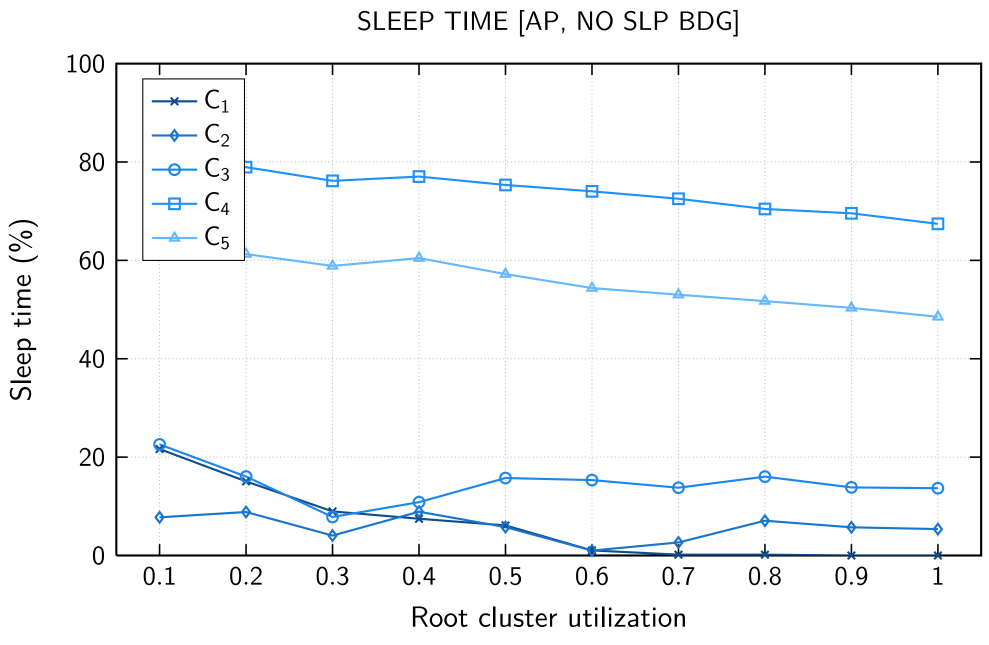

Figure 6.13. *Node sleep times with periodic and aperiodic traffic and no guaranteed sleep budget.*

traffic generated is sent, then it starts to decrease until U = 0.7 and from then onwards it keeps almost constant to 65% of the total generated amount. This because, even when the root cluster is highly utilized, the aperiodic traffic fills the small gaps in the time slots of the nodes that have been left by the periodic messages that missed the deadline.

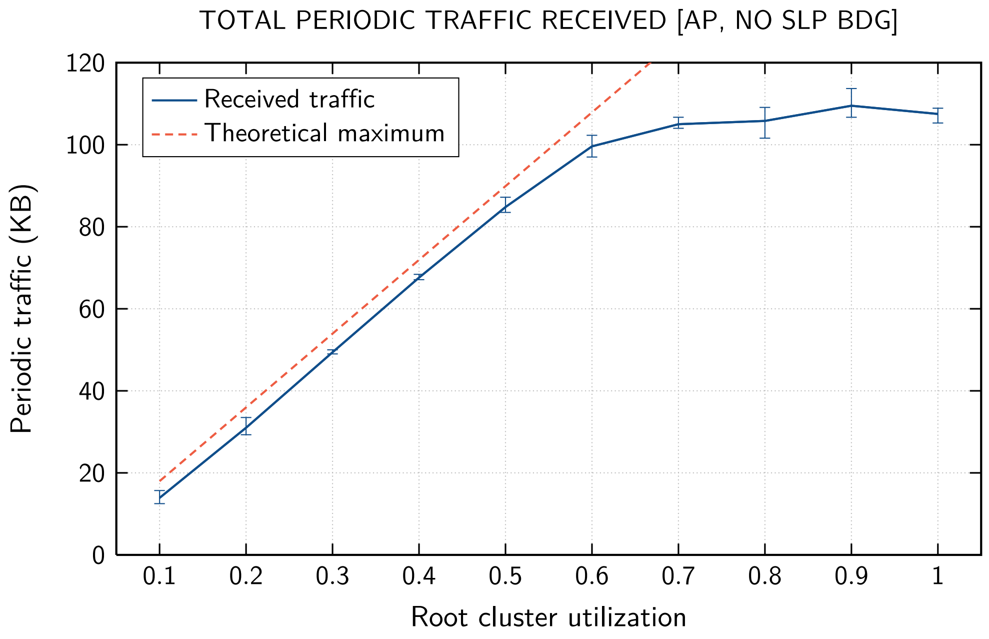

Figure 6.14. *Total periodic traffic received at the root node with periodic and aperiodic traffic and no guaranteed sleep budget.*

6.2 With aperiodic, sleep budget

The last scenario employs both aperiodic traffic and a guaranteed 10% power saving period. Again, there are no big changes in the latencies as shown in figure 6.16 with respect to the previous scenarios. This confirms that when a message is sent, it is guaranteed to be delivered within a number of message periods equal to the number of hops between the source and the destination. Conversely, the ADMR graph, in figure 6.17, shows the worst performance among all the four scenarios. Already when U = 0.5, the ADMR is not zero and assumes the value of 2%. When U = 0.8 the ADMR is almost equal to 40% and when the U = 1.0 it reaches 52%, that is the highest value seen among all the simulations, that means that half of the messages expire and they are not sent within the deadline.
The node sleep time, shown in figure 6.18, displays the same trend of the scenario without the aperiodic traffic while the total periodic traffic received,

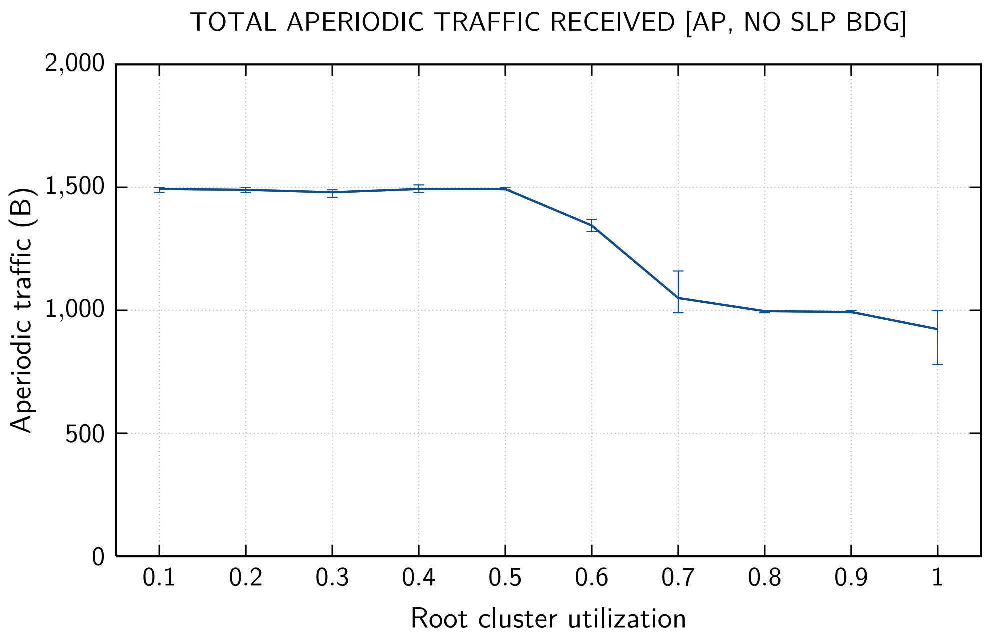

Figure 6.15. *Total aperiodic traffic received at the root node with periodic and aperiodic traffic and no guaranteed sleep budget.*

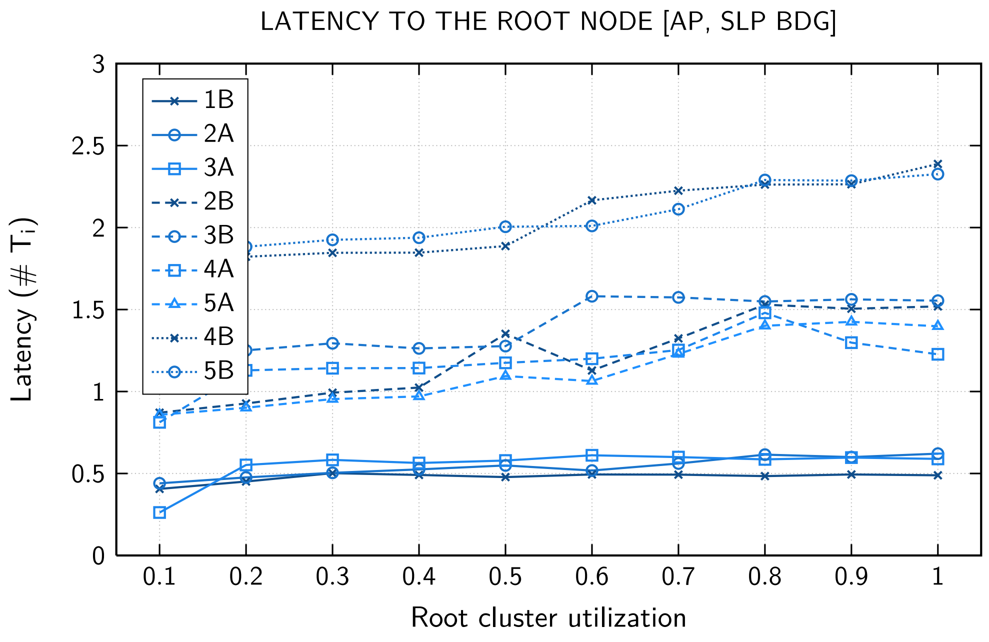

Figure 6.16. *Message latencies from the nodes to the root node with periodic and aperiodic traffic and guaranteed sleep budget.*

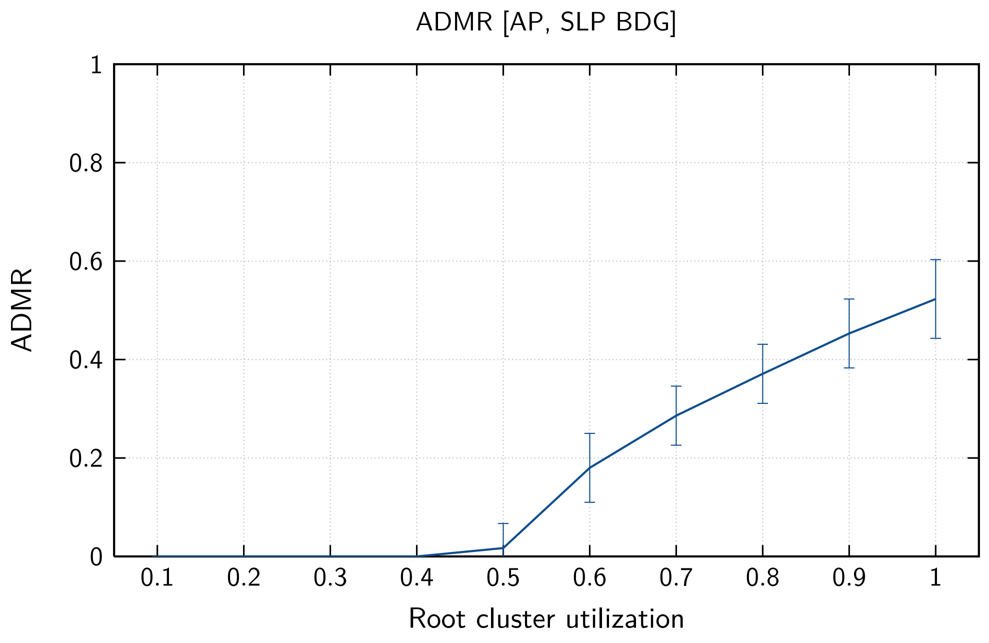

Figure 6.17. *Average Deadline Miss Ratio with periodic and aperiodic traffic and guaranteed sleep budget.*

shown in figure 6.19, with U > 0.6, displays a further 5% decrease with respect to the same scenario without aperiodic traffic.
The last graph on the received aperiodic traffic, in figure 6.20, shows no differences with the previous one except a steeper decrease from 100% to the 65% mostly concentrated in the utilization interval from 0.5 to 0.6.

6.2 Summary

After having discussed all the scenarios in detail, let's now summarize all the results. Figure 6.21 shows a comparison among the ADMR in all the four scenarios. As one could expect the best performance is obtained when there is no aperiodic traffic and no guaranteed sleep time and the worst one when both the conditions are present. When U > 0.5, adding aperiodic traffic worsen the ADMR of 5% in almost all the situations while forcing a 10% of guaranteed sleep time worsen the ADMR of around the same amount, 10%.

The graph in figure 6.22 shows the estimated amount of energy saved during the simulation monitoring interval of 10 seconds for each node in the cluster C_1. The computed value is based on the CC2420 transceiver datasheet and it is calculated as follows:

- $I_{RX} = 18.8$ mA is the current consumption in receive mode

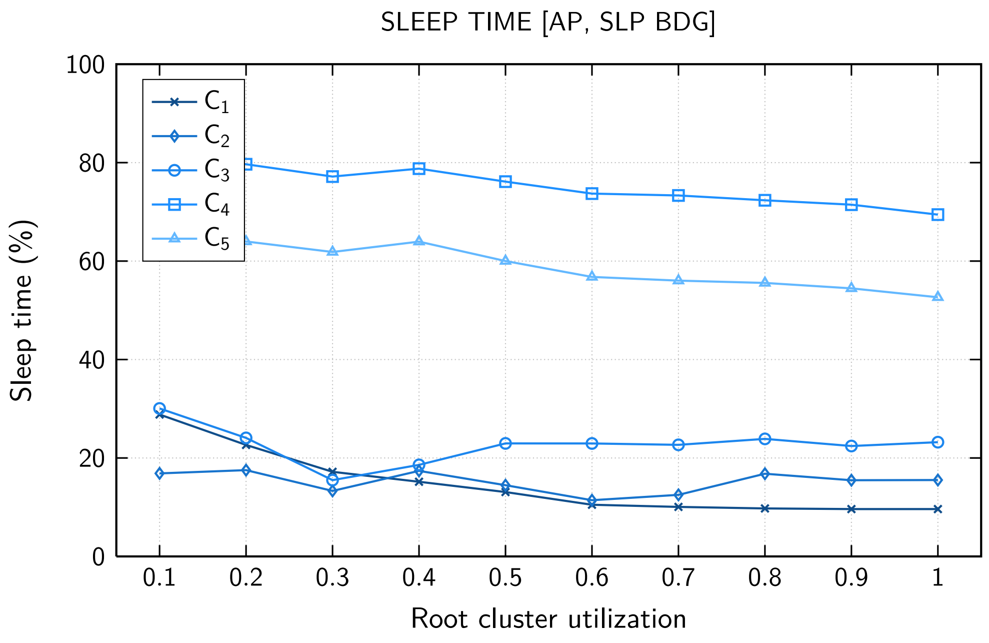

Figure 6.18. *Node sleep times with periodic and aperiodic traffic and guaranteed sleep budget.*

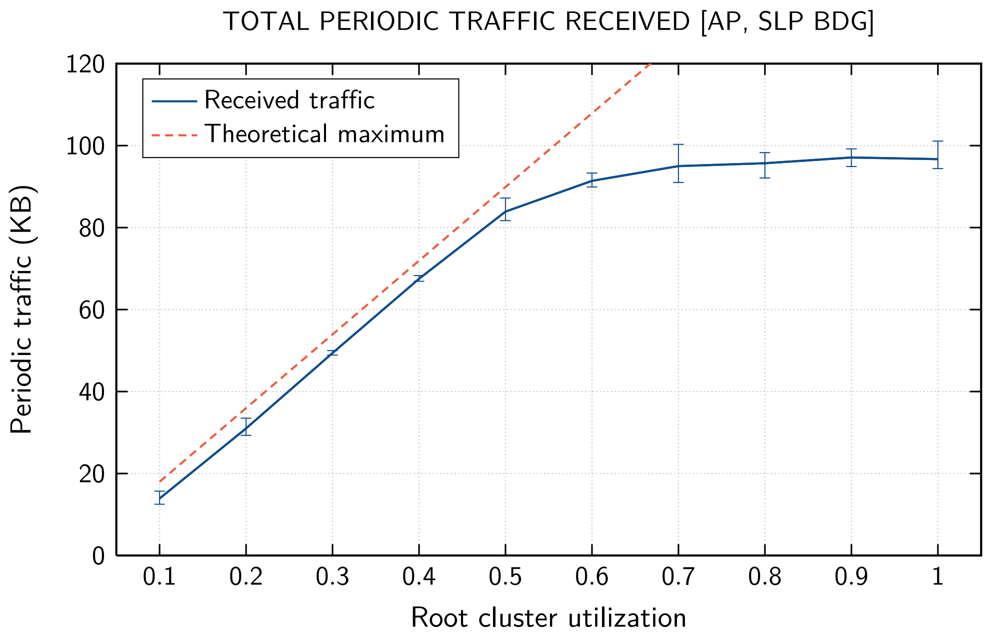

Figure 6.19. *Total periodic traffic received at the root node with periodic and aperiodic traffic and guaranteed sleep budget.*

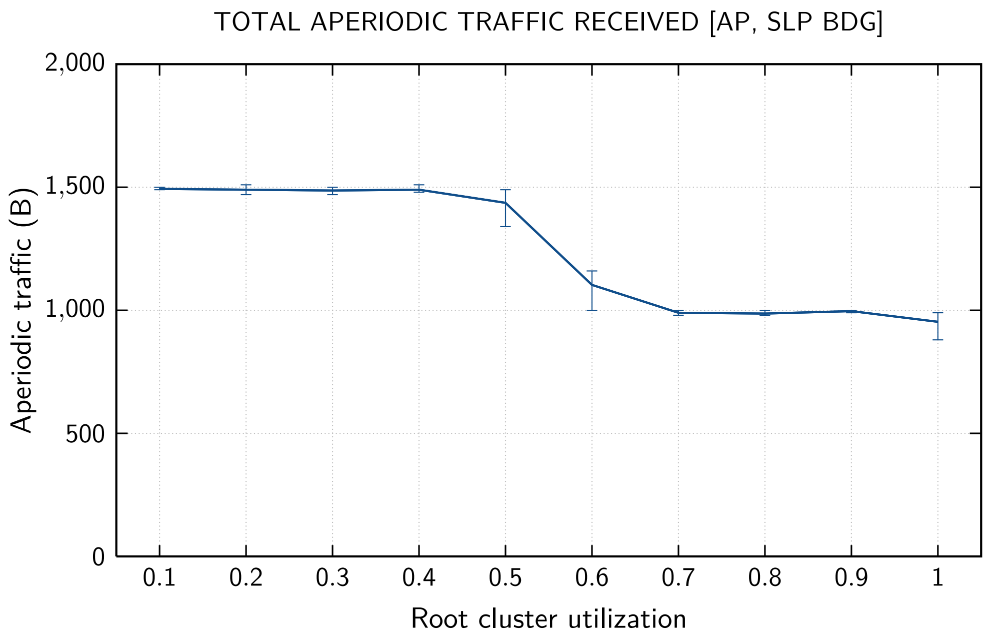

Figure 6.20. *Total aperiodic traffic received at the root node with periodic and aperiodic traffic and guaranteed sleep budget.*

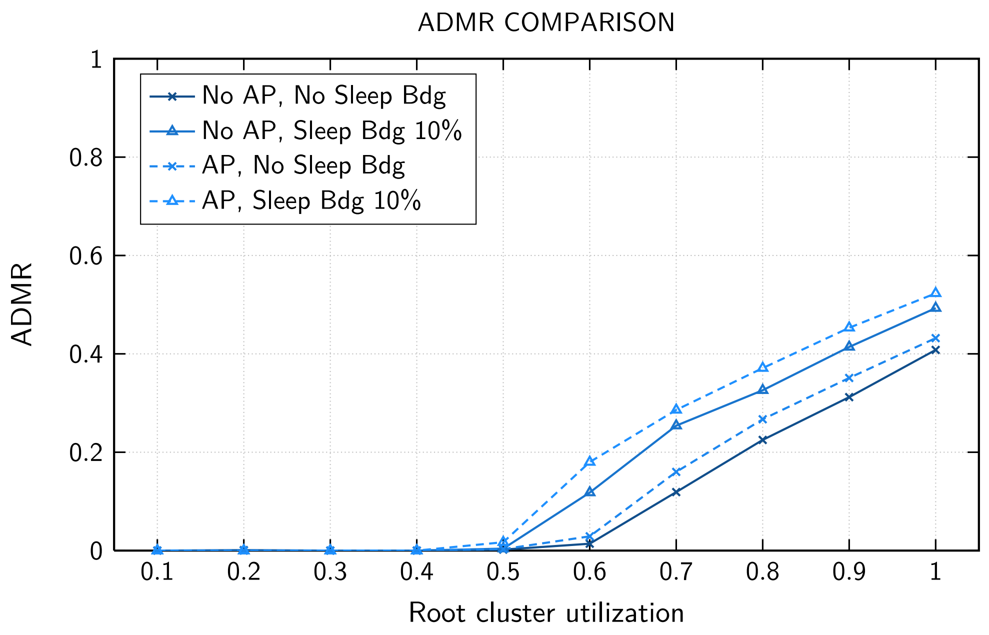

Figure 6.21. *Average Deadline Miss Ratio comparison among the four scenarios.*

- $I_{IDLE} = 426\ \mu$A is the current consumption in idle mode
- $V_{IN} = 1.8$ V is the working voltage of the transceiver

therefore:

- $P_{RX} = V_{IN} \cdot I_{RX} = 33.84$ mW is the power consumption in receive mode
- $P_{IDLE} = V_{IN} \cdot I_{IDLE} = 0.76$ mW is the power consumption in idle mode

Finally, given T^{SLP} the total sleeping time, the saved energy is calculated as:

$$E_{SAVED} = (P_{RX} - P_{IDLE}) \cdot T^{SLP}$$

As can be seen from the graph, the aperiodic traffic has a really small influence on the saved energy, while the guaranteed sleep budget brings a noticeable improvement on the saved energy.

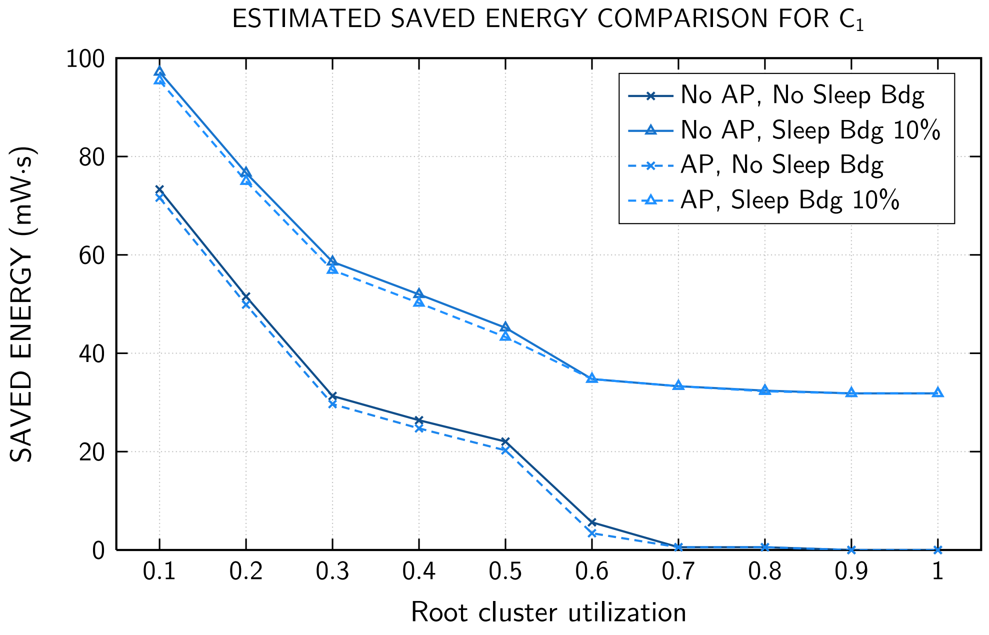

Figure 6.22. *Saved energy comparison among the four scenarios for the cluster C_1.*

7 Conclusions

This work presents the WBuST protocol for real-time, multi-hop wireless sensor networks with power awareness capabilities. The network is composed of clusters of nodes which are organized as a tree. The protocol provides the routing operations to transfer the packets among the nodes, guaranteeing the real-time constraints such as the messages deadline verification and a bounded communication latency. A bandwidth reclaiming mechanism is implemented in order to reuse the portion of the time slots not used by a stream and given to the next one that is waiting to transmit. The power saving capability allows also to turn the transceiver off when the nodes have no more traffic to transmit and the communication window is not finished yet, maximizing the lifetime of the device that, in this kind of networks, is usually battery powered. The work is based and assessed on a consistent theoretical analysis that provides the tools to verify whether a given amount of real-time traffic, described by a stream set, can be guaranteed by the protocol.

An extensive set of tests, carried out in the lab using ten FLEX boards, confirmed the results obtained by the former analysis of the system that was going to be developed. Several parameters have been studied such as the average deadline miss ratio, the communication latencies, the amount of periodic and aperiodic traffic received, the total sleep time and the energy saved. Many others have been collected but not included due their less relevance but still useful to verify the protocol correctness. Besides, four scenarios have been simulated with ten different loads and several stream sets in order to have a good statistical coverage of the possible configurations. The protocol has been proved to be able to manage real-time and best-effort traffic, guaranteeing a predictable transmission time for both intra-cluster and inter-cluster communications, while reducing the energy consumption, according to the theoretical model.

Further future work addresses both the implementative part and the experimental testing part. For what concerns the possible additional development, we highlight: a dynamic node join and leave mechanism for the multi-hop version, a backup cluster-coordinator node that replaces the original in case it experiences a failure, a technique for data encryption that will be used when the data transmitted on the network is confidential and requires secrecy, the automatic switching of the cluster channel when the one currently in use experiences too much interference and a more robust error correction code that allows to recover corrupted packets. About the experimental testing, it would be useful to perform further tests on the power saving capability varying the guaranteed sleep budget for each cluster and to carry out simulations with many more nodes to study the protocol performance on very vaste wireless sensor networks.

Bibliography

[1] K.S. Prabh. *Real-time wireless sensor networks.* Dept. of Computer Science, University of Virginia. 2007.

[2] J. Stankovich e K. Ramamrithan. *Tutorial on Hard Real Systems.* IEEE Computer Society Press, 1988.

[3] G. Buttazzo. *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications.* Springer Verlag, 2005.

[4] Henrik Bengtsson, Elisabeth Uhlemann, Per-Arne Wiberg. *Protocol for Wireless Real-Time Systems.* ECRTS 1999.

[5] Sinem Coleri Ergen. *ZigBee/IEEE 802.15.4 Summary.* September 2004.

[6] K. Kredo and P. Mohapatra. *Medium Access Control in Wireless Sensor Networks.* Technical Report, University of California, Davis, 2006.

[7] J. Zheng and M. J. Lee. *A Comprehensive Performance Study of IEEE 802.15.4.* 2004.

[8] S. Zhang, A. Burns, J. Chen and E. Lee. *Hard Real-Time Communication with the Timed Token Protocol: Current State and Challenging Problems.* Real-Time Systems. September, 2004.

[9] I. Lee, J.Y.T. Leung and S. H. Son. *Handbook of real-time and embedded systems.* CRC Press, 2007.

[10] J. Stankovic et al. *Real-time communication and coordination in embedded sensor networks.* Proceedings of The IEEE Volume 91, 2003.

[11] X. Zhuang, Y. Yang and W. Ding. *A TDMA-based protocol and implementation for avoiding inter-cluster interference of wireless sensor*

network. Proceedings of the 2009 IEEE International Conference on Industrial Technology, pp. 1-6, IEEE 2009.

[12] M. Caccamo, L. Y. Zang, L. Sha and G. Buttazzo, *An implicit priorized access protocol for wireless sensor networks*. Proceedings of IEEE Real-Time Systems Symposium, 2002.

[13] M. Gleeson. *A real time implementation of TBMAC using IEEE 802.11b*. University of Dublin, Trinity College, 2004.

[14] T.L. Crenshaw et al. *A Robust Implicit Access Protocol for Real Time Wireless Collaboration*. IEEE Real-Time Systems, 2005.

[15] G. Korkmaz, E. Ekici, and F. Ozguner. *Black-burst-based multihop broadcast protocols for vehicular networks*. IEEE Transactions for Vehicular Technology, 2007.

[16] M. Wang, L. Ci, P. Zhan and Y. Xu. *Multi-Channel MAC Protocols in Wireless Ad Hoc and Sensor Networks*. 2008 ISECS International Colloquium an Computing, Communication, Control and Managment, 2008.

[17] P. Kumarawadu, D. Dechene, M. Luccini and A. Sauer. *Algorithms for Node Clustering in Wireless Sensor Networks: A Survey*. Proceedings of the 4th International Conference on Information and Automation for Sustainability, December 2008.

[18] G. Franchino and G. Buttazzo. *WBuST: a Real-Time Energy-Aware MAC layer Protocol for Wireless Embedded Systems*. Proceedings of the 17th IEEE Conference on Emerging Technologies and Factory Automation, 2012.

[19] G. Franchino, G. Buttazzo and T. Facchinetti. *BuST: Budget Sharing Token protocol for hard real-time communication*. IEEE Conference on Emerging Technologies and Factory Automation, 2007.

[20] N. Malcom and W. Zhao. *The Timed-Token Protocol for Real-Time Communications*. Texas A&M University, January 1994.

[21] G. Agrawal, B. Chen, and W. Zhao. *Local Synchronous Capacity Allocation Schemes for Guaranteeing Message Deadlines with the Timed Token Protocol*. Proceedings of the 12th Annual Joint Conference of the IEEE Computer and Communications Societies, 1993.

[22] M. Hamdaoui and P. Ramanathan. *Selection of Timed Token Parameters to Guarantee Message Deadlines*. IEEE/ACM Trans. on Networking, 1995.

[23] S. Zhang, A.Burns and A. Wellings. *An Efficient and Practical Local Synchronous Bandwidth Allocation Scheme for the Timed Token MAC Protocol*. Proceedings of the IEEE Infocom '96, pp. 920-927, 1996.

[24] D. Chen, E. Chan and C. H. Lee. *Timing Properties of the FDDI-M Medium Access Protocol for a Class of Synchronous Bandwidth Allocation Schemes*. Proceedings of the 7th International Conference on Computer Communications and Networks, 1998.

[25] G. Franchino, G. Buttazzo and T. Facchinetti. *Token Passing Techniques for Hard Real-Time Communication*. IN-TECH, 2010.

[26] G. Franchino, G. Buttazzo and T. Facchinetti. *Time Properties of the BuST Protocol under the NPA Budget Allocation Scheme*. Proceedings of the Conference on Design, Automation and Test in Europe, March 2008.

[27] G. Franchino, G. Buttazzo and T. Facchinetti. *Properties of BuST and Timed Token Protocols in Managing Hard Real-Time Traffic*. Proceedings of the 13th IEEE Conference on Emerging Technologies and Factory Automation, September 2008.

[28] E. Bini and G. Buttazzo, *Biasing effects in schedulability measures*. ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems, (Washington, DC, USA), pp. 196–203, IEEE Computer Society, 2004.

[29] Evidence FLEX Board, `http://www.evidence.eu.com/it/products/flex.html`

[30] Microchip dsPIC33FJ Manual,
`http://ww1.microchip.com/downloads/en/DeviceDoc/70287C.pdf`

[31] Texas Instuments CC2420. `http://www.ti.com/product/cc2420`

[32] Michael Kunz. *OSEK OS*. March 2009.