Supporting Different QoS Levels in Multiple-Cluster Wireless Sensor Networks

A DISSERTATION

Submitted in partial fulfillment of the

requirements for the award of the degree

of

INTEGRATED DUAL DEGREE

in

COMPUTER SCIENCE & ENGINEERING

By

MANISH KUMAR BATSA

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE – 247667 (INDIA)

JANUARY 2010

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my heartfelt gratitude to my guides **Dr. A. K. Sarje**, Professor, and **Dr. Rajdeep Niyogi**, Assistant Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for their trust in my work, their able guidance, regular source of encouragement and assistance throughout this dissertation work. I would also like to express my gratitude to my supervisor at IPP-HURRAY!, **Dr. Mário Alves**, for his outstanding supervision, counsel, advice, support, inspiration, patience, and for always being available during the course of my work in IPP-HURRAY! Research Group. I must thank my co-supervisor at IPP-HURRAY, **Ricardo Severino**, for his guidance and help on a day to day basis. I must state that the dissertation work would not have been in the present shape without his inspirational support.

I also wish to thank the IPP-HURRAY! team and management, for giving me the opportunity to work in their lab and use the facilities. Also, I would like to thank my colleagues at Hands-On lab, Joel Gonçalves, with whom I shared the lab and Ricardo Gomes, for his help in hardware configuration. I also wish to thank my friends at IIT Roorkee, especially Rahul Bishnoi for his valuable suggestions and timely help.

Finally, I would like to thank my parents for encouraging me to take my own decisions and for helping me maintain a balanced perspective in life. I would also like to thank my elder brother for guiding me throughout my student life and my sister for being interested, understanding and encouraging in my academic and non-academic endeavours.

MANISH BATSA

ABSTRACT

Recent advancement in technology and rapid reduction in costs have led to the uses of Wireless Sensor Networks (WSNs) for applications with requirements significantly differing form traditional monitoring applications. Sensor nodes are now being increasingly used for real-time embedded applications having stringent QoS requirements in terms of timeliness and reliability. However, most of the current set of communication protocols use best effort service and do not provide any real-time guarantees on data delivery. Real-time communication and QoS support in WSN remains an open issue and are the focus of this work.

Towards this end, IEEE 802.15.4/Zigbee protocols are considered among the most promising candidates and have been under recent investigations. However, the attempts to evaluate the protocols by implementing them over TinyOS, the most popular operating system for sensor nodes, encountered several problems, mainly because of the limitations of the OS, namely lack of task pre-emption and prioritization. To provide a more reliable platform for a better evaluation of the protocol, we first implement the stack over ERIKA, a real-time operating system with support for task prioritization and priority based preemption. In order to support cluster-tree formation in synchronized mode, we additionally implemented a Time Division Beacon Scheduling mechanism.

While IEEE 802.15.4 does provide options for guaranteed bandwidth by providing contention free time slots, its usefulness is severely restricted for large scale distributed applications with even distribution of critical message. For the rest of the period the protocol uses CSMA/CA algorithm for channel access, without any provisions of QoS support. In this dissertation we extend the QoS to Contention Access Period by introducing priority based service differentiation in CSMA/CA.

CONTENTS

LIST OF FIGURES

CHAPTER 1
OVERVIEW

1.1 Introduction

A Wireless Sensor Network (WSN) consists of multiple battery-powered devices, equipped with one or several kinds of sensors, capable of wireless communication, data storage, and limited amount of computation. According to the traditional view of a WSN application, a large numbers of such randomly deployed devices would then collectively carry out sensing and computations and forward data to a sink. Till recent years, most of the research in this area has been focused on issues relating to such applications, e.g. ad-hoc network formation, mobility, scalability, self organization, routing, energy efficiency.

Recent advancements in technology (e.g. memories, energy scavenging, and hardware design) and rapid reduction in its cost have, however, pushed sensor networks towards an increased use in a much wider range of applications. Sensors are now closely integrated with real-world applications, and interact directly with the environment. Examples of such applications include home and building automation, industrial process control and automation, healthcare applications, disaster response and management and numerous other such real-time monitoring applications. These applications pose stricter timing and reliability requirements than traditional WSN applications (e.g. environmental monitoring, precision agriculture). This Thesis focuses on this set of applications, and it aims at providing the architectural means to support the QoS requirements (with respect to timing and reliability) of such time-critical WSN applications [1].

To satisfy these requirements, timing guarantees must be provided on computation, i.e., at node level, as well as in communication, i.e., at network level. At node level, it comes down to the operating system and its scheduling policy, which should be able to produce predictable timing behavior of tasks. TinyOS [2], one of the most widely used operating system for sensor nodes, however, assumes a non-preemptive scheduling policy, thus providing no real-time guarantees on computation. Previous attempts to provide time-bounded communication relying on TinyOS have met with problems precisely because of this reason [3]. More recent operating systems, e.g Contiki [4], Nano-RK [5] and ERIKA [6] have been designed with real-time properties and are being widely considered especially in applications requiring real-time support.

Real-time support is also required on networking side, since most WSN applications are expected to involve a group of nodes communicating with each other. The protocol used must provide not only an efficient mechanism for sharing the channel but also some means of support for time-bounded communication. While various novel frameworks have been proposed [7-9] to achieve this, none have yet gained significant uses.

IEEE 802.15.4 [10] and ZigBee [11], on the other hand, though originally designed for Low-Rate Wireless Personal Area Networks (LR-WPANs), are fairly established technologies and have shown excellent potentials to fit the requirements of time-critical WSN applications [12]. Efforts [13-14] have been made to evaluate the suitability of the protocol to meet sensor network requirements by implementing it over TinyOS. However, as mentioned before, difficulties arose because of the non real-time nature and lack of preemption of the operating system, causing the loss or delay of critical tasks under heavy duty cycles.

The need for a stack implementation over a real-time operating system has thus been vindicated for a fair assessment of the adequateness of the protocol for WSNs [3]. This dissertation intends to fill this gap by providing an implementation of IEEE802.15.4/ZigBee protocol stack over ERIKA real-time operating system. It further addresses the issue of QoS enhancement for time-critical messages using traffic differentiation strategies.

1.2 Research Context

This dissertational work lies within the context of the ART-WiSE (Architecture for Real-Time communications in Wireless Sensor Networks) research framework [15-16], aiming to specify a scalable two-tiered communication architecture for improving the timing and reliability behavior of WSNs. In this line, the work hereby presented has been carried out within a collaborative research between IPP-HURRAY group[17], based at School of Engineering (ISEP), Polytechnic Institute of Porto (IPP), Portugal and Real-Time Systems Laboratory[18], of Scuola Superiore Sant'Anna, Italy.

1.3 Research Objectives

The major objectives of this dissertation are:

1. To provide an ERIKA implementation of the IEEE 802.15.4 and a set of the ZigBee network layer services, in order to provide a reliable platform for the evaluation of the adequateness of the protocol for WSNs in the line of the ART-WiSe Framework.

2. To implement and demonstrate a priority based QoS differentiation mechanism in the Contention Access Period (CAP) [10] of IEEE 802.15.4 protocol.

1.4 Research Contributions

The major contributions of this Thesis are:

- Implementation and validation, of the following IEEE 802.15.4/ZigBee features in ERIKA[1]:
  - IEEE 802.15.4
    - Acknowledged and Indirect data Transmission mechanism
    - Guaranteed Time Slot(GTS) allocation and De-allocation mechanism;
    - Association and Addressing mechanisms
  - ZigBee Network Layer
    - Network formation
    - Tree routing and addressing mechanism
- Implementation of a Time Division Beacon Scheduling Mechanism [19] to support cluster-tree operation in beacon enabled mode.
- Implementation and validation of a traffic differentiation mechanism [20] in IEEE 802.15.4 slotted CSMA/CA, providing multiple level QoS support in the CAP.
- Design, implementation and validation of a Testbed to carry out the performance evaluation of the above mechanism.

1.5 Organization of the Dissertation

The rest of the thesis is organized as follows:

Chapter 2 gives an overview of the IEEE 802.15.4 and ZigBee protocols. Chapter 3 discusses the hardware and software tools used in the development and analysis. Chapter 4 outlines the implementation of the protocol stack and the TDBS mechanism. Chapter 5 discusses the traffic differentiation mechanism, namely the implementation, evaluation and results obtained. Chapter 6 concludes the Thesis.

CHAPTER 2
IEEE 802.15.4 AND ZIGBEE: AN OVERVIEW

This chapter provides a brief description of some of the important features of the IEEE 802.15.4 and ZigBee protocols. It focuses on the IEEE 802.15.4 Data Link and ZigBee Network Layers, which are relevant in the context of this Thesis.

2.1 Overview

The IEEE 802.15.4 [10] and ZigBee [11] standards together complete the communication protocol stack for Low-Rate Wireless Personal Area Networks (LR-WPANs). The IEEE 802.15.4 defines the Medium Access Control (MAC) and the Physical (PHY) layers while the ZigBee standard specifies the Network (NWL) and the Application (APL) layers. Figure 2.1 shows the layered architecture of the complete stack. The following sections provide brief descriptions of both the standards.

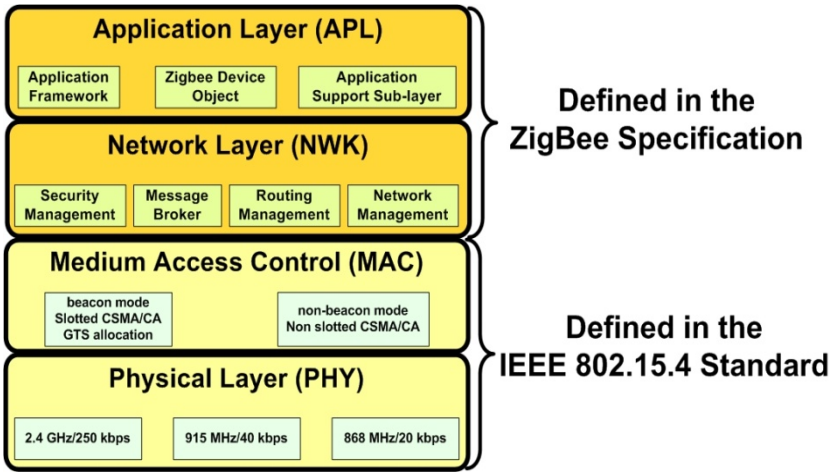


Figure 2.1: The IEEE 802.15.4/ZigBee protocol stack architecture [11]

2.2 IEEE 802.15.4 Physical and MAC Layers

The IEEE 802.15.4 specification defines two different types of devices: the Full Function Devices (FFDs) that implement the full protocol stack and the Reduced Function Devices (RFDs) that implement a subset of the stack.

An FFD can assume one of the following three roles in the network:

(1) *The Personal Area Network (PAN) Coordinator:* Controls the Personal Area Network (PAN), identifying the network and its configurations;

(2) *The Coordinator* : Provides synchronization services by transmitting beacons; must associate to a PAN Coordinator and does not create its own network;

(3) *The End-Devices* : The leaves of the network; must associate with a Coordinator but cannot associate other devices

The RFDs implement only the minimal functionalities of the IEEE 802.15.4 and can only act as end devices. They are intended to support simple tasks, and usually do not have to send or process large amounts of data. One RFD can only associate with a single FFD at a time.

2.2.1 Physical Layer

The Physical Layer (PHY) provides two services: the PHY data service and PHY management service. The PHY data service enables the transmission and reception of PHY protocol data units (PPDU) across the physical radio channel. The management service provides the interfaces between the MAC and the PHY used for exchanging management information.

There are three operational frequency bands (Figure 2.2): 2.4 GHz, 915 MHz and 868 MHz. There is 1 channel between 868 and 868.6 MHz, 10 channels between 902 and 928 MHz, and 16 channels between 2.4 and 2.4835 GHz. Lower frequencies are more suitable for longer transmission ranges whereas higher frequency means higher throughput.

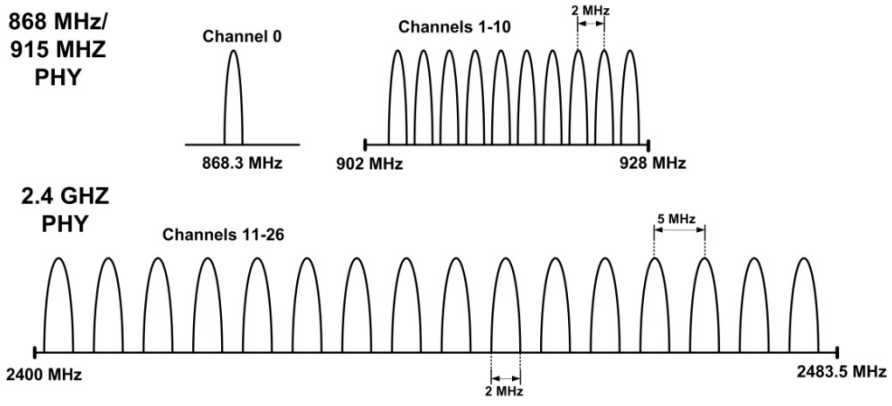


Figure 2.2: Operating frequencies and bands [10]

According to the standard, the physical layer is responsible for the following tasks:

- *Activation and deactivation of the radio transceiver*: turning the transceiver ON or OFF, on the request from higher layers. The turnaround time should be less than 12 symbol periods, where each symbol is made up of 4 bits.
- *Energy Detection (ED)*: the estimation of the received signal power in any particular channel.
- *Link Quality Indication (LQI):* characterization of the strength of a received packet. It may be implemented using receiver ED, a signal-to-noise ratio estimation or a combination of both.
- *Clear Channel Assessment (CCA):* the determination of the current state of the medium: busy or idle. It can be performed using Energy Detection, Carrier Sense or Carrier Sense with Energy Detection. CCA is used in CSMA/CA algorithm.
- *Channel Frequency Selection***:** the ability to tune the transceiver into one of the 27 channels, as requested by a higher layer.

2.2.2 MAC Layer

The MAC protocol supports two operational modes (Figure 2.3):

- *Beacon-enabled mode*: In this mode, beacons are periodically transmitted by the Coordinator to synchronize the nodes, and to identify the PAN. The part of the time frame between two consecutive beacons is called a superframe. Medium access is governed by slotted CSMA/CA mechanism in Contention Access Period (CAP) and Guaranteed Time Slot (GTS) mechanism in Contention Free Period (CFP).
- *Non-beacon-enabled mode*: As suggested by the name, there are no beacons or superframes in non beacon-enabled mode. Medium access is governed by the unslotted CSMA/CA mechanism.

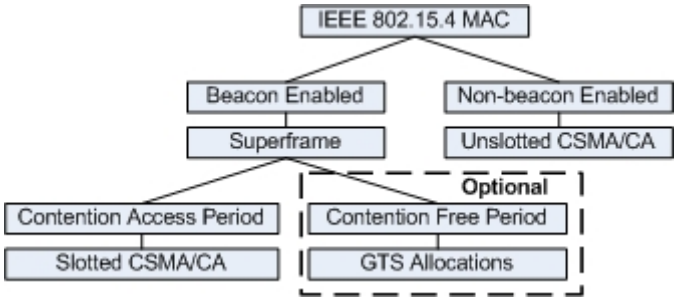


Figure 2.3: IEEE 802.15.4 Operational Modes

Superframe Structure

- In beacon-enabled mode the superframe is defined as the time period between any two beacon frames and has an active and an inactive period, as shown in figure 2.4. The active portion is divided into 16 time slots, and can be made of the following three parts:*Beacon*: the beacon frame is transmitted at the start of slot 0. It contains the information on the addressing fields, the superframe specification, the GTS fields, the pending address fields and other PAN related information.
- *Contention Access Period (CAP):* the CAP starts immediately after the beacon frame and ends before the beginning of the CFP. All transmissions during the CAP, with the exception of acknowledgement and indirect transmission, are made using the Slotted CSMA/CA.
- *Contention Free Period (CFP): T*he CFP starts immediately after the end of the CAP and ends at the end of the superframe. Transmissions are made by any device in the slot specifically allotted to it by the Coordinator, and hence are contention free. *Allocations/deallocations ar*e managed by the Coordinator.

Construction of the superframe is determined by two parameters: the *Beacon Order* (*BO*) and the *Superframe Order* (*SO*). These in turn determine the *Beacon Interval* (*BI*) and *Superframe Duration* (*SD).*

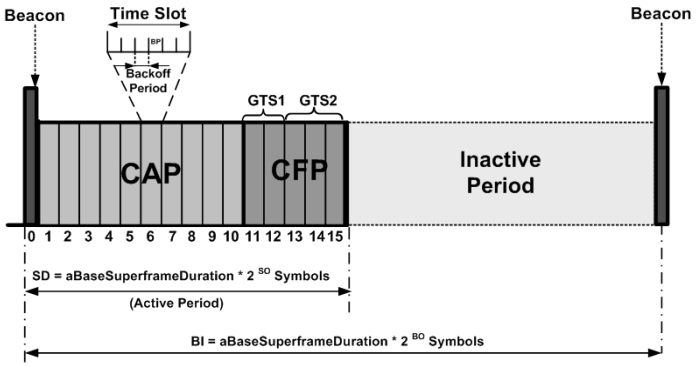


Figure 2.4: IEEE 802.15.4 Superframe structure [10]

The relationship between the two is given by the following two equations:

$$\left.\begin{aligned} BI &= aBaseSuperframeDuration \times 2^{BO} \\ SD &= aBaseSuperframeDuration \times 2^{SO} \end{aligned}\right\} for\ 0 \le SO \le BO \le 14 \qquad (2.1)$$

BI defines the time between two consecutive beacon frames whereas *SD* defines the active portion of the superframe, both in terms of *aBaseSuperframeDuration*, which is equal to 15.36 ms (assuming 250 kbps in the 2.4 GHz frequency band), also equal to the minimum duration of the superframe (corresponding to *SO*=0*).* An inactive period can be configured by setting *BO* > *SO*, in which all nodes may enter the sleep mode. This is useful for WSNs, since energy efficiency is often a factor.

Association Mechanisms

To communicate in a PAN, a device must be associated with a Coordinator. The association procedure begins with the requesting device sending an association request command frame. The Coordinator on receiving the request decides on whether to admit the device and generates the association response frame. For successful association, the response frame contains the short address to be assigned to the device and the Coordinator adds the new device in its neighbor table. For unsuccessful associations the response frame contains the problem status information. The response frame transmission is indirect (figure 2.5), which means that when the Coordinator has the response frame ready for transmission, it puts the recipient's address in the pending address field of the forthcoming beacon. The End device on receiving its address in the beacon

transmits a data request command frame, followed by the transmission of the association response by the coordinator. After a successful association, the associated device stores all the information about the new PAN by updating its MAC PAN Information Base (MAC PIB). The newly assigned short address is used for all future communication purposes. Figure 2.5 shows the message sequence chart for the association mechanism.

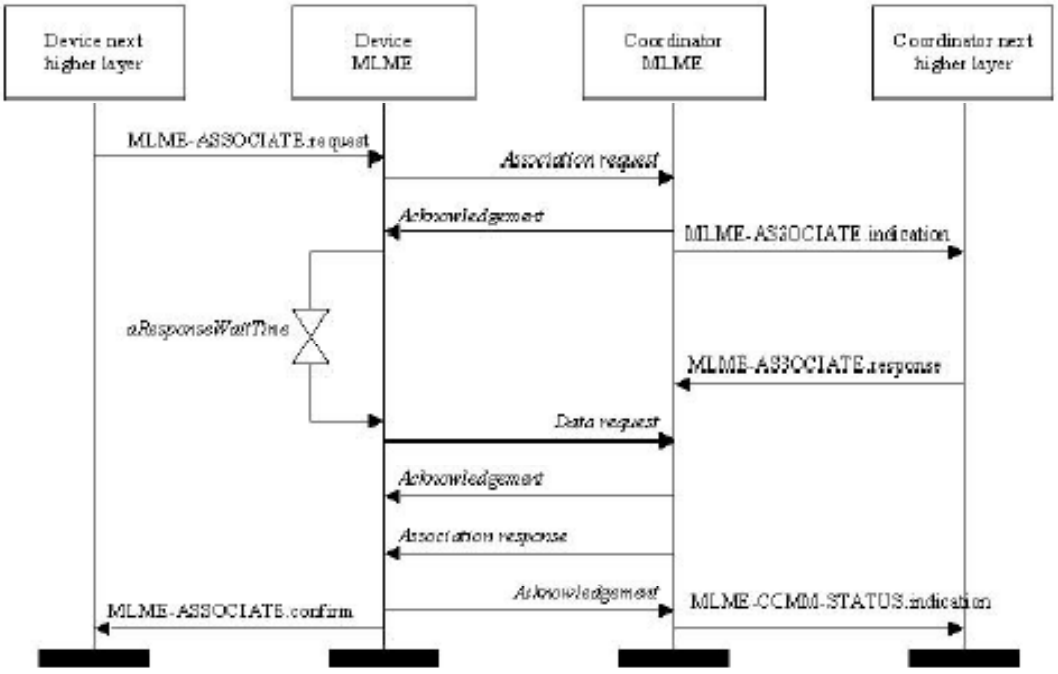


Figure 2.5: Association message sequence chart [10]

Guaranteed Time Slot (GTS) mechanism

The GTS mechanism allows devices to operate in the medium without contention by having portions of the superframe dedicated to a particular device, in which no other devices can operate. Slots are allocated by the Coordinator and can be used only for communications with the Coordinator. Each GTS may contain one or more time slots and up to seven GTSs may be allocated in any superframe. Each GTS slot can have only one direction: either from the device to the Coordinator (transmit) or from the Coordinator to the device (receive). Figure 2.6 shows message sequence chart of GTS allocation procedure.

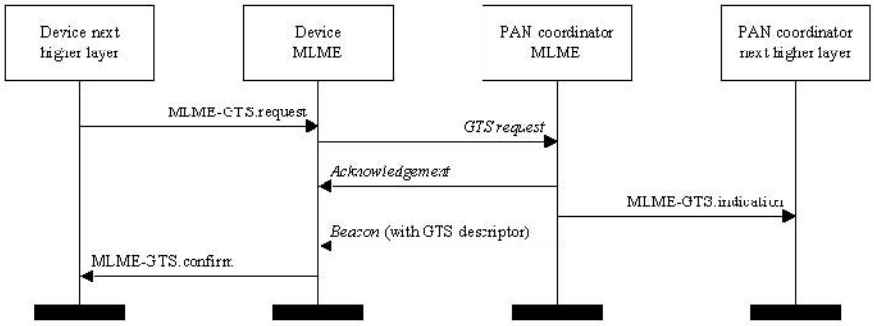


Figure 2.6: GTS allocation message sequence diagram [10]**Error! Reference source not found.**

The Coordinator is responsible for performing the GTS management and can deallocate the allocated slots at any time on its own discretion. The device that originally requested the GTS allocation can also request for dellocation. For each GTS, the Coordinator stores the starting slot, length, direction, and associated device address. Only one transmit and/or one receive GTS are allowed for each device. Upon the reception of the deallocation request the Coordinator updates the GTS descriptor list by removing the previous allocated slot and rearranging the remaining allocation starting slots.

The Coordinators monitor GTS activity and if there are no transmissions during a defined number of time slots the GTS allocation expires. The expiration occurs if no data or no acknowledgement frames are received by the device or by the Coordinator, on every *2*n* superframes, where *n* is defined as:

$$\begin{cases} n = 2^{(8-macBeaconO\ rder)}, if\ 0 \le macBeaconO\ rder \le 8 \\ n = 1, if\ 9 \le macBeaconO\ rder \le 14 \end{cases} \quad (2.2)$$

CSMA/CA Mechanism

In IEEE 802.15.4, contention-based MAC access can be governed by slotted or unslotted CSMA/CA, depending on the network operation behaviour: beacon-enabled or non beacon-enabled, respectively. The CSMA/CA mechanism is based on backoff periods (with the duration of 20 symbols). Three variables are used to schedule medium access:

- *Number of Backoffs* (*NB*), representing the number of failed attempts to access the medium;
- *Contention Window* (*CW*), representing the number of backoff periods that must be clear before starting transmission;
- *Backoff Exponent* (*BE*), enabling the computation of the number of wait backoffs before attempting to access the medium again.

Figure 2.7 shows a flowchart describing the slotted version of the CSMA/CA mechanism. It can be summarized in five steps:

1. Initialization of the algorithm variables: *NB* equal to 0; *CW* equals to 2 and *BE* is set to the minimum value between 2 and a MAC sub-layer constant (*macMinBE*);
2. After locating a backoff boundary, the algorithm waits for a random defined number of backoff periods before attempting to access the medium;
3. Clear Channel Assessment (CCA) to verify if the medium is idle or not.
4. The CCA returned a busy channel, thus *NB* is incremented by 1 and the algorithm must start again in Step 2;
5. The CCA returned an idle channel, *CW* is decremented by 1 and when it reaches 0 the message is transmitted, otherwise the algorithm jumps to Step 3.

In the slotted CSMA/CA, when the battery life extension is set to 0, the CSMA/CA must ensure that, after the random backoff (step 2), the remaining operations can be undertaken and the frame can be transmitted before the end of the CAP. If the number of backoff periods is greater than the remaining in the CAP, the MAC sub-layer pause the backoff countdown at the end of the CAP and defers it to the start of the next superframe. If the number of backoff periods is less or equal than the remaining number of backoff periods in the CAP, the MAC sub-layer applies the backoff delay and re-evaluate whether it can proceed with the frame transmission. If the MAC sub-layer do not have enough time, it defers until the start of the next superframe, continuing with the two CCA evaluations (step 3). If the battery life extension is 1, the backoff countdown must only occur during the first six full backoff periods, after the reception of the beacon, as the frame transmission must start in one of these backoff periods.

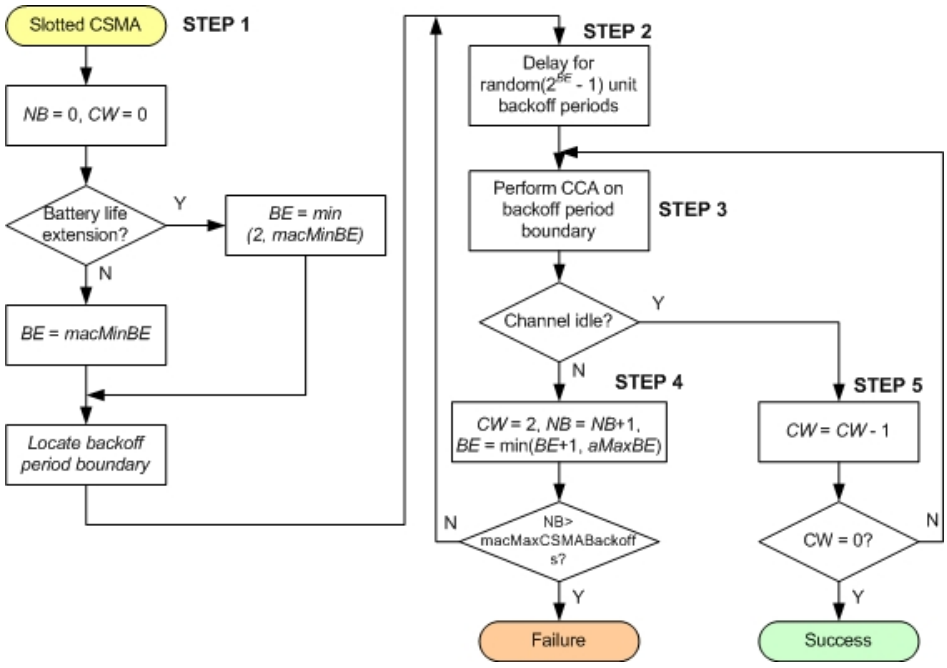


Figure 2.7: The Slotted CSMA/CA mechanism [10]

Transmission Scenarios

The IEEE 802.15.4 standard enables three different types of transmissions:

- *Direct transmissions*: frames are transmitted to the medium without any channel assessment. Used in the transmission of the beacon frames, the acknowledgment frames and the frames in the GTS time slots.
- *Indirect transmissions*: the frames are stored in the Coordinator to which the destination device is associated. The information about the pending transmission is then added to the pending addresses field of the beacon frame. The device having pending data in the Coordinator can then request it by sending a data request command frame and the stored frame is transmitted by the Coordinator.
- *Normal transmissions*: the frames are transmitted to the medium with contention, by applying the CSMA/CA algorithm. Applied to the data frames and command frames transmitted during the CAP.

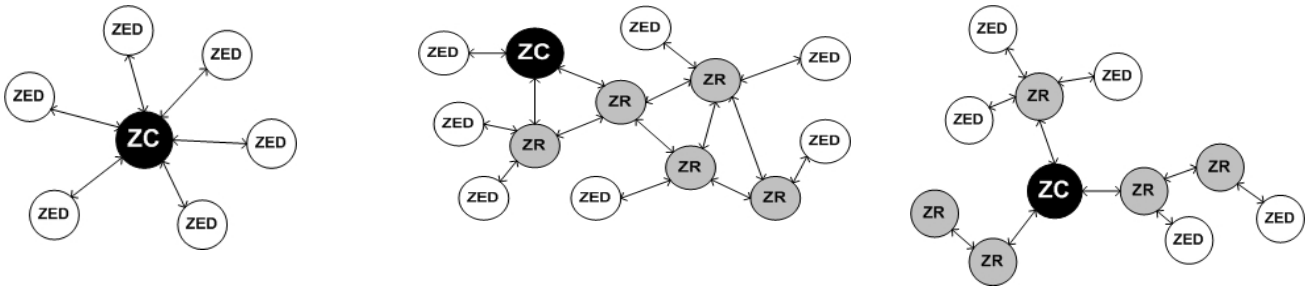
2.3 ZigBee

2. 3.1 Topology and Device Types

ZigBee defines 3 types of devices [11]:

- *ZigBee Coordinator* (ZC): Each ZigBee Network has one ZC which initiates and configures network formation and also acts as an IEEE 802.15.4 Personal Area Network (PAN) Coordinator. It must be a Full Functional Device (FFD)
- *ZigBee Router* (ZR): ZR participates in multi-hop routing of messages in mesh and Cluster-Tree networks. It must associate with a ZC or with a previously associated ZR in Cluster-Tree topologies. It also acts as an IEEE 802.15.4 PAN Coordinator and has to be a Full Functional Device (FFD)
- *ZigBee End Device* (ZED): ZED does not allow other devices to associate with it and does not participate in routing. It can be a Reduced Function Device (RFD)

Three network topologies are supported: star, mesh and cluster-tree; as shown in Figure 2.8.



a) star topology b) mesh topology c) cluster-tree topology

Figure 2.8: ZigBee network topologies

In the star topology (Figure 2.8 a), a single node starts the network, operating as a ZC. It chooses a unique PAN identifier (not being used by any other ZigBee network in the range). The communication paradigm of the star topology is centralized, i.e. each device must send its data to the ZC first, which then transmits it to the destination node. In mesh topology (Figure 2.8 b) also the communication is decentralized, i.e. each node can directly communicate with any other node

within its radio range. Thus mesh topology enables networking flexibility, but at the cost of additional complexity. The cluster-tree network topology (Figure 2.8 c) is a special case of a mesh network with a single routing path between any pair of nodes. The ZC identifies the entire network and every cluster is managed by a separate ZR. In beacon enabled mode the ZRs must provide synchronization to the nodes in its cluster while avoiding collision with other clusters.

2.3.2 ZigBee Network Layer

The ZigBee Network Layer supports two service entities: The Network Layer Data Entity (NLDE) and Network layer management entity (NLME). NLDE-SAP provides services specific to data transmission over the network whereas the NLME-SAP provides network management services and maintenance of Network Information Base (NIB), as shown in figure 2.9.

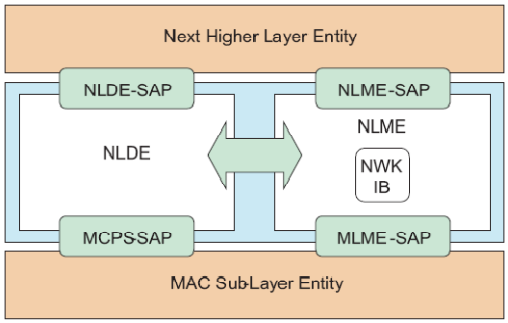


Figure 2.9: Network Layer reference model **Error! Reference source not found.**

According to the standard, NLDE should provide the following services [11]:

- *Generation of the Network level PDU (NPDU):* The ability to generate an NPDU from an application layer PDU through the addition of an network protocol header.
- *Topology specific routing:* The ability of transmitting an NPDU to the next device on the route to the final destination.

NLME is responsible for:

- *Configuring a new device*: The ability to sufficiently configure the stack for operation as required. Configuration options include beginning operation as a ZigBee coordinator or joining an existing network.

- *Starting a network:* The ability to establish a new network.
- *Joining and leaving a network:* The ability to join or leave a network as well as the ability for a ZigBee coordinator or ZigBee router to request a device to leave the network.
- *Addressing:* The ability of ZigBee coordinators and routers to assign addresses to devices joining the network.
- *Neighbor discovery:* The ability to discover, record and report information pertaining to the one-hop neighbors of a device
- *Route discovery:* The ability to discover and record paths through the network whereby messages may be efficiently routed.
- *Reception control:* The ability for a device to control when the receiver is activated and for how long, enabling MAC sub-layer synchronization or direct reception.

The ZigBee Coordinator also defines some additional network parameters e.g. the maximum number of children (C_m) any device is allowed to have, maximum number (R_m) router-capable devices. Every device has an associated depth, representing the number of hops a transmitted frame must travel from itself to reach the ZigBee Coordinator. The ZC has a depth of 0, while its children have a depth of 1. The ZC also determines the maximum depth (L_m) of the network. The maximum number of children, routers and network depth are used for calculating the addresses of the devices in the network, in a distributed address scheme [101].

Short Address Assignment

A parent device uses the values of C_m, R_m, and L_m to determine the sizes of the address sub-blocks distributed by each parent, calculated using *Cskip* function applied on the depth (d) of the network. For a given network depth *d*, *Cskip(d)* is calculated as follows[11, 102]:

$$Cskip(d) = \begin{cases} 1 + Cm \cdot (Lm - d - 1), & if\ Rm = 1 \\ \dfrac{1 + Cm - Rm - Cm \cdot Rm^{Lm-d-1}}{1 - Rm}, & Otherwise \end{cases} \qquad (2.3)$$

A parent device that has a *Cskip(d)* value of zero is not capable of accepting children and must be treated as an end device. A parent device that has a *Cskip(d)* value greater that zero must accept devices and assign addresses if possible. A parent device assigns an address that is one greater than its own to the first router that associates. The next router receives an address that is

separated by the return value of the *Cskip(parent depth)* function. The maximum number of associated routers is defined by the network parameter *nwkMaxRouters* (R_m).

Considering a parent node with a depth *d* and an address of A_{parent}, the number of child devices *n* is between 1 and C_m-R_m.

$$1 \le n \le (C_m - R_m) \tag{2.4}$$

The A_{child} address of the nth child router is calculated according to Eq. 2.5(*n* is the number of child routers):

$$\begin{aligned} A_{child} &= A_{parent} + (n-1) \times Cskip\,(d) + 1, n = 1 \\ A_{child} &= A_{parent} + (n-1) \times Cskip\,(d), n > 1 \end{aligned} \tag{2.5}$$

The A_{child} address of the nth child end device is calculated according to Eq. 2.6 (*n* is the number of child end devices):

$$A_{child} = A_{parent} + Rm \times Cskip(d) + n \tag{2.6}$$

Figure 2.10 depicts an example of an address assignment scheme. The parameters used in the address assignment are the following: maximum depth (L_m) = 3, maximum children (C_m) = 6 and maximum routers (R_m) = 4.

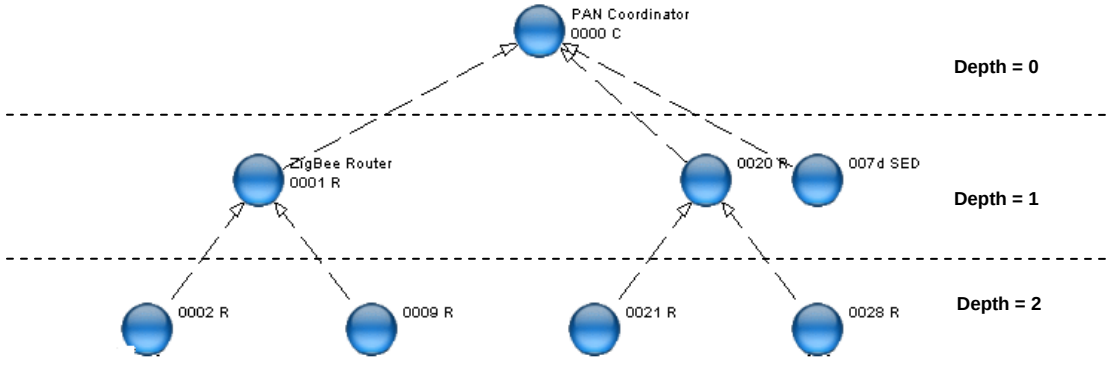


Figure 2.10: Address assignment scheme example [36]

Figure 2.11 shows the ZigBee Coordinator (0x0000) available addressing scheme. Considering the above network parameters, the ZigBee Coordinator is allowed to associate up to A4 routers and 2 end devices in its available address pool. On the other hand, the ZR (0x0020) is allowed to associate up to 4 ZRs and 6 ZEDs.

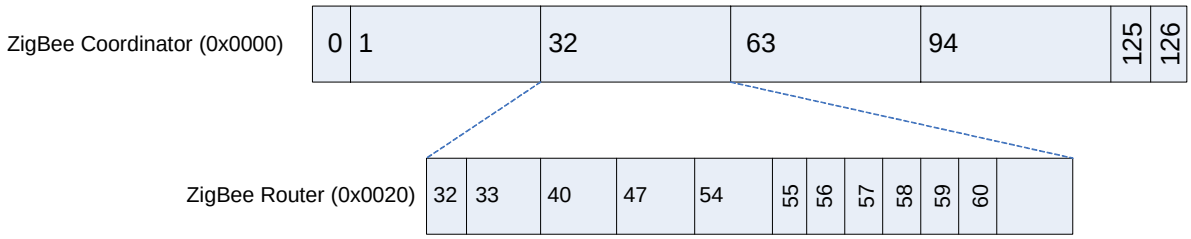


Figure 2.11: ZigBee Coordinator addressing scheme [36]

Tree-Routing

This routing mechanism is based on the short addressing scheme and was initially proposed by MOTOROLA [20]. Each device, upon the reception of a data frame, reads the routing information fields and checks the destination address. If the destination is a child of the device (neighbour table check), the device relays the packet to the appropriate address. If the destination address is not a child, the device must check if the address is a descendent using the condition in 2.7, where *A* is device network address, *D* the destination address and *d* the device depth in the network.

$$A < D < A + Cskip(d-1) \tag{2.7}$$

The next hop (*N*) address when routing down is given by:

$$N = A + 1 + \left\lfloor \frac{D-(A+1)}{Cskip(d)} \right\rfloor \times Cskip(d) \tag{2.8}$$

If the destination address is not a descendant, the device relays the packet to its parent and so on.

CHAPTER 3
TECHNOLOGICAL PLATFORMS AND DEVELOPMENT TOOLS

This chapter introduces the hardware and software platforms: the FLEX boards [21], the ERIKA real-time OS [6]; and the tools used for development, debugging and analysis: RT-Druit [22], the MPLAB In-circuit Debugger (ICD) [23], and the Chipcon packet sniffer [24]. The Open-ZB implementation [25] of IEEE 802.15.4/ZigBee protocol over TinyOS is described in brief in the end.

3.1 The FLEX Board

FLEX embedded development board was the basic hardware used on which ERIKA was installed. Embedded with a Michrochip dsPIC microcontroller, the FLEX is able to support real-time kernels.

Its main features are:

- DsPIC33FJ256MC710 Microcontroller with 40 MHz frequency [26];
- Flexipanel EASYBEE IEEE 802.15.4 Transceiver module [27];
- 256 KB of Programmable flash memory ;
- Modular hardware architecture ;
- In-circuit programmer connectors;
- Support of the ERIKA real-time kernel, provided by Evidence Srl [28];

The FLEX device can be configured by mounting various components on the Base Board. In our case, it mounts a Microchip dsPIC micro-controller, and exports almost all the pins of the micro-controller. As depicted in Figure 3.1, several daughter boards can be connected in piggyback to the Flex Base Board. The daughter boards can have different features and they can be easily combined to obtain complex devices.

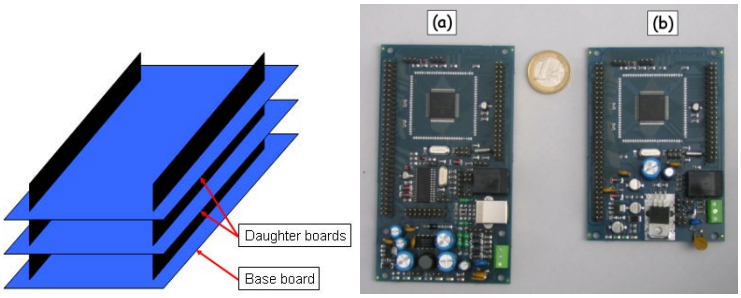


Figure 3.1: The FLEX programming board [21]

3.2 ERIKA

Erika is a multi-processor kernel architecture, running a real-time scheduler and resource managers, thus allowing predictable timing behavior of the tasks. It implements a number of Application Programming Interfaces (APIs), closely matching the OSEK/VDX standard [29] for automotive embedded controllers. It supports several microcontrollers including the Microchip dsPIC used in the FLEX board.

Erika provides support for,

- Four OSEK conformance classes to match different application requirements
- Preemptive and non-preemptive multitasking
- Fixed priority scheduling
- Shared resources, including stack
- Periodic activations of tasks using alarms
- Centralized error handling

The kernel provides a minimal set of primitives which can be used to implement a multithreaded environment. It supports OIL (OSEK Implementation Language) as a standard configuration language, used for the static definition of the RTOS objects which are instantiated and used by the applications. This can be used to configure tasks to match the requirements of real-time applications.

Tasks in ERIKA are scheduled according to statically assigned priorities, and share resources using Immediate Priority Ceiling protocol. Interrupts can always preempt running tasks to execute operations required by the peripherals.

3.3 RT-DRUIT

RT-Druid is the Eclipse-based development environment for the ERIKA RTOS, used to write, compile, and analyze an application. RT-Druid is composed by a set of plug-ins for the Eclipse Framework [30]. The RT-Druid Core plug-in contains all the internal metamodel representation, providing a common infrastructure for the other plug-ins, together with ANT scripting support.

The RT-Druid Code Generator plug-in implements the OIL file editor and configurator (for more detail on OSEK/VDX and OIL, see [29]), together with target independent code generation routines for ERIKA. The RT-Druid Schedulability Analysis plug-in provides the Schedulability Analysis framework, implementing algorithms like scheduling acceptance tests, sensitivity analysis, task offset calculation; and provides a set of tools for modelling, analyzing, and simulating the timing behaviour of embedded real-time systems.

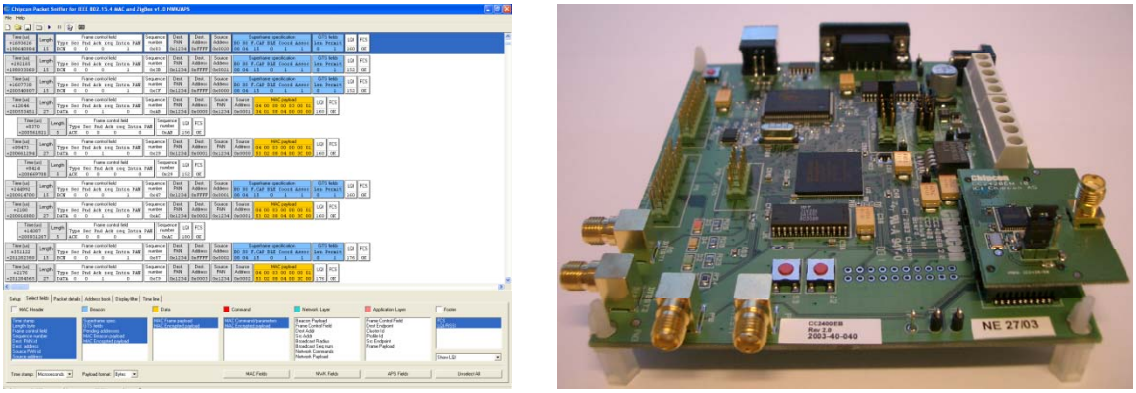
3.4 Microchip MPLAB ICD

The MPLAB In-Circuit Debugger is the hardware debugger/programmer for Microchip Flash Digital Signal Controller (DSC) and microcontroller (MCU) devices. It provides MPLAB Integrated Development Environment (IDE), with a graphical user interface to debug and program PIC Flash microcontrollers and dsPIC DSCs .The ICD probe is connected to the PC containing the program using a high-speed USB 2.0 interface and is connected to the target with a connector compatible with the MPLAB ICD 2. Program or Debug mode is chosen, as required, and the binary file is loaded to the target device.

Figure 3.2: The MPLAB In-Circuit debugger [31]

3.5 IEEE 802.15.4/ZigBee Protocol Analysers

The IEEE 802.15.4 compliant Chipcon CC2420 Packet Sniffer has been used throughout the development process for validation and debugging purposes by interpreting the packets being transmitted through the channels. It was also used for performance analysis purposes where sniffer output files recording packets transmitted during various experiments were parsed to measure throughput and average delays.



a) Snapshot of the sniffer application b) CC2420 EB with a CC2420EM

Figure 3.3: Chipcon IEEE802.15.4/ZigBee packet sniffer [24]

Figure 3.3a shows a snapshot of the sniffer application. It provides:

- Raw list of the received packets with timestamp information
- Interpretation of the packets information, highlighting the different packet fields
- Packet fields filtering
- Device list

Chipcon also provides a tool used to test the transceivers by allowing viewing and interacting with the CC2420 transceiver memory registries.

3.6 Open-ZB TinyOS Protocol Stack

The Open-ZB stack implementation [13] includes IEEE 802.15.4 Data Link Layer and a part of the ZigBee Network Layer. The ERIKA implementation of the protocol stack is loosely based on this implementation, with some basic changes and enhancements (more detail in chapter 4).

The Open-ZB stack has been implemented for MicaZ [31] as well as TelosB [32] motes, and has three main blocks: (1) the hardware abstraction layer, containing the IEEE 802.15.4 physical layer and the timer modules (2) the IEEE 802.15.4 MAC sub-layer; and (3) the ZigBee Network Layer. The IEEE 802.15.4 implementation includes the slotted CSMA/CA implementation, the different types of transmission scenarios (direct, indirect and GTS transmissions), association of the devices, channel scans and beacon management.

The Network layer supports data transfer between the Network Layer and the MAC sub-layer, the association mechanisms and the network topology management (e.g. cluster-tree support by the ZigBee Addressing schemes) and routing (e.g. neighbour routing and tree-routing). Security is not supported.

The difficulties encountered in implementation of the stack as well as observations based on performance evaluations are listed in [3]. There were hardware related limitations e.g. memory constraints, transceiver limitations and problems with the consistency and accuracy of timers; however, the biggest and most important of these was considered to arise because of the nature of the TinyOS task scheduler. TinyOS does not support tasks prioritization and the scheduler is non pre-emptive. The tasks invoked by various events are posted to the queue and are processed in FIFO order. This significantly impacts the behavior of the protocol stack, as sharing the microcontroller between all protocols tasks is very demanding, specially for high duty cycles, and there is no way to guarantee execution of critical tasks on time. For example, processing and transmitting the beacon frame is essential for the network stability, and should take precedence over other tasks. This doesn’t happen when a FIFO scheduler is used and under heavy load, beacon frames may be delayed in transmission, processing, or lost in both cases. This results in the loss of synchronization. This has been one of the major motivations to implement the stack on ERIKA, which supports task prioritization and preemption.

CHAPTER 4
Protocol Stack Implementation

This chapter summarizes the implementation details, highlighting some of the most important features of the IEEE802.15.4/ZigBee stack implementation on ERIKA and its software architecture. Section 4.1 describes the implementation architecture, including a brief description of the system components and customized libraries provided by the RETIS Lab [18] to support the stack implementation. Section 4.2 shows the OS configuration and describes the tasks and alarms created to support the implementation. The implementation of IEEE802.154 protocol functionalities is described in section 4.3 and the ZigBee implementation is outlined in section 4.4. Section 4.5 describes the TDBS implementation.

4.1 Implementation Architecture

4.1.1 System Overview

The implementation follows a layered architecture. Each layer makes use of the services provided by the lower layers and provides services to the upper layers [33]. Figure 4.1 shows various elements of the stack, including the system components and their interactions. Table 4.1 summarizes services provided by each of these components.

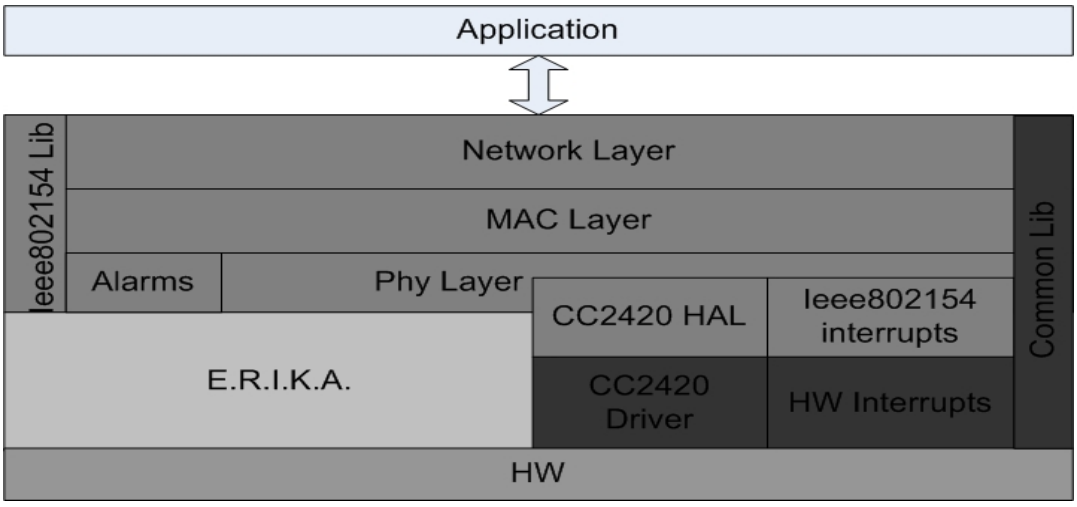


Figure 4.1: Protocol stack layered architecture

Layer		Description
HW		Represents the hardware components used: the FLEX development board, the dspic33F microcontroller, and the CC2420 transceiver.
HW Interrupts		Component handling all the hardware interrupts. Implements interrupt service routines (ISRs).
CC2420 Drivers		Provides an abstraction for the upper layers to use the CC2420 transceiver. The transceiver conforms to the IEEE 802.15.4, providing most of the functions required to implement the protocol.
ERIKA		The Operating System: manages hardware; provides task management, resource management and timer management services.
CC2420 HAL		Provides an additional layer of abstraction over the CC2420 driver, enabling the Physical and MAC layers to communicate using format specific to the communication protocol.
Alarms		Provide software abstraction for the timers. Used to activate periodic tasks required for handling various activities of the slotted mode.
Common Lib		Provides common utilities such as printing data on the console, dynamic memory management, and the basic data structure implementations.
Ieee802154 Lib	phy	Implements IEEE802.15.4 PD-SAP and PLME-SAP primitives.
	mac	Implements IEEE802.15.4 MCPS-SAP and MLME-SAP primitives.
	nwl	Implements ZigBee Network Layer management, addressing and routing primitives.
Apl		Used for implementing Test applications.

Table 4.1: Services provided by various components

The ieee802154 Lib implements the communication protocols. It includes the physical and MAC layers of IEEE 802.15.4 and basic network layer functions of ZigBee. These layers are concerned only with the higher level implementation details specific to the communication

protocol; lower level details related to hardware, timers, interrupts and memory management are handled by the underlying layers, as described in Table 4.1.

4.1.2 File System Architecture

Figure 4.2 shows the implementation file system architecture. The source code is located in the */contrib/ieee802154/libscrc* directory while corresponding header files are placed in */contrib/ieee802154/inc*. The *inc* directory also contains additional header files of constant declarations (*mac_const.h*) and enumerations (e.g. *mac_enumerations.h*). The entire file system is placed in the root ERIKA installation directory.

The *common* subdirectory (Figure 4.2) implements general utilities, including modules to control the cc2420 transceiver (*cc2420.c*), access console and serial ports (*console.c*, *ee_radio_spi.c*, *eeuart.c*), and memory management functions (*sralloc.c*, *netbuff.c*). The circular queue implementation (*cqueue.c*) needed to support transmission buffers is also placed in this directory. The *hal* directory contains *hal_cc2420.c* and *hal_interrupts.c,* implementing modules to read and write from transceiver, and interrupt service routines respectively.

The IEEE 802.15.4 and ZigBee implementation files are placed in *phy*, *mac* and *nwl* directories. The *phy* directory (Figure 4.2) contains Physical Layer Data Service (*PD_DATA.c*) as well as Physical Layer Management Services (*PLME_CCA.c, PLME_ED.c, PLME_GET.c, PLME_SET_TRX_STATE.c, PLME_SET.c*) implementation. *Phy.c* implements physical layer initialization modules. Similarly, the MAC layer implementation files are placed in the *mac* directory, which contains files implementing MCPS Data Component (*MCPS_DATA.c*) as well as Mac Layer Management Services (*MLME_ASSOCIATE.c, MLME_GET.c, MLME_GTS.c, MLME_SET.c, MLME_START.c*). *Mac.c* contains initialization modules, functions implementing CSMA/CA, beacon management functions, and the functions to process received data and command frames. The *mac_func.c* file implements auxiliary utility function. The ZigBee network layer functions have been implemented in Nwl.c.

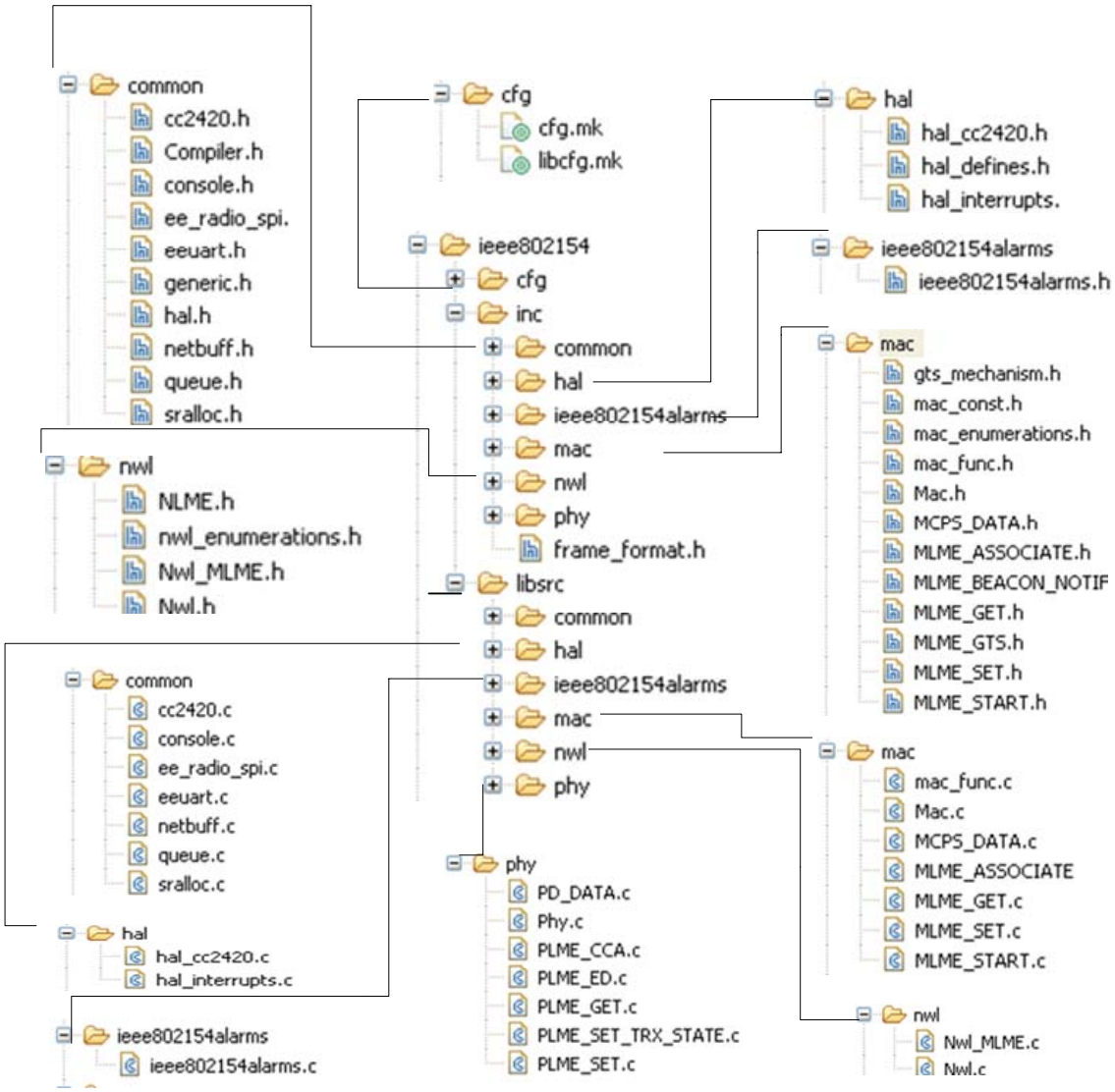


Figure 4.2: Implementation file system architecture

4.2 Configuring ERIKA

4.2.1 OS Configurations using OSEK

Erika supports a reduced OSEK/VDX API [34], providing support for real-time thread activation, mutual exclusion, alarms, and counting semaphores. Objects can be declared and configured using OSEK Implementation Language (Oil), also used to assign task priorities and specify scheduling policy. Code Example 4.1 shows a snapshot of the *conf.oil* file, used to configure system components for the stack.

```
CPU mySystem {

	OS myOs {
		EE_OPT = "DEBUG";
		EE_OPT = "__HAS_TYPES_H__";
		EE_OPT = "__ADD_LIBS__";
		LIB = ENABLE { NAME = "ieee802154"; };

		CFLAGS = "-DDEVICE_TYPE_COORDINATOR";  //DEVICE TYPE
COORDIANTOR
		//CFLAGS = "-DDEVICE_TYPE_END_DEVICE"; //DEVICE TYPE END
DEVICE

		//---------------------------------//

		CPU_DATA = PIC30 {
			APP_SRC = "code.c";
			MULTI_STACK = FALSE;
			ICD2 = TRUE;
		};

		MCU_DATA = PIC30 {
			MODEL = PIC33FJ256MC710;
		};

		BOARD_DATA = EE_FLEX {
			USELEDS = TRUE;
		};

		KERNEL_TYPE = FP;

	};

	COUNTER myCounter;
};
```

Code Example 4.1: Configuration of System Components

This file defines and configures the Oil objects and resources, including the programming board, Micro Controller Unit, the OS, Counters, Tasks, and Alarms. It also defines the operating system and its properties. It also selects the device type for the node, which can be either Coordinator, Router or End Device.

Figure 4.3 shows the compilation procedure generating the executable file. The system generator translates the configuration file into *C* code which is then used by the compiler along with the application source code to generate the object code.

The executable is finally loaded to the board using MPLAB In-Circuit Debugger.

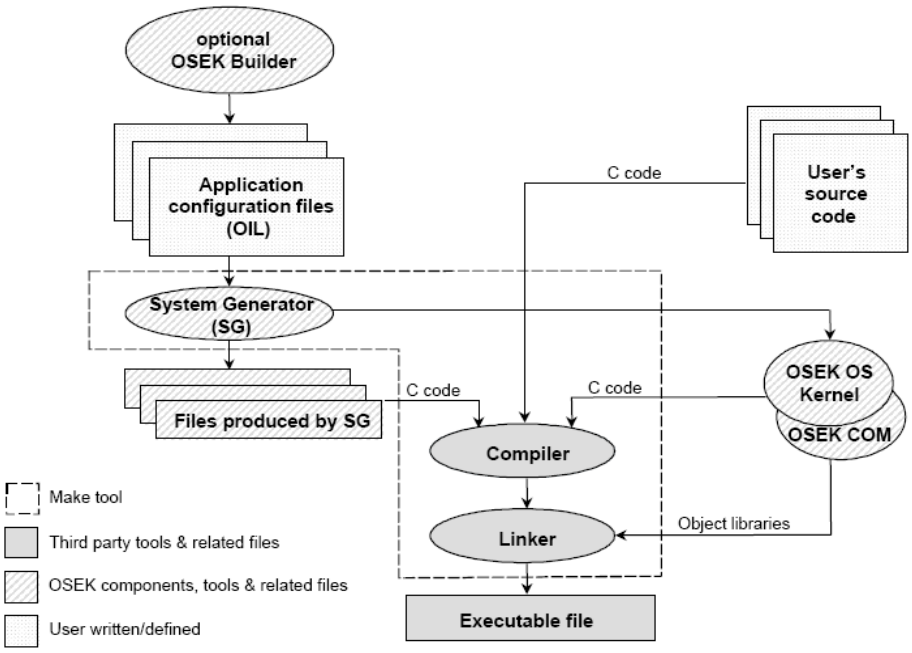


Figure 4.3: Compilation of an application in ERIKA [34]

4.2.2 Task Creation and Alarms

The *conf.oil* file also defines the tasks needed for the stack implementation. Code Example 4.2 shows syntax of Task configuration.

```
TASK TaskName {
      PRIORITY = …;
      STACK = …;
      SCHEDULE = …;
};
```

Code Example 4.2: Task declaration format

The *PRIORITY* attribute defines the priority of a task. It is used in the scheduling of tasks. *SCHEDULE* defines the preemptiveness, with the enum variable "FULL" declaring a preemptable task and "NON" declaring a non-preemptable task. The stack is specified to be shared among all tasks by setting the *STACK* attribute to SHARED for every task. Figure 4.4 shows the portion of the *conf.oil* file declaring tasks. Tables 4.1 and 4.2 describe the communication services each of these tasks are used for.

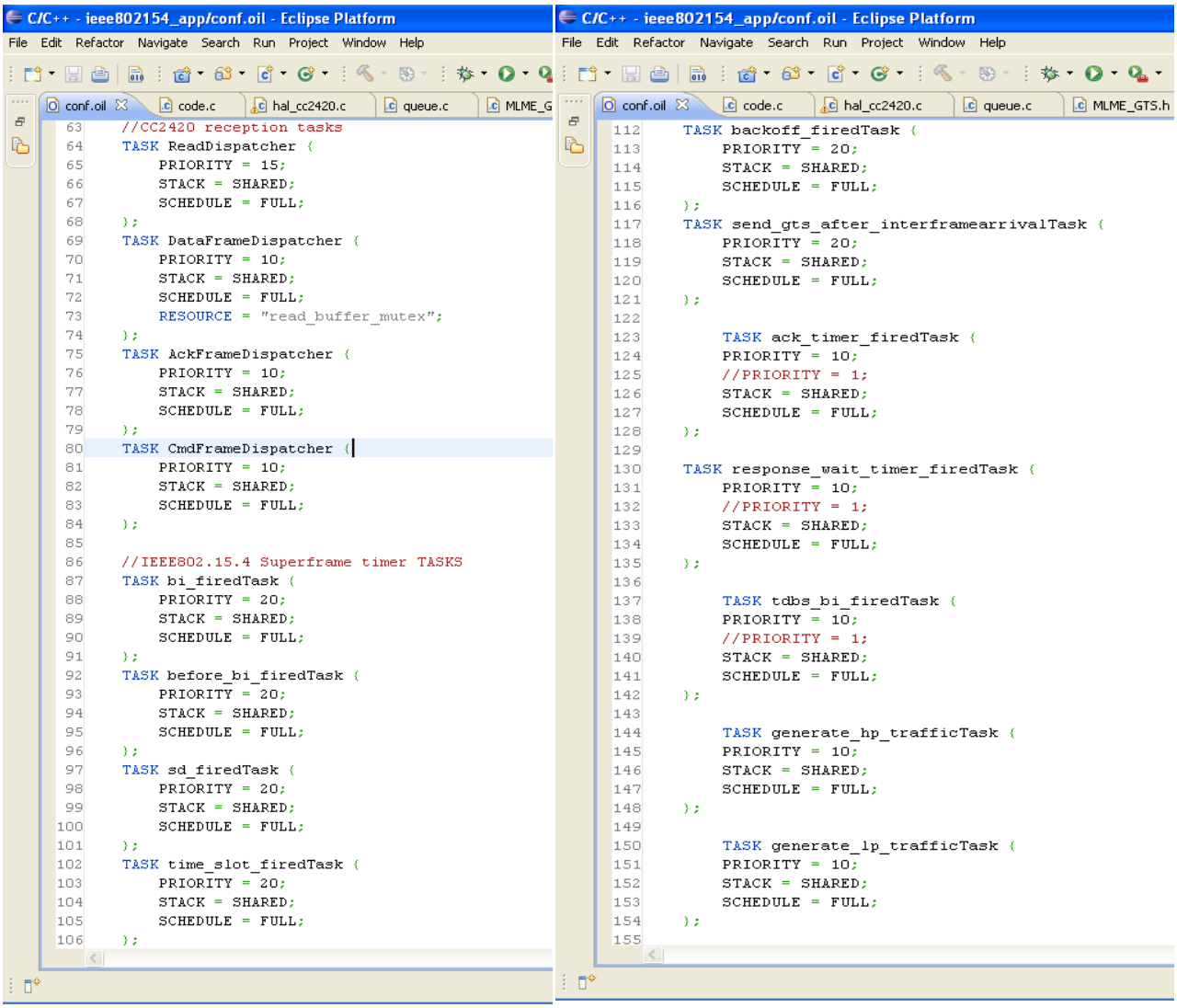


Figure 4.4: Snapshot of conf.oil showing task configurations

Table 4.1 describes the C2420 reception tasks. The *ReadDispatcher* task is activated when there is an interrupt from the transceiver indicating reception of a frame. *ReadDispatcher* calls the *read_Data* function, which retrieves the frame from the transceiver memory. Subsequently, it checks the frame type of the received packet by parsing the data type field and posts one of the *DataFrameDispatcher*, *AckFrameDispatcher* or *CmdFrameDispatcher* tasks depending on whether the frame received is of data, acknowledgement or command type. *DataFrameDispatcher*, *AckFrameDispatcher* and *CmdFrameDispatcher*, upon being activated, call *process_data()*; *process_ack()* and *process_cmd()* functions respectively. The functions process the received frame with further processing depending on frame type.

Task Name	Description	Associated Alarms	Period
ReadDispatcher	Task posted upon the reception of the FIFO ISR	N/A	N/A
DataFrameDispatcher	Task posted in the ReadDispatcher to process data frames	N/A	N/A
AckFrameDispatcher	Task posted in the ReadDispatcher to process acknowledgment frames	N/A	N/A
CmdFrameDispatcher	Task posted in the ReadDispatcher to process command frames	N/A	N/A

Table 4.1: CC2420 reception tasks

In case the received frame is a beacon, it is processed directly (instead of posting a task) with a call to *process_beacon* function. This is achieved by having a separate memory buffer for beacon frames, as distinct from the send and receives buffers. Since these memory locations are shared by different modules and require mechanisms for mutual exclusion and synchronization (using ERIKA "resources"), having separate buffer for beacons enables on-time processing of beacon frames.

Table 4.2 describes the tasks used for the creation of superframe. Many of these tasks are periodic with fixed inter arrival rates, with periods depending on the selection of the beacon order and the superframe order (except for *backoff_firedTask*). The *backoff_firedTask* has a has a period of 320us, equivalent to 20 symbol duration in a 2.4 Ghz band. Periodic tasks are activated using alarms, which can be set to fire periodically at specified intervals. The associated alarms with the periodic superframe creation Tasks are shown in Table 4.2.

Task Name	Associated Alarms	Description	Period
backoff_firedTask	backoff_firedAlarm	Fired on every backoff. Used to implement the slotted CSMA/CA	320 us
before_bi_firedTask	Before_bi_firedAlarm	Fired before every beacon interval to switch the transceiver on RX (non coordinator/router) or TX (coordinator/router)	BEFORE_BI_TICKS – Beacon Interval
bi_fired_Task	bi_fired Alarm	Fired on every beacon interval	Beacon Interval
sd_firedTask	sd_fired Alarm	Fired at the end of the superframe	N/A
before_time_slot_fired Task	Before_time_slot_fired Alarm	Fired before each time slot to set the transceiver to RX or TX during the GTS period	N/A
time_slot_firedTask	time_slot_fired Alarm	Fired on every time slot of the superframe	Superframe Duration / 16

Table 4.2: IEEE802.15.4 Superframe time-triggered TASKS

The alarms use counter *mycounter*, whose granularity is set to 320 microseconds, equal to the backoff period in the 2.4 GHz band. The pattern of correspondence between alarms and tasks is that of <alarm_name>Alarm calling the task <alarm_name>Task. The *ieee802154alarms* component implements the necessary functions to configure and manage the alarms.

Code Example 4.3 shows the definition of *backoff_firedAlarm*. All other alarms have been similarly defined in correspondence with the respective tasks.

```
ALARM backoff_firedAlarm {
        COUNTER = "myCounter";
        ACTION = ACTIVATETASK { TASK = "backoff_firedTask"; };
};
```

Code Example 4.3: Example of an Alarm Definition

The activation of each of the tasks, with the exception of the dispatcher tasks (*DataFrameDispatcher*, *AckFrameDispatcher*, *CmdFrameDispatcher*), is controlled by having an explicit alarm. These alarms activate the tasks depending on the selected cycle value parameters of the alarm.

4.3 IEEE 802.15.4 Implementation

The implementation is broadly based on the open-ZB implementation of IEEE 802.15.4 in nesC/TinyOS [35] with some important differences and optimizations. One of the basic differences is in the timer abstraction provided to the upper layers. While Open-ZB implements a centralized timer component to handle all timer dependent events, Erika implementation uses independent alarms for each task. Queuing and Buffering mechanisms are also different and have been improved in the Erika implementation. While TinyOS implementation uses global arrays and variables to maintain buffers, Erika provides a more sophisticated and efficient mechanism by implementing a circular queue for all buffering purposes.

Changes have also been made in the implementation of higher level protocol features. The association mechanism uses the indirect transmissions, as specified by the standard. GTS implementation has been optimized by implementing a dynamic reshuffling of available GTS slots with deallocations. Minor changes have also been made in the slotted CSMA/CA implementation. The following sections describe some the important aspects of the implementation.

4.3.1 Superframe Creation

The superframe creation uses the following 6 Alarms (also see Figure 4.5), to mark critical points in the active portion of the superframe:

- *bi_firedAlarm* – Marks the start of a beacon interval. Period depends on the BI parameter.

- *before_bi_firedAlarm* – Activated before the *bi_firedAlarm* in order to enable and set the transceiver in RX mode for End Devices or TX in the case of the Coordinator. The offset parameter, relative to the *bi_firedAlarm*, is defined by the *BEFORE_BI_INTERVAL* constant.
- *sd_firedAlarm* – Marks the end of the active portion of the superframe. Period depends on the SD parameter.
- *time_slot_firedAlarm* – Alarm to mark time slots. Period equals to the time slot duration, which in turn depends on the parameter *SD*.
- *before_time_slot_firedAlarm* – Similar to the *before_bi_firedAlarm*, this alarm is used to activate the transceiver before the beginning of each time slot. Used for the implementation of the GTS mechanism. The offset parameter, related to the *time_slot_firedAlarm*, is defined by the constant *BEFORE_TS_INTERVAL*.
- *backoff_firedAlarm* – Fires on every timer tick. The period of this alarm is 320 us as it is defined by the IEEE 802.15.4 standard. The *backoff_firedAlarm* is used to implement the slotted CSMA/CA.

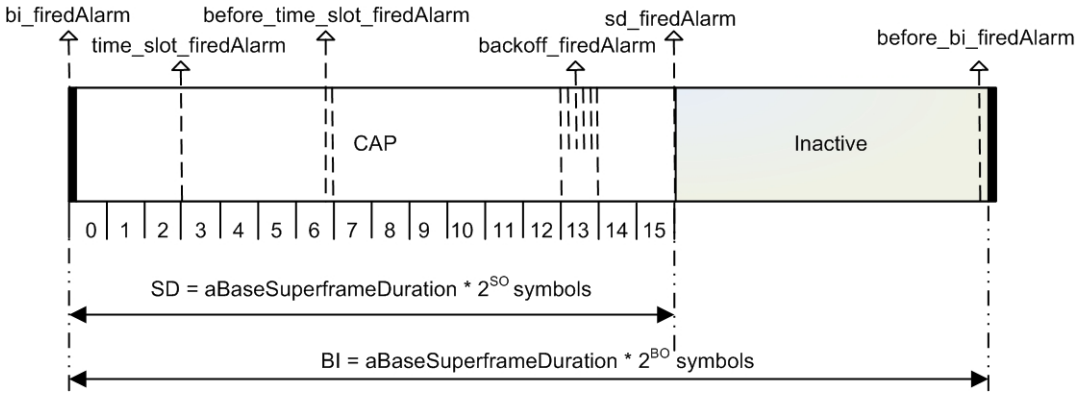


Figure 4.5: Superframe structure alarms

These alarms use the hardware counter, with a granularity of 320 us per tick, equal to one backoff duration. The ieee802154alarms component implements the necessary functions to configure the fire period of the alarms given a certain superframe configuration (*BI* and *SD*). The priorities associated with the tasks posted by the superframe alarms are highest to ensure that all nodes remain synchronized during the whole superframe.

4.3.2 Frame Construction

The MAC frame format, as defined by the standard, is shown in Figure 4.6. It is composed of a Mac Header (MHR), a MAC payload and a MAC Footer (MFR). The fields of MHR appear in a fixed order, but the addressing fields can be of zero (source device being coordinator) or variable (short or long address) length.

<table>
<tr><td>Octets: 2</td><td>1</td><td>0/2</td><td>0/2/8</td><td>0/2</td><td>0/2/8</td><td>variable</td><td>2</td></tr>
<tr><td rowspan="2">Frame control</td><td rowspan="2">Sequence number</td><td>Destination PAN identifier</td><td>Destination address</td><td>Source PAN identifier</td><td>Source address</td><td rowspan="2">Frame payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td></tr>
<tr><td colspan="6">MHR</td><td>MAC payload</td><td>MFR</td></tr>
</table>

Figure 4.6: General MAC frame format [10]

The frame control field (Figure 4.7) is of 16 bits and contains information defining the frame type, addressing fields and other control flags. The frame thus depends on the type of the frame, and other type-dependent parameters.

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame type	Security enabled	Frame pending	Ack. request	Intra-PAN	Reserved	Dest. addressing mode	Reserved	Source addressing mode

Figure 4.7: Frame control field format [10]

To deal with this variety of combinations, the strategy used was to adopt a generic frame format structure. This was defined as MPDU, and has the format as shown in Code Example 4.4

```
typedef struct MPDU
{
      EE_UINT8* length;
      EE_UINT8 frame_control1;
      EE_UINT8 frame_control2;
      EE_UINT8 seq_num;
```

```
        EE_UINT8 data[120];
        EE_UINT8 retransmission;
        EE_UINT8 indirect;
}MPDU;
```

Code Example 4.4: MPDU structure used for creation of frames

To construct a frame in general, an MPDU variable is created followed by the assignment of the packet length and the frame control fields. This is done using the auxiliary function *set_frame_control*, which takes as argument the subfields of the frame control field (e.g. frame_type, security etc) and sets the corresponding bits of the MPDU accordingly. The source and destination addresses are written in the data array, whose size may vary depending on the type of address used, and is specified in the frame control field for the destination device to be able to parse it. The pointer is advanced and the frame payload, which again depends on the frame type, is written.

4.3.3 Buffer Management

There are four buffers in the implementation:

sendBuffer: to store messages to be sent using normal (direct) transmission procedure;
receiveBuffer: to store messages received, but yet to be processed;
indirect_trans_queue: to store messages to be sent using indirect transmission method;
gts_send_buffer: to store messages to be sent during CFP period, using GTS mechanism.

The management of each of these buffers is described below.

Send and Receive Buffers

The send and receive buffers (*sendBuffer*, *receiveBuffer)* are defined as instances of circular queue structure, which is defined in cqueue.h file under common library. The queue structure has three elements: an array to store messages, and front and rear pointers, as shown in Code Example 4.5.

**The type EE_UINT<number of bits> is an E.R.I.K.A. type definition and represents an unsigned integer with the size of <number of bits>.*

```
typedef struct {
      MPDU arr[ARR_SIZE];
      EE_INT8 rear,front;

}c_queue;
```

Code Example 4.5: Circular queue structure

The circular queue structure consists of an array that contains the items in the queue, and two array indexes representing the front and rear pointers. The front pointer points to before the first element in the queue, and the rear pointer points to the last element in the queue (Figure 4.8).

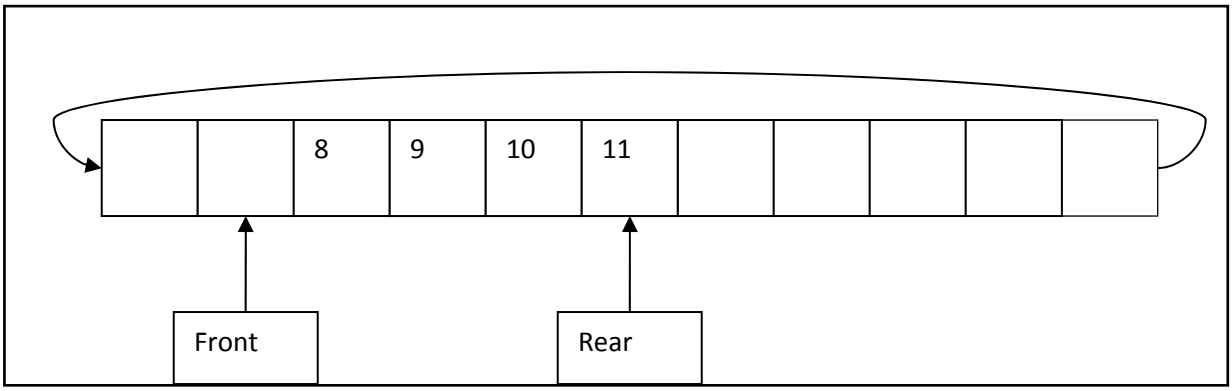


Figure 4.8: Circular queue for send and receive buffers

If the front pointer is before the rear pointer, the queue is full. The array is defined to be of maximum *ARR_SIZE,* which is 15. The ADT provides functions to insert, remove and retrieve the data into/from the queue. Additional functions to insert data at a relative position within each block are also implemented.

Indirect Transmissions Buffers

The buffer used for the indirect transmissions is defined as an array of indirect transmission elements, whose structure shown below (Code Example 4.6):

```
typedef struct
{
      EE_UINT8 handler;
      EE_UINT16 transaction_persistent_time;
      EE_UINT8 frame[127];

}indirect_transmission_element;
```

Code Example 4.6: indirect transmission element structure

An array of *INDIRECT_BUFFER_SIZE* size is maintained by the Coordinator to store messages to be sent using indirect transmission. When it has to send a message using indirect transmission, it searches by going through the indirect transmission queue, comparing the destinations addresses of every message in the queue to the device requesting indirect transmission. If found, the message is removed from the indirect transmission queue, and is inserted into the direct transmission buffer. The request is ignored if there are no messages for the requested address. Handler is used to identify whether a particular message has already been sent, specified by a value of zero. *Transaction_persistent_time* is the amount of time for which each message is kept in the buffer, and is deleted after this time is elapsed.

GTS Buffer and Management

The GTS buffer implementation differs for end device and coordinator. For end device, a FIFO queue of MPDUs, *gts_send_buffer* of *SEND_GTS_BUFFER_SIZE* is maintained, along with two pointers *in* and *out*, indicating rear and front, and a variable representing the total number of messages in the buffer. If the device has a GTS allocated and the *gts_send_buffer* is not empty, the message is sent in the allotted slot and the variables updated. For Coordinator, along with the *gts_send_buffer* storing messages, another array *gts_slot_list* is used, which maintains the available time slots. Each element in the *gts_slot_list* array represents one GTS, and there can be a maximum of seven. The structure of the GTS slot element is as shown in Code Example 4.7:

```
typedef struct gts_slot_element
{
      EE_UINT8 element_count;
      EE_UINT8 element_in;
      EE_UINT8 element_out;
      EE_UINT8 gts_send_frame_index[GTS_SEND_BUFFER_SIZE];

}gts_slot_element;
```

Code Example 4.7: *gts_slot_element* structure

The *gts_slot_element* defines a FIFO buffer used to store indexes that reference positions in the gts_send_buffer, and it is maintained as the GTS send and receive buffers. The array *available_gts_index* stores the available indexes.

Figure 4.9 shows sniffer snapshots showing allocation of GTS slots on the *Alloc* request from the end device. The allocated slots are listed in the *GTS* field of the beacons.

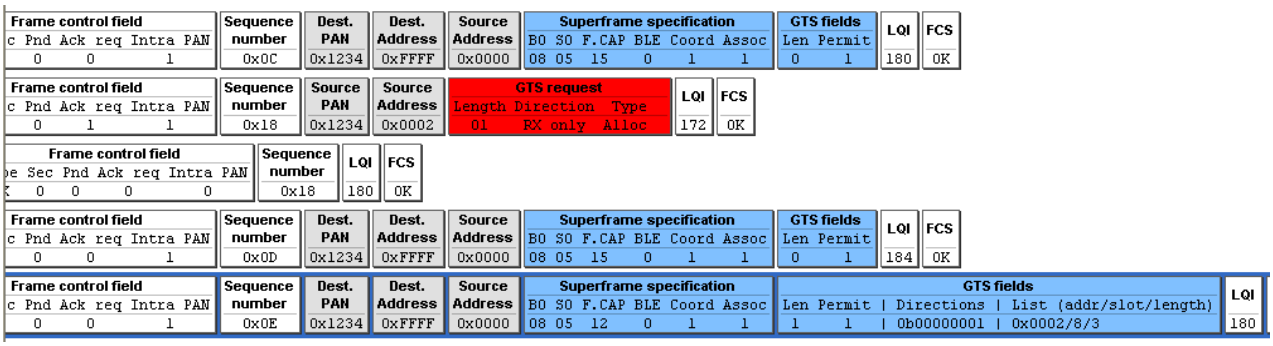


Figure 4.9: GTS allocation

Figure 4.10 shows deallocation of GTS slot on *Dealloc* request. As shown, the subsequent beacons do not have the allocated frame in the *GTS* field.

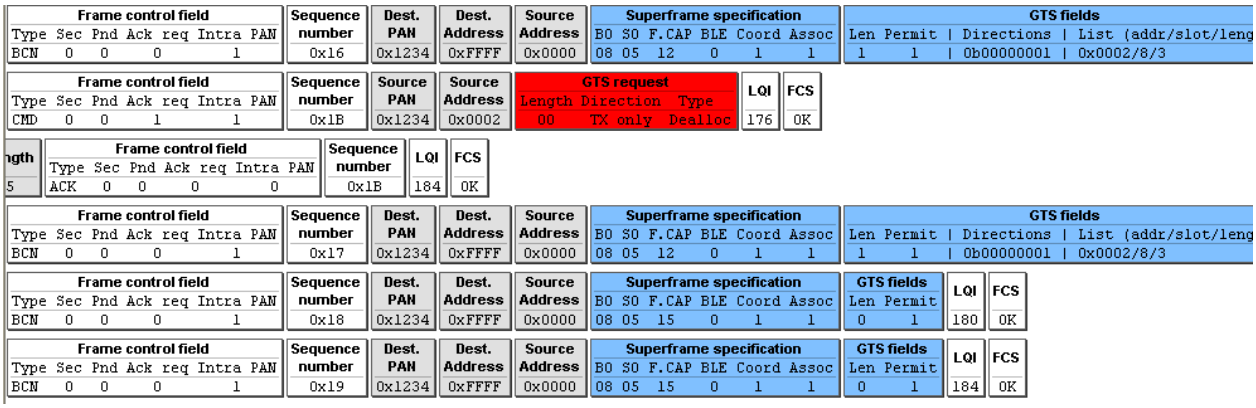


Figure 4.10: GTS deallocation

4.3.4 Beacon Management

Beacons are transmitted periodically by the Coordinator in order to synchronize the devices in the network as well to broadcast the general PAN information (BO, SO, GTS descriptors, pending data information etc). Figure 4.11 shows the structure of a beacon frame.

Octets: 2	1	4/10	2	variable	variable	variable	2
Frame control	Sequence number	Addressing fields	Superframe specification	GTS fields (Figure 38)	Pending address fields (Figure 39)	Beacon payload	FCS
MHR			MAC payload				MFR

Figure 4.11: Beacon frame format [10]

The Mac payload consists of superframe specification fields, GTS descriptor list, pending address information and optional beacon payload. The superframe specification field includes values of BO, SO, Battery life extension information, PAN Coordinator (indicating whether the device transmitting beacons is the PAN Coordinator) and Association Permit (indicating whether the new devices are allowed to join).

The beacon management implementation includes primitives to create beacons (in Coordinator) and to process beacons (in End Devices). The *create_beacon* function is used by the Coordinator to construct beacon frames. In order to avoid any delay, the beacon is created in advance, during inactive period, and stored at a separately allocated memory space (*mac_beacon_txmpdu*). This avoids delays that could occur had a common buffer been used. The created beacon frame is sent without contention when the next *bi_firedAlarm* fires.

The flowchart in figure 4.12 shows the steps of beacon creation: (1) MAC header is written; (2) Superframe specification is written, including the PAN parameters such as BO and SO; (3) GTS descriptor field is constructed, indicating the allocated and deallocated GTS, if any; (4) pending address descriptors are added, if any; (5) beacon payload is added, if the length is specified to be non-zero. The Coordinator can modify the PAN parameters using the *MLME-START_request* primitive, passing new values, which are updated in the following superframes.

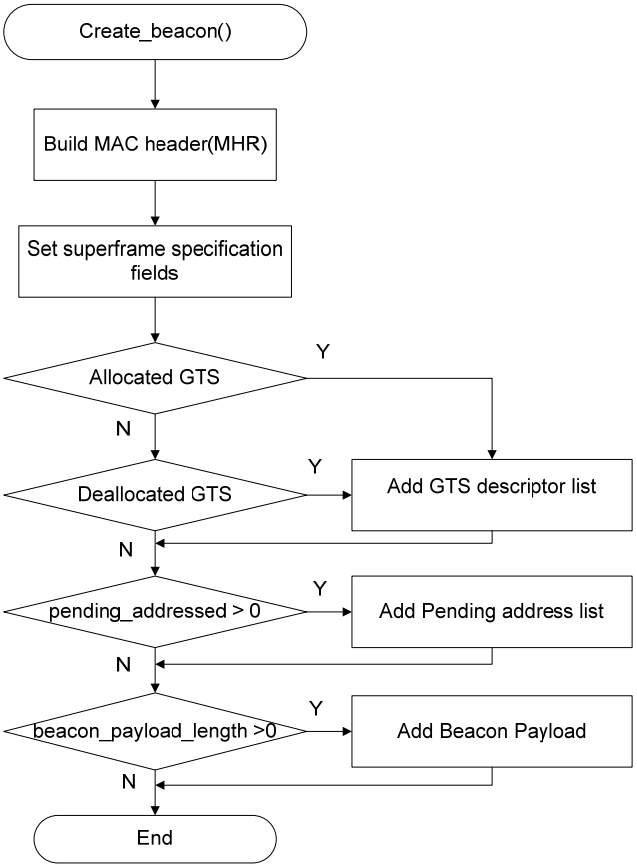
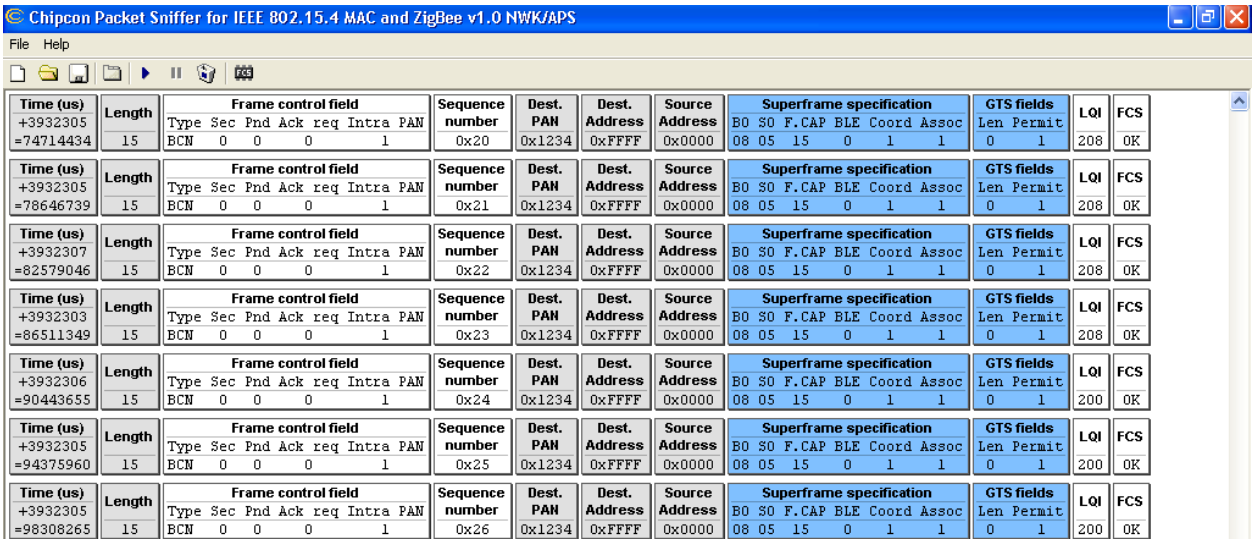


Figure 4.12: Beacon creation

Figure 4.13 shows a sniffer snapshot showing coordinator transmitting beacons.



Time (us)	Length	Frame control field	Sequence number	Dest. PAN	Dest. Address	Source Address	Superframe specification	GTS fields	LQI	FCS
		Type Sec Pnd Ack req Intra PAN					BO SO F.CAP BLE Coord Assoc	Len Permit		
+3932305 =74714434	15	BCN 0 0 0 1	0x20	0x1234	0xFFFF	0x0000	08 05 15 0 1 1	0 1	208	OK
+3932305 =78646739	15	BCN 0 0 0 1	0x21	0x1234	0xFFFF	0x0000	08 05 15 0 1 1	0 1	208	OK
+3932307 =82579046	15	BCN 0 0 0 1	0x22	0x1234	0xFFFF	0x0000	08 05 15 0 1 1	0 1	208	OK
+3932303 =86511349	15	BCN 0 0 0 1	0x23	0x1234	0xFFFF	0x0000	08 05 15 0 1 1	0 1	208	OK
+3932306 =90443655	15	BCN 0 0 0 1	0x24	0x1234	0xFFFF	0x0000	08 05 15 0 1 1	0 1	200	OK
+3932305 =94375960	15	BCN 0 0 0 1	0x25	0x1234	0xFFFF	0x0000	08 05 15 0 1 1	0 1	200	OK
+3932305 =98308265	15	BCN 0 0 0 1	0x26	0x1234	0xFFFF	0x0000	08 05 15 0 1 1	0 1	200	OK

Figure 4.13: Sniffer snapshot showing beacon transmission

4.3.5 The Slotted CSMA/CA Mechanism

All transmissions in the Contention Access Period, with the exception of beacon and acknowledgement frames, follow the slotted CSMA/CA mechanism. Its implementation involves several functions as described below:

- *send_frame_csma()* : Called after a message is enqueued in the send buffer, this function checks if there is a slotted CSMA/CA execution chain already started by checking the global variable *performing_csma_ca*. If not, it sets the variable and initiates the slotted CSMA/CA procedure. Figure 4.14 shows the implementation with the help of a flowchart.

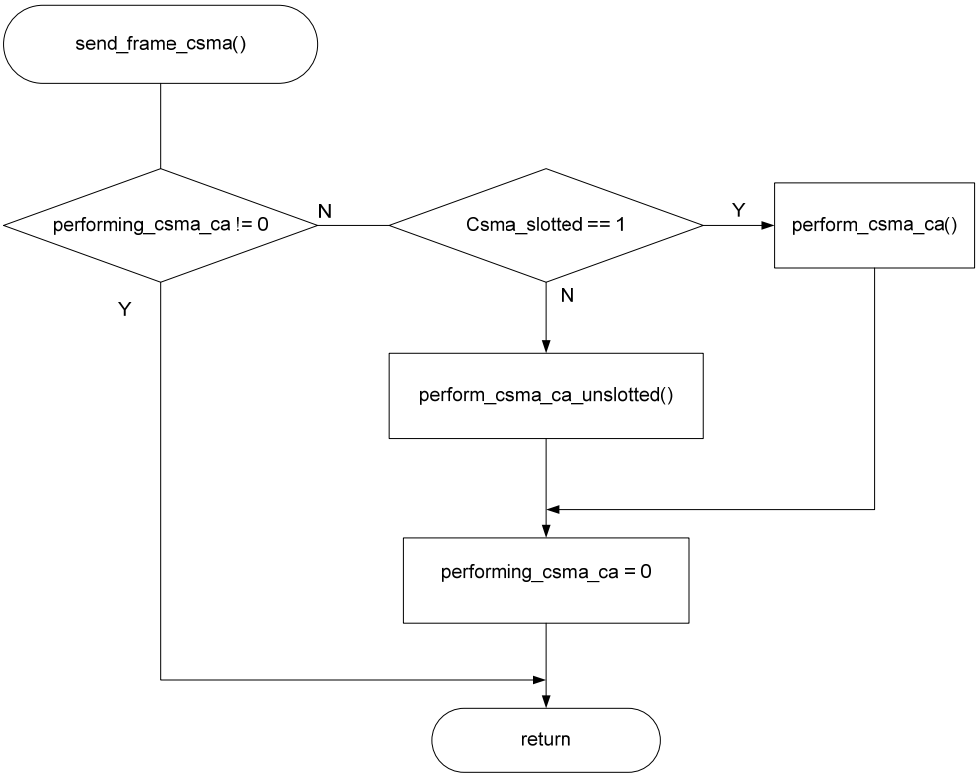


Figure 4.14: *send_frame_ca()* function flowchart

- *perform_csma_ca()*: Called from *send_frame_csma(),* it first initializes the slotted CSMA/CA variables by calling the *init_csma_ca()* function. It also initializes *BE* based on battery life extension support and finally sets the variable *csma_locate_backoff_boundary*, which triggers the final steps of the slotted CSMA/CA from the next backoff boundary. Figure 4.15 shows the implementation using a flowchart.

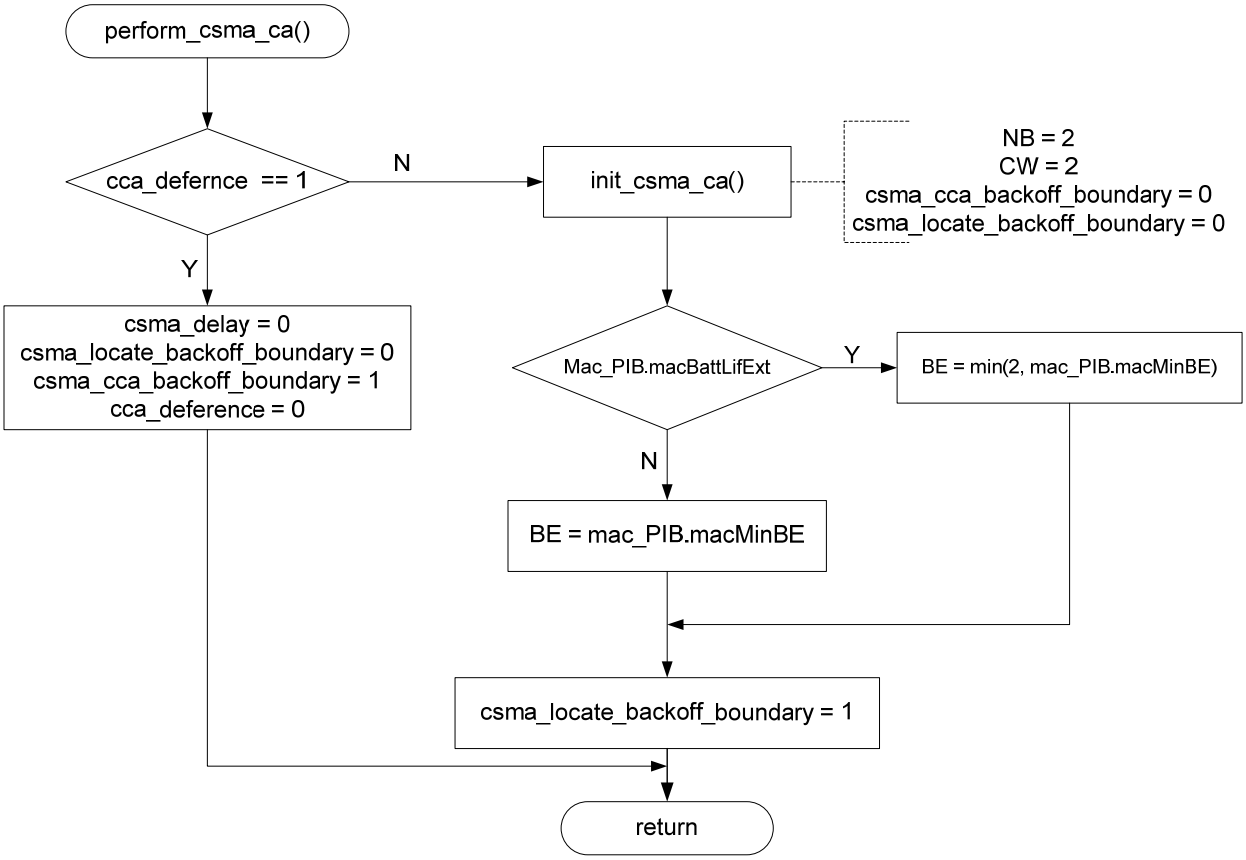


Figure 4.15**:** *perform_csma_ca()* function flowchart

- *init_csma_ca (EE_UINT8 slotted)*: Used to intialize the slotted CSMA/CA variables including NB, BE and CW;
- *backoff_fired_check_csma_ca()*: On the firing of the *backoff_fired* alarm, this function checks if there is a message to be sent in CAP. If yes, it initiates the final stages of the algorithm by counting the number of backoffs and calling *perform_csma_ca_slotted()*. The steps are described in the flowchart of Figure 4.16.
- *check_csma_ca_send_conditions()*: Used to evaluate the conditions necessary to send a messages in the CAP period. It calculates whether a message can be sent by adding the frame length and correspondent IFS symbols (also adding the acknowledgment turnaround time if the message requires an acknowledgment). Returns true if there is enough time to send the message;
- *perform_csma_ca_slotted( )*: It is called from the *backoff_fired_check_csma_ca()* function and executes the final steps of the slotted CSMA/CA procedure.

Figure 4.16: *backoff_fired_check_csma_ca()* function flowchart

4.3.6 Indirect Transmissions

For indirect transmissions, an array (of *INDIRECT_BUFFER_SIZE*) is maintained by the Coordinaotor, storing messages to be sent using indirect transmission procedure. The structure of the elements of the buffer (*indirect_transmission_element*) is shown in Code Example 4.8. *Handler* is used to identify whether the message has already been sent (specified by a value of zero). The *transaction_persistent_time* element is the amount of time for which each message is

to be kept in the buffer waiting for the indirect transmission request. The *frame* array stores the message.

```
typedef struct
{
      EE_UINT8 handler;
      EE_UINT16 transaction_persistent_time;
      EE_UINT8 frame[127];

}indirect_transmission_element;
```

Code Example 4.8: Indirect transmission element structure

When a device has an indirect transmission message, the message is stored in the indirect transmission buffer. At the time of the creation of the next beacon, the buffer is checked for messages and if found, recipient's address is specified in the beacon's pending addresses field. The destination device, upon receiving its address in the beacon, sends a data request command. On receiving this request, the Coordinator searches in the indirect transmission buffer for the correct message. The procedure is to go through the indirect transmission queue, comparing the destinations addresses until it finds the correct message. The request is ignored if there are no messages for the requested address. If found, the message is sent and removed from the buffer. It is also removed if *transaction_persistent_time* time is elapsed without any indirect transmission request received.

The following functions collectively perform the above tasks:

- *void init_indirect_trans_buffer()*: Initializes the indirect transmission buffer
- *void send_ind_trans_addr(EE_UINT32 DeviceAddress[])*: Called upon the reception of the indirect transmission request command from a perspective recipient; it searches for the message in the indirect transmission buffer and inserts it into the normal transmission buffer.
- *EE_UINT8 remove_indirect_trans (EE_UINT8 handler)*: Used to remove a message from the indirect transmission queue.
- *void increment_indirect_trans()*: Called at the end of every superframe, this function increments the transaction persistent time of every message in the indirect transmission

queue. If its value for any message reaches *macTransactionPersistenceTime*, the message is discarded.

Figure 4.17 shows indirect transmission of the association response command. The Coordinator prepares the response frame after receiving the request and indicates it to the End device by adding its address to the pending address field of the beacon. The End device then sends the data request command, to which the Coordinator responds by sending the association response command frame.

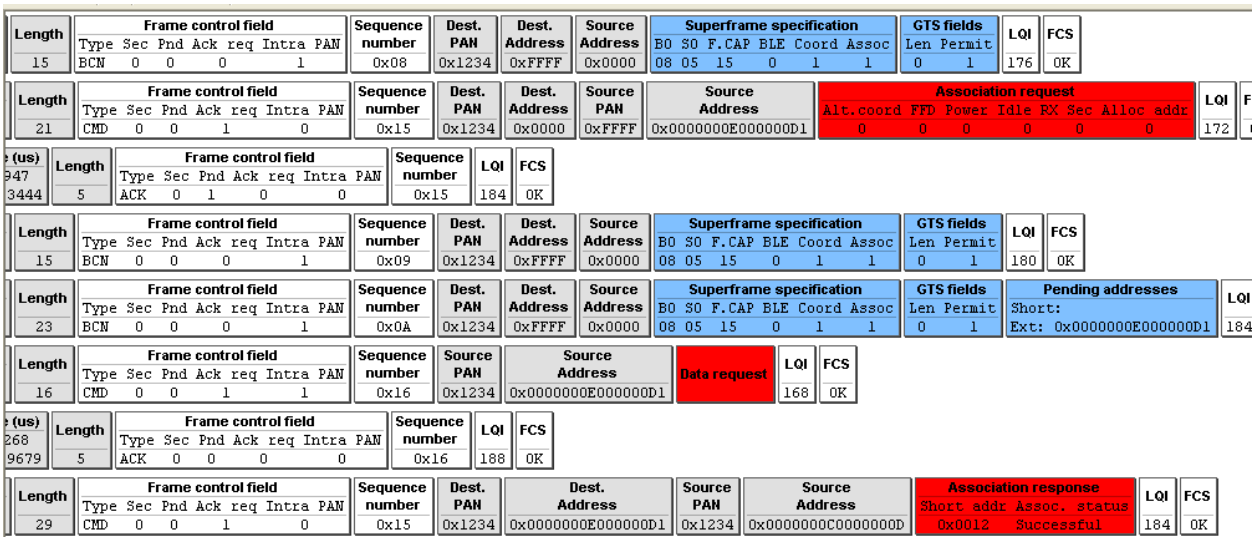


Figure 4.17: Indirect transmission sniffer snapshot

4.3.7 Acknowledgement and Retransmission

The acknowledgement and retransmission mechanism is implemented using the *ack_timer_firedAlarm* Alarm. The alarm *ack_timer_firedAlarm* is set before the call to *PD_DATA_request()* in *perform_csma_ca_slotted()* function when a frame having its *ack_reuest* bit set is being transmitted. The alarm is aperiodic with a duration of *ackwait_period*, given by

ackwait_period = mac_PIB.macAckWaitDuration/20;

20 beuing the number of symbols in a backoff period. The alarm is cancelled if an acknowledgement frame is received within *ackwait_period*, with the sequence number of the last transmitted frame. If the acknowledgement frame is not received within this period, a

retransmission is initiated. A transmission failure is reported if the number of re-transmission exceeds *aMaxFrameRetries*. The flowchart of Figure 4.18 shows the implementation approach.

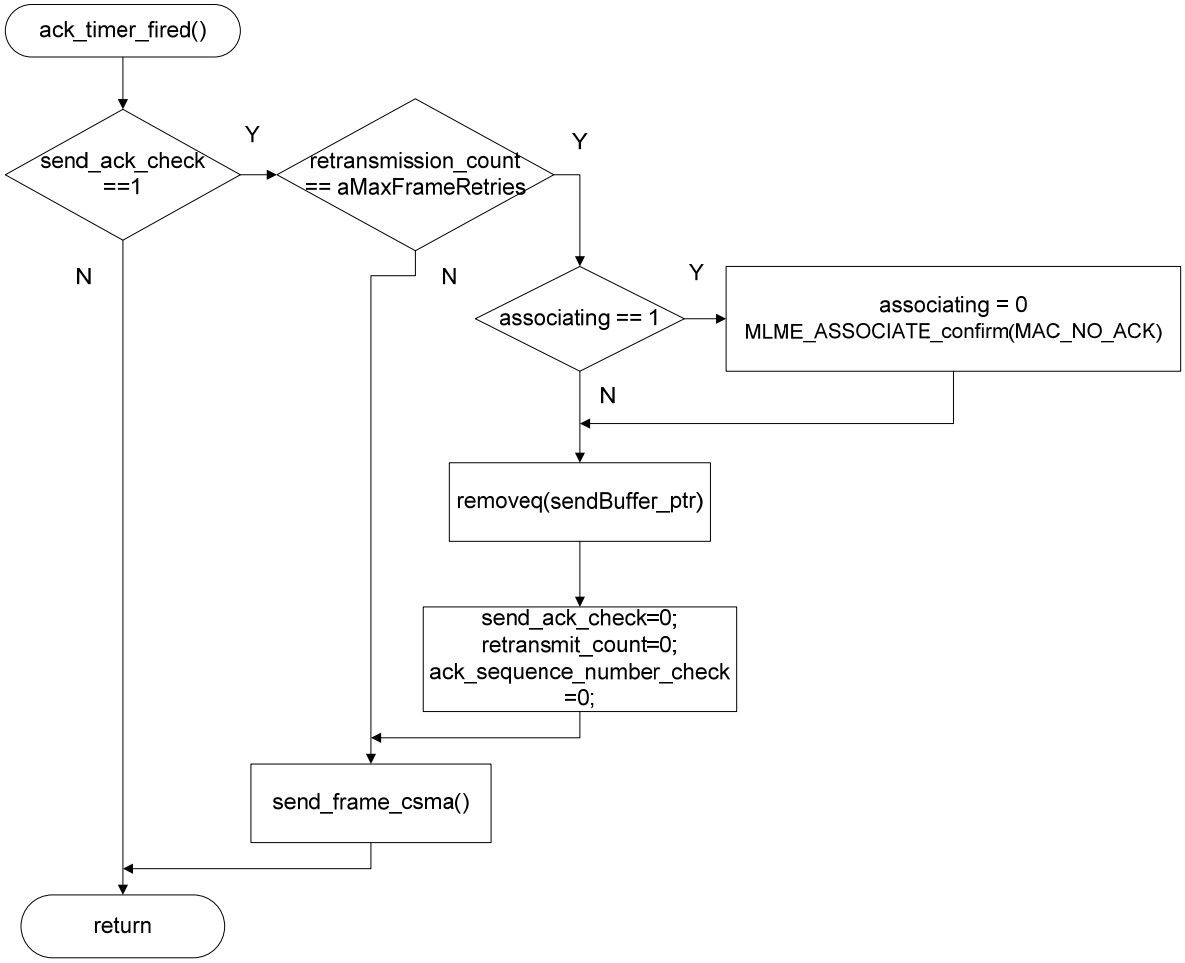


Figure 4.18: Acknowledged transmission and retransmissions flowchart

The following variables are used to control the retransmission procedure:

- *send_ack_check*: set if the current transmission requires an acknowledgment;
- *retransmit_count*: keps count of the number of retransmission attempts for the current frame;
- *send_indirect_transmission*: variable stating that the current transmission is an indirect transmission and doesn't require a retransmission if the first transmission fails;
- *ack_sequence_number_check*: current transmission frame sequence number.

The variable '*associating*' is used for the special case of the transmitted message being an association request command. If the requesting device doesn't get an acknowledgment after having transmitted *aMaxFrameRetries* times, the upper layer is notified with a failure status.

The variables *send_ack_check*, and *ack_sequence_number_check* are initialized in the function *send_frame_csma(),*which is used to start the transmission of the next frame. Figure 4.19 shows the transmission of a data frame with acknowledgement request and the transmission of acknowledgement by the receiving device.

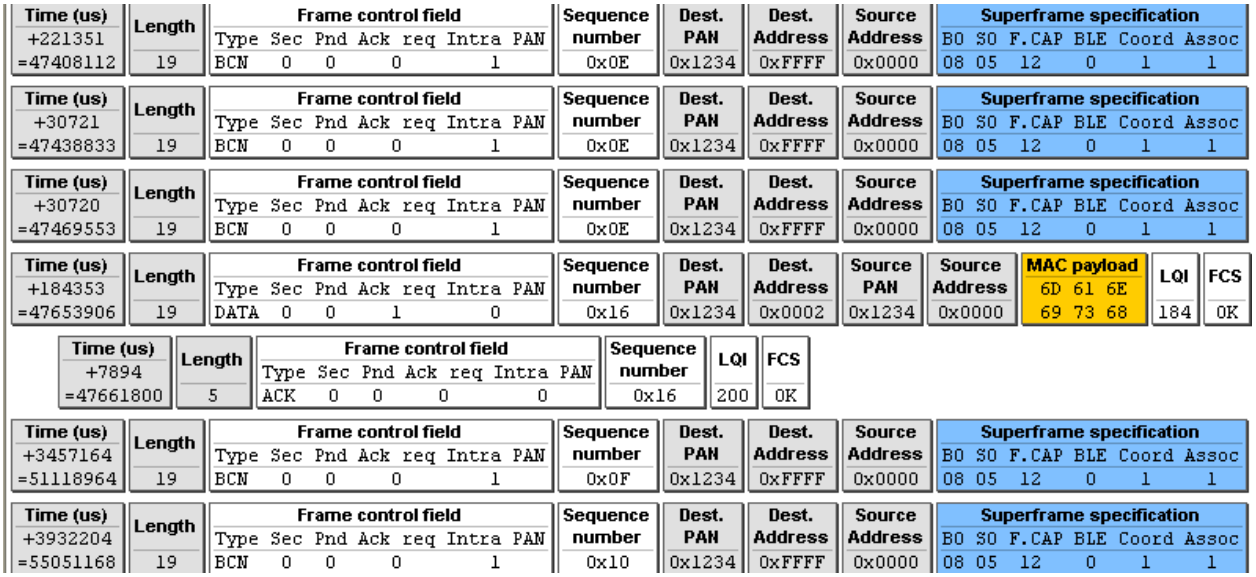


Figure 4.19: Acknowledged data transmission sniffer snapshot

4.4 ZigBee Network Layer

The current network layer implementation supports tree-routing using a distributed address assignment mechanism. Network discovery functions are implemented statically since channel scan is not supported by lower layers. The implementation is broadly based on the Open-ZB implementation of ZigBee Network Layer (in nesC), which is documented in [36].

4.4.1 Association and address assignment

To be able to transmit data in a PAN, a device must join a network by associating with a Coordinator. The association occurs with the candidate device sending an association request command to the Coordinator and the Coordinator responding with an association response sent using indirect transmission. If the procedure is successful, the end device is assigned a short

address, which is used for all future communication within the PAN. In the Network Layer, the decision on association and address assignment is made in the *MLME_ASSOCIATE.indication* primitive which is called from MAC layer when an association request command is received. The flowchart of Figure 4.20 summarizes the procedure.

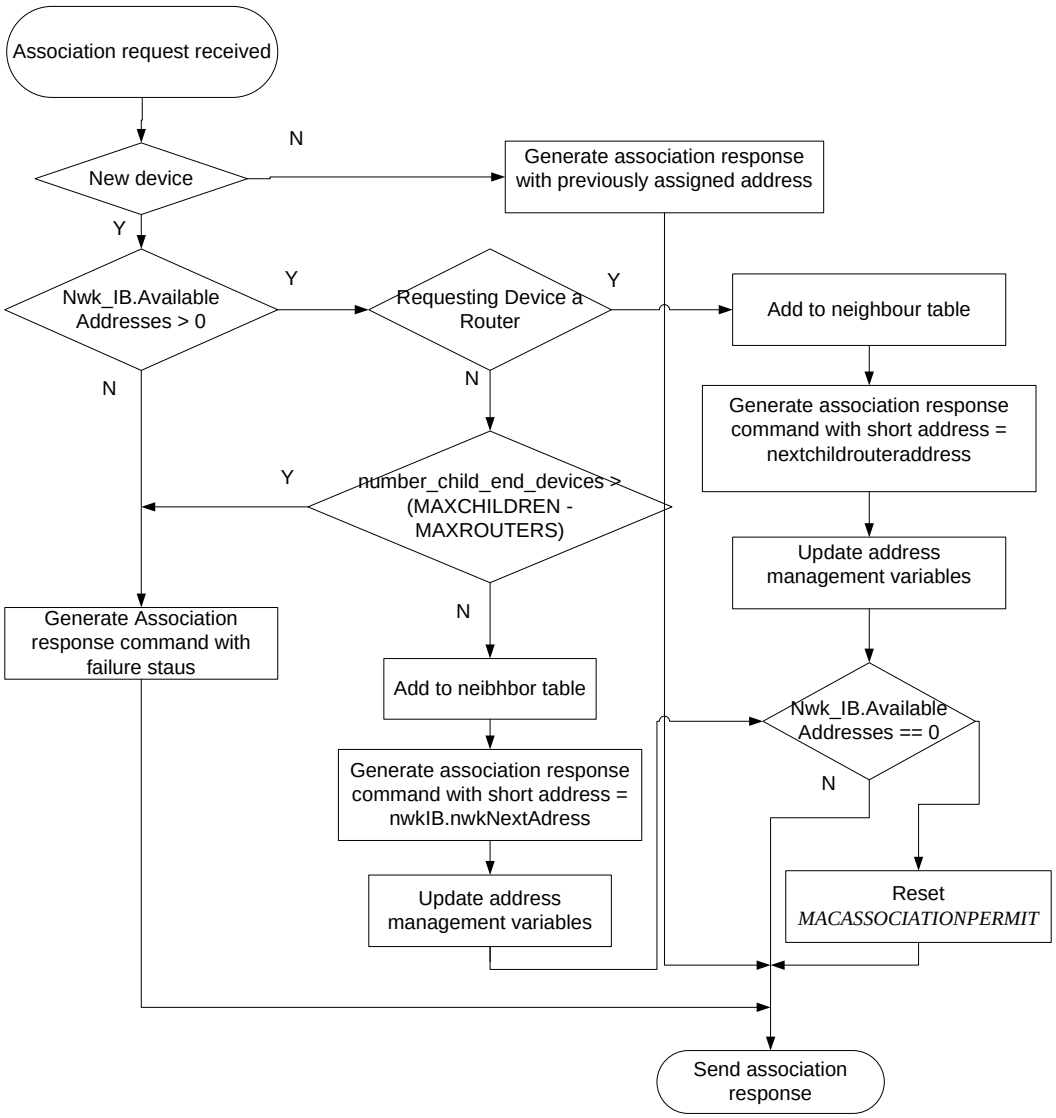


Figure 4.20: Network Layer association and address assignment flowchart

Upon receiving the association request, the parent first checks if the requesting device is new by searching its address in the neighbor table. If the request is from a device already associated, association response is generated with previously assigned short address. If it is a new device, the parent will check if there are new addresses available by checking the NWK PAN variable

nwk_IB.nwkAvailableAddresses. If available, new address of the associating device will be determined depending on the type of the device (end device or router). The *Cskip* function (described in Chapter 2) is used to compute the address.

After generating the association response command, the parent device updates the address assignment variables. *nwk_IB.nwkAvailableAddresses* is decremented and if it becomes equal to zero, *MACASSOCIATIONPERMIT* flag is reset to zero. This flag is used to set association permit bit of the beacons, which indicates whether new devices are to be accepted or not.

Figure 4.21 shows a sniffer snapshot of an end device associating with the Coordinator of a PAN. As described above, the association response can be seen being transmitted using the indirect transmission method, with the Coordinator waiting to receive the data request command before transmitting the association response frame.

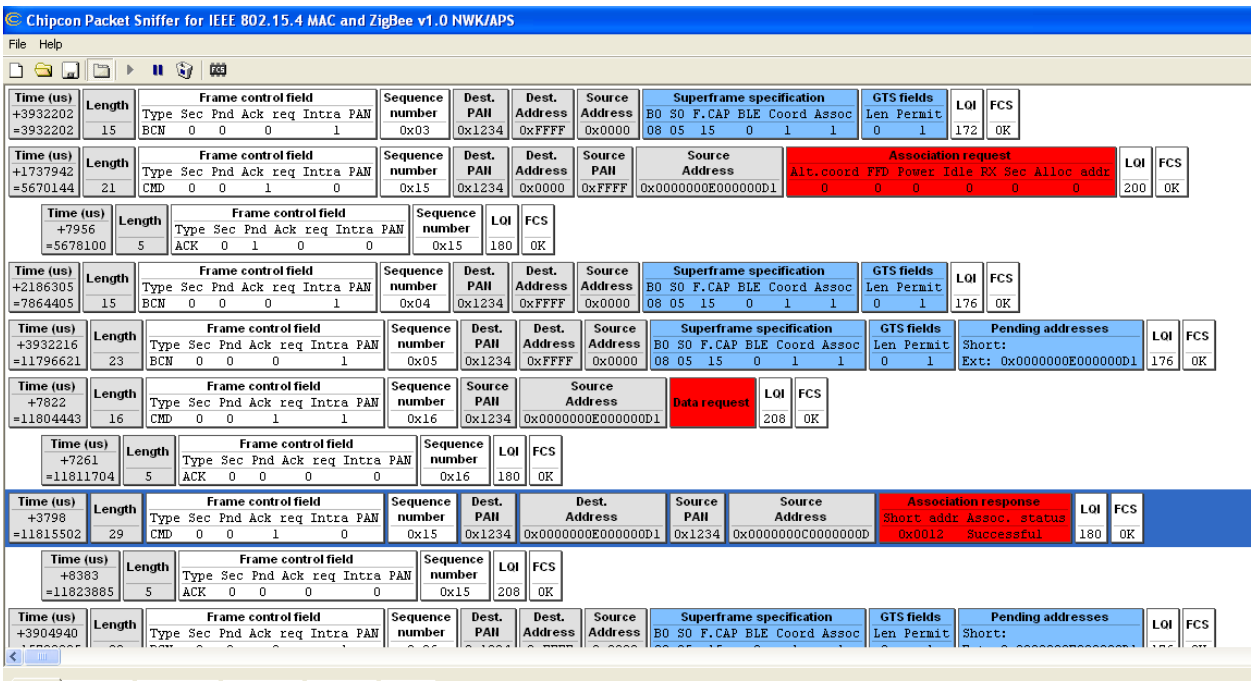


Figure 4.21: Association sniffer snapshot

4.4.2 Tree Routing

The routing procedure is based on the scheme described in 2.3.2. Routing is implemented in the network layer in the function *MCPS_DATA_indication*, called by MAC layer when a data frame is received. The procedure is described using the flowchart of Figure 4.22.

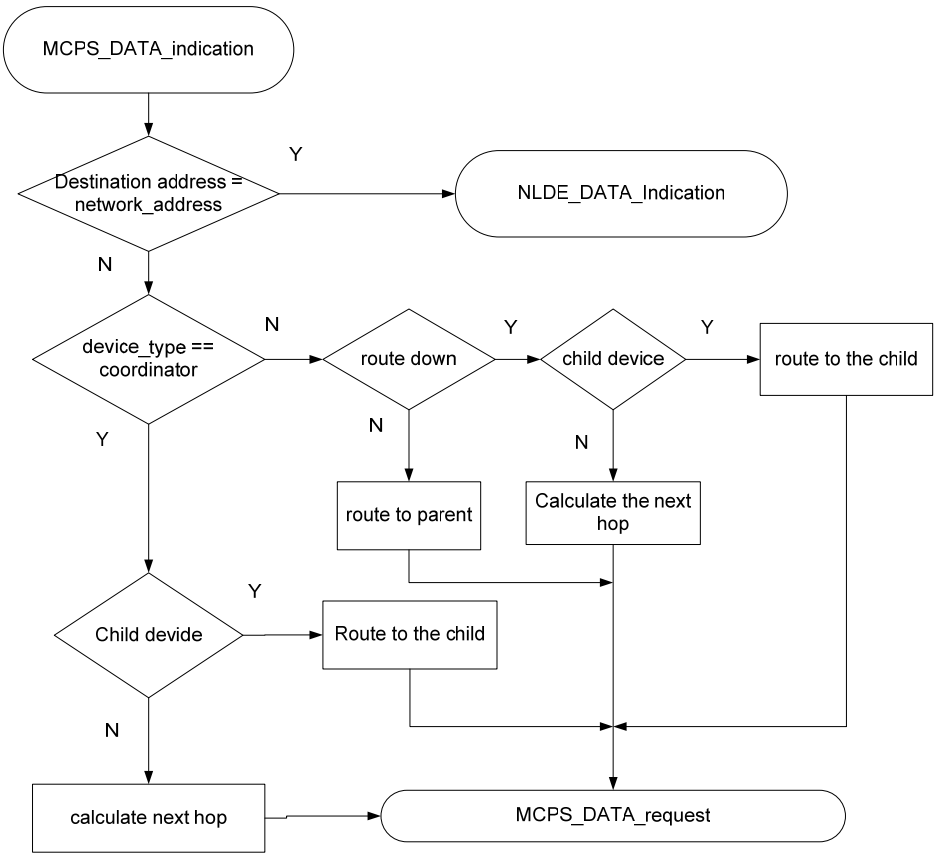


Figure 4.22: *MCPS_DATA_indication()*flowchart

On the reception of a data frame, the network layer first checks if the packet was destined to it by comparing routing destination field (in the network header) to its own short address. If it matches, the data payload is transferred to the upper layer, using *NLDE_DATA_indication* primitive. If the routing destination address is different, and the device is an end device, it will forward the data to its parent device. If the current device is a ZigBee Coordinator, it will check if the if the final destination is one of its child, by comparing the destination address to the addresses in the neighbor table. If found it will set the next hop address to the child device. Otherwise, it calculates the next hop by applying the Tree Routing formula of equation 2.8. Based on the next hop address, it will route the packet to its parent or to one of its child routers.

The data frame transmission procedure is similar to the routing mechanism. After the creation of the frame, the device assigns a destination address to the routing fields. If the device is a ZED, this address is of its parent. Otherwise, if the device is the ZigBee Coordinator or a ZigBee Router, it checks if the destination is a child device; if not, it calculates the next hop address using the Tree Routing formula (Section 2.3.2).

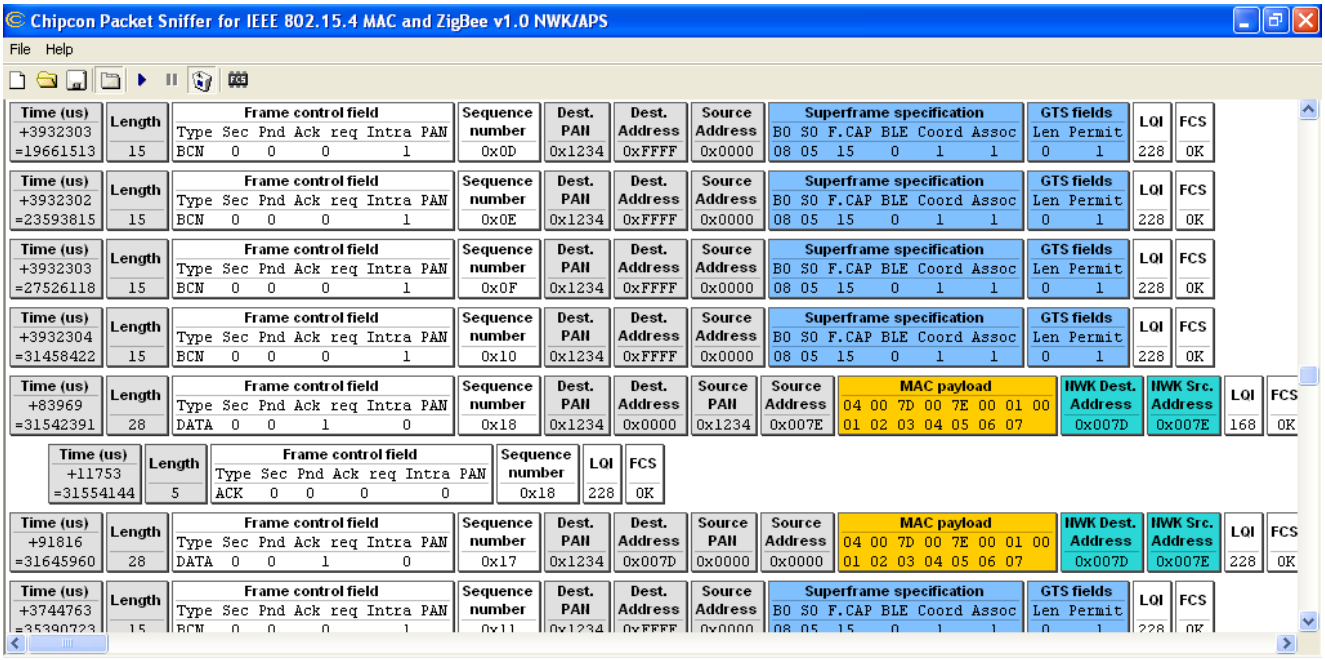


Figure 4.23: Routing sniffer snapshot

Figure 4.23 shows the NWK fields of a packet being transmitted from source 0x7D to destination 0x7E, by hopping through the common coordinator parent, 0x0000. The 'Source Address' and 'Destination Address' fields represent the source and destination of the current hop, whereas the 'NWK Src' and 'NWK Dest' fields represent the original source and destination addresses.

4.5 Cluster-Tree Network Formation

The IEEE 802.15.4/ZigBee specifications specify the network formation in the beacon-enabled mode for star-based networks only, which lack scalability. Although Cluster-Tree network concept is mentioned, there is no description of how it can be implemented. The difference between the star-based networks and Cluster-Tree is that in Cluster-Tree networks there are

multiple routers which act as IEEE 802.15.4 Coordinators. It does not make any difference in non beacon-enabled mode but in beacon-enabled mode all these router generate beacons to synchronize their clusters of nodes, which may result in collisions if their timings are not coordinated centrally. Such collision beacons of and frames from different clusters may result in the loss of synchronization of the clusters. Therefore, a beacon frame scheduling mechanism is required to avoid collisions of beacons and frames from different clusters.

4.5.1 Time Division Beacon Scheduling Mechanism

Although no mechanism to avoid such collisions is specified in the ZigBee standard, two approaches were proposed to avoid the collisions by Task Group 15.4b: (1) a time division approach and, (2) a beacon-only period approach.

We have implemented the first approach of time division. In this approach, time is divided among the Coordinators in a way that the active period of any Coordinator falls in the inactive period of all other Coordinators of the network (Figure 4.24).

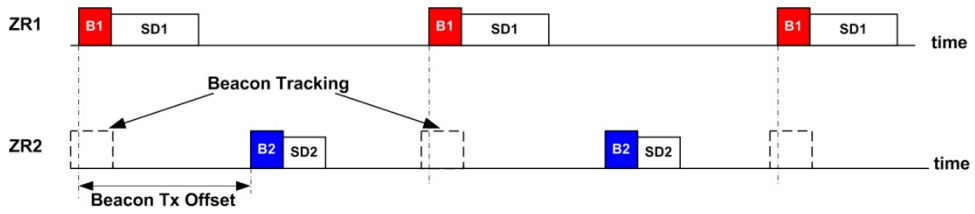


Figure 4.24: Beacon Frame Collision Avoidance - The Time Division Approach [19]

Each Coordinator uses a starting time relative to the Coordinator beacon (*Beacon_Tx_Offset)* to transmit its own beacon frames. The beacon offsets is different for each router to ensure that their active periods do not overlap. Communication between different clusters is accomplished by the using indirect transmissions: every Coordinator wakes up both in its own active period and in its parent's active period.

The scheduling relies on a negotiation prior to beacon transmission. After successfully associating with a network, the ZigBee Router (ZR) sends a negotiation message to the ZigBee Coordinator(ZC), embedding the envisaged (*BO, SO*) pair, and requesting a beacon broadcast

permit. The ZC replies with a negotiation response message containing a beacon transmission offset (the instant when the ZR must start transmitting the beacon) for successful negotiations. In case of rejection, the ZR must disassociate from the network.

4.5.1 TDBS Implementation

The TDBS implementation follows the implementation of the same mechanism, over Open-ZB stack, as described in [36]. A few minor changes had to be made in the MAC and Network Layer SAP namely the addition of *StartTime* argument in M*LME-START_request* and *NLME-START-ROUTE_request* primitives. It is used as a transmission offset with respect to the parent ZigBee Route. Figure 4.25 shows the negotiation mechanism.

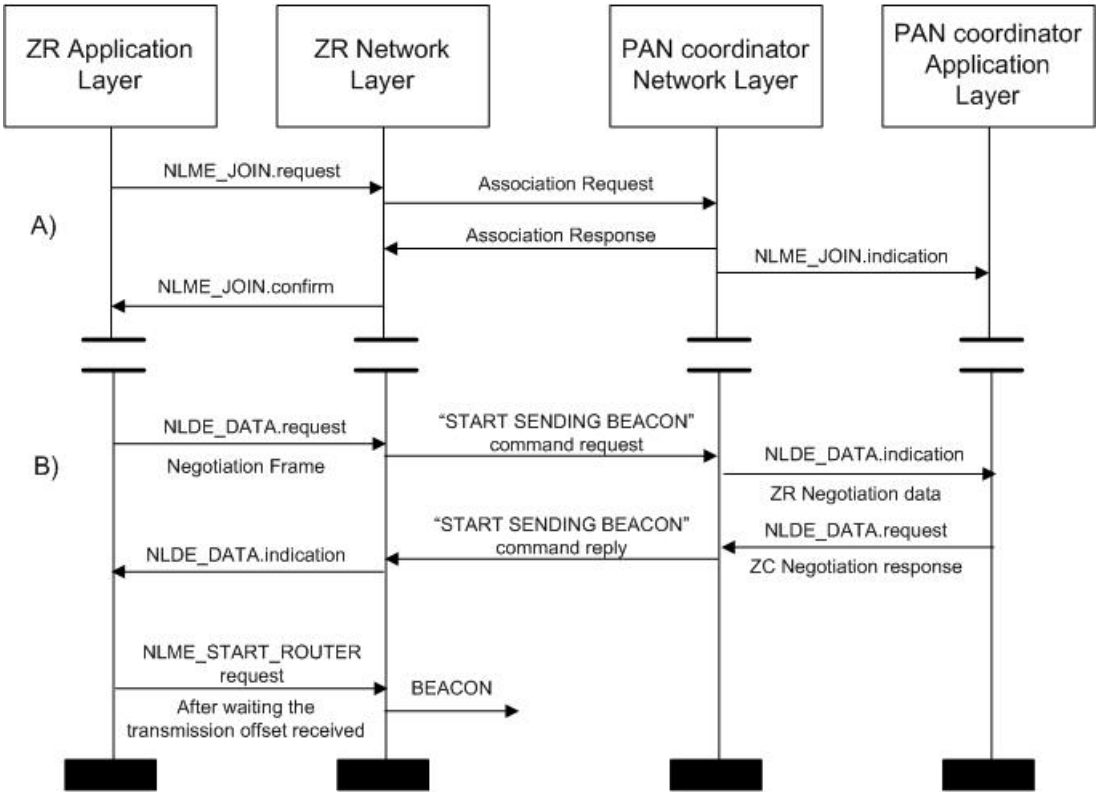


Figure 4.25: Time Division Beacon Scheduling Negotiation diagram [19]

After a successful negotiation, the ZR has two active periods:

I. It´s own superframe duration, in which ZR is allowed to transmit frames to its associated devices or relay frames to the descendant devices in the tree, and
II. Parent's superframe duration, in which the frames to be sent upstream are sent.

To achieve this, the buffering mechanism at Mac had to be changed. Instead of one send buffer, with TDBS it has two: one for downstream messages, and the other for upstream messages. Which of these two is to be used can be specified from the network layer using a reserved bit of *TxOptions* (transmission options) parameter in MCPS_DATA_request primitive.

CHAPTER 5
Supporting Different QoS Levels in CAP

This chapter discusses the second major contribution of this thesis, namely the implementation, validation and evaluation of traffic differentiation mechanism.

5.1 Introduction

IEEE 802.15.4 uses CSMA/CA for medium access control in Contention Access Period (CAP), which in its standard form does not provide any means of QoS support. Although the GTS mechanism provides an option of guaranteeing timing and reliability in Contention Free Period (CFP), it is with many limitations. The first is the restriction on the amount and distribution of traffic that can avail this service. In a superframe, a maximum of eight GTS slots can be allocated; implying that in a PAN only a maximum of eight devices can have guaranteed slots in any particular superframe. Other devices can only transmit in CAP, without any QoS support. Second, GTS is not very useful if the messages requiring QoS support are evenly distributed over time. It can only provide guaranteed services in bursts, limited for any device to the guaranteed slots allocated to it. Third, even in applications where GTS slots can be considered sufficient, the commands requesting guaranteed slot are themselves to be transmitted in the CAP, and are thus susceptible to delays and losses.

Thus, while GTS is considered a good solution for the QoS requirement of the low-rate WPAN applications (for which IEEE 802.15.4 was originally designed), the requirements of dense sensor networks (especially at high and distributed traffic load) demand a more flexible mechanism. This need arises because, in most sensor applications, there can often be found a grade of cruciality among the messages being transmitted. It is even more common to find a set of messages whose transmission is critically more important compared to the rest. In the case of a fire detector application, for example, successful transmission of messages indicating an abrupt rise in temperature is critical for the application. Such distinctions can also be made among the protocol data units. For example, it can be argued the PAN management commands, needed for the synchronization and control of a network, are more important than the regular data frames. These critical messages, distributed over time, require that QoS support be extended to CAP. The

same can be said of the large scale applications, where getting a guaranteed slot in CFP is not possible for most nodes.

5.2 Related works

The need to extend QoS support to CAP has drawn considerable attention from the research community in recent past and many proposals have been put forward. Since the CSMA/CA algorithm is used for medium access in CAP, most of these proposals focus on enhancement of the CSMA/CA. It should be noted that similar proposals had also been made for introducing QoS support in IEEE 802.11 [37], which also uses CSMA/CA for medium access. Consequently, the 802.11e amendment [38] proposing Hybrid Coordination Function (HCF) has been approved and incorporated in IEEE 802.11-2007[39]. However, since the behavior of the slotted CSMA/CA used in 802.15.4 is different from the unslotted version used in 802.11, only the proposals specific to the slotted version are discussed here. One such proposal [40] suggests introducing Priority Toning strategy, which requires that the node having high priority packet transmit a "tone signal" in the backoff slot immediately preceding the next beacon transmission. The Coordinator wakes up in this particular slot every superframe to listen for the tone, and if detected, it transmits an alarm signal in the next beacon indicating other devices to defer their transmissions. This period of deference is used exclusively by the original node having high priority packet to transmit the urgent data. The authors propose another modification in [41], which advocates that the high priority frames perform only one CCA, instead of the standard two, to determine the idleness of the channel before transmission.

While the simulation results of both of these approaches indicate improved timing and reliability for high priority frames, their implementation require fundamental changes in the protocol. The first approach, Priority Toning, needs special hardware support, which is not available in many chipsets including the CC2420 transceiver used by us. The second major drawback is its incompatibility with the standard version of the protocol. If a network has nodes programmed according to the current standard as well as priority toning, it may result in collisions in the deference period since the nodes programmed with current standard will not recognize the tone signal and may thus transmit in the reserved slot. The third disadvantage is that the high priority frame is to be transmitted in the superframe next to the one in which request is made, introducing intrinsic delay. Also, if multiple devices send such requests during the same superframe, the

mechanism fails. The second approach of CCA reduction requires Frame Tailoring, i.e. adjusting data packet length in such a way that one CCA becomes sufficient to detect any acknowledgement frame transmission. While this method reduces the CCA overhead by half, problem of backward incompatibility remains.

In [20], the authors propose an alternative approach which is compatible with the existing standard while providing priority based service. It proposes two means of service differentiation: one at node level, and another at network level. At node level, it proposes priority queuing to reduce queuing delays of high priority traffic. High priority frames are given preference and sent before the low priority frames. At network level, it applies the idea of priority based parameter tuning, i.e. initialization of relevant CSMA/CA parameters on per packet basis. The selection of parameters is based on a previous study of the slotted CSMA/CA by the authors [42], in which it was observed that the timing and reliability can be significantly affected by the initialization values of the following parameters: (i)macMinBE: the minimum backoff exponent; (ii)aMaxBE: the maximum backoff exponent; (iii)CWinit: the initial value of the CW; and (iv)macMaxCSMABackoffs: the maximum number of backoffs.

The proposed differentiation service was simulated using OPNET simulator [43], generating positive results, described in [20]. We adopted this approach to introduce QoS support in CAP by adding priority queuing and per-packet parameter control in our implementation of slotted CSMA/CA.

It should be noted that a similar strategy was also adopted in [44]. However, the effect of each parameter was not studied separately. Also, the implementation was built over a TinyOS implementation of the protocol stack which we found unreliable for traffic generation at high traffic load, making it difficult to study precisely the impact of parameter variations. In comparison, Erika provides reliable timing behavior through very high traffic generation rate, making it possible to make a more precise study. We also assess each case with and without priority queuing, helping distinguish between impacts parameter tuning and priority queuing.

5.3 Differentiation Strategy and Implementation

5.3.1 Strategy

The implementation of Traffic Differentiation (referred to as TRADIF henceforth) mechanism is based on two principles: priority based parameter tuning in slotted CSMA/CA and priority queuing in transmission buffers. Figure 5.1 presents a pictorial view.

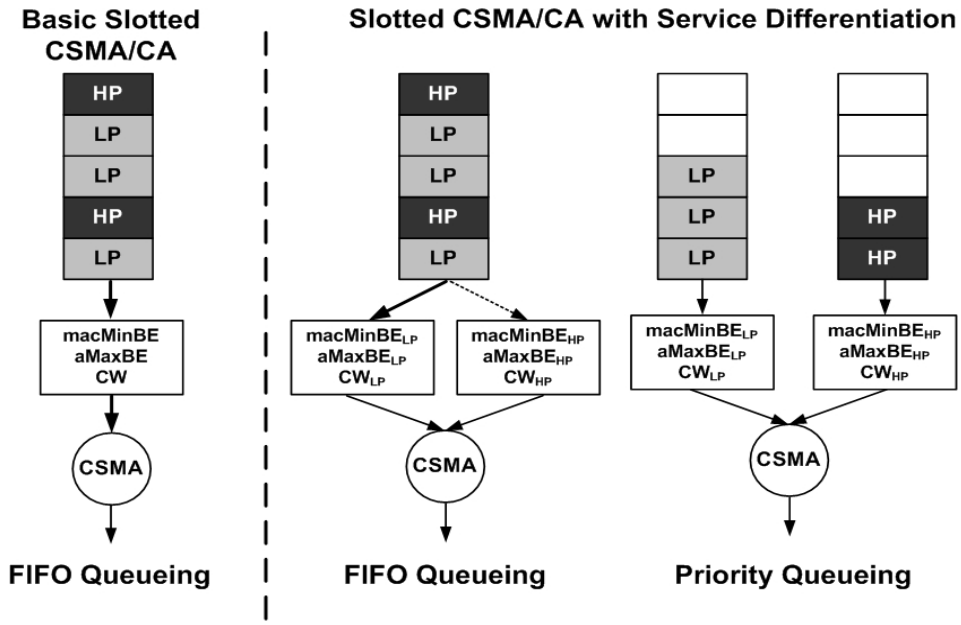


Figure 5.1: Service Differentiation Strategies [20]

1. Priority based parameter tuning: As already mentioned, the behavior of the slotted CSMA/CA is affected by its initialization parameters, changing the values of which impacts its performance. The idea is to choose different values of these parameters for high and low priority packets so as to increase the probability of success and reduce delays for high priority frames. The set of parameters, listed below, and their initialization values for both cases are based on the studies carried out in [42] and [20]. The subscript HP and LP denote High Priority and Low Priority respectively.
   (i) macMinBE: {$macMinBE_{HP}$, $macMinBE_{LP}$}
   (ii) aMaxBE: {$aMaxBE_{HP}$, $aMaxBE_{LP}$}
   (iii) CWinit: {CW_{HP} , CW_{LP}}

2. Priority Queuing: Priority Queuing is applied to reduce the queuing delays of high priority frames. The priority scheduling means that if there are high priority frames in transmission queue, they will be picked ahead of low priority frames, irrespective of the order of arrival. In FIFO mode, on the other hand, frames are transmitted in the order they arrive. The implementation supports both FIFO and Priority Queuing (PQ) modes, and can be specified by the user.

5.3.2 Implementation

To add TRADIF to our IEEE802.15.4 stack implementation, modifications have been made primarily in two set of modules: queuing and CSMA/CA mechanism.

Since only two priority levels are assumed, Priority Queuing support has been provided by maintaining two transmission queues: High Priority (HP) queue and Low Priority (LP) queue. The high priority frames are enqueued in the HP queue, and low priority frames are enqueued in the LP queue. In TRADIF mode of operation, every transmission starts with an examination of the high priority queue, and if non empty, the frame is selected from it.

The changes in slotted CSMA/CA implementation involves modifying the following functions (described in section 4.5): *send_frame_csma(), perform_csma_ca(), init_csma_ca(), backoff_fired_check_csma_ca(),* and *perform_csma_ca_slotted().*The modified version supports CSMA/CA in both standard as well as TRADIF mode. TRADIF in turn is supported by both modes of queuing: FIFO and PQ.

Only the changes from the implementation of slotted CSMA/CA (section 4.5) are mentioned here. In the standard (non-TRADIF) mode, when a frame is to be sent, it is enqueued in the send buffer and *send_frame_csma()* is called to initiate the process of transmission. This is unchanged for the FIFO mode of TRADIF. In Priority Queuing mode, when a frame is to be sent, it is enqueued in the High Priority (HP) or Low Priority (LP) Queue, depending on the priority of the frame. In our implementation, commands frames have been treated as high priority traffic and data frames as low priority by default. However this can be easily modified to support prioritization of traffic generated at application level (which was done for testing of TRADIF, as discussed in next section).

The modified (TRADIF) version of the *send_frame_csma()* function is shown in the flowchart of Figure 5.2. To understand the changes, recall from section 4.5 that there are two independent threads of executions to complete the execution of CSMA/CA procedure. The thread starting from *send_frame_csma()* and including *perform_csma_ca(), init_csma_ca()* are responsible for the initialization of of the CSMA parameters, including *NB, BE* and *CW*. The thread starting from *backoff_fired(),* and including functions *csma_check_backoff_fired()* and *perform_csma_ca_slotted()* implement backoff countdown, CCA and CCA deference, and finally the transmission of the frame.

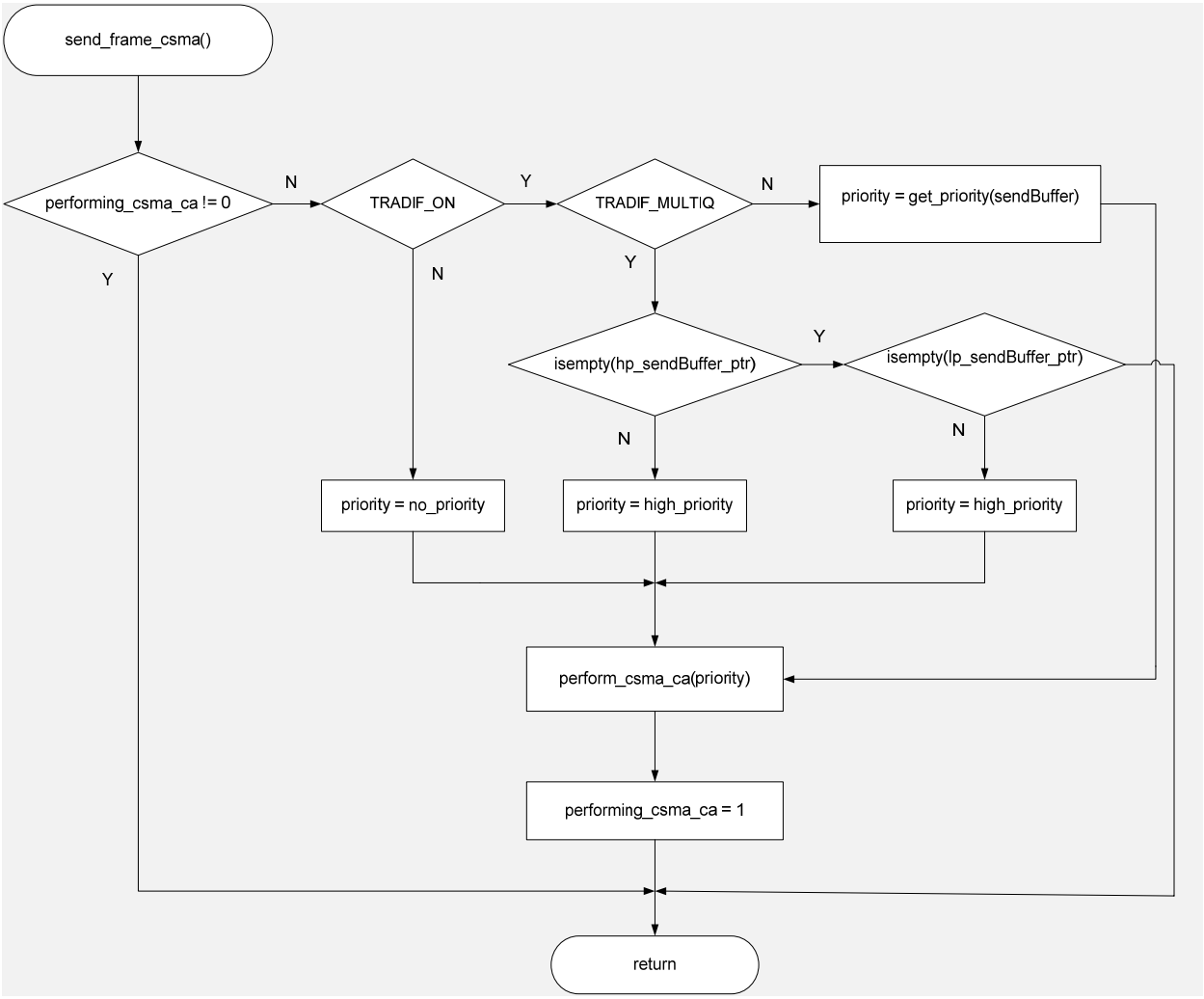


Figure 5.2: TRADIF *send_frame_csma()* flowchart

The TRADIF version of *send_frame_csma()* (Figure 5.2) ensures that the parameters initialized by the thread correspond to the priority of the frame to be picked by the

backoff_fired_check_csma thread. In FIFO mode, it is the default case since new frames are placed at the rear end of the queue and are handled only after the frame under processing at front is transmitted and removed from the queue. In Priority Queuing mode, which uses two queues, it is achieved with the help of two global variables: *performing_csma_ca* and *pkt_priority*. *Performing_csma_ca* is set at the start of the CSMA/CA procedure, i.e., in the *send_frame_csma()* function, indicating that the a CSMA chain is already in execution. Any other attempt to initiate the procedure is rejected until the current operation is completed (Figure 5.2, first step), thus preventing the mid-way re-initialization of parameters. Variable *performing_csma_ca* is reset at the end of *perform_csma_ca_slotted()*. This check is sufficient for FIFO queue models (both TRADIF and non-TRADIF). In multiple queuing (PQ mode), however, another check is needed to avoid the rare but nevertheless possible case of a frame being enqueued in the HP queue after initialization of the parameters by the first thread (corresponding to a lower priority frame) but before the second thread picks the packet from the send buffer. Here, parameter consistency is maintained using *pkt_priority* variable, which is used to pass the priority of the packet at the front of the queue to the second thread. This variable is set after the initialization of the parameters (Figure 5.3) and used to select the queue in *perform_csma_ca_slotted()* (Figure 5.5).

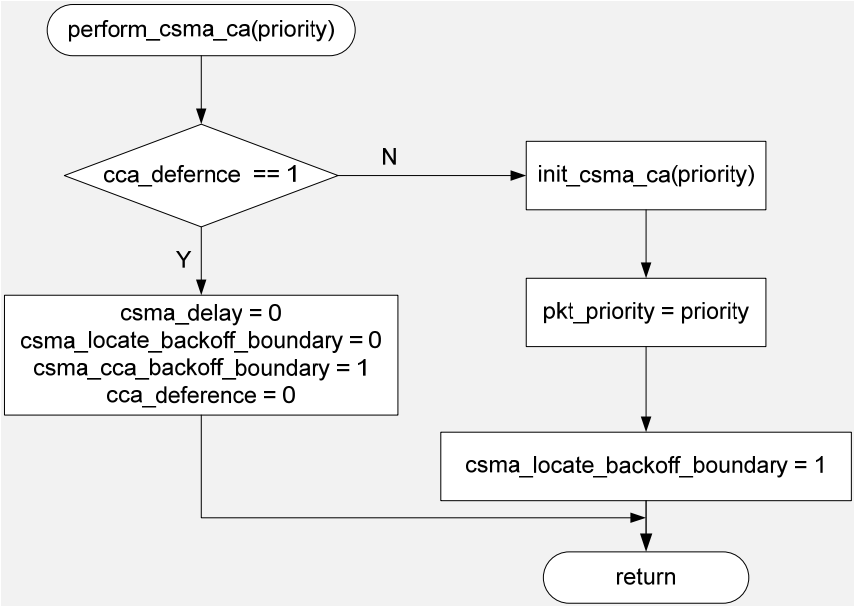


Figure 5.3: TRADIF *perform_csma_ca()* flowchart

Perform_csma_ca, shown in Figure 5.3, is similar to the previous version (Figure 4.15) with two differences: addition of *pkt_priority* variable, just described, and removal of initialization of *BE*, which has been moved to *init_csma_ca()*. This is done to move all TRADIF parameters to one module. *Init_csma_ca()*,shown in Figure 5.4, has been extended to include the initialization of the tradif parameters *macMinBE, aMaxBE , CW_init* and also *BE*.

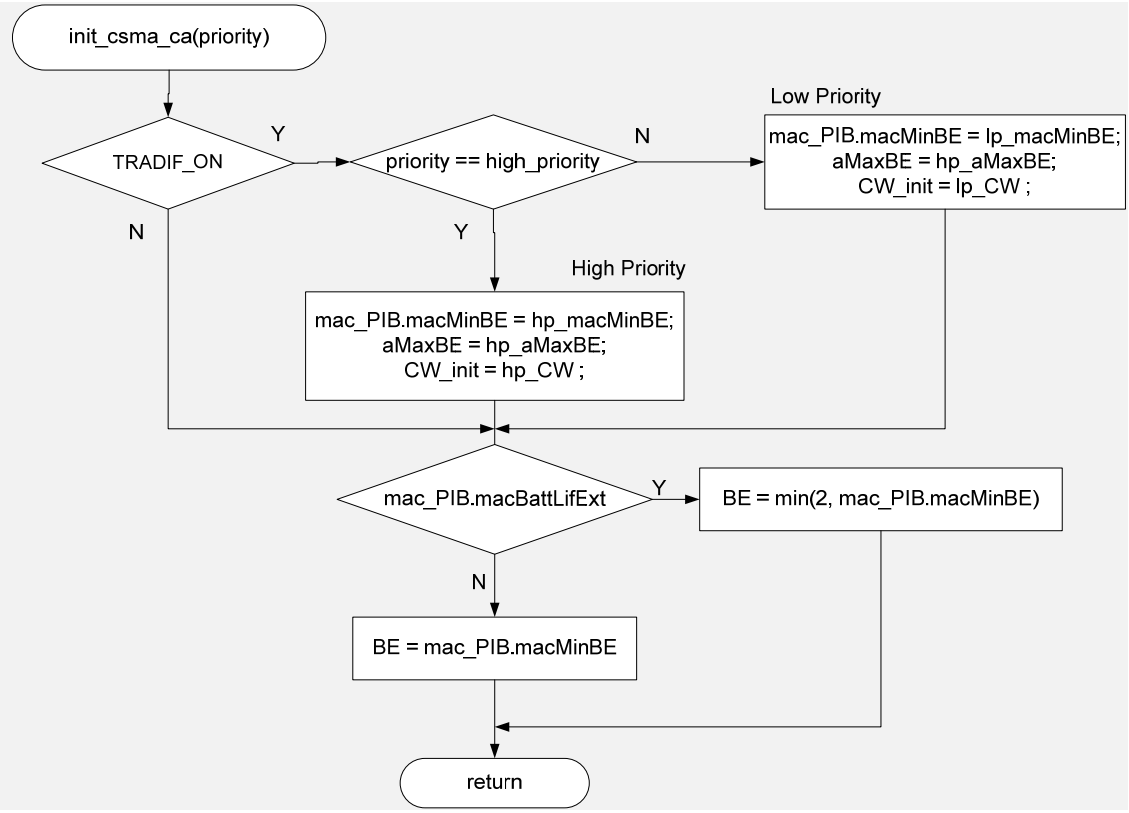


Figure 5.4: TRADIF *init_csma_ca()* flowchart

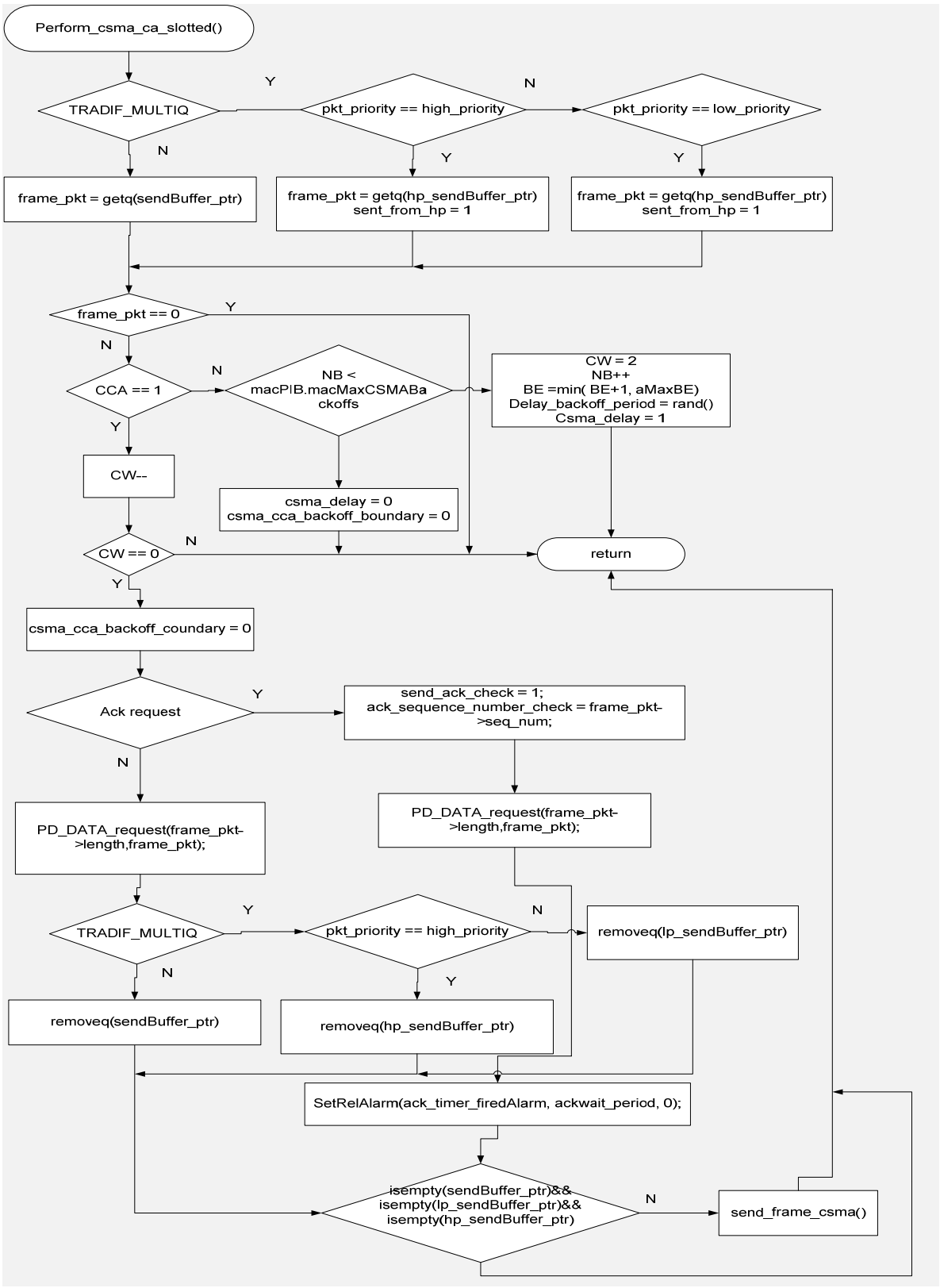


Figure 5.5: TRADIF *perfrom_csma_ca_slotted()* flowchart

5.4 Performance Evaluation

To study the effects of parameter tuning and priority queuing, we measured the success probability of packet delivery for high priority packets against increasing traffic load in various scenarios.

5.4.1 Experimental setup

The experimental setup consisted of five devices programmed with TRADIF as described in the preceding section. One of these was programmed as Coordinator and the rest four as end devices. The end devices were used to generate traffic, both high and low priority, while the Coordinator, apart from synchronizing the devices by generating beacons, was also used to manage the experiment by transmitting control information through beacon payload. This included the amount and type of traffic to be generated by the end devices and signals to start and end the experiment. The traffic generated by the end devices contended for the medium using TRADIF version of slotted CSMA/CA algorithm with different values of CSMA parameters in different scenarios. The Chipcon packet sniffer was used to read the packets transmitted through the medium and throughput measurements were obtained by parsing the sniffer readings.

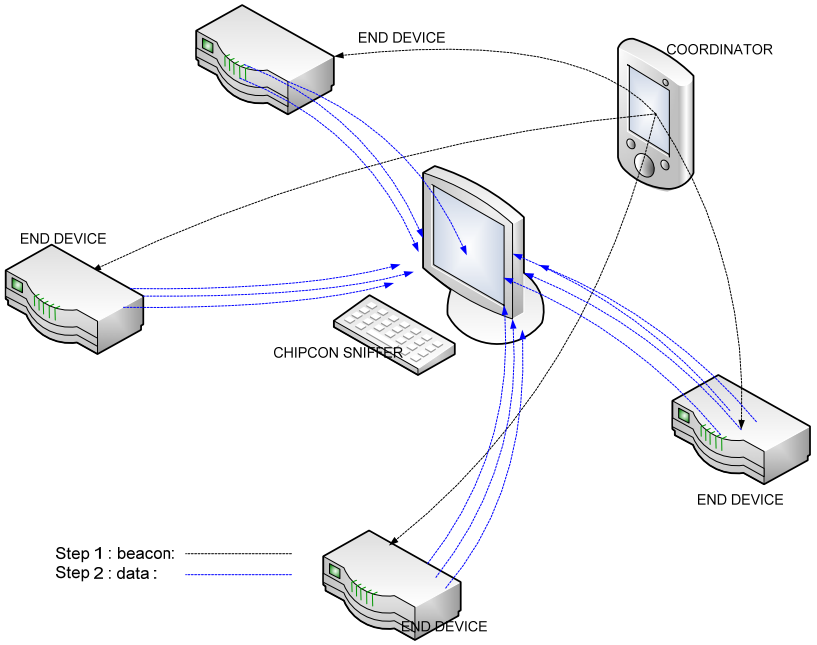


Figure 5.6: Experimental setup for TRADIF evaluation

To start an experiment, the end devices were first reset, set to receive beacons. The Coordinator was then set to transmit beacons and control information in payload, specifying traffic loads of both kinds to be generated. The end devices, upon receiving the beacon, would set the traffic generator alarms (of both high and low priority), with intervals as specified in the beacon payload. The alarms *generate_hp_trafficAlarm* and *generate_lp_trafficAlarm* have been implemented for the purpose of traffic generation and call *generate_hp_traffic()* and *generate_lp_traffic()* functions upon firing. These functions generate high priority and low priority frames respectively, setting the application payload to indicate priority. Although the TRADIF implementation by default treats commands as high priority and data as low priority traffic, it was modified for the purpose of testing where data frames were used for both high and low priority traffic, with the application payload field defined to determine traffic type.

The rate of traffic generation was determined by the traffic generator alarms frequency, with a unit corresponding to a backoff period. Thus a frequency of 100 for high priority traffic generator would mean that a high priority data packet would be generated every 100 backoff period. However, since at higher generation rates these traffic generator tasks could themselves be preempted by higher priority tasks, it was not an accurate measure of traffic actually generated. The actual traffic generated was thus calculated by inserting traffic generation counters in application payload when frames were constructed. The counters are described in more detail in the next section.

5.4.2 Measurements Technique

To measure output parameters such as throughput, queue overflows and delays, the strategy used was to insert counters at various stages of the transmission procedure, starting from the traffic generation at application layer to transmission from the physical layer. For example, high priority packet counter at application level, *hp_app_counter* was used to count the number of high priority frames generated by an end device from the beginning of the experiment to the instant of current frame creation. It was incremented with every call to *generate_hp_traffic()* and inserted into the network payload of the high priority frames. The following counters were used:

- *lp_app_counter*, *hp_app_counter*: Aplication layer counters for high and low priority traffic, incremented with the each high priority frame generated

- *lp_queued, hp_queued*: Counters representing the number of high and low priority packets successfully enqueued
- *lp_mac_sent, hp_mac_sent*: Counters representing the number of packets transmitted after completing the CSMA/CA procedure
- *lp_csma_fail, hp_csma_fail*: Counters representing failed CSMA/CA transmissions.
- *lp_last_csma_delay_backoff_period, hp_last_csma_delay_backoff_period:* Counters representing the CSMA delay in the last transmission of respective priorty classes, in terms of the number of backoffs

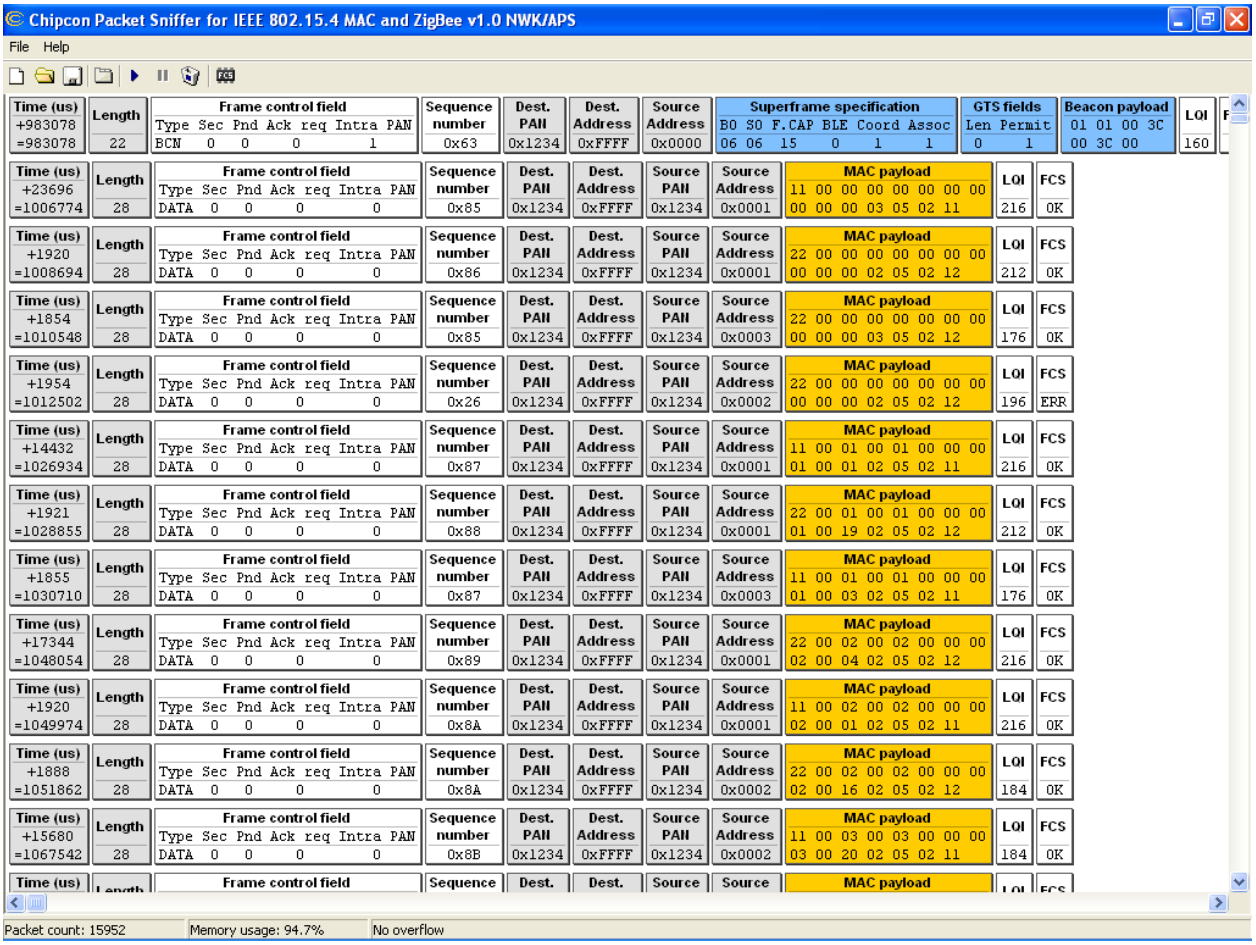


Figure 5.7: Sniffer snapshot showing counters in data frames

These counters were read by parsing the sniffer files (Figure 5.7) and reading the Mac payload of each successive frame. From these counters, the number of frames generated of each type,

number of frames enqueued, number of frames successfully completing CSMA/CA, and number of backoff delays for each frame were calculated. Numbers of successfully transmitted frames of each type are given by the number of packets received at the sniffer. Required measurements were then obtained using the following formula:

Application Layer traffic

*Gapp_lp_data = (g_lp_app_packet_nbr * DATA_PACKET_SIZE) / (*250000*total_time_sec);*
*Gapp_hp_data = (g_hp_app_packet_nbr * DATA_PACKET_SIZE) / (*250000*total_time_sec);*
*Gapp_data = (g_app_packet_nbr * DATA_PACKET_SIZE) / (*250000*total_time_sec);*

Gapp_lp_data denotes low priority traffic generated by the Application Layer; *Gapp_hp_data* denotes high priority traffic generated by the Application Layer and *Gapp_data* total traffic generated by the Application Layer, all three as fractions of the overall network capacity (250 kbps).

Traffic Enqueued

*Gmac_lp_queued = (g_lp_queued * DATA_PACKET_SIZE) / (250000*total_time_sec);*
*Gmac_hp_queued = (g_hp_queued * DATA_PACKET_SIZE) / (250000*total_time_sec);*
*Gmac_queued = (g_total_queued * DATA_PACKET_SIZE) / (250000*total_time_sec);*

Gmac_lp_queued, *Gmac_hp_queued* and *Gmac_queued* denote low priority, high priority and total traffic, respectively, successfully enqueued in send buffers, all as fractions of total network capacity.

Mac Traffic *(Undergoing CSMA/CA)*

*Gmac_lp_data = (g_lp_mac_sent * DATA_PACKET_SIZE) / (250000*total_time_sec);*
*Gmac_hp_data = (g_hp_mac_sent * DATA_PACKET_SIZE) / (250000*total_time_sec);*
*Gmac_data = (g_mac_sent * DATA_PACKET_SIZE) / (250000*total_time_sec);*

Similarly, *Gmac_lp_data*, *Gmac_hp_data* and *Gmac_data* denote low priority, high priority and total traffic, undergoing CSMA/CA procedure, all as fractions of total network capacity.

Total Traffic **(**Including Beacons)

*Gmac_all= ((g_mac_sent+g_beacon_nbr) * DATA_PACKET_SIZE)/ 250000*total_time_sec);*

Successful Transmission

*S_lp = (g_lp_tx_success * DATA_PACKET_SIZE) / (250000*total_time_sec);*
*S_hp = (g_hp_tx_success * DATA_PACKET_SIZE) / (250000*total_time_sec);*
*S = (g_rx_packet_nbr * DATA_PACKET_SIZE) / (250000*total_time_sec);*

S_lp and *S_hp* and *S* denote the amount of low priority, high priority and total traffic successfully transmitted (received by the sniffer).

Application layer traffic probability of success

Ps_lp_app = S_lp / Gapp_lp_data;
Ps_hp_app = S_hp / Gapp_hp_data;
Ps_app = S/Gapp_data;

Ps_lp_app, *Ps_hp_app* and *Ps_app* represent the average probability of success of a low, high and any frame generated by the Application layer.

Mac layer traffic probability of success

Ps_lp_mac = S_lp / Gmac_lp_data;
Ps_hp_mac = S_hp / Gmac_hp_data;
Ps_mac = S/Gmac_data;

Ps_lp_mac, *Ps_hp_mac* and *Ps_mac* represent the average probability of success of a low, high and any frame undergoing CSMA/CA.

5.5 Results and Discussions

The first set of experiments consist of varying low priority traffic while keeping high priority traffic constant, and measuring throughput of the high priority traffic for various differentiation scenarios. The values of CSMA parameters used for each of these scenarios are listed in table 5.1. Each case was examined for FIFO as well as Priority Queuing scheduling policies.

The traffic generation was controlled using traffic generator alarms, with one high priority frame generated every 60 backoffs and varied from 1000 to 5 backoffs for low priority traffic. However, since the alarm frequency alone does not accurately determine the generated and transmitted traffic (due to tasks preemptions and queuing losses), we measured transmission success probability against traffic generated at Application layer as well as traffic undergoing CSMA/CA at MAC layer ('Application layer traffic' – 'queuing losses'). In the following discussions, Application layer traffic is denoted by *Gapp* and the MAC layer traffic by *Gmac.* Similarly, *Gapp_hp* and *Gapp_lp* are used to denote Application layer high priority and low priority traffics, and *Gmac_hp*, *Gmac_lp* used for MAC layer high and low priority traffic, respectively.

Scenario	[$macMinBE_{LP,}$ $macMinBE_{HP}$]	CW_{LP}	CW_{HP}
Sc1	[2,2]	2	2
Sc2	[2,2]	3	2
Sc3	[2,0]	2	2
Sc4	[2,0]	3	2

Table 5.1: Test Scenarios

The four graphs of Figures 5.8 show the success probabilities of Application layer high priority frames with increasing Application traffic (*Gapp*), for each of the four test scenarios of Table 5.1,with FIFO Queuing. The graphs of Figures 5.9 show the success probabilities of high priority frames undergoing CSMA/CA (*Gmac*) for each of the four test scenarios, again with FIFO Queuing. The four graphs of Figures 5.10 show the success probabilities of Application layer high priority frames against with increasing *Gapp*, for each of the above test scenarios with Priority Queuing, and finally, the the graphs of Figures 5.11 show the success probabilities of high priority frames undergoing CSMA/CA in Mac (*Gmac*), again with Priority Queuing.

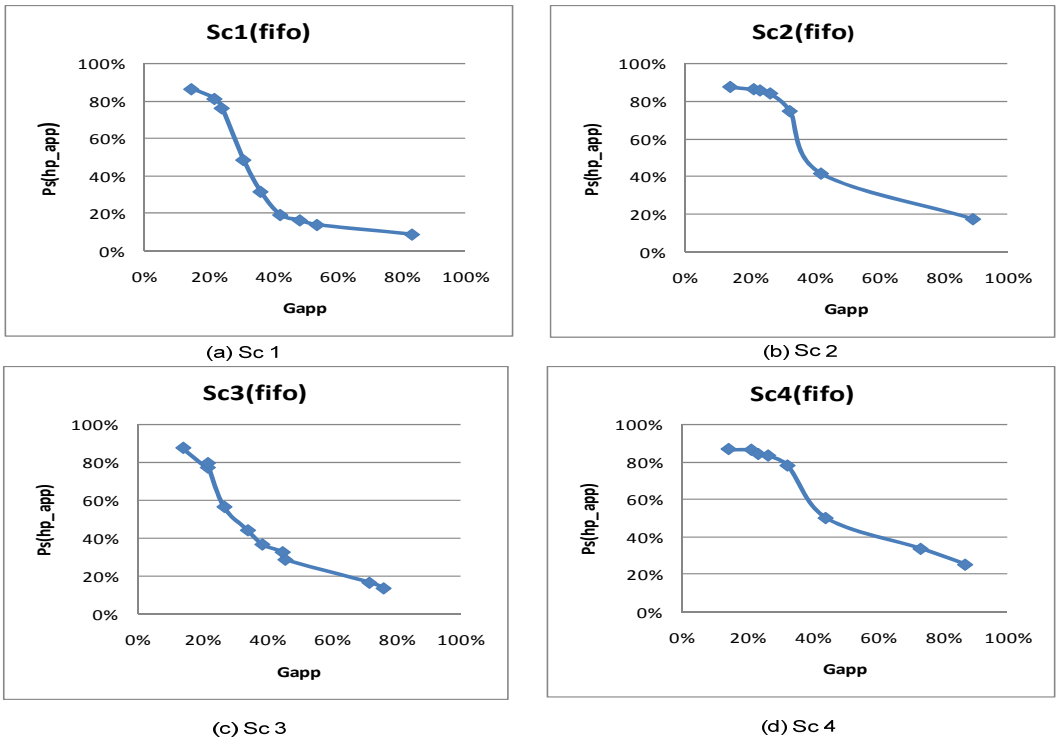


Figure 5.8: Success probability of Application traffic: Sc [1-4] with FIFO Queuing

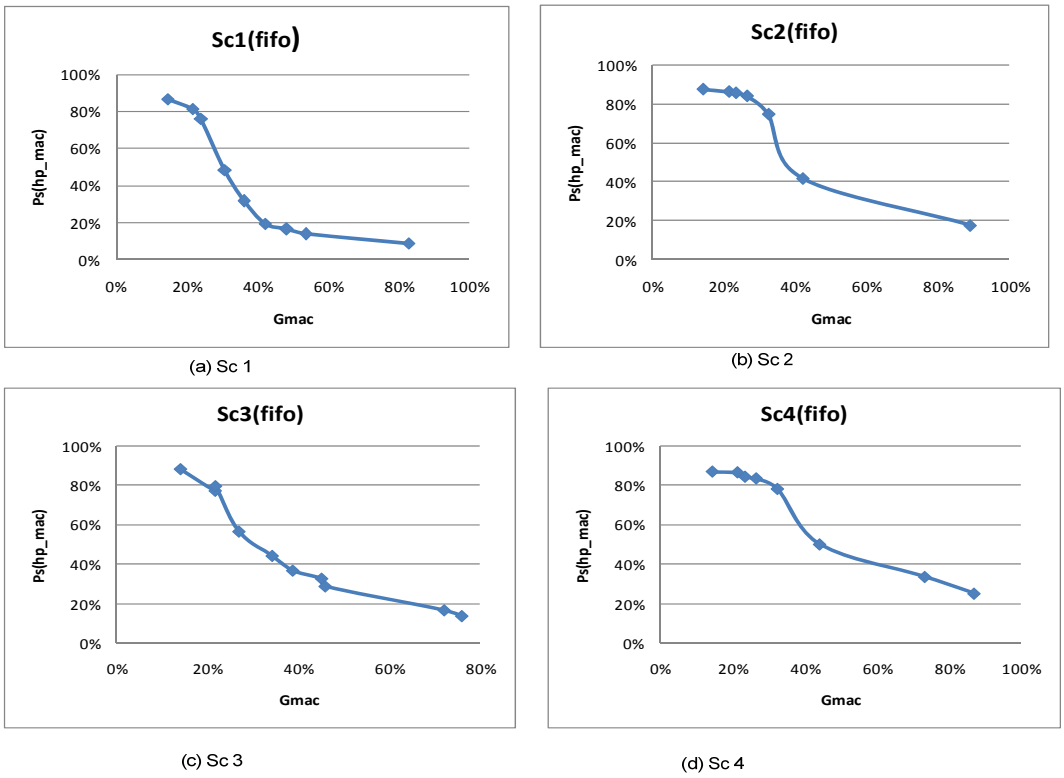


Figure 5.9: Success probability of MAC traffic: Sc [1-4] with FIFO Queuing

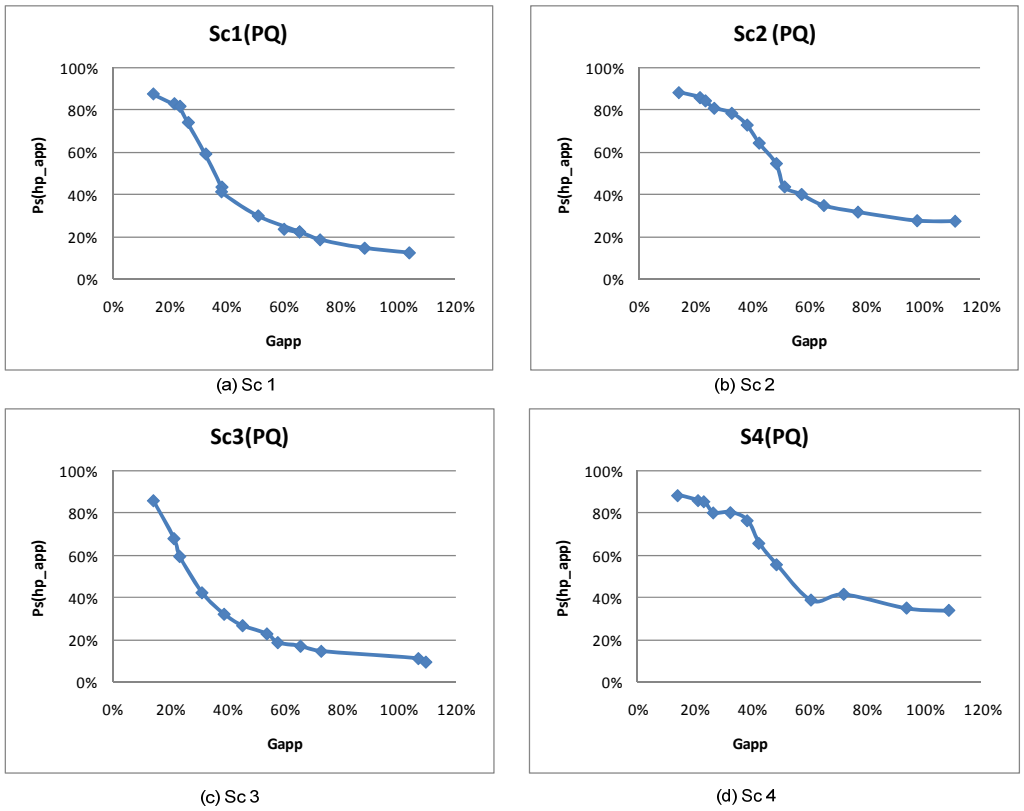


Figure 5.10: Success probability of Application traffic: Sc [1-4] with Priority Queuing

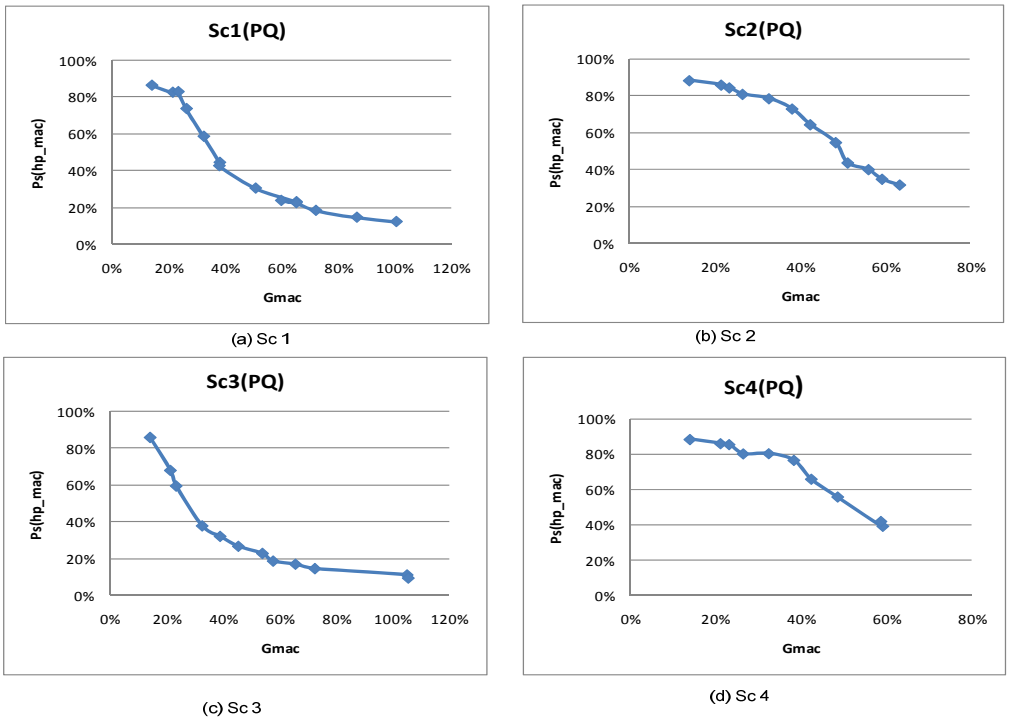


Figure 5.11: Success probability of MAC traffic: Sc [1-4] with Priority Queuing

Figure 5.12 shows the comparison of the success rates of the high priority Application traffic of the four scenarios of Figure 5.8. Queuing mode in all four cases is FIFO. The contention windows size for high priority frames is kept 2 (standard value) in all cases, while it is increased to 3 for low priority frames in Sc2 and Sc4. On the other hand, the value of *macMinBE* is kept constant (2, standard value) for low priority traffic in all cases, whereas it is set to 0 for high priority traffic in Sc3 and Sc4.

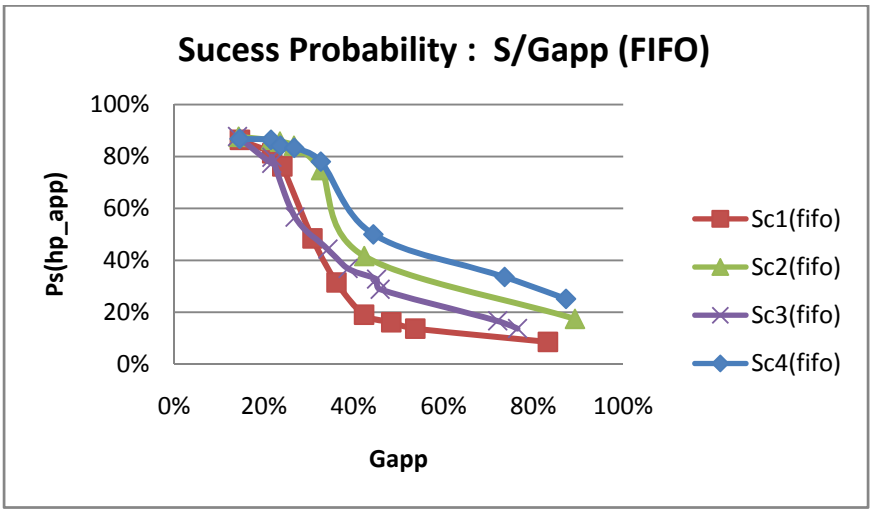


Figure 5.12: Success Probability (*Gapp*): Comparing four scenarios with FIFO Queuing

From the graphs it can be observed that all three scenarios of parameter tuning (Sc [2-4]) result in higher success rates compared to the standard case (Sc1). The order of increasing success probabilities is *P(Sc4)>P(Sc2)>P(Sc3)>P(Sc1)*. Sc1, which is the standard case, has the lowest success probability. Sc3, in which $macMinBE_{HP}$ is decreased to 0, results in improved success rates, but it is still very close to the standard case (change of 0-5%). This is so because setting $macMinBE_{HP}$ lower than $macMinBE_{LP}$ means lower backoff delays for high priority traffic (refer to slotted CSMA/CA algorithm, Figure 2.7), but the number of backoffs and contention window size, which are directly related to the contention success probability, are unchanged. On the other hand, setting CW_{LP} greater than CW_{HP} means that high priority traffic need the channel to remain idle for shorter time before transmitting, which means higher probability of success in every sensing attempt. The comparatively higher success rates in Sc2 and Sc4 (improvement of 20-

25%) reflect this, showing greater improvement in performance by setting $CW_{LP} > CW_{HP}$, compared to by changing $macMinBE_{HP}$.

Figure 5.13 compares the success probabilities of the four cases against *Gmac*. The queuing mode in this case is FIFO. The difference from the previous case (Figure 5.12) is that while there the packets lost because of queue overflow were counted among failed deliveries, here those are excluded from the calculations, and the failed deliveries are of channel contention. Thus it is a more accurate reflection on the effects of parameter tuning on CSMA/CA performance. However, the results are very close to those in Figure 5.12 because queuing losses were negligible (less than 1%). The order of success probabilities, as in the previous case, is *P(Sc4)>P(Sc2)>P(Sc3)>P(Sc1).* This confirms the greater dependency of the contention window size on the performance, compared to the value *macMinBE*.

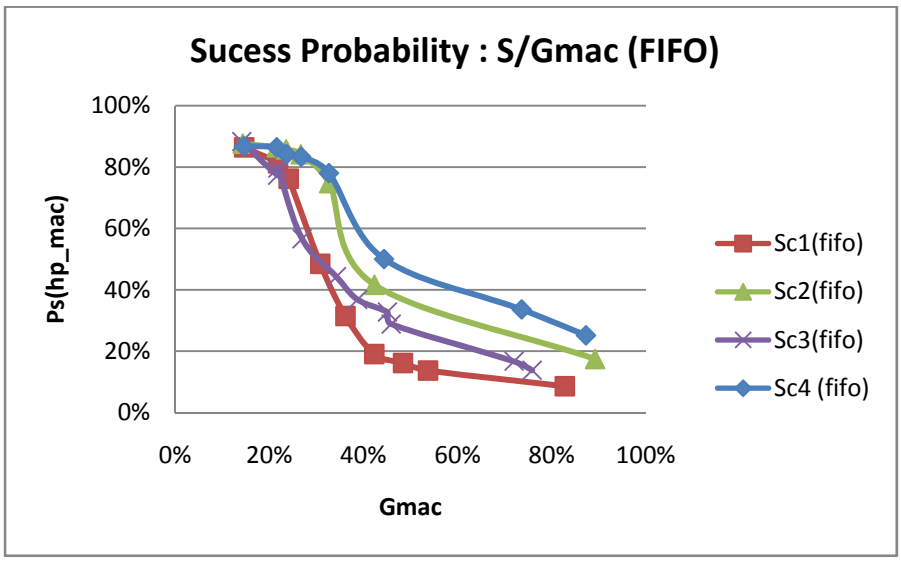


Figure 5.13: Success Probability (*Gmac*): Comparing four scenarios with FIFO Queuing

Figure 5.14 compares the success probabilities of high priority Application layer traffic for the four cases against in Priority Queuing mode. One of the noticeable changes from the FIFO cases is the fall of success probability of Sc3. This is so because with priority queuing, high and low priority frames go to separate queues and the high priority frames are picked first irrespective of the order of arrival or the state of the low priority queue. As such, the effect of changing $macMinBE_{HP}$, which would decrease the backoff delay of high priority packet, does not make

any difference on contention success. Ideally, Sc1 and Sc3 should have the same success rates, which is the case at higher traffic loads.

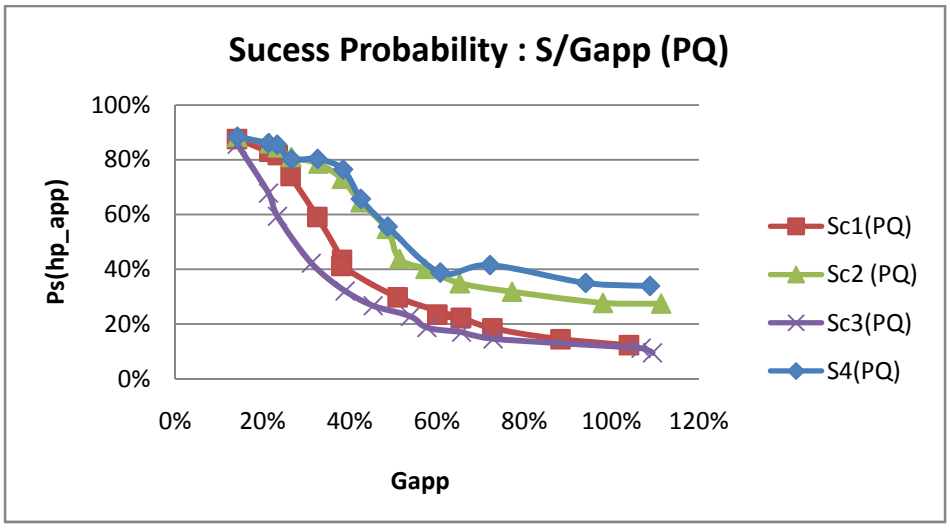


Figure 5.14: Success Probability (*Gapp*): Comparing four scenarios with Priority Queuing

Sc2 and Sc4 again have better success rates since setting having CW_{HP} lesser than CW_{LP} means that high priority traffic need the channel to remain idle for shorter time before transmitting and hence has more chances of success. In this case again, changing CW_{LP} to 3 improves the success rate of high probability packet by 15 to 20%.

Figure 5.15 shows the comparison of the success probabilities of high priority Mac layer traffic in priority queuing mode. Sc2 and Sc4 again have an improvement of 15-20% over Sc1 and Sc3 till the traffic reaches around 55-60%. However, as can be seen from the graph, the traffic in Sc2 and Sc4, in this case, do not go beyond 60%. This is so because increased low priority contention window size (CW_{LP}) results in increased delays in the transmission of low priority frames which in turn means that the low priority frames are removed from the low priority queue at a lower rate. Beyond this rate, the frames generated by the Application layer end up as queue overflow, without any increase in total Mac layer traffic.

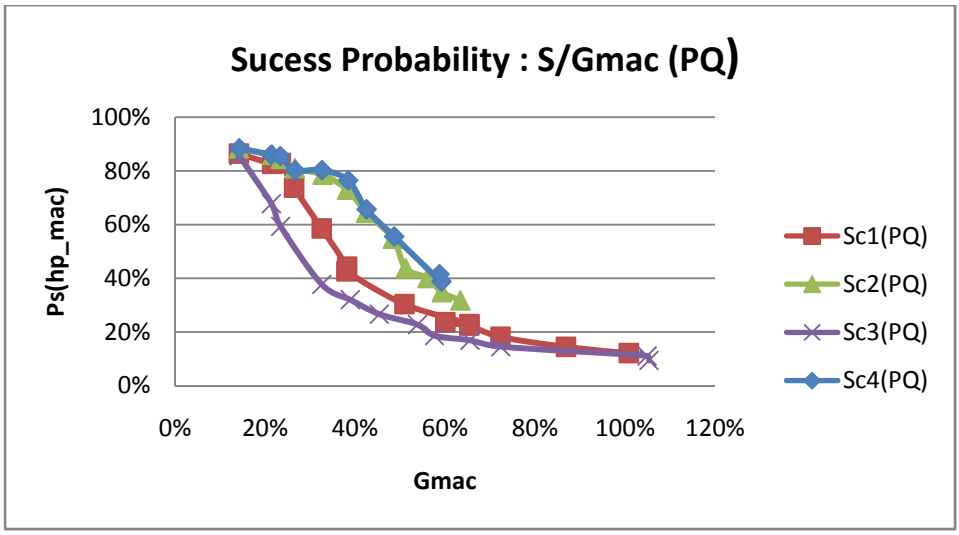


Figure 5.15: Success Probability (Gmac): Comparing four scenarios with Priority Queuing

To separately evaluate the effect of priority queuing mechanism, a single sender was used to generate equal amount of high and low priority frames. The queue size for both high and low priority queues were set to be 15. The Application layer traffic generation rate was increased at equal rate. The number of packets enqueued of both types were calculated by parsing the output file of the sniffer used to receive packets. Figure 5.16 shows the packets enqueued against the packets generated by the application of both high and low priority. It can be seen that beyond 20% of channel capacity, while the low priority frames are dropped due to queue overflow, the

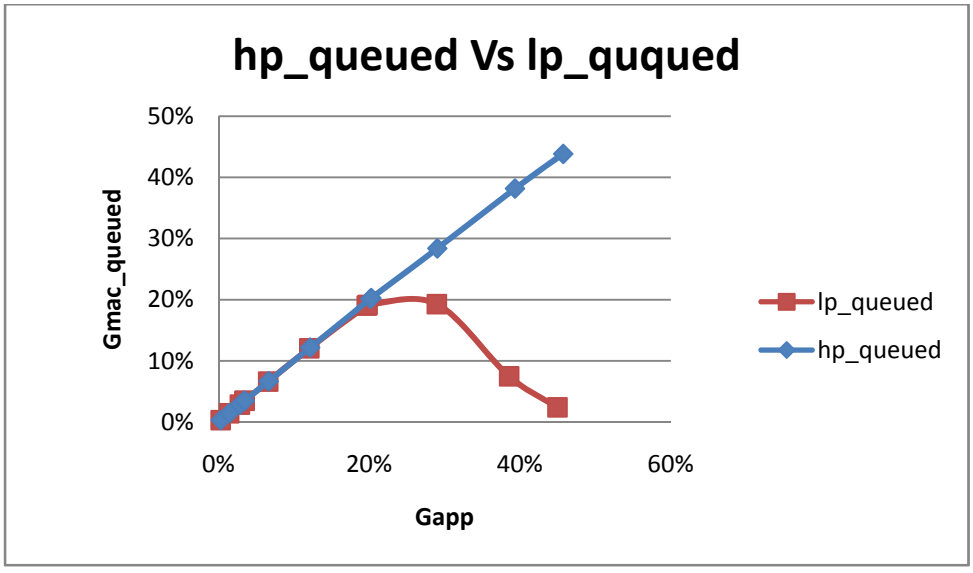


Figure 5.16: Comparing queuing success in Priority Queuing mode

high priority frames are unaffected. This indicates that at high traffic load, priority queuing has an important role in ensuring the precedence of high priority frames.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Conclusions

Increased uses of WSNs in time-critical applications such as industrial automation and process control have led to the demands of real-time support from WSN communication protocols. The IEEE802.15.4/ZigBee protocol set, by virtue of their dynamically adjustable duty-cycles and support for real-time bandwidth allocation using the GTS (Guaranteed Time Slot) mechanism, show excellent potential to meet such demands. However, previous attempts to build the stack over TinyOS (the most popular OS for sensor nodes) could not produce predictable temporal behavior of tasks because of the lack of real-time support from the kernel. To overcome these limitations, the protocol stack was implemented over ERIKA real-time Operating System. The implemented functionalities include: Data Transmission (direct and indirect) in Contention Access Period (CAP) and Contention Free period (CFP); Transmission with acknowledgement requests (and retransmission for acknowledgement failure); Association mechanism, GTS Allocation and Deallocation, Address Assignment mechanisms, ZigBee Network formation and Tree Routing. All of the implemented functionalities were validated using IEEE802.15.4 compliant packet sniffers. Additionally, Time Division Beacon Scheduling mechanism [19] was added to support Cluster-tree formation in synchronized mode.

While the stack implementation provides support for guaranteed bandwidth allocations using GTS mechanism, its scope is limited to Contention Free Period and also to a limited number of devices. To extend the QoS support to Contention Free Period and an unlimited number of devices, priority based service differentiation was introduced in the slotted CSMA/CA (the channel access algorithm used in CAP). This included addition of priority queuing to lower queuing delays of high priority frames as well as priority based parameter tuning to favour the high priority frames in channel contention. A number of test scenarios with different parameter and queuing combinations were studied and the results confirmed the achievement of a greater success rate for high priority frames.

6.2 Suggestions for Future Work

The Zigbee specifications restrict multi-hop networking in the beacon-enabled mode to star-based networks. Additional mechanisms are thus needed to support operational cluster tree formation in beacon enabled mode. In this direction, Time Division Beacon Scheduling mechanism [19] has been implemented which enables cluster tree formation in beacon-enabled mode, but the current operation is limited to unidirectional communication. The next step is the splitting of transmission buffers into upstream and downstream buffers to enabling bi-directional communication in Cluster-tree networks. This will allow large scale distributed operation of nodes in synchronized mode. In traffic differentiation evaluation, practical difficulties were encountered in high rate traffic generation because of the small size of the test bed. An evaluation on a larger test bed is desirable. A larger test bed will also enable the evaluation of the mechanism for hidden node scenarios [20].

REFERENCES

[1] Manish Batsa, Ricardo Severino, and Mário Alves, “Supporting Different QoS Levels in Multiple-Cluster WSNs,” in Proceedings of the 10th Portuguese Thematic Network on Mobile Communications Workshop (RTCM), Porto, Portugal, June, 2009

[2] TinyOS, www.tinyos.net, 2009

[3] André Cunha, Ricardo Severino, Nuno Pereira, Anis Koubâa, and Mário Alves” ZigBee over TinyOS: implementation and experimental challenges,” 8th Portuguese Conference on Automatic Control (CONTROLO’2008), Vila Real, Portugal, 21-23 July, 2008.

[4] A. Dunkels, B. Grnvall, and T. Voigt. “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I), Tampa, Florida, USA, November 2004.

[5] A. Eswaran, A. Rowe and R. Rajkumar. “Nano-rk: An energy-aware resourcecentric operating system for sensor networks,” in Proceedings of IEEE Real-Time Systems Symposium, 2005.

[6] ERIKA Real-time operating system, http://erika.sssup.it/, 2009

[7] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He, “RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks,” in IEEE Real-Time and Embedded Technology and Applications Symposium, 2002.

[8] T. Abdelzaher, J. Stankovic, S. Son, B. Blum, T. He, A. Wood, and C. Lu, "A communication architecture and programming abstractions for real-time embedded sensor networks," in Proceedings of the 23rd International Conference on Distributed Computing Systems, Washington DC, USA: IEEE Computer Society, 2003, p. 220.

[9] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher, "Speed: a stateless protocol for real-time communication in sensor networks," in Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003, pp. 46-55.

[10] IEEE-TG15.4, "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," IEEE standard for Information Technology, 2003.

[11] ZigBee Specification 2006, http://www.zigbee.org/

[12] J. Zheng and J. L. Myung, “Will IEEE 802.15.4 Make Ubiquitous Networking a Reality? A Discussion on a Potential Low Power, Low Bit Rate Standard,” IEEE Communications Magazine, vol. 42, No. 6, 2004, pp. 140- 146.

[13] A. Cunha, A. Koubâa, R. Severino, M. Alves, “Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS,” in Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS´07), Pisa, Italy, October 2007.

[14] J. H. Hauer, “An implementation of the ieee 802.15.4 mac in tinyos 2,” Telecommunication Networks Group, Technische Universit¨at Berlin, Tech. Rep. TKN-08-003, Feb. 2008.

[15] A. Koubâa, M. Alves, E. Tovar, “A Two-Tiered Architecture for Real-Time Communications in Large-Scale Wireless Sensor Networks.”, WIP Session on the 17th Euromicro Conference on Real-Time Systems (ECRTS’05), Palma de Mallorca, Spain, 2007.

[16] The ART-WiSe Framework, www.hurray.isep.ipp.pt/art-wise/, 2009

[17] IPP-HURRAY!, http://www.hurray.isep.ipp.pt/

[18] RETIS Lab, http://retis.sssup.it/

[19] A Koubâa , A Cunha , M Alves, and E Tovar, “TDBS: a time division beacon scheduling mechanism for ZigBee cluster-tree wireless sensor networks,” Real-Time Systems, v.40 n.3, December 2008, pp. 321-354.

[20] A. Koubaa, M. Alves, B. Nefzi, and Y.-Q. Song, “Improving the ieee 802.15.4 slotted csma/ca mac for time-critical events in wireless sensor networks,” in Proceeding of the

Workshop of Real-Time Networks (RTN 2006), Satellite Workshop to (ECRTS 2006), July 2006.

[21] FLEX: Microchip dsPIC evaluation board, “FLEX Embedded Platform Reference Manual,” [Online]. Available: http://www.evidence.eu.com/content/view/114/204/ [Accessed: July, 2009].

[22] RT-DRUIT, “RT-Druid Code generator Plugin Reference Manual,” [Online]. Available: www.evidence.eu.com [Accessed: March, 2009].

[23] MPLAB, “MPLAB ICD 2 In-Circuit Debugger User’s Guide,” [Online]. Available: www.microchip.com/icd3 [Accessed: March, 2009]

[24] Chipcon, Texas Instruments Incorporated, “Chipcon Packet Sniffer for IEEE 802.15.4,” [Online]. Available: www.chipcon.com [Accessed: March, 2009]

[25] Open-ZB, “Open-ZB open-source toolset for the IEEE 802.15.4/ZigBee protocols,” [Online]. Available: http://www.open-zb.net [Accessed: March, 2009]

[26] Microchip, “dsPIC33F Family Data Sheet,” [Online]. Available: www.microchip.com [Accessed: March 2009]

[27] Flexipanel, “2.4GHz ZigBee ready IEEE 802.15.4 RF transceiver,” [Online]. Available: www.flexipanel.com. [Accessed: June, 2009]

[28] Evidence, “Evidence srl,” [Online]. Available: http://www.evidence.eu.com [Accessed: March 2009]

[29] OSEK, “OSEK/VDX-STANDARD,”, [Online]. Available: http://portal.osek-vdx.org [Accessed: July, 2009]

[30] Eclipse, “Eclipse – An open development platform,” [Online] Available :www.eclipse.org, [Accessed: March, 2009]

[31] Crossbow Technology, “MICAz Datasheet,” [Online]. Available: www.xbow.com, [Accessed: July, 2009]

[32] Crossbow Technology, “TelosB Datasheet”, [Online]. Available: www.xbow.com, [Accessed: July, 2009]

[33] P. Pagano, et al., “ERIKA and OpenZB: an implementation for real-time wireless networking,” in 24th ACM Symposium on Applied Computing (SAC 2009), Poster Session, March 2009, pp 1687-1688.

[34] OIL, “OIL: OSEK Implementation Language Version 2.5,” [Online]. Available: http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf [Accessed: July, 2009]

[35] A. Cunha, M. Alves, and A. Koubaa., “An IEEE 802.15.4 protocol implementation (in nesC/TinyOS): Reference Guide v1.2,” IPP-HURRAY Technical Report, HURRAY-TR-061106, Nov 2006.

[36] A. Cunha, M. Alves, and A. Koubaa, “Implementation of the ZigBee Network Layer with Cluster-tree Support,” IPP-HURRAY Technical Report, HURRAY-TR- 070510, May 2007.

[37] IEEE 802.11, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Standard,IEEE, Aug. 1999

[38] IEEE Std 802.11e/D13.0, "Draft supplement to standard for telecommunications and information exchange between systems-LAN/MAN specific requirements. Part 11: Wireless medium access control and physical layer specifications: Medium access control enhancements for quality of service," Apr. 2005.

[39] IEEE 802.11-2007: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2007

[40] Kim, D. Lee, J. Ahn, and S. Choi, "Priority toning strategy for fast emergency notification in IEEE 802.15.4 LR-WPAN," in Proceedings of the 15th Joint Conference on Communications & Information (JCCI), April, 2005.

[41] T. Kim and S. Choi, "Priority-based delay mitigation for event-monitoring IEEE 802.15.4 LR-WPANs," IEEE Communications Letters, Nov. 2005, pp. 213-215.

[42] A. Koubaa, M. Alves, E. Tovar, “A Comprehensive Simulation Study of Slotted CSMA/CA for IEEE 802.15.4 Wireless Sensor Networks,” In IEEE WFCS 2006, Torino (Italy), June 2006, pp.183-192.

[43] OPNET Technologies, Inc., “Opnet Modeler Wireless Suite - ver. 11.5A,” [Online] Available: http://www.opnet.com [Accessed: July 2009]

[44] D. Kipnis, A. Willig, J. H. Hauer, and N Karowski,” The ANGEL IEEE 802.15.4 Enhancement Layer: Coupling Priority Queueing and Service Differentiation,” In Proceedings of 14th European Wireless Conference, Prague, June 2008, pp.1-7.