Technical Report

Implementation of the ZigBee Network Layer with Cluster-tree Support

André CUNHA

Mário ALVES

Anis KOUBÂA

TR-070510

Version: 1.0

Date: 26-05-2007

Implementation of the ZigBee Network Layer (in nesC/TinyOS)

André CUNHA, Mário ALVES, Anis KOUBÂA

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: arec@isep.ipp.pt, mjf@isep.ipp.pt, anis@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

While the IEEE 802.15.4/Zigbee protocol stack is being considered as a promising technology for low-cost low-power Wireless Sensor Networks (WSNs), several issues in their specifications are still open. One of those ambiguous issues is how to build a synchronized cluster-tree network. In fact, the current IEEE 802.15.4/Zigbee specifications restrict the synchronization in the beacon-enabled mode (by the generation of periodic beacon frames) to star-based networks, while they support multi-hop networking using the peer-to-peer mesh topology, but with no synchronization. Even though both specifications mention the possible use of cluster-tree topologies the description on how to effectively construct such a network topology is missing.

This technical describes the implementation details of the ZigBee Network Layer on top of our implementation IEEE 802.15.4 for nesC/TinyOS. This implementation enables the cluster-tree network topology with our proposed mechanism for beacon scheduling in order to enable an efficient use of synchronized cluster-tree networks. This implementation induces minor changes to our IEEE 802.15.4 implementation, the open-ZB (www.open-zb.net), in order to implement our proposed Time Division Beacon Scheduling approach.

This technical report describes the implemented mechanisms and gives an intuition how to effectively use this implementation with a ZigBee Cluster-tree network topology.

Contents

Figures

Tables

Acronyms and abbreviations

ACL	access control list
AES	advanced encryption standard
BE	backoff exponent
BER	bit error rate
BI	beacon interval
BO	beacon order
BPSK	binary phase-shift keying
BSN	beacon sequence number
CAP	contention access period
CBC-MAC	cipher block chaining message authentication code
CCA	clear channel assessment
CFP	contention-free period
CID	cluster identifier
CLH	cluster head
CRC	cyclic redundancy check
CSMA-CA	carrier sense multiple access with collision avoidance
CTR	counter mode
CW	contention window (length)
DSN	data sequence number
DSSS	direct sequence spread spectrum
ED	energy detection
FCS	frame check sequence
FFD	full-function device
FH	frequency hopping
FHSS	frequency hopping spread spectrum
GTS	guaranteed time slot
IFS	interframe space or spacing
LAN	local area network
LIFS	long interframe spacing
LLC	logical link control
LQ	link quality
LQI	link quality indication
LPDU	LLC protocol data unit
LR-WPAN	low-rate wireless personal area network
LSB	least significant bit
MAC	medium access control
MCPS MAC	common part sublayer
MCPS-SAP	MAC common part sublayer-service access point
MFR	MAC footer
MHR	MAC header
MLME	MAC sublayer management entity
MLME-SAP	MAC sublayer management entity-service access point
MSB	most significant bit
MSC	message sequence chart
MPDU MAC	protocol data unit
MSDU MAC	service data unit
NB	number of backoff (periods)
OSI	open systems interconnection
PAN	personal area network
PD-SAP	PHY data service access point
PDU	protocol data unit
PER	packet error rate
PHR	PHY header
PHY	physical layer
PIB	PAN information base
PLME	physical layer management entity
PLME-SAP	physical layer management entity-service access point
PPDU	PHY protocol data unit
PSDU	PHY service data unit
RF	radio frequency
RFD	reduced-function device
RSSI	received signal strength indication
RX	receive or receiver
SAP	service access point
SD	superframe duration
SPDU	SSCS protocol data units
SDU	service data unit
SFD	start-of-frame delimiter
SHR	synchronization header
SIFS	short interframe spacing
SO	superframe order
SRD	short-range device
SSCS	service specific convergence sublayer
TRX	transceiver
TX	transmit or transmitter
WLAN	wireless local area network
WPAN	wireless personal area network

1 Introduction

This technical report gives a brief overview of the ZigBee Network Layer mechanisms[1], namely the association and addressing schemes, the tree-routing and some features of the Network Layer Information Base.

The proposed implementation supports the cluster-tree network topology using a beacon scheduling mechanism, as proposed on [2], to effectively schedule the transmission offsets of every ZigBee router in the cluster-tree. In this implementation, we opted on a basic approach for the beacon scheduling based on time division and on a negotiation mechanism before beacon transmission. The ZigBee Coordinator (ZC) is responsible for the scheduling. Currently, the ZC has a static definition of the schedule based on the routers short addresses. For getting the implementation details of the IEEE 802.15.4[3], the interested reader is referred to [4,5].

Note that in this implementation, some add-ons were needed in the IEEE 802.15.4 MAC sublayer[3], namely the division of the send buffer into two new buffers: the *downstream buffer* for the transmissions downstream in the network tree and the *upstream buffer* for the upstream transmissions. In addition, note that some of the implemented network layer mechanisms were modified, namely, the inclusion of a new active period shared with the parent device.

2 Overview of the IEEE 802.15.4/ZigBee Address Assignment and Tree Routing mechanisms

2.1 Introduction

In ZigBee Networks there are 3 types of devices:

- **ZigBee Coordinator – ZC**
  - One for each ZigBee Network;
  - Initiates and configures the Network formation;
  - Acts as an IEEE 802.15.4 Personal Area Network (PAN) coordinator;
  - Acts as ZigBee Router (ZR) once the network is formed;
  - Is a Full Functional Device (FFD) – implements the full protocol stack.
- **ZigBee Router – ZR**
  - Not used in star topology networks;
  - Associates with ZC or with previously associated ZR;
  - Acts as an IEEE 802.15.4 PAN coordinator;
  - Participates in multi-hop routing of messages.
  - Is a Full Functional Device (FFD) – implements the full protocol stack.
- **ZigBee End Device –ZED**
  - Does not allow other devices associate with it;
  - Does not participate in routing;
  - Can be a Reduced Function Device (RFD) – implements a reduced subset of the protocol stack.

Throughout this document the names of the devices and the acronyms are used interchangeably.

The tree-routing relies on a distributed address assignment mechanism that provides to each potential parent (ZC and ZRs) a finite sub-block of unique network addresses based on the maximum number of children, depth and the number of routers in the PAN.

For setting up a Cluster-Tree Wireless Sensor Network (WSN), 3 Network parameters must be defined at the ZC. The addressing and tree routing mechanisms will operate according to these parameters, outlined next:

- the maximum number of children (*Cm*) of a ZR;
- the maximum number of child routers (*Rm*) of a ZR;
- the depth of the network (*Lm*).

2.2 Association Mechanism

The association procedure takes place when a node (ZR or ZED) device wants to join the network. To proceed with the association the device must scan all radio channels, so that it can select the most suitable PAN. The association is necessary if the device wants to transmit data in the PAN. As a result of the association mechanism, the device is assigned with a short address allowing it to transmit in the PAN.

The device associates with the PAN Coordinator, defining its characteristics in the capability information field of the association request command. Figure 1 presents the association request frame command format.

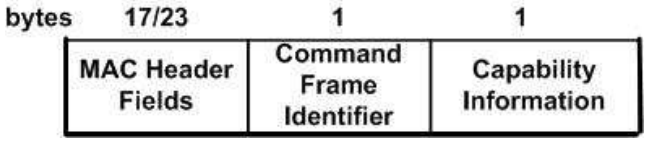

Figure 1 – Association request command format [3]

Besides the standard MAC frame fields, the frame has an addressing field, a command frame identifier field and the capability information of the device.

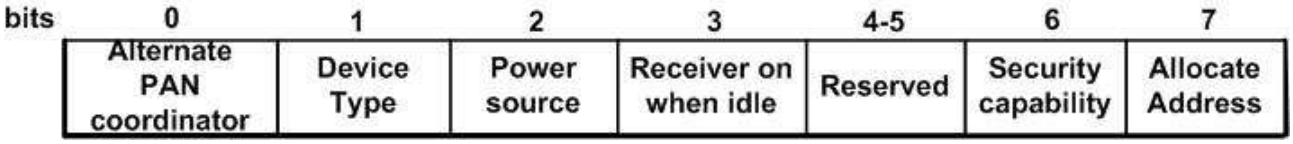

Figure 2 – Association request capability information field format [3]

The capability information field contains the following information:

- Alternate PAN coordinator – 1 if the device is capable of becoming a PAN coordinator (assuming that the device can be a router);
- Device type – 1 – FFD; 0 –RFD;
- Power source – 1 if the device is main powered;
- Receiver on when idle – 1 if the receiver is on during the inactive period;
- Security – 1 if the device is capable of sending and receiving secured MAC frames with a security suite;
- Allocate address – 1 – if the device wants a short address; 0 if the device wants to communicate with the 64 bits extended address.

Upon the reception of the association request command frame, the ZC will process the command and signal the network layer using the *MLME_ASSOCIATE.indication* primitive. The network layer will process the request, by allowing/disallowing the association and assigning a short address (according to the address assignment functions - refer to section 2.2), and issues the MAC layer with the *MLME_ASSOCIATE.request* primitive stating the result of the request. The command is stored in the indirect transmission buffer being transmitted after a data request from the associating device.

Figure 3 presents the association response command format, received by the device requesting the association.

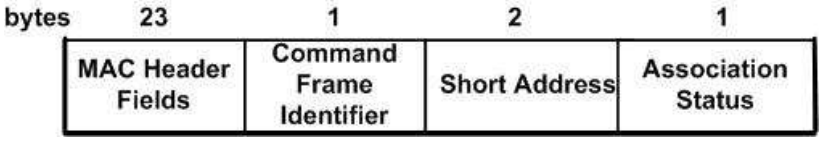

Figure 3 – Association response command format [3]

Besides the standard MAC frame fields, the association response command frame contains the assigned short address (the possible values are presented in figure 4) and the association status stating the reason of success or failure of the association procedure.

Value of *macShortAddress*	Description
0 x 0000–0 x fffd	The device shall use short addressing mode.
0 x fffe	The device shall use 64 bit extended addressing mode with an address consisting of *aExtendedAddress*.
0 x ffff	The device is not associated and shall not communicate on the PAN.

Figure 4 – Association request short address values [3]

Note that the ZigBee Coordinator short address is always 0x0000.

After a successful association there is an update of the MAC PAN Information Base (PIB), the MAC layer of the device stores the following association parameters:

- *a_LogicalChannel* – Logical channel of the PAN;
- *a_CoordAddrMode* – Coordination address mode;
- *a_CoordPANId* – Coordinator PAN id;
- *a_CoordAddress* – Coodinator address, depending on the address mode;
- *a_CapabilityInformation* – Capability information of the device when the association request was sent.
- *a_securityenable* – Security enable stating if the device is using security or not.

In the network layer, each device maintains a neighbour table with the information on every device within its transmission range. In our implementation each device neighbour table contains the following information:

- *PAN_Id* – PAN 16 bits short address
- *Extended Address* – Device 64 bits extended address (if possible);
- *Network Address* – Device 16 bits short address;
- *Device Type* – Device Type (Coordinator; Router; End Device);
- *Relationship* – Relation between the neighbour and the current device (Parent; Child; Sibling; Other);
- *Depth* – Optional Field – Depth of the device in the network;
- *Permit Joining* – Information about the device availability to accept associations;
- *Logical Channel* – Device Logical Channel;
- *Potential Parent* – Indication of whether the device has been ruled out as a potential parent due to a failed join attempt.

2.3 Short Address Assignment

A parent device uses the *Cm*, *Rm*, and *Lm* values to compute a *Cskip* function defining the size of the address sub-block that is distributed by each parent depending on its depth (*d*) in the network. For a given network depth *d*, *Cskip(d)* is calculated as follows:

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

A parent device that has a *Cskip(d)* value of zero will not be capable of accepting children and shall be treated as an end device.

A parent device that has a *Cskip(d)* value greater that zero shall accept devices and shall assign addresses if possible.

A parent device will assign an address that is one greater than its own to the first router that associated. The next associated router will receive an address that will be separated according to the return value of the *Cskip(parent depth)* function. The maximum number of associated routers is defined in the network parameter nwkMaxRouters (*Rm*).

Considering a parent node with a depth *d* and an address of *Aparent*, the number of child devices *n* is between 1 and *Cm-Rm*.

$$1 \leq n \leq (Cm - Rm)$$

The Achild address of the n^{th} child router is calculated according to (*n* is the number of child routers):

$$Achild = Aparent + (n-1) \times Cskip(d) + 1 \quad , n = 1$$

$$Achild = Aparent + (n-1) \times Cskip(d) \quad , n > 1$$

The Achild address of the n^{th} child end device is calculated according to (*n* is the number of child end devices):

$$Achild = Aparent + Rm \times Cskip(d) + n$$

The next figure presents an example of an address assignment scheme. Note that the network parameters are the following:

- Maximum depth: 3
- Maximum children: 6
- Maximum routers: 4

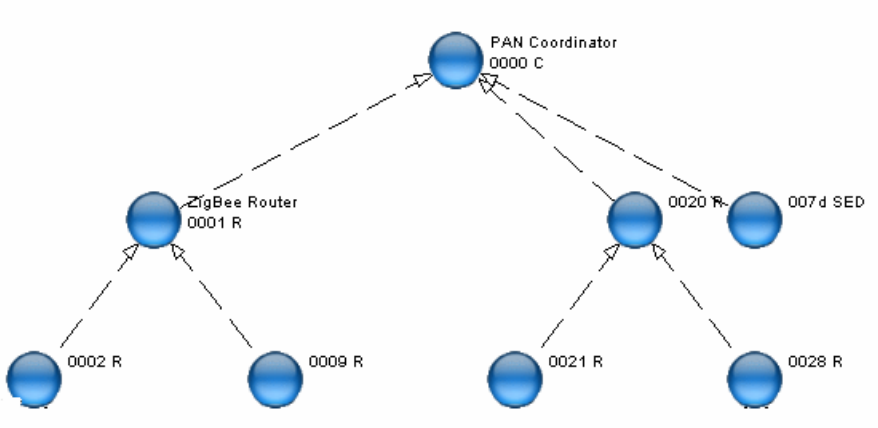

Figure 5 – Address assignment scheme example.

Figure 6 presents the PAN Coordinator available addressing scheme. With the above network parameters the coordinator is allowed to associate 4 routers and 2 end devices in its available address pool.

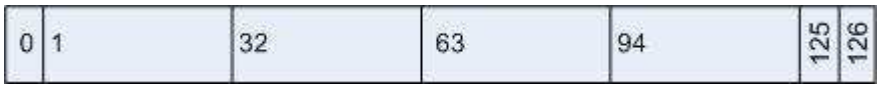

Figure 6 – PAN Coordinator addressing scheme (decimal values).

2.4 ZigBee Tree Routing Mechanism

Our current implementation only supports the tree-routing (mesh routing is not supported yet). This routing mechanism is based on the addressing scheme of the network.

Each device, upon the reception of a data frame, reads the routing information fields (Figure 7) and checks the destination address.

Octets: 2	2	2	1	1	Variable
Frame Con-trol	Destination Address	Source Address	Radius	Sequence Number	Frame Payload
	Routing Fields				
NWK Header					NWK Payload

Figure 7 – Network layer frame format [1]

If the destination address is equal to its own address, the device will signal the upper layer with the NLDE_DATA.indication primitive along with the frame payload as argument. If the destination is a child of the device (neighbour table check), the device relays the packet to the appropriate child address. If the destination address is not a child, the device must check if the address is a descendent using the following condition, being *A* the device network address, *D* the destination address and *d* the device depth in the network.

$$A < D < A + Cskip(d-1)$$

The device address of the next hop when route down is given by:

Error! Objects cannot be created from editing field codes.

If the destination address is not a descendant, the device will relay the packet to its parent.

Consider Figure 5 and a network with the following parameters: a maximum depth 3; children 6; routers 4. The *Cskip* values in the network are presented next:

Depth	Cskip(Depth)
0	31
1	7
2	1

If the ZR 0x0002 wants to transmit a message to ZR 0x0028 the tree-routing protocol will behave as follows:

1. ZR 0x0002 creates the data frame and sends it to it parent (0x0001). The most relevant fields of the data frame are outlined next:
    a. MAC destination address – 0x0001;
    b. MAC source address – 0x0002;

c. Network Layer Routing Destination Address – 0x0028;

d. Network Layer Routing Source Address – 0x0002;

2. ZR 0x0001 receives the data frame and realizes that the message in not for him and has to be relayed. The device checks its neighbour table for the routing destination address trying to find the destination is one of its child devices. Then, the device checks if the routing destination address is a descendant by verifying the condition $A < D < A + Cskip(d-1)$ that will result in:

$$0x0001 < 0x0028 < 0x0001 + 7$$

Note that the ZR 0x0001 is a depth 1 device in the network. After verifying that the destination is not a descendant, the ZR 0x0001 will route the data frame to it parent, the ZC 0x0000.

The most relevant fields of the data frame are outlined next:

a. MAC destination address – 0x0000;

b. MAC source address – 0x0001;

c. Network Layer Routing Destination Address – 0x0028;

d. Network Layer Routing Source Address – 0x0002;

3. The ZC 0x0000 receives the data frame and will verify if the routing destination address exists in its neighbour table. After realizing that the destination device is not a neighbour the ZC, that is on top of the tree and cannot route up, the next hop address is calculated as follows:

$$N = 0x0000 + 1 + \left\lfloor \frac{0x0028 - (0x0000 + 1)}{31} \right\rfloor \times 31$$

The next hop address results in $N = 32$ (decimal) = 0x0020 (N is the address of the next hop).

The most relevant fields of the data frame are outlined next:

a. MAC destination address – 0x0020;

b. MAC source address – 0x0000;

c. Network Layer Routing Destination Address – 0x0028;

d. Network Layer Routing Source Address – 0x0002;

4. The ZR 0x0020 receives the data frame and checks its neighbour table for the routing destination address. After verifying that the address is its neighbour, the message is routed to it. The next hop is assigned with the short address present in the selected neighbour table entry.

The most relevant fields of the data frame are outlined next:

a. MAC destination address – 0x0028;

b. MAC source address – 0x0020;

c. Network Layer Routing Destination Address – 0x0028;

d. Network Layer Routing Source Address – 0x0002;

3 Network Layer Implementation

3.1 Overview

The ZigBee Network layer is implemented in TinyOS/nesC[5,6] using our own implementation of the IEEE 802.15.4 protocol.

Figure 9 illustrates the TinyOS implementation diagram, respecting the layered structure presented in Figure 8. The Physical, Data Link and Network layers (gray modules in Figure 9) are implemented by us. The hardware drivers of the CC2420 radio transceiver are already provided by TinyOS.

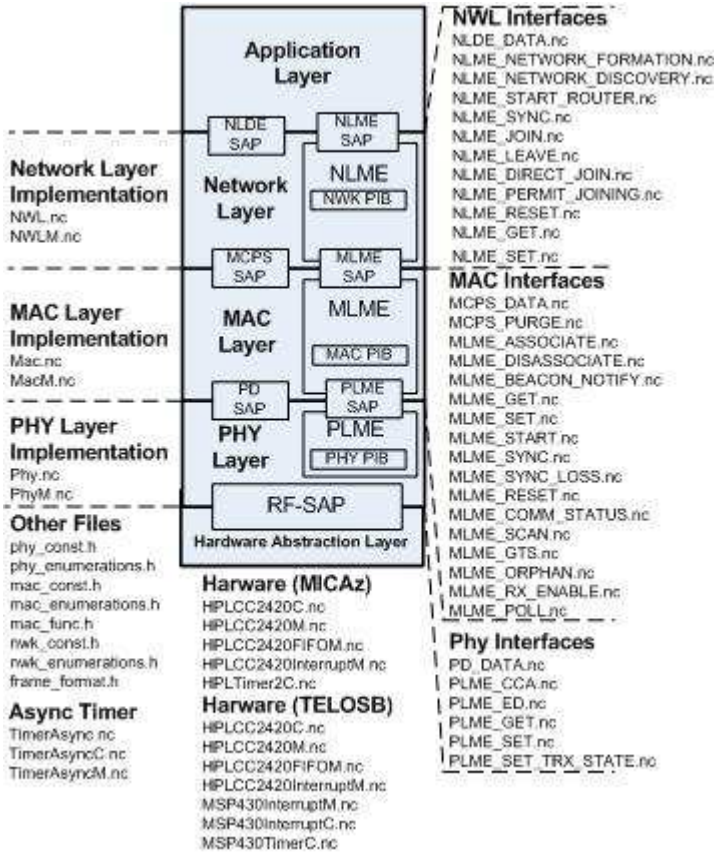

Figure 8 – TinyOS implementation file structure

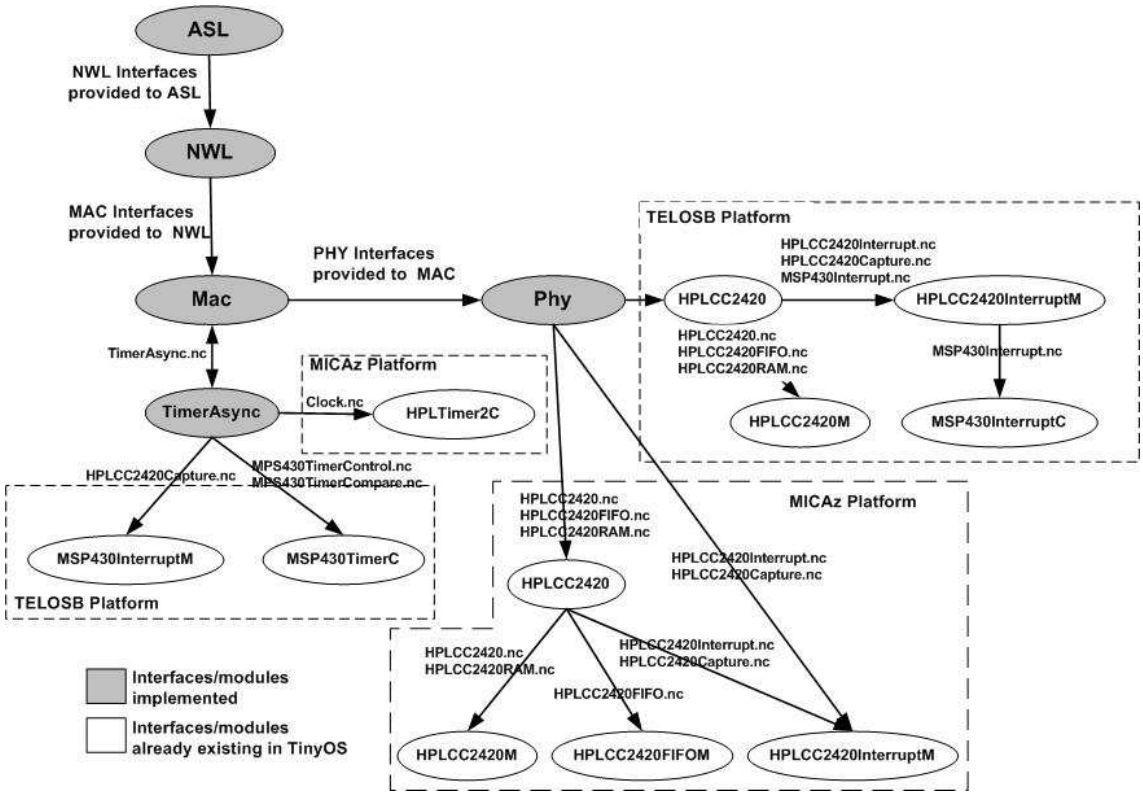

Figure 9 - TinyOS implementation diagram

Figure 9 depicts the relations between different components of the IEEE 802.15.4/Zigbee protocol stack implementation. Note that some components used in our implementation are already part of the TinyOS operating system, namely the hardware components (e.g. the HPL<…>.nc and the MSP430<…>.nc modules).

In this implementation, there is no direct interaction with the hardware, in fact, TinyOS already provides hardware drivers forging a hardware abstraction layer used by the Phy component. In Figure 9, observe that the components highlighted in white are hardware components already provided by the TinyOS operating system.

3.2 Network Layer Reference Model

The next figure presents the network layer reference model.

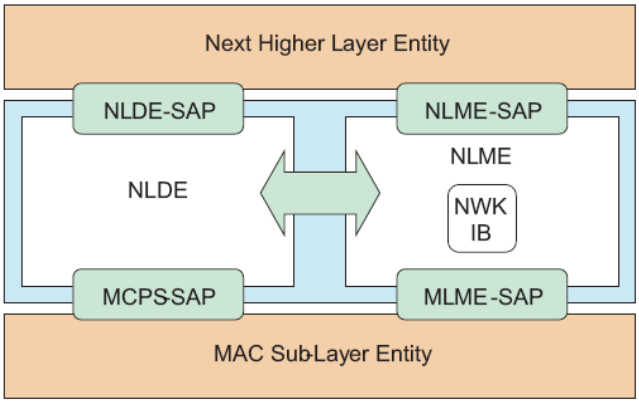

Figure 10 – Network layer reference model

The network layer provides two service entities. The Network Layer Data Entity (NLDE) provides a data service for allowing the transmission of data frames and topology specific routing. The Network Layer Management Entity (NLME) provides a management service allowing the application interface layer to interact with the network layer stack parameter. The management services provided are the following:

- Configuring a new device – Start the device operation as a ZigBee Coordinator/Router/End Device;
- Starting a Network – Establish a new network with the desired parameters;
- Joining and leaving a network – Association/disassociation procedures;
- Addressing – The ability for Coordinator or Routers to correctly assign addresses;
- Neighbour discovery – Maintenance of a neighbour table of all the devices one-hop away.
- Reception Control – Control the MAC layer operation mode for data reception.
- Route discovery – The ability to store a routing table. (not supported in our current implementation)

The Network Layer Data Entity Service Access Point (NLDE-SAP) comprehends the data transfer between the network layer and the application support sublayer (APS). The file included in the interfaces for the NLDE-SAP is the following and is located under the *contrib.hurray.tos.interfaces.ZigBee.nwk* directory:

- NLDE-DATA [2 pag 247] – exchange data frames between the NWK and the APS

The next table summarizes the primitives supported by the NLDE-SAP [2 pag 247]

Interface Name	Request	Indication	Response	Confirm
NLDE-DATA	X	X		X

Table 1 - NLDE-SAP primitives

The Network Layer Management Entity Service Access Point (NLME-SAP) comprehends the exchange of management commands between the NWK layer and the APS. The files included in the interfaces for the NLDE-SAP are the following and are located under *contrib.hurray.tos.interfaces.ZigBee.nwk* directory:

- NLME_NETWORK_DISCOVERY [3 pag. 254] – Used by the APS to perform a discovery of networks operating within the device personal operating space (POS);
- NLME_NETWORK_FORMATION [3 pag. 257] – Used by the APS of a ZigBee Coordinator to start a network and to change the superframe configuration
- NLME_PERMIT_JOINING [3 pag.262] – Used by the APS of a ZigBee Coordinator or Router to allow/deny the acceptance of new associations.
- NLME_START_ROUTER [3 pag.263] – Used by the APS of a ZigBee Router to start broadcasting beacon frames and to change the superframe configuration
- NLME_JOIN [3 pag. 265] – Used by the APS of a device request a join procedure (or association) to a network.
- NLME_DIRECT_JOIN [3 pag. 273] – Used by the APS of a ZigBee Coordinator or Router to request to directly join another device to its network
- NLME_LEAVE [3 pag. 275] – Used by the APS to request or inform a disassociation to the network.
- NLME_RESET [3 pag. 279] – Used to reset the NWK layer parameters;

- NLME_SYNC [3 pag. 283] – Allows the notification of the APS of the loss of synchronization to the device parent.
- NLME_GET [3 pag.287] – Used by the APS to read the values of an attribute of the Network Layer Information Base;
- NLME_SET [3 pag. 288] Used by the APS to write the values of an attribute of the Network Layer Information Base;

The next table summarizes the primitives supported by the NLME-SAP [3 pag 253]

Interface Name	Request	Indication	Response	Confirm
NLME_NETWORK_DISCOVERY	X			X
NLME_NETWORK_FORMATION	X			X
NLME_PERMIT_JOINING	X			X
NLME_START_ROUTER	X			X
NLME_JOIN	X	X		X
NLME_DIRECT_JOIN	X			X
NLME_LEAVE	X	X		X
NLME_RESET	X	X		X
NLME_SYNC	X			X
NLME_GET	X			X
NLME_SET	X			X

Table 2 - NLME-SAP primitives

3.3 Components NWK and NWKM

The Network layer is implemented in two files. There file are located under *contrib.hurray.tos.lib.nwk*

NWK.nc – Component wiring the interfaces to the implementation on the component NWKM

NWKM.nc – Component that implements the Network layer functions that will be provided to the upper layer.

3.4 Component NWK

3.4.1 Provided Interfaces

- NLME_NETWORK_DISCOVERY
- NLME_NETWORK_FORMATION
- NLME_PERMIT_JOINING

- NLME_START_ROUTER
- NLME_JOIN
- NLME_DIRECT_JOIN
- NLME_LEAVE
- NLME_RESET
- NLME_SYNC
- NLME_GET
- NLME_SET

3.4.2 Component Graph

The network layer was developed as a TinyOS component (NWK) and each SAP as an interface. The MAC component provides the NWK the MAC SAP interfaces and the NWK provides the upper layer with the NWK SAP interfaces. Figure 11 depicts the component diagram that connects the NWK module with the MAC module (this diagram was generated by the *nesdoc* tool provided by TinyOS [7]).

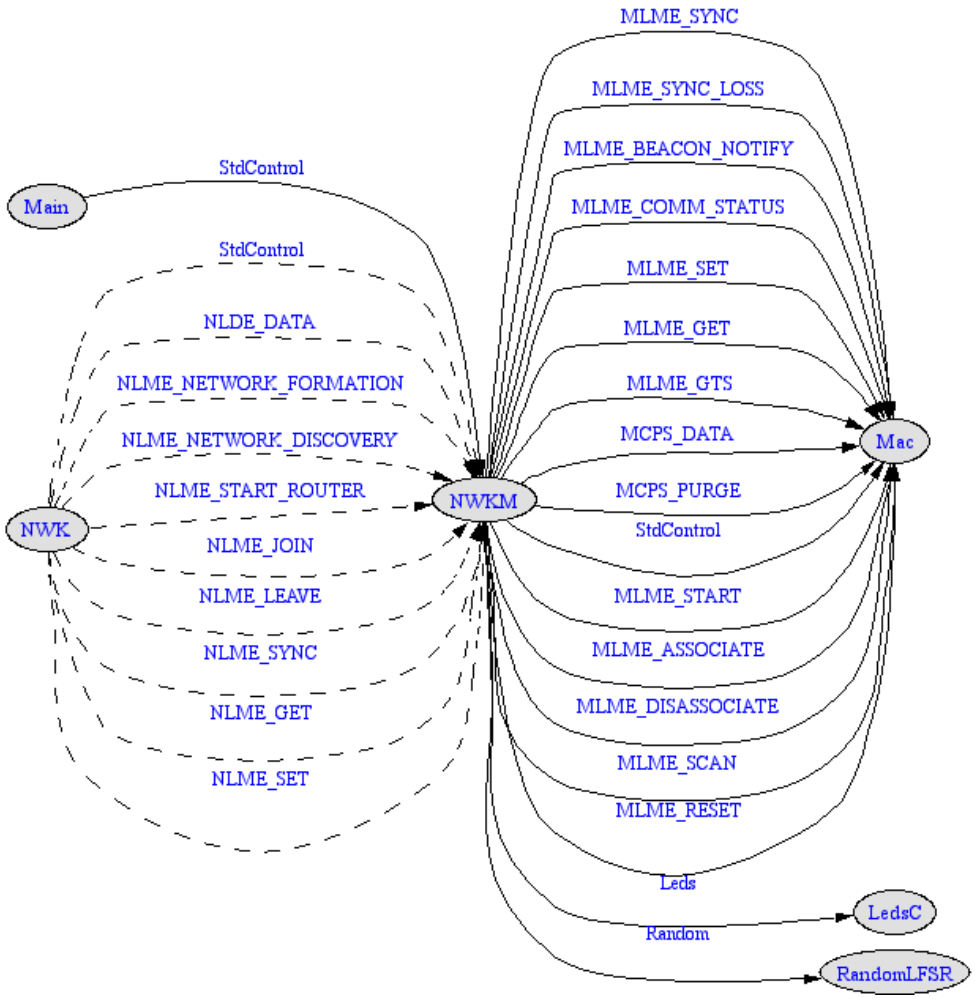

Figure 11 – TinyOS NWK component diagram

Only a subset of the network layer was developed. The functionalities implemented are the following:

- Network association mechanisms – tree association scheme;

- Neighbour table – information about the parent node and the associated child devices only;
- NWK IB – Network layer information base;
- Tree routing.

The network discovery functions were implemented statically because, in our current implementation of the IEEE 802.15.4, the channel scan mechanism is not implemented. The network parameters were defined as constants.

3.5 Component NWKM

3.5.1 Required Interfaces

- MLME_START [3 pag. 100] – Used for the coordinator to start sending beacons or use a new superframe configuration.
- MLME_ASSOCIATE [3 pag. 64] – Used to create an association request directed to the coordinator.
- MLME_DISASSOCIATE [3 pag. 71] – Used to create a disassociation requent directed to the coordinator.
- MLME_SYNC [3 pag. 104] – Used to enable the MAC layer to start synchronizing the coordinator by always keep track of the beacon.
- MLME_SYNC_LOSS [3 pag. 105] – Used by the MAC layer to inform the upper layer about the loss of synchronization with a coordinator.
- MLME_SCAN [3 pag. 92] - Implements the channel scan mechanist in order to inform the upper layer about the energy detection on each channel.
- MLME_RESET [3 pag. 88] – Used to request a reset operation in the MAC layer.
- MLME_BEACON_NOTIFY [3 pag. 75] – Used by the MAC layer to inform the upper layer about the PAN descriptor and pending addresses contained in the beacon received.
- MLME_COMM_STATUS [3 pag. 96] – Used by the MAC layer to inform the upper layer about the communication status.
- MLME_SET [3 pag. 98] - Used to write in the attributes of the MAC PAN Information Base.
- MLME_GET [3 pag. 78] – Used to read the attributes of the MAC PAN Information Base.
- MLME_GTS [3 pag.79] – Used to create a GTS allocation request directed to the coordinator. This interface also informs the MAC upper layer about the status of the allocation.
- MCPS_DATA [3 pag. 56] – Implement the data exchange between the MAC layer and the next upper layer.
- MCPS_PURGE [3 pag. 61] – Used to purge a data frame from the transaction queue.

3.5.2 Provided Interfaces

- NLME_NETWORK_DISCOVERY
- NLME_NETWORK_FORMATION
- NLME_PERMIT_JOINING
- NLME_START_ROUTER
- NLME_JOIN
- NLME_DIRECT_JOIN
- NLME_LEAVE
- NLME_RESET

- NLME_SYNC
- NLME_GET
- NLME_SET

3.5.3 Variables

- nwkIB nwk_IB – Structure used to store the NWK PAN informations. The attributes contain in this structure can be accesses by the upper layers thought the *NLME-GET* and *NLME-SET* interface primitives. The structure definition is located in the file *nwk_const.h* located under the *contrib.hurray.tos.lib.nwk*.
- uint8_t device_type – Variable used to define the device type in the network layers. This can assume the values of COORDINATOR, ROUTER or END DEVICE;
- neighbortableentry neighbortable[7] – Neighbour table list that will contain all the neighbours of the device. Each element on this list is defined as a *neighbortableentry*. The structure definition is located in the file *nwk_const.h* located under the *contrib.hurray.tos.lib.nwk*.
- uint8_t neighbour_count – Number of elements in the neighbour table.
- uint8_t parent – Index of the parent device in the neighbour table.
- uint16_t networkaddress – Short address of the device;
- uint8_t depth – Depth of the device in the tree.
- uint8_t cskip – Value of the cskip function used in the address assignement by the ZigBee Coordinator and Routers.
- uint8_t cskip_routing - Value of the cskip function used and in the tree-routing protocol by the ZigBee Coordinator and Routers.
- uint8_t parent_index
- uint16_t panid – PAN id of the network.
- uint8_t beaconorder – Beacon order that the ZigBee Coordinator and Routers is using.
- uint8_t superframeorder – Superframe order that the ZigBee Coordinator and Routers is using.
- uint16_t next_child_router_address – The short address of the next child router device that will request an association.
- uint8_t number_child_router – Number of child routers currently associated.
- uint8_t number_child_end_devices – Number of child end devices currently associated.
- uint8_t joined – Boolean variable defining if the device in joined to a network or not.
- uint8_t sync_loss – Boolean variable defining if the device has lost synchronization to its parent.

3.5.4 Function description

- void init_nwkIB(void) - Function that initialized the NWK PAN Information base to its default values.
- uint8_t check_neighbortableentry(uint8_t addrmode, uint32_t Extended_Address0, uint32_t Extended_Address1) - Function used to search in the neighbour table (*nwk_IB*) for a device. This function takes as arguments an extended address or a short address. This function returns the index in the neighbour table list if the device was found, otherwise it returns zero.
- void add_neighbortableentry(uint16_t PAN_Id, uint32_t Extended_Address0, uint32_t Extended_Address1, uint32_t Network_Address, uint8_t Device_Type, uint8_t Relationship) - Function used to add a new element to the neighbour table.

- void update_neighbortableentry(uint16_t PAN_Id, uint32_t Extended_Address0, uint32_t Extended_Address1, uint32_t Network_Address, uint8_t Device_Type, uint8_t Relationship) – Function used to update an element in the neighbour table.
- uint8_t find_suitable_parent(void) – Function used to find a suitable parent in the neighbour table. Currently this function finds the first device where the relationship field contains the value NEIGHBOR_IS_PARENT. This function returns the index in the neighbour table list if the device was found, otherwise it returns zero.
- uint16_t Cskip(uint8_t d) - Function used to compute the Cskip value.
- uint16_t nexthopaddress(uint16_t destinationaddress, uint8_t d) - Function used to calculate the next hop address of the appropriate child if route down is required.
- void list_neighbourtable(void) – Debug function that list all the elements in the neighbour table.

3.6 Implementation of the protocol functionalities

This section presents a brief explanation of the implemented network layer mechanisms. It starts with the Time Division Beacon Scheduling Mechanism that is necessary in the construction of the cluster-tree because it provides an effective way to schedule the beacon transmissions of all the ZigBee Routers. Other mechanisms like the joining to a network, the formation of a network and the routing are briefly explained.

3.6.1 The Time Division Beacon Scheduling Mechanism

The Time Division Beacon Scheduling Mechanism does not introduce relevant changes in the protocol specification. It relies on a negotiation mechanism based on command frames embedded in data frames.

When implementing this mechanism we assume the following:

1. The ZigBee network layer supports the tree-routing mechanism, thus and the network addresses of the devices are assigned accordingly.
2. The ZigBee Coordinator is the first node broadcasting beacons in the network.
3. The ZigBee Routers start to send beacons only after a successful negotiation.
4. The same Beacon Interval (BI) is used by every router.

For the negotiation of the beacon transmission, each ZR must complete the following steps:

1. The ZR must successful associate with its parent and temporarily behaves as a ZigBee End Device (ZED), without sending beacons.
2. The ZR initiates the negotiation protocol by sending a “START SENDING BEACON” request command.
3. The ZC receives the request and determines the schedule of the ZR.
4. After the schedule process, the ZC replies by sending a “START SENDING BEACON” response command with the status of the negotiation (SUCCESS or FAIL) and the transmission offset value to the requesting ZR
5. The ZR receives the command from the ZC and if the negotiation its successful it starts sending beacon in the defined offset related to its parent.

In order to implement this mechanism, some changes are needed in the Network and Mac layer Service Access Point (SAP) primitives. Thus, is necessary to add a *StartTime* argument in the MLME-START.request primitive, as already proposed in the ZigBee standard [1, pag 245]. This primitive is used by the upper layer to request the MAC layer to start sending beacons to or use a new superframe configuration. The new format of the primitive is as follows:

MLME-START.request (
PANID,
LogicalChannel,
BeaconOrder,
SuperframeOrder,
PANCoordinator,
BatteryLifeExtention,
CoordRealignment,
SecurityEnable,
StartTime)

The *StartTime* parameter will be used as a transmission offset referring to the ZigBee Router (ZR) parent. In the ZC the value of this parameter is 0.

The *StartTime* parameter size is 3 bytes and is specified in symbols.

In the NLME-START-ROUTER.request primitive there is also the need to add a *StartTime* parameter. The new format of the primitive is as follows:

NLME-START-ROUTER.request(
BeaconOrder,
SuperframeOrder,
BatteryLifeExtension,
StartTime)

The primitive is requested by the Network upper layer of the ZR to start the beacon transmission. The *StartTime* parameter is obtained after a successful negotiation with the ZC for beacon broadcasting. Note that, in case of an unsuccessful negotiation the ZR will not be allowed to send beacons, therefore can only act as a ZED.

After a successful negotiation of the beacon transmission, the ZR will have two active periods: its own (the superframe duration) and the parent's superframe duration. In its own active period the ZR is allowed to transmit frames to its associated devices or relay frames to the descendant devices in the tree. The frames destined upstream are sent during its parent's active period. To accomplish this behaviour there is a need to implement a different buffer mechanism for each message flow - the downstream to the device descendants and the upstream to the device ascendants.

The buffer mechanism is implemented in the MAC layer that uses the downstream buffer or the upstream buffer depending of the transmission options parameter of the MCPS_DATA.request primitive. The transmit options or *TxOptions* parameter, last argument of the primitive, define the transmission options for the data frame, allowing the frame to be sent in the GTS or during the CAP period.This parameter will also define if the transmission will use the upstream or the downstream buffer.

As defined in the IEEE 802.15.4 standard [3] the transmission options parameters has the following format:

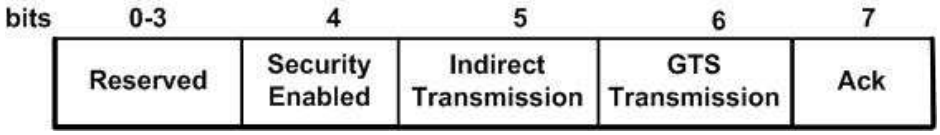

Figure 12 – MCPS_DATA.request TxOptions format [3]

In order to inform the MAC layer of which buffer to use, we have changed the transmission format including an upstream parameter. The Time Division Beacon Scheduling transmission option parameter has the following format:

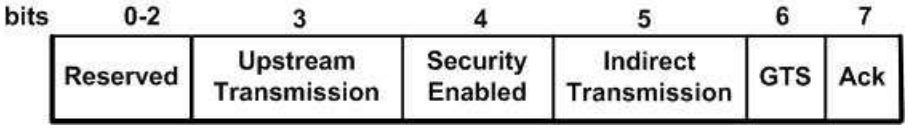

Figure 13 – Time Division Beacon Scheduling MCPS_DATA.request TxOptions format

During the ZR superframe all the frames that need to be transmitted to its parent will be stored in the upstream buffer. When the device enters the parent superframe, it tries to transmit the messages.

To enable the use of the 2 message buffers the device must wake up in the parent's superframe. This is accomplished by adding two new timer events to the MAC layer. One is triggered at the beginning of the parent's superframe and turns on the transceiver in receive mode and another at the end turning the transceiver off.

In our current implementation we did not include the implementation of the scheduling algorithm yet. Instead, the application running on the ZC has the offset values pre-established for each node address that requests a negotiation for a time window slot.

The negotiation of the beacon transmission is performed thought a simple protocol that uses the data frames payload with a predefined format. The format of the negotiation fields is the following:

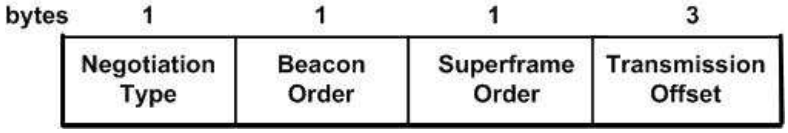

Figure 14 – Negotiation Fields

- Negotiation type – Indicates the type of the negotiation command. This field can have the following values; 1 for a negotiation request, 2 for a negotiation accept and 3 for a negotiation deny;
- Beacon Order – Indicates the beacon order of the ZR device;
- Superframe Order – Indicates the superframe order of the ZR device;
- Transmission Offset – Indicates the transmission offset schedule by the ZC in a negotiation accept command.

In the negotiation request, the Beacon Order and Superframe Order fields indicate the intended ZR superframe configuration. In the negotiation response the configuration may not be the same as the requested, instead, the ZC can assign a different configuration according to its scheduling.

The next diagram presents the sequence of Network layer events from the association of the ZR (A) until the beacon transmission after a successful negotiation (B).

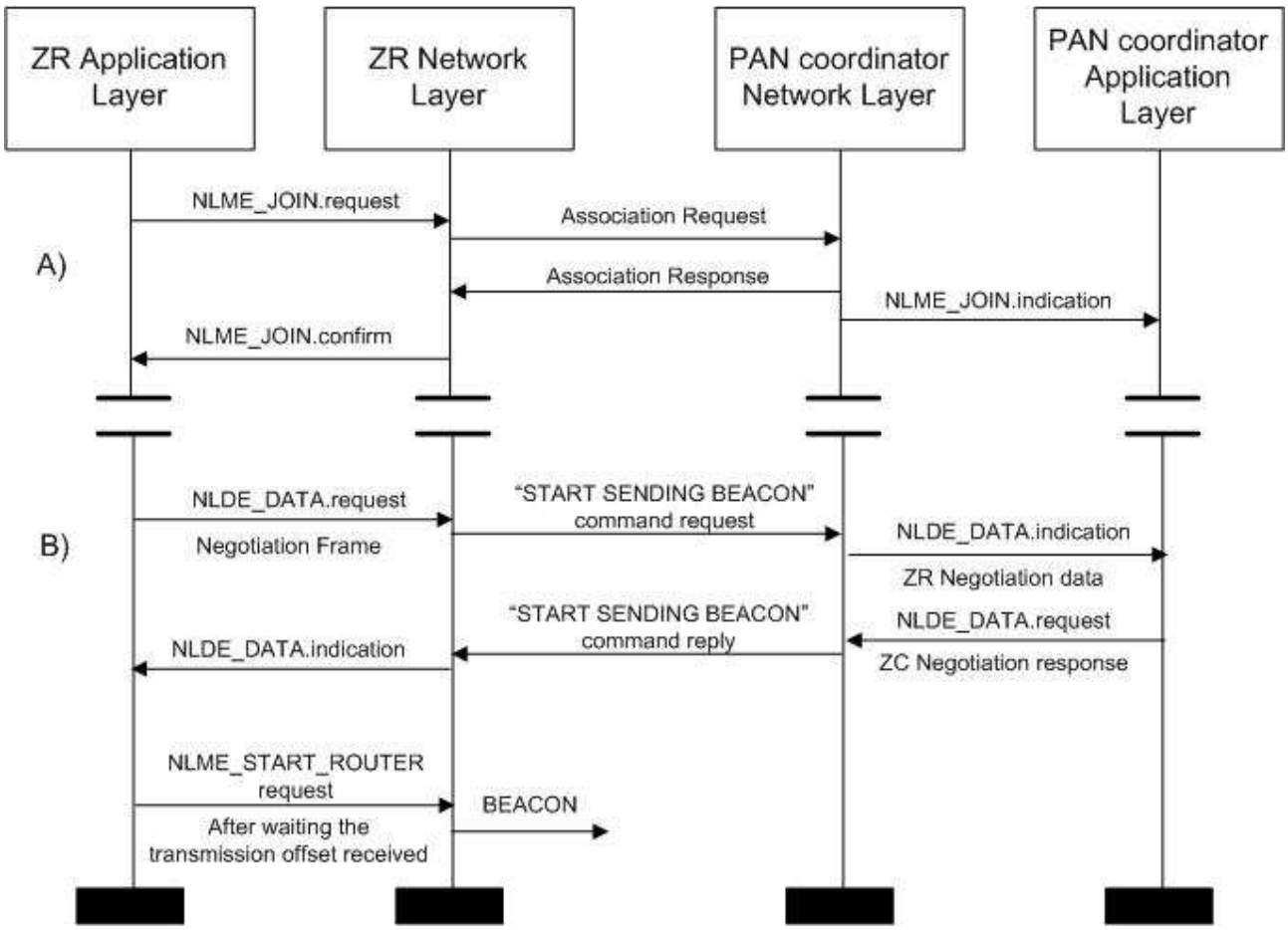

Figure 15 – Negotiation diagram

3.6.2 Creating a new network as a ZigBee Coordinator

The creation of a new network is done by issuing the NLME_NETWORK_FORMATION.request. When this primitive is requested the ZigBee router will perform the following actions:

1. Set the NWK variable device_type to COORDINATOR
2. Set the MAC layer PAN information (e.g. beacon order, superframe duration, maximum beacon length, PAN ID, short address);
3. Calculate the cskip value for the address assignment and for the routing. This values are calculated only in this procedure to avoid the computation during operational behaviour of the mechanisms.
4. Initialize the address assignment variables; The address assignment will include
5. Call the MLME_START.request primitive to start broadcasting beacons.

3.6.3 Start Sending beacons as a ZigBee Router

Each ZigBee router before starting to broadcast beacons must join the network. As this implementation uses the Time Division Beacon Scheduling Mechanism the device will behave as a ZED and will negotiate with the ZC for beacon broadcasting. Is the broadcast is successful the device will call the

NLME_START_ROUTER.request primitive that will perform the same operations as the ZC when starting a new network except when the device issues the MLME_START.request primitive to the MAC layer it will include the StartTime argument with the values received during the negotiation. This value will be used by the MAC layer to calculate the beacon transmission offset after the reception of the devices parent beacon.

3.6.4 Joining a Network

The join procedure is necessary for every ZigBee Router and ZigBee End Device. Only the ZC and ZRs are allowed to associate devices. The join procedure in the network layer, as explained in section 2.3 of this technical report, is based of a distributed address scheme.

When a device receives an association request command frame the MAC layer issues the *MLME_ASSOCIATE.indication* primitive to the network layer. Figure 16 depicts a flow chart showing the association procedure of a parent device.

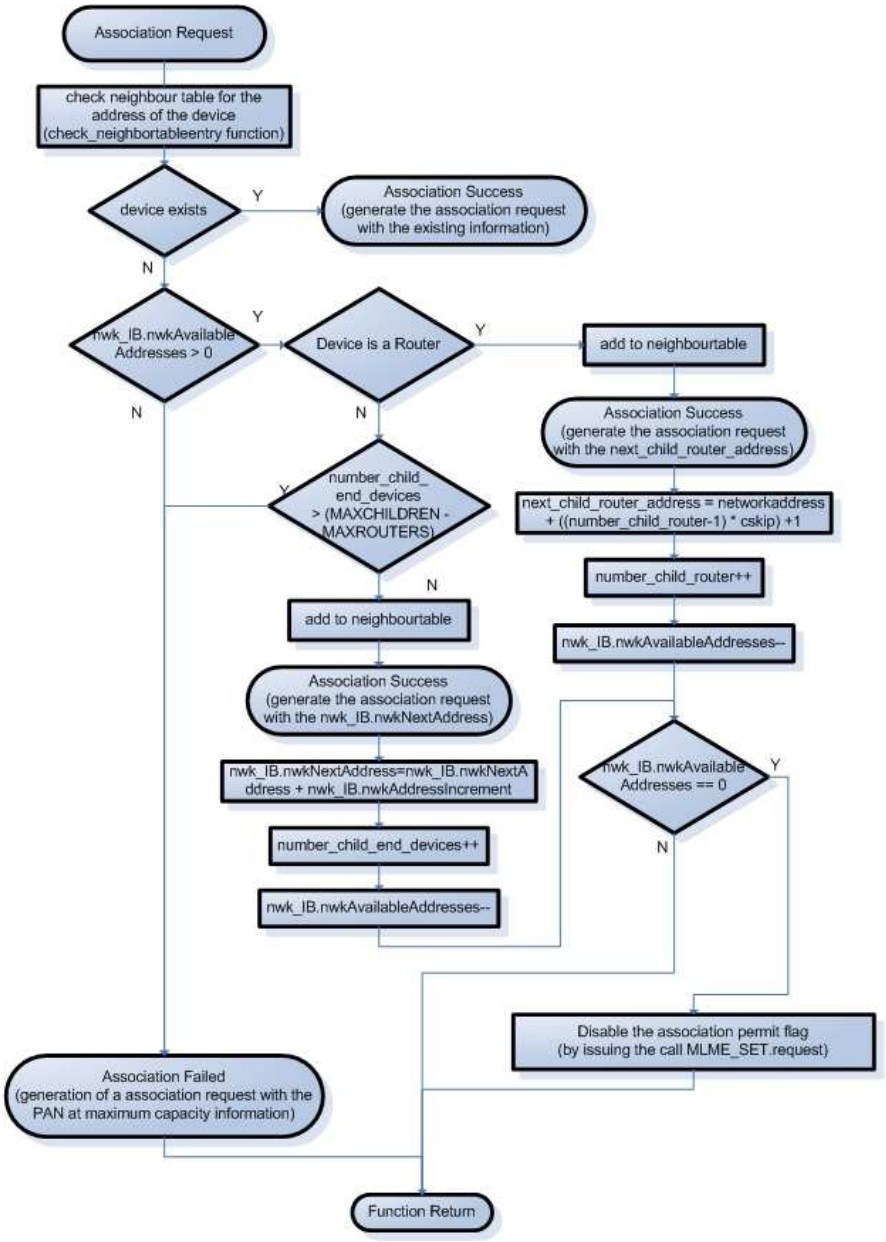

Figure 16 - MLME_ASSOCIATE.indication flow chart

In the *MLME_ASSOCIATE.indication* primitive the parent will first check for associating device address in the neighbour table, verifying if it is a reassociation and in that case the association is successful and the parent generates and association response command frame with the already stored information. If the device does not exist, the parent must verify what type of device is trying the association, if it's a router or an end device. The type of device will define the address that must be assigned. Nevertheless, for a successful association the variable *nwk_IB.nwkAvailableAddresses* in the NWK PAN Information base must be grater than zero.

If the associating device is a router, the parent will add its information to the neighbour table and send the association response command with the short address previously calculated in the *next_child_router_address* variable. Then, the parent updates the future child router address by calculating the next formulation: *next_child_router_address = networkaddress + ((number_child_router-1) * cskip) +1* followed by an increment of the number of child routers associated (*number_child_router*). Note that the *cskip* parameter is calculated in the initialization of the node and the *networkaddress* variable is the parents device short address.

If the association device is an end device, the parent will also add its information to the neighbour table and generate the association respond command with the short address previously calculated in the *nwk_IB.nwkNextAddress* variable. After the response generation, to *nwk_IB.nwkNextAddress* variable will be added the *nwk_IB.nwkAddressIncrement* variable followed by an increment of the number of child end devices.

There is a need to differentiate the type of associating device and separately count their numbers and their assigned addresses to comply with the address schemes. This address distribution will enable the construction of the cluster-tree topology

3.6.5 Cluster-tree Routing

The cluster-tree routing procedure, as briefly explained in section 2.4 of this technical report, is based on the addresses of the devices. When a MAC layer of a device receives a data frame it issues the MCPS_DATA.indication primitive to the NWK layer. Figure 17 depicts a flow chart showing the procedures when the NWK receives a data frame.

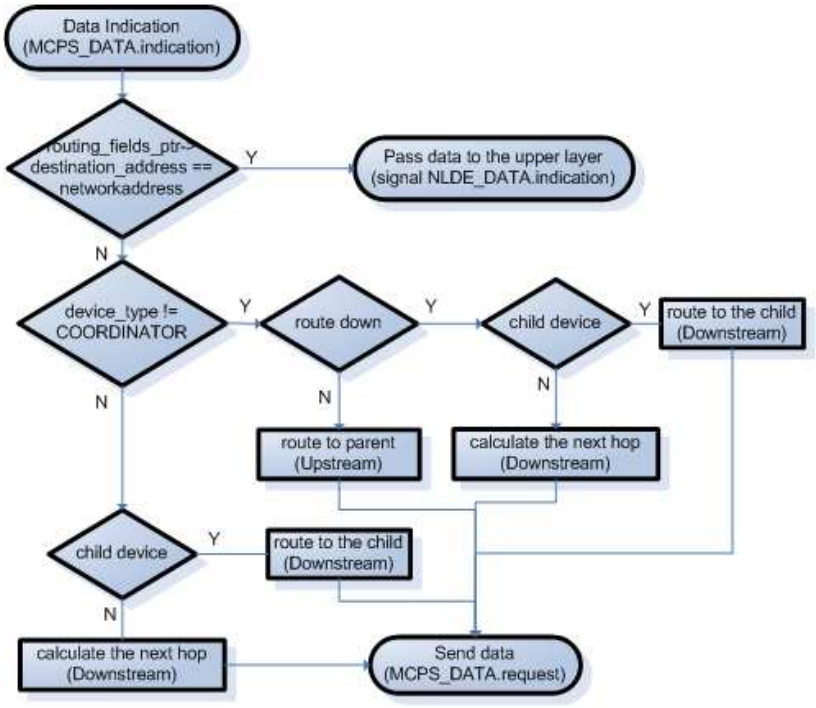

Figure 17 - MCPS_DATA.indication flow chart

The NWK layer, upon the reception of a data frame will first verify the routing destination field equals its own short address and if true it will pass the data payload to the upper layer, by issuing the

NLDE_DATA.indication primitive. If the routing destination address is not for it self, the device must calculate the next hop destination address. In the case of a ZigBee Coordinator, if the destination of the data frame is its own child, after checking in the neighbour table, it will assign the next hop with the short address present in the neighbour tables of the respective child, otherwise it need to calculate the next hop by applying the cluster-tree routing formula for that effect (shown in section 2.4). In all the cases the routing is always downstream because the ZC does not have any parent.

In the case of a ZigBee Routers there is an initial verification if the destination is up or down in the tree. This verification is done by applying the following conditions:

networkaddress < routing_fields_ptr->destination_address

AND

routing_fields_ptr->destination_address < (networkaddress + cskip_routing)

If the above conditions are true then the routing is down the tree. The device checks it the destination is a child device (by consulting its neighbour table) and if not it will calculate the next hop by applying the cluster-tree routing formula for that effect (shown in section 2.4).

If the conditions are false the device just routes up the network to its parent.

After the next hop decision the message is transmitted by issuing the *MCPS_DATA.request* primitive to the MAC layer.

3.6.6 Data Transmission

The data transmission procedure is similar to the routing mechanism. After the creation of the frame the device must assign a destination address to the routing fields. Depending on the type of device, if its an end device the only option is to route the its parent (upstream in the cluster-tree network), otherwise if the device is the ZigBee Coordinator or a ZigBee Router, it must check if the destination is a child device or must calculate the next hop by applying the cluster-tree routing formula for that effect (shown in section 2.4). Figure 18 depicts a flow chart showing the procedures the NWK upper layer requests a data transmission.

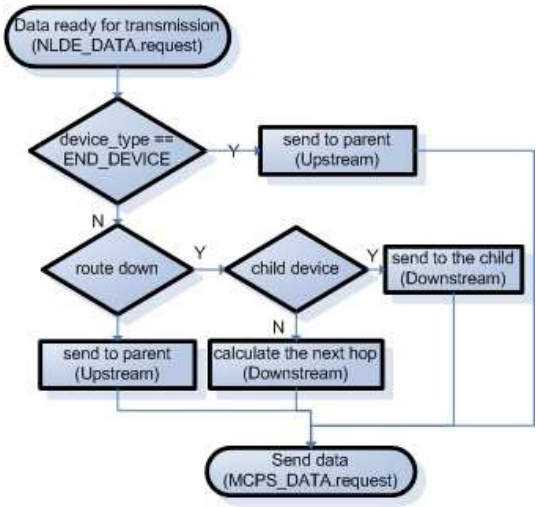

Figure 18 - NLDE_DATA request flow chart

4 Auxiliary Files (Under contrib.hurray.tos.lib.nwk)

nwk_const.h

This file contains data structures definition used and protocol constants definition related with the NWK layer along with auxiliary constants.

The NWK protocol constants defined are the described in the next table [1 pag. 315].

Constant	Value	Description
nwkcCoordinatorCapable	Set at build time	A Boolean flag indicating whether the device is capable of becoming the ZigBee Coordinator (0x01 indicates the device is capable)
nwkcDefaultSecurityLevel	ENC-MIC-64	The default security level to be used.
nwkcDiscoveryRetryLimit	0x03	The maximum number of times a route discovery will be retried.
nwkcMaxDepth	0x0f	The maximum depth of the device
nwkcMinHeaderOverhead	0x08	The minimum number of octets added by the NWK layer to a NSDU
nwkcProtocolVersion	0x02	The version of the ZigBee NWK protocol in the device
nwkcWaitBeforeValidation	0x500	Time duration in milliseconds, on the originator of a multicast route request, between receiving a route reply and sending a message to validate the route
nwkcRepairThreshold	0x03	Maximum number of allowed communication errors after which the route repair mechanism is initiated
nwkcRouteDiscoveryTime	0x2710	Time duration in milliseconds until a route discovery expires
nwkcMaxBroadcastJitter	0x40	The maximum broadcast jitter time measured in milliseconds.
nwkcInitialRREQRetries	0x03	The number of times the first broadcast transmission of a route request command frame is retried.
nwkcRREQRetries	0x02	The number of times the broadcast transmission of a route request command frame is retried on relay by an intermediate ZigBee router or ZigBee coordinator
nwkcRREQRetryInterval	0xfe	The number of milliseconds between retries of a broadcast route request command frame
nwkcMinRREQJitter	0x01	The minimum jitter, in 2 millisecond slots, for broadcast retransmission of a route request command frame
nwkcMaxRREQJitter	0x40	The maximum jitter, in 2 millisecond slots, for broadcast retransmission of a route request command frame

Table 3 - ZigBee Network layer constant description

The next table describes auxiliary constants used in the MAC layer.

Constant	Value	Description
TYPE_DEVICE		Defines the type of device (COORDINATOR; ROUTER or ENDDEVICE)
DEVICE_DEPTH		Defines the device current depth in the network
LOGICAL_CHANNEL		Defines de logical radio channel where the device is operating

PANID		The 16 short address of the PAN.
AVAILABLEADDRESSES	0x04	Number of available addresses for nodes association.
ADDRESSINCREMENT	0x0001	Address increment value
MAXCHILDREN	0x06	Maximum number of child devices allowed for the ZC and each ZR:
MAXDEPTH	0x04	Maximum depth in the cluster-tree network;
MAXROUTERS	0x04	Maximum number of child routers allowed for the ZC and each ZR:
BEACON_ORDER	8	Beacon order;
SUPERFRAME_ORDER	4	Superframe order;
D<x>_PAN_EXT0 D<x>_PAN_EXT1		Static definition of the 32 bit extended address of the device parent
D<x>_PAN_SHORT		Static definition of the 16 bit short address of the device parent
NEIGHBOUR_TABLE_SIZE	7	Number of entries in the neighbour table
SCHEDULING_REQUEST	0x01	Negotiation mechanism message type.
SCHEDULING_ACCEPT	0x02	Negotiation mechanism message type.
SCHEDULING_DENY	0x03	Negotiation mechanism message type.

Table 4 - NWK layer auxiliary constants description.

The next table describes the structures defined in this file.

Structure Name	Attributes	Description
nwkIB	uint8_t nwkSequenceNumber; uint8_t nwkPassiveAckTimeout; uint8_t nwkMaxBroadcastRetries; uint8_t nwkMaxChildren; uint8_t nwkMaxDepth; uint8_t nwkMaxRouters; uint8_t nwkNetworkBroadcastDeliveryTime; uint8_t nwkReportConstantCost; uint8_t nwkRouteDiscoveryRetriesPermitted; uint8_t nwkSymLink; uint8_t nwkCapabilityInformation; uint8_t nwkUseTreeAddrAlloc; uint8_t nwkUseTreeRouting; uint16_t nwkNextAddress; uint16_t nwkAvailableAddresses; uint16_t nwkAddressIncrement; uint16_t nwkTransactionPersistenceTime;	NWK PAN Information base [1 pag 317]
neighbortableentry	uint16_t PAN_Id; uint32_t Extended_Address0;	Neighbour table entry [1 pag 342]

	uint32_t Extended_Address1; uint16_t Network_Address; uint8_t Device_Type; uint8_t Relationship; uint8_t Depth; uint8_t Permit_Joining; uint8_t Logical_Channel; uint8_t Potential_Parent;	
networkdescriptor	uint16_t PANId; uint8_t LogicalChannel; uint8_t StackProfile; uint8_t ZigBeeVersion; uint8_t BeaconOrder; uint8_t SuperframeOrder; uint8_t PermitJoining;	PAN network descriptor
beacon_scheduling	uint8_t request_type; uint8_t beacon_order; uint8_t superframe_order; uint8_t transmission_offset[3];	Negotiation for beacon transmission frame structure.

Table 5 - Structure definitions on the nwk_const.h file

nwk_enumerations.h

This file contains the enumeration values used in the NWK layer. The following tables describe the enumerations and their usage [1 pag. 243].

Enumeration	Value	Description
NWK_SUCCESS	0x00	A request has been executed successfully
NWK_INVALID_PARAMETER	0xc1	An invalid or out-of-range parameter has been passed to a primitive from the next higher layer
NWK_INVALID_REQUEST	0xc2	The next higher layer has issued a request that is invalid or cannot be executed given the current state of the NWK layer
NWK_NOT_PERMITTED	0xc3	An NLME-JOIN.request has been disallowed
NWK_STARTUP_FAILURE	0xc4	An NLME-NETWORK-FORMATION.request has failed to start a network
NWK_ALREADY_PRESENT	0xc5	A device with the address supplied to the NLMEDIRECT-JOIN.request is already present in the neighbor table of the device on which the NLMEDIRECT- JOIN.request was issued
NWK_SYNC_FAILURE	0xc6	Used to indicate that an NLME-SYNC.request has failed at the MAC layer
NWK_NEIGHBOR_TABLE_FULL	0xc7	An NLME-JOIN-DIRECTLY.request has failed because there is no more room in the neighbour table
NWK_UNKNOWN_DEVICE	0xc8	An NLME-LEAVE.request has failed because the device addressed in the parameter list is not in the neighbor table of the

		issuing device
NWK_UNSUPPORTED_ATTRIBUTE	0xc9	An NLME-GET.request or NLME-SET.request has been issued with an unknown attribute identifier
NWK_NO_NETWORKS	0xca	An NLME-JOIN.request has been issued in an environment where no networks are detectable
NWK_LEAVE_UNCONFIRMED	0xcb	A device failed to confirm its departure from the network
NWK_MAX_FRM_CNTR	0xcc	Security processing has been attempted on an outgoing frame, and has failed because the frame counter has reached its maximum value
NWK_NO_KEY	0xcd	Security processing has been attempted on an outgoing frame, and has failed because no key was available with which to process it
NWK_BAD_CCM_OUTPUT	0xce	Security processing has been attempted on an outgoing frame, and has failed because security engine produced erroneous output

Table 6 - General NWK enumeration description.

The next table enumerates the device types in ZigBee networks [1 pag 342].

Enumeration	Value	Description
COORDINATOR	0x00	ZigBee Coordianator
ROUTER	0x01	ZigBee Router
END_DEVICE	0x02	ZigBee End Device

Table 7 – Device Types enumeration.

The next table enumerates the possible device relationship [1 pag. 342].

Enumeration	Value	Description
NEIGHBOR_IS_PARENT	0x00	The neighbour device is the parent
NEIGHBOR_IS_CHILD	0x01	The neighbour device is a child
NEIGHBOR_IS_SIBLING	0x02	The neighbour device is a sibling
NEIGHBOR_IS_NON	0x03	The neighbour device has no relationship
NEIGHBOR_IS_PREVIOUS_CHILD	0x04	The neighbour device was a child

Table 8 – Neighbour device relationship enumerations

NWK PIB attributes enumerations description table used in the NLME_GET and NLME_SET primitives [1 pag. 317].

Enumeration	Value	Description
NWKSEQUENCENUMBER	0x81	
NWKPASSIVEACKTIMEOUT	0x82	
NWKMAXBROADCASTRETRIES	0x83	
NWKMAXCHILDREN	0x84	

NWKMAXDEPTH	0x85	
NWKMAXROUTERS	0x86	
NWKMETWORKBROADCASTDELIVERYTIME	0x88	
NWKREPORTCONSTANTCOST	0x89	
NWKROUTEDISCOVERYRETRIESPERMITED	0x8a	
NWKSYMLINK	0x8e	
NWKCAPABILITYINFORMATION	0x8f	
NWKUSETREEADDRALLOC	0x90	
NWKUSETREEROUTING	0x91	
NWKNEXTADDRESS	0x92	
NWKAVAILABLEADDRESSES	0x93	
NWKADDRESSINCREMENT	0x94	
NWKTRANSACTIONPERSISTENCETIME	0x95	

Table 9 - NWK GET/SET reference PIB enumerations description.

5 Example Application

The sample application used to test the NWK implementation is the *Test_APL*, located under the *contrib.hurray.apps.Test_APL*.

This application uses the interfaces provided by the NWKM component and currently is customized to use with the TELOSB mote due to the interfacing with the mote user button. The TELOSB mote needs to "warmup" before entering into normal operational behaviour, so, the user button is used to start the mote operation either by starting to send beacons, in the case of the ZigBee Coordinator, or to associate to a network in the case of ZigBee Routers or End Devices.

In order to test the cluster-tree approach we have forced the association to a specific parent device by assigning some static parameters to the device. These parameters are located in the nwk_const.h file under the *contrib.hurray.tos.lib.nwk* and are the following:

- TYPE_DEVICE – selecting the role of the device in the network;
- DEVICE_DEPTH – selecting the depth of the device in the network. This parameter in be used in computing the cskip functions used for the address assignment and for the tree-routing. This value will also be used to select the appropriate parent selected for the association.

Depending of the selected depth the device will select the statically defined parent. The parent values are assigned in the *NLME_NETWORK_DISCOVERY.request* primitive.The parents addresses (short address and extended address) are defined in the following variables:

Activated when the device depth is 0x01

- D1_PAN_EXT0 0x00000001
- D1_PAN_EXT1 0x00000001
- D1_PAN_SHORT 0x0000

Activated when the device depth is 0x02

- D2_PAN_EXT0 0x00000002
- D2_PAN_EXT1 0x00000002
- D2_PAN_SHORT 0x0001

Activated when the device depth is 0x03

- D3_PAN_EXT0 0x00000003
- D3_PAN_EXT1 0x00000003
- D3_PAN_SHORT 0x0002

Activated when the device depth is 0x04

- D4_PAN_EXT0 0x00000006
- D4_PAN_EXT1 0x00000006
- D4_PAN_SHORT 0x0022

In order for a cluster-tree to work properly there is a need to schedule the beacon frames. This is done by assigning a time offset to each routers (refer to section 3.6.1). The device assigned as a ZigBee Coordinator will accept the negotiation requests for beacon transmission. Upon the reception of these messages the ZC will execute the *process_beacon_scheduling* function that already has an offset list for each device (based on the short address). This function can be replaced with a scheduling algorithm.

6 References

[1] ZigBee-Alliance, "ZigBee specification," http://www.ZigBee.org/, 2006.

[2] Anis KOUBAA, Andre CUNHA, Mário ALVES, “A Time Division Beacon Scheduling Mechanism for IEEE 802.15.4/Zigbee Cluster-Tree Wireless Sensor Networks”, to be presented in Euromicro Conference on Real-Time Systems (ECRTS 2007), Pisa(Italy), July 2007

[3] IEEE 802.15.4 Standard-2003, "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)", IEEE-SA Standards Board, 2003.

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

[5] The OPEN ZigBee implementation - www.open-zb.net

[6] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler, “The nesC language: A holistic approach to network embedded systems”, in PLDI’03.

[7] www.tinyos.net

[8] Crossbow Technologies INC. http://www.xbow.com

[9] ATmega128L 8-bit AVR Microntroller Datasheet, Atmel ref: 2467MAVR-11/04, http://www.atmel.com

[10] Chipcon, SmartRF CC2420 Datasheet (rev 1.3), 2005.http://www.chipcon.com

[11] Chipcon, “Chipcon Packet Sniffer for IEEE 802.15.4”, 2006

[12]Daintree Networks, "Sensor Network Analyser, www.daintree.net," 2006