6LoWPAN
IPv6 for battery-less building networks

Abbasi, N.A.

Award date:
2009

Technische Universiteit Eindhoven

Department of Mathematics and Computer Science
and Department of Electrical Engineering

6LoWPAN: IPv6 for Battery-less Building Networks

By
Naveed A. Abbasi

Supervisors

Prof. Dr. J.J. Lukkien
Dr. P.D.V. van der Stok
Dr. Ir. M.C. W. Geilen
Dr. Ir. P.H.F.M Verhoeven

August 2009

Contents

1 Introduction **1**
1.1 Lighting Networks 1
1.2 Wireless Sensor Networks 4
1.3 Wireless Lighting Control Networks 4
1.4 Objectives 5
1.5 Outline of thesis 6

2 IEEE 802.15.4 MAC and PHY **7**
2.1 Device Types 7
2.2 Network Topologies 8
2.3 MAC Frame Format 9
2.4 PHY Specification 10
2.5 MAC Specification 11
2.5.1 MAC Data Service 11
2.5.2 MAC Management Service 12
2.6 Network Formation 13
2.7 PAN Setup 13
2.8 Robustness 15

3 Internet Protocol and ZigBee **16**
3.1 Internet Protocol version 6 16
3.1.1 Header Format 17
3.1.2 Address types 17
3.1.3 Address Representation 19
3.2 ZigBee 20
3.2.1 Device Types 20
3.2.2 Traffic Types 21
3.2.3 Addressing 22

3.3 IPv6 and ZigBee : A comparison of standards 22

4 6LoWPAN 23

4.1 IEEE 802.15.4 for IP 23
4.2 Addressing Modes 24
4.3 Stateless Address Autoconfiguration 24
4.4 Routing 25
4.5 Constraints 26
4.6 6LoWPAN Adaptation Layer 26
4.6.1 Header Types 27
4.6.2 Header Compression 30
4.7 6LoWPAN Neighbor Discovery 32
4.7.1 Node Registration/Confirmation 33
4.8 Router Solicitation Message 34
4.9 Router Advertisement Message 34
4.10 6LoWPAN-ND Message Options 34
4.11 6LoWPAN and ZigBee: A comparison 35

5 Development Environment 36

5.1 FreeRTOS 36
5.1.1 Scheduler 36
5.1.2 Inter-task Communication 37
5.1.3 Memory Management 38
5.2 SAND Node 38
5.3 BSN Node 39
5.4 MSPGCC 39
5.5 Debugging and Visualization 40

6 Implementation 41

6.1 Pre-porting Analysis 41
6.1.1 The 6LoWPAN Stack 41
6.1.2 FreeRTOS 42
6.1.3 Drivers 42
6.1.4 MAC 42
6.2 The 6LoWPAN Stack 43
6.2.1 sixLowPanLayer 43
6.2.2 ipv6Layer 44
6.2.3 udpLayer 44

6.2.4 socket 44
6.2.5 buffer 44
6.2.6 routingUnderIp 45
6.2.7 routingTable 45
6.3 Investigation Methodology 45
6.4 6LoWPAN Stack Extension 45
6.4.1 6LoWPAN Adaptation Layer 46
6.4.2 ICMPv6 Layer 46
6.5 UDP Chat Application 47
6.5.1 Operation 47
6.6 WASP Demonstration 48
6.6.1 Code Size Reduction 51

7 Self-powered Switch and Sensors 52
7.1 Requirements 53
7.2 Procedure 53
7.3 Problems 55
7.4 Acknowledgment Requirements 58
7.5 Timing Results 58
7.5.1 MSP430 Timing Results 59
7.5.2 AquisGrain2 Timing Results 59
7.5.3 Timing Analysis 60
7.6 Conclusion 60

8 Conclusion 62

A Header and Frame Formats 64
A.1 IPv6 Header 64
A.2 LOWPAN_IPHC Details 66
A.3 LOWPAN_UDP Details 67
A.4 Node Registration/Confirmation 68
A.5 Router Solicitation 70
A.6 Router Advertisement 71

B Packet Details 73
B.1 Route Request/Reply Packet 73
B.2 UDP Packet 74
B.3 Router Solicitation and Router Advertisement 75

C UDP Application **76**
C.1 Client 76
C.2 Server 77

D Error Log **78**

Bibliography **82**

Bibliography **82**

List of Figures

1.1 A Comparison on using different Physical Layer Architectures 2
1.2 Wireless Lighting Network overview 5

2.1 MAC Data Service 12
2.2 The Association Procedure 14

3.1 Unicast and Broadcast 18
3.2 Multicast and Anycast 19
3.3 The ZigBee Stack 21

4.1 6LoWPAN Stack Overview 28

5.1 SAND Node and BSN Node 39

6.1 Directory Structure of the Code 42
6.2 Driver Abstraction of CC2420 43
6.3 The 6LoWPAN Stack 44
6.4 Network and Software Architecture of 6LoWPAN 49
6.5 BSN Node and PC Interaction 50

7.1 Battery-less Switch Based Lighting Network 54
7.2 Battery-less Switch Circuitry 55
7.3 MSP430 Timing Results 59
7.4 AquisGrain 2 Timing Results 60

B.1 Route Request Message 73
B.2 UDP Packet with short addresses 74
B.3 Router Solicitation and Router Advertisement 75

C.1 UDP Client Block Diagram 76

C.2 UDP Server Block Diagram 77

List of Tables

1.1 A Comparison of LR WPAN with other wireless technologies 6

2.1 MAC Packet Structure 9
2.2 Frame Control Field 10

3.1 IPv6 Address Types 20

4.1 Dispatch Value Bit Pattern 27
4.2 Stack Comparison as in [1] 35

6.1 Memory Utilization on BSN Node 51

A.1 IPv6 Header 65
A.2 LOWPAN_IPHC Encoding 66
A.3 LOWPAN_UDP Encoding 67

Abstract

6LoWPAN is a protocol specification to enable IPv6 packets to be communicated on top of low power wireless networks, specifically IEEE 802.15.4. The rationale behind this protocol is that bulky IPv6 headers can be compressed to only few bytes by introducing an adaptation layer between IP stack's link and network layer, therefore reducing IP overhead. The transmission of 1280 bytes long IPv6 Maximum Transmission Unit (MTU) over IEEE 802.15.4 is also made possible using fragmentation and reassembly provided by this adaptation layer.

Philips Lighting is particularly interested in IPv6 connectivity for battery-less switches. These switches operate on the technology that converts the pressing action into small amount of energy. This energy can then be used by a radio transceiver to transmit a signal to luminaries. Within Philips Research a 6LoWPAN stack has been developed for the SAND node platform.

The purpose of the present assignment was to port the existing 6LoWPAN stack from the SAND node platform to the MSP430 platform, to enhance the stack capabilities by providing support for other features which were missing from existing implementation. Moreover the Neighbor Discovery Protocol for 6LoWPAN was to be investigated and implemented. Apart from these, a demonstration of 6LoWPAN based lighting network utilizing a self powered switch was also carried out as part of this project.

Chapter 1

Introduction

Recent advancements in communication technologies and network equipment have greatly expanded the horizons for business and residential connectivity for voice, internet and telecommunication. Network components are available at affordable prices, configuration is easy and help is readily available via websites and consulting services to install and manage such networks. The situation in building automation systems (BAS) is however, much different.

A BAS describes the functionality provided by the control system of a building. A BAS is an example of a distributed control system. It comprises a computerized, intelligent network of electronic devices and a control system to automate various constituents of BAS. The difficulties that are faced in BAS are:

- Many proprietary protocols are not open and their connectivity with other systems is cumbersome.
- The control network configuration is not flexible.
- Different systems exist next to each other and make it difficult to maintain and install.

When these networks employ different technologies, then the cost of installation and operation increases. Moreover connectivity becomes complicated as well.

We will discuss Lighting Networks as a part of BAS in more detail.

1.1 Lighting Networks

Lighting Networks are an important constituent of BAS. Traditionally lighting control networks have been operated via wired connections. Different standards evolved over time but their key

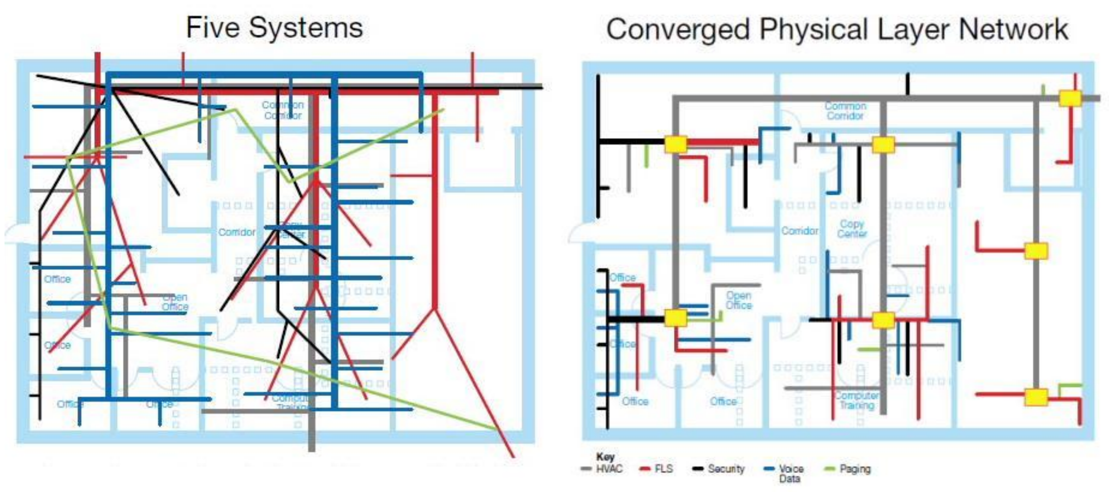


Figure 1.1: A Comparison on using different Physical Layer Architectures

characteristics remain the same. They are normally isolated from the external world and form a small, independent control group. Two major wired lighting control systems are DMX512 and DALI.

DMX512 is a standard wired communication protocol used extensively in industrial lighting applications, most commonly to control stage lighting and exhibition lighting. Developed by the Engineering Commission of USITT, the standard started in 1986. With subsequent revisions in 1990, the standard was named as *USITT DMX512/1990*. The revised edition of standard namely *"Entertainment Technology USITT DMX512A Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories"*, was approved by ANSI in November 2004. This current standard is more commonly known as DMX512-A.

The physical interface consists of 3 to 5 wires, with 1 data signal constructed from two differential lines (D+, D-), a common/ground and an optional second set of Data lines. It supports data rate up to 250 kbit/sec. It can support up to 512 devices in one network. DMX512 is unidirectional and does not include automatic error checking and correction, so it is not safe to use for applications involving life safety, such as controlling pyrotechnics or laser lighting display where audience or performer safety is involved [2].

DALI stands for Digital Addressable Lighting Interface. It is a standard wired communication protocol used for home and building lighting applications. It is an open standard, being standardized in accordance with International Electrotechnical Commission IEC 62386 [3].

The physical interface consists of two wires, DALI Power and DALI Data. The DALI system consists of one DALI control module with up to 63 DALI devices. It supports data rate up to 1200 bits/sec.

The limitations with existing lighting control and communication systems is their limited inter operability and their lack of connectivity with external world. A step towards improving connectivity is taken recently in the form of Architecture for Control Networks (ACN), a suite of network protocols for theatrical lighting control, being developed by Entertainment Services and Technology Association (ESTA). It may replace DMX as the control protocol for lighting systems. This protocol is designed to be layered on top of UDP/IP and therefore will run over standard, inexpensive Ethernet and 802.11 network links.

With ACN it is possible to have connectivity with external world, directly. DALI networks can connect to external networks via a gateway. However when it comes to flexibility and inter operability, the Internet Protocol has a proven track record. IP can be used for connectivity of all major systems and control protocols can be run on top of it, thus providing compatibility of various control systems.

Lighting connectivity and control systems exist in wireless world as well. Z-Wave, EnOcean and recently introduced ZigBee based control systems are some among those. Although all of them operate on wireless networks, they are not inter operable due to the physical characteristics of wireless networks as well as specifications of these protocols.

In a house with a different control network each for lighting, HVAC, security and automation, the layout and installation is complicated. Moreover operating all these different control systems will be a difficult task as well, as all of them will have different controllers. Having a common communication protocol for all different networks will ease the procedure of installation, operation and maintenance. Figure 1.1 depicts how cabling cost and configuration is simplified when using integrated system [4].

Philips Lighting is particularly interested in utilizing wireless battery-less switches in lighting networks. These switches operate on the principle of piezoelectric energy harvesting in which mechanical strain is converted into electrical current. Therefore whenever these switches are pressed, they produce a small amount of current. A microcontroller based node containing a radio transceiver utilizes this current to transmit wireless control messages to the luminaries. Philips Lighting in cooperation with Philips Research is also investigating the usage of IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN)[5] in these lighting applications employing battery-less switch, thus standardizing and having connectivity with the Internet. These wireless networks are low powered, lossy, exhibit low data rates and have limited communication range. These lighting control wireless networks are a special case of Wireless Sensor Networks (WSN).

1.2 Wireless Sensor Networks

A Wireless Sensor Network (WSN) is a network of spatially distributed small autonomous systems, called sensor nodes which cooperate over a wireless medium to support at least one common application. These wireless sensor networks are used in various industrial and civilian application areas, including but not limited to, process monitoring and control, environment and health monitoring, healthcare applications, home automation and traffic control. In addition to sensors each node contains a radio transceiver, a small microcontroller and limited energy source, either a battery or an energy scavenging device. Their sizes vary depending on the nature of application. Similarly their costs also vary depending on the size of network, the size of business and the complexity required for application.

WSN are different from traditional wireless networks, such as Mobile Ad Hoc Networks due to various principle differences, mainly due to their nature of applications, environment interaction, scale of deployment, energy costs, self configurability, data centricity, resource scarceness and mobility[6].

IEEE 802.15.4 is a standard which specifies physical and media access control layer for low rate wireless personal area networks. It is maintained by the IEEE 802.15.4 working group. It provides the basis for ZigBee[7], WirelessHART and MiWi specifications. It is also an active area of investigation for provision of Internet Protocol version 6 (IPv6). The availability of IPv6 over WSN ensures connectivity of such networks with the Internet. By supporting IPv6 these nodes will be able to communicate with any IPv6 enabled host over the Internet, benefit from standardized and already established services, network management tools and end-to-end reliable communication through existing transport protocols. However IEEE 802.15.4 is ideal for low power and low data rate networks, a comparison with other wireless networks is made in Table 1.1. IPv6 requires more complicated headers which are large in nature, and impose heavy overhead. Therefore a tailored version of IPv6, namely 6LoWPAN is being defined and implemented on IEEE 802.15.4.

1.3 Wireless Lighting Control Networks

Wireless Lighting Control Networks employ wireless communication between control systems installed in luminaries and switches. These control systems are microcontroller based devices containing transceivers. They are installed inside luminaries and can control them. The switches are also built on similar principle, however unlike control systems in luminaries, they require human interaction. A new generation of switches does not require batteries, instead they generate a small amount of energy when pressed. This energy can be used to power-up the chip attached to the switch and then using its radio, transmit the control packets to the luminary controllers.

In this assignment a battery-less switch was deployed in a lighting control network, which operates on 6LoWPAN based network. The structure of the network is shown in Figure 1.2. Whenever the

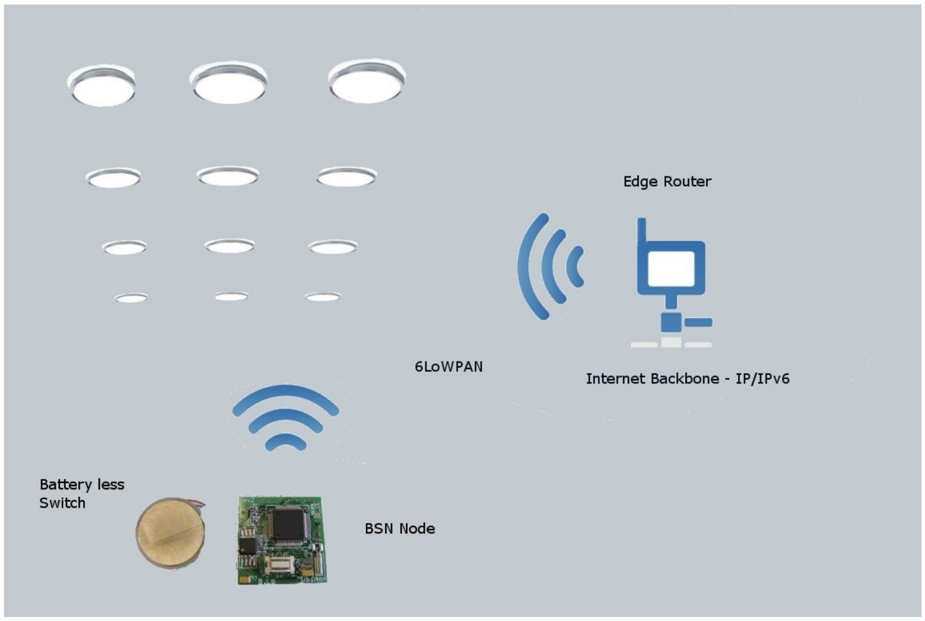


Figure 1.2: Wireless Lighting Network overview

switch is pressed, it powers up the attached BSN node[8]. This BSN node then sends control packets to the luminaries via 6LoWPAN stack. These luminaries can also be accessed through internet using Edge Router, which provides interface from 6LoWPAN to the internet. Messages directed to the luminaries pass through Edge Router and messages addressed to the nodes outside pass through it as well. In practice, a user can check the status of the lightings anywhere just with the basic access of the internet. User will send a command to the luminaries routed through Edge Router and in reply, the luminaries will send status via Edge Router.

1.4 Objectives

Philips and TU/e are participating in a European Union project WASP, which stands for Wirelessly Accessible Sensor Populations. This project is aimed at narrowing the mismatch between research at the application level and the node and network level by covering complete technology chain from sensor node hardware to the implementation of application [9].

Within WASP, a common platform for development is in the form of Imperial College BSN nodes which are based on MSP430 microcontroller derivatives [8]. Every node contains a radio transceiver, for which different MAC versions can be used. One such implementation employs MAC libraries provided by Chipcon. Within Philips a 6LoWPAN stack with limited functionality was already developed, for the Small Autonomous Network Devices (SAND)[10] architecture .

This thesis project has following objectives.

Table 1.1: A Comparison of LR WPAN with other wireless technologies

	$WLAN$	$BT - basedWPAN$	LR WPAN
Range	˜ 100 m	˜ 10-100 m	10 m
Data throughput	˜ 2-11 Mb/s	1 Mb/s	less than 0.25 Mb/s
Power Consumption	Medium	Low	Ultra low
Size	Larger	Smaller	Smallest

- Port the existing 6LoWPAN stack implementation from the SAND node to the MSP430 architecture.
- Develop the stack so as to comply with predefined standard and proposed draft [11].
- Investigate and develop the Neighbor Discovery for 6LoWPAN nodes as proposed in [12].
- To provide the wireless IP based lighting demonstration to Philips Lighting employing a battery-less switch as mentioned in 1.3.
- To assist in Philips-WASP project 6LoWPAN demonstration, i.e. inter operability of 6LoWPAN stack with IPv6 wired network.

1.5 Outline of thesis

The remaining report is structured as follows. Chapter 2 will give details about the IEEE 802.15.4 MAC and PHY protocol. In Chapter 3, Internet Protocol version 6 is discussed and compared with ZigBee. Chapter 4 summarizes the 6LoWPAN standard and the challenges that needed to be overcame in order for successful deployment over IEEE 802.15.4. Details about the environment are given in Chapter 5. Implementation details and some applications are given in Chapter 6. Philips Lighting project demonstration and details are given in Chapter 7. Results and concluding remarks are given in the Chapter 8.

Chapter 2

IEEE 802.15.4 MAC and PHY

IEEE 802.15.4 is a standard which specifies the physical layer and media access layer for low-rate wireless personal area networks (LR-WPANs). It is maintained by the IEEE 802.15 working group. It is meant to operate at lower power for extended duration of time. The data rate for 802.15.4 varies from 20 to 250 kbps. These devices are designed to be of extremely low cost and small size.

Medium Access Control (MAC) provides addressing and channel control system. The fundamental task of a MAC protocol is to regulate the access of a number of nodes to a shared medium in a way to meet some performance goal. In some networks these goals are throughput, delay and fairness, while in WSN major criteria is energy.

Physical Layer (PHY) consists of basic hardware transmission technologies of a network, providing means to transmit raw bits rather than logical data packet. It mediates between the transmission and the reception of wireless waveforms and the processing of digital data in the remaining higher level layers.

The MAC protocol is the first protocol layer above PHY and consequently MAC protocols are heavily influenced by its properties [6]. Therefore to achieve best possible performance, the MAC layer is optimized for specific characteristics of underlying PHY layer for various parameters including energy, delay, efficiency and throughput.

This chapter summarizes various features of the IEEE 802.15.4 MAC and PHY layer.

2.1 Device Types

IEEE 802.15.4 describes two device types:

- Full Function Device (FFD)
  Such a device has implemented the all primitives and functions, which are defined in the standard. An FFD can act as Personal Area Network (PAN) coordinator, a (cluster) coordinator

or an ordinary device.

- Reduced Function Device (RFD)
  In contrast to FFD, the RFD only consists of minimal implementation of the standard. So it is not possible for such a device to act as coordinator.

As a result, an FFD can talk to any device in the network, be it an RFD or another FFD, whereas an RFD can only talk to an FFD. The use of RFDs is typically limited to simple application scenarios in which individual nodes possess basic communication capability but little computational capability. Hence for a FFD more memory capacity is needed and a consequence of that, the power consumption and hardware costs are higher. An RFD has low power consumption and low hardware requirements.

The distinction between these two device types offers the way to design systems in a flexible and economic way.

2.2 Network Topologies

IEEE 802.15.4 standard supports multiple network topologies, including both star and peer-to-peer networks. The topology is an application design choice; some applications which require low latency will preferably be deployed using star topology however applications requiring large area coverage may require peer-to-peer networking, as in perimeter security. Multiple address types, including both physical (i.e. 64 bit IEEE) and short (i.e. 16 bit network assigned) are provided.

- Star Topology
  In the networks with star topology, the communication is between devices and a single central controller called the PAN coordinator. In this topology a device is either an initiator or the end receiver of network communication. A PAN coordinator is an FFD. It can initiate, terminate or route communication around the network. It is the primary controller of the network. If a device wants to communicate with another device in the PAN it sends the message to the PAN coordinator which then forwards the message to the destination device.

- Peer-to-Peer Topology
  In a peer-to-peer network each device can communicate with every other device in its environment directly. It also has a PAN coordinator; however it is not the central point of communication. The technology allows the more complex network formations to be implemented, include mesh topology. If a device wants to communicate with another device which is not in its immediate environment, a routing scheme is employed.

2.3 MAC Frame Format

Each MAC layer packet consists of the header, which provides administrative and security information, a variable length payload and a footer which contains the frame check sequence. The details of the MAC layer packet structure are shown in Table 2.1.

Table 2.1: MAC Packet Structure

Element	*Field*	Length (in bytes)
Header	Frame Control	2
	Sequence Number	1
	Destination PAN Identifier	0 or 2
	Destination Address	0, 2 or 8
	Source PAN Identifier	0 or 2
	Source Address	0, 2 or 8
	Auxiliary Security Header	0, 5, 6, 10 or 14
Payload	Frame Payload	Variable
Footer	Frame Check Sequence	2

The first field of the MAC header is the frame control field. It indicates the type of MAC frame being transmitted, specifies the format of the address field and controls the acknowledgment. Further details about the frame control field are given in Table 2.2. The values not used in the table are reserved for future use.

The sequence number field contains the beacon sequence number, in case of a beacon frame, or a data sequence number that is used to match an acknowledgment frame to the corresponding data or MAC command frame. A frame transmission is considered successful only if the sequence number in the acknowledgment frame is the same as in the data frame. The size of an address field may vary between 0 and 20. For example, a data packet may contain both source and destination address information, i.e. address and PAN identifier. While the return acknowledgment frame does not contain any information at all. On the other hand a beacon frame may only contain the source address information. In addition, short 16 bit or 64 bit IEEE addresses may be used. The short address value of `0xFFFF` is used for broadcast transmission, which should be accepted by all devices currently receiving on that channel. A radio channel is a specific frequency, pair or band of frequencies, named with a number. IEEE 802.15.4-2003 has 27 channels, numbered 0 through 26.

The payload field is variable in length; however the complete MAC frame may not exceed 127 bytes in length. The data contained in the packet is dependent on the frame type. IEEE 802.15.4 has four different frame types. These are the beacon frame, data frame, acknowledgment frame and MAC command frame. Only the data and beacon frame actually contain information sent by higher layers; the acknowledgment and MAC command frame are originated in MAC layer and are used for MAC peer-to-peer communication. The FCS in an IEEE 802.15.4 MAC frame is a 16 bit International Telecommunication Union - Telecommunication Standardization Sector (ITU-T)cyclic

Table 2.2: Frame Control Field

Subfield	*BitNo.*	*Values*	Meaning
Frame Type	0-2	000	Beacon
		001	Data
		010	Acknowledgment
		011	MAC Command
Security Enabled	3	1	Frame is protected
Frame Pending	4	1	more data is pending
Acknowledgment Request	5	1	Acknowledgment is requested
PAN ID Compression	6	1	Dest. and Source PAN ID equal
			the latter can be omitted
Destination Addressing Mode	10-11	00	PAN ID and address not present
		10	16 bit short addresses used
		11	64 bit extended addresses used
Frame Version	12-13	00	Frame compliant with 2003 standard
		01	Frame compliant with 2006 standard
Source Addressing Mode	14-15	00	PAN ID and address not present
		10	16 bit short addresses used
		11	64 bit extended addresses used

redundancy check.

2.4 PHY Specification

The IEEE 802.15.4 PHY is responsible for the following tasks.

- Activation and deactivation of the radio transceiver
- Data transmission and reception
- Channel clear assessment (CCA) for CSMA/CA
- Link quality index (LQI) for received packets
- Energy Detection (ED) for current channel

The PHY provides an interface between MAC sublayer and physical radio channel, via RF firmware and RF hardware. The PHY includes a management entity called the Physical Layer Management Entity (PLME). This entity provides the layer management service interface through which layer management functions may be invoked. The PLME is also responsible for maintaining a database PHY PAN information base (PIB). This database contains information about managed objects.

The PHY provides two services:

- the PHY data service

- the PHY management service

Following are some of the PHY management services.

- Clear Channel Assessment
  Before sending a frame, the PHY needs to do a clear channel assessment. This is part of CSMA/CA, the collision management protocol. It prevents the sending of a frame when there is already another frame being transmitted and thus avoids corrupting it.
- Get/Set PHY PIB
  The PHY PAN Information Base (PIB) contains information related to the PHY configuration. Most of these values are constants. Some of the values it contains are as following.
  - Current Channel
  - Channels Supported
  - Transmit Power
- Set Tx/Rx State
  This service allows the PHY transceiver to be enabled, disabled, set in Rx mode or set in Tx mode.

2.5 MAC Specification

The IEEE 802.15.4 MAC is responsible for the following tasks.

- Generating network beacons if the device is a coordinator
- Synchronizing to the beacons
- Support PAN association and disassociation
- Employ CSMA/CA mechanism for channel access

The MAC provides two different types of service: the MAC data service and the MAC management service. These services utilize underlying PHY layer's data and management services. An FFD must implement the entire MAC data service, but there are some capabilities in MAC management service that are optional for FFDs. There are a number of capabilities in both data and management service that are optional for RFDs. A brief overview is given as follows.

2.5.1 MAC Data Service

The important primitives that supported and required by every application are as following.

- Data Request
  This is a transmit function. On calling this request, the data is taken from high level and put into the outgoing FIFO of the radio.
- Data Confirm
  When data is transmitted, then the sending radio gives some indication of the status to MAC. The status information will tell whether data was transmitted successfully or it failed due to interference or some other catastrophic failure.
- Data Indication
  This is signaled by the Rx Data ISR. When data arrives in radio's inbound FIFO, then it will trigger the interrupt. From the ISR, MAC is normally signalled that data was received and needs processing.

Figure 2.1 denotes a channel access by two nodes using MAC Data Service.

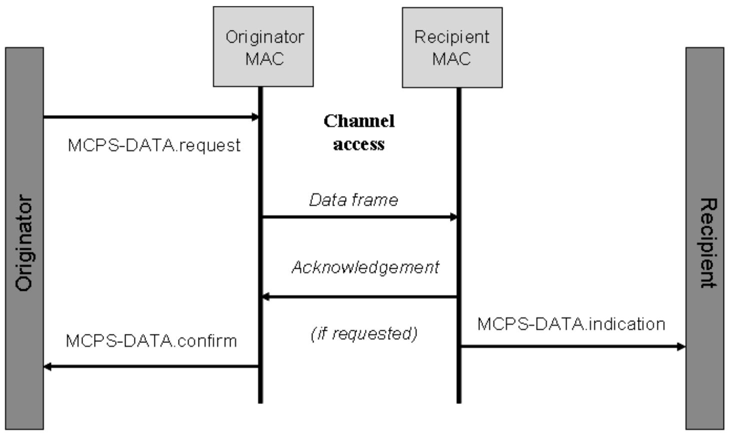


Figure 2.1: MAC Data Service

2.5.2 MAC Management Service

Some management services offered by the MAC layer are follows:

- Association and disassociation
- Beacon Notification
- Request data from coordinator
- Initiate a channel scan over a given list of channels
- Enable receiver for a finite period of time

Higher layers use these services to request PLME to perform above mentioned actions.

2.6 Network Formation

The association is a process in which a device joins a network. The MAC layer provides the association procedures to the Network layer. The Network layer manages the network formation. IEEE 802.15.4 MAC layer provides four service primitives for MAC association procedure.

- Associate Request
  This primitive is used by the Network layer of the device that requests joining a coordinator. This request also provides the list of capabilities of the device that requests to join the network.
- Associate Indication
  When the MAC of the coordinator receives Associate Request from a node, it uses Associate Confirm primitive to inform its Network layer about the pending request.
- Associate Response
  Upon receiving Associate Indication, Network layer uses Associate Response to inform about the decision to its own MAC layer.
- Associate Confirm
  The response from PAN coordinator is not sent immediately. Instead the requesting node waits for a certain amount of time and then requests the result of association of request. The MAC of coordinator responds with the result. Associate Confirm is used by the MAC layer of requesting node to inform its own Network layer about the decision of PAN coordinator.

This procedure can be seen in Figure 2.2.

2.7 PAN Setup

When a device boots up, it can either become a coordinator or associate to another coordinator. Coordinators can operate in a peer-to-peer fashion and multiple coordinators can form a Personal Area Network (PAN). A PAN is identified by a 16-bit PAN identifier and one of its coordinators is designated as a PAN coordinator.

Before a node can associate, it first needs to know about the reachable PANs and their link quality which is determined by the scan process. This can be done by either active or passive scan. In passive scan the device turns on its receiver and listens to the channel for a defined period of time. After that it switches to another channel and performs the scan again. When all requested channels are scanned, the MAC sublayer informs the next higher layer about all discovered PANs. The active scan is quite similar to passive scan, however in this case the node transmits a beacon request frame and waits for a definite period of time, for a beacon frame of the coordinator. Hence the active scan is used for non beacon enabled PANs. If a PAN coordinator in a non-beacon enabled PAN receives a beacon request, it will reply with a single beacon frame.

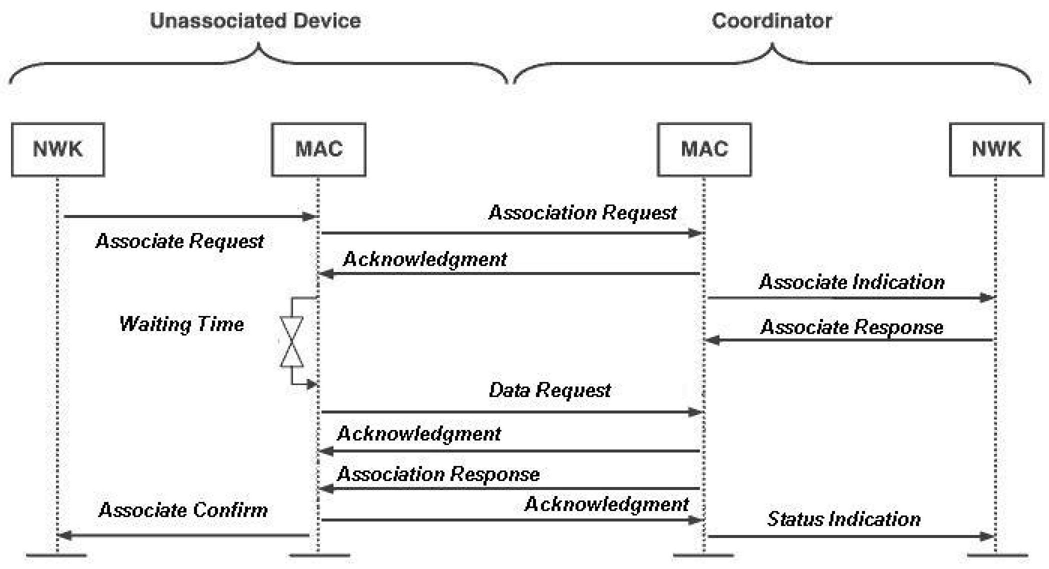


Figure 2.2: The Association Procedure

When a device discovers a PAN coordinator through a scan, it can start the association process. Before the request, the internal state of the device must be updated. A higher layer in the new device sends an association request to its own MAC layer, which is relayed to the coordinator. If the coordinator's MAC layer receives this message, it informs the next higher layer. The higher layer decides whether or not the association can be made. Depending on this decision, the coordinator's MAC layer sends a response to the device which requested the association. When the device receives this message, it informs its higher layer. If the association was successful, the received short address is stored into its internal data structure.

If a device wants to disassociate from a PAN, the next higher layer sends the disassociation frame to MAC. The MAC layer generates the disassociation command frame and sends to the PAN coordinator. After receiving the acknowledgment frame from the coordinator, the originating node's MAC layer informs the next higher layer about successful disassociation. If the device does not receive an acknowledgment frame, it indicates disassociation anyway.

On the other hand, if a coordinator wants a device to be disassociated, it sends such request to its own MAC layer. The MAC layer sends a disassociation notification to that node. This node should send an acknowledgment frame back to the coordinator. Upon reception of such acknowledgment frame, the MAC layer of the coordinator informs the next higher layer about the disassociation. If no such acknowledgment is received, the MAC layer of the coordinator indicates the next higher layer the disassociation anyway.

2.8 Robustness

Robustness in an IEEE 802.15.4 based network is achieved by using various channel access mechanisms, data verification and optional acknowledgments. The channel access can be done either in the beacon mode or non-beacon mode. In the beacon mode, the PAN coordinator regularly transmits frame beacon packets announcing the PAN identifier, a list of outstanding frames and other parameters [6]. The frame beacon includes a superframe specification describing the length of various components of the superframe that follows the beacon frame. The superframe is divided into active period and inactive period. During the inactive period all nodes including the coordinator can switch off their transceivers and go to sleep mode. The active period is subdivided into a Contention Access Period (CAP) and Guaranteed Time Slots (GTSs). During CAP slotted CSMA/CA is used.

In non-beacon networks, transmission is done using a unslotted CSMA/CA channel access mechanism. Whenever a node has to transmit a frame, it has to wait for a random period of time. After that it checks if the channel is idle or not. If it is idle, the frame is transmitted. If the channel is busy, the node will wait for another random period of time before retrying. Beacon enabled networks use slotted CSMA/CA. The backoff slots are aligned with the start of beacon transmission. A device wishing to transmit waits for a random number of backoff slots. If the channel is idle after this waiting period, the packet is transmitted. In case the channel is busy, the node again waits for a random number of slots. The data verification in 802.15.4 based network is done using CRC. It is performed on every frame to detect bit errors. The acknowledgment frames can be requested by the transmitter for data and command frames. GTS can be used for the data that requires timeliness and real-time guarantees.

Chapter 3

Internet Protocol and ZigBee

LR-WPAN defines only two layers of the 7-layer OSI model: the physical layer and MAC sub layer. The upper layers are left for developers to implement. A variety of host protocols can be implemented on top of IEEE 802.15.4. Among them, ZigBee and IPv6 are two distinct and prominent choices. The IPv6 implementation is only possible through an adaptation layer namely 6LoWPAN, which is discussed in Chapter 4. ZigBee is a specification for a suite of high level communication protocols to operate on top of the IEEE 802.15.4 standard. It is maintained by the ZigBee alliance. In this chapter, we will discuss the Internet Protocol and ZigBee.

3.1 Internet Protocol version 6

Internet Protocol Version 6 is a best effort layer 3 network protocol. It neither guarantees delivery, nor assures proper sequencing or avoid duplicate delivery. These aspects are taken care of by upper layer protocols such as UDP and TCP. IPv6 is a successor of IPv4. IPv6 introduces changes in address size, header formats and address assignment. Network security is integrated into the design of IPv6.

IPv6 has a much larger address space than IPv4: addresses in IPv6 are 128 bits long, compared to 32-bit addresses in IPv4. It also has a goal to reduce time required to configure and manage systems. An IPv6 system can participate in stateless autoconfiguration, where it can create its guaranteed unique IP address by combining its MAC address with a prefix provided by the network router. Therefore DHCP is not needed in this configuration for address assignment, whereas in IPv4 it was a requirement. IPv6 headers are fixed size and more regular than IPv4 counterparts. Thus resulting in improved performance and more extension possibilities. Multicast is a requirement in IPv6 specifications and although supported in IPv4, not every router and host supports it. Moreover Neighbor Discovery is a part of IPv6. It specifies a set of ICMPv6 messages, which replaces Address Resolution Protocol, ICMP Router Discovery and ICMP Redirect used in IPv4.

3.1.1 Header Format

An IPv6 packet includes an IPv6 header. It is placed at the beginning of the IPv6 packet and is 40 bytes long. The structure is given in Appendix A.1.

IPv6 contains optional information in separate extension headers. They are placed between the IPv6 header and the IPv6 payload. The presence is indicated in the `Next Header` field. Every extension header also has this field. Hence a sequence of headers can be formed. These headers can contain Hop-by-Hop Options, Destination Options, Routing, Authentication and Destination Options among many other possible options. The value 59 in `Next Header` field means that there is nothing following that header.

The IPv6 layer requires the underlying layer to support an MTU of at least 1280 bytes.

3.1.2 Address types

IPv6 addresses are 128 bits long. Unlike IPv4, it does not use classes. However it defines three different IP address types

- Unicast
  A unicast address defines a single recipient and a packet addressed to such address is only delivered to that specific recipient. IPv6 unicast addresses can be classified according to their scopes. A scope determines the applicable area. An IPv6 unicast address can have global, site-local or link-local scopes. A global scope means that the address is valid for communication with any node on the IPv6 internet. Site-local address is defined to be configured in a networked site made up of several subnets. Link-local address is used within a link.

- Multicast
  Multicast addresses are used to allow a single device to send a packet to a group of recipients. These recipients must belong to a multicast group. The packet addressed to this group will be delivered to all nodes belonging to this group. There is one-to-many relationship between network addresses and network end points. Each destination identifies a set of receiver end-points, to which all information is replicated.

  IPv4 multicast address starts with `1110`. Remaining 28 bits identify multicast group address. In contrast, IPv6 start with `0xFF`, 4 bits for Flags and 4 bits for Scope ID. Remaining 14 bytes identify actual group address. Flags denote the nature of multicast address. Multicast addresses can be *well-known* when allocated by IANA or *transient* when they are not permanent. Multicast address can have one of the following different scopes.

  - Interface-local: meaning packet with this destination address may not be sent over any network link, but must remain within current node. It is equivalent to unicast loopback address.

  - Link-local: meaning packet with this destination address may only be sent to local neighbors.
  - Admin-local: meaning packet with this destination address must be administratively configured.
  - Site-local is restricted to local physical network.
  - Organization-local: restricted to networks used by the organization administrating the local network.
  - Global: eligible to be routed over the public internet.

  In IPv4 a specific address, `0xFFFFFFFF` is used for broadcast. IPv6 does not specify an address for broadcast. This can be achieved by defining a link-local multicast group and sending a multicast message on that address.

- Anycast

  An anycast address is used to identify a set of recipients where a packet sent to the anycast address is routed to the nearest recipients belonging to that group, depending on the distance of the routing algorithm. While a unicast address requires the message to be sent to a specific recipient and a multicast address requires the message to be sent to *every* member in that multicast group, an anycast requires the message to be sent to *one* member of the anycast group. The destination is chosen by router for efficiency in term of routing distance.

  Like multicast, in anycast there is also one-to-many relationship between network addresses and end points, only one of those end points is chosen at any given time to receive information from any sender.

The above types are shown in Figure 3.1 and Figure 3.2.

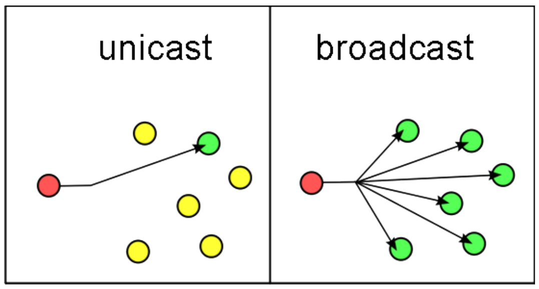


Figure 3.1: Unicast and Broadcast

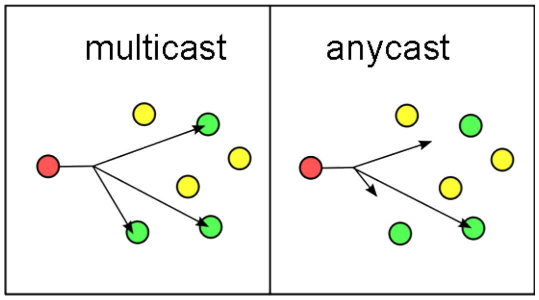


Figure 3.2: Multicast and Anycast

3.1.3 Address Representation

A common representation of IPv6 address is `xx:xx:xx:xx:xx:xx:xx:xx`, where each `x` denotes a byte. Also it can be represented as eight group of four hexadecimal digits. An example of an IPv6 address is as following

```
2001:1db8:85e3:0000:0000:8a2a:1370:2354
```

To shorten the writing and presentation of addresses, some simplifications to the notation are permitted.

- Any leading zeroes in a group may be omitted, thus the above example becomes

```
2001:1db8:85e3:0:0:8a2a:1370:2354
```

- Furthermore consecutive groups of 0 values can be replaced by two colons `::`.

```
2001:1db8:85e3::8a2a:1370:2354
```

However only one "::" simplification is allowed in any address, to avoid ambiguity.

To represent a prefix, the format `ipv6_address/prefix_length` is used. where

- `ipv6_address` is an IPv6 address in any of notations given above
- `prefix_length` is a decimal value specifying how many of remaining most contiguous bits of the address comprise of the prefix.

For example, the following are legal representations of 32-bit prefix `2001D0D0` (hexadecimal):

```
2001:D0D0:0000:0000:0000:0000:0000:0000/32

2001:D0D0:0:0:0:0:0:0/32

2001:D0D0::/32
```

Table 3.1: IPv6 Address Types

Address Type	Binary Prefix	IPv6 Notation
Unspecified	00...00 128 bits	::/128
Loopback	00...01 128 bits	::1/128
Multicast	11111111	FF00::/8
Link-local Unicast	1111111010	FE80::/10
Global Unicast	Everything else	Empty

A list of IPv6 address prefixes and notation is given in Table 3.1.

The IPv6 header does not provide a checksum. However it provides a pseudo-header which can be used by upper layers for checksum calculation. A pseudo-header contains IP source address, destination address, protocol and length fields. A byte consisting of zeros is added to this header and then this header is added to the actual message. Checksum is computed over the combination of pseudo-header and real message and the value is placed in the checksum field. The pseudo-header is only used for computation of checksum and not transmitted along with real message. The destination higher layer performs same action and computes a checksum to be compared in the transmitted header.

3.2 ZigBee

ZigBee is built on top of the IEEE 802.15.4 standard. Therefore it is tied to this specification. It uses standard's functionalities and adds its own features on top of it. It provides specification for network and application layer along with providing security features. It is maintained by the ZigBee alliance. The ZigBee alliance was established in 2002. It is an association of companies working together to enable reliable, cost-effective, low power and wirelessly networked products based on IEEE 802.15.4. The ZigBee stack is shown in Figure 3.3.

3.2.1 Device Types

ZigBee distinguishes devices either on the basis of physical characteristics or logical role.

1. Physical Device Types
   To provide low cost implementation options, the ZigBee Physical Device type distinguishes the type of hardware on the basis of IEEE 802.15.4 definition of devices. These types are fully functional device (FFD) and reduced function device (RFD).

2. Logical Device Types
   The ZigBee Logical Device type distinguishes the Physical Device Types (FFD or RFD) on the basis of their role in the ZigBee network. The device types are ZigBee Coordinators, ZigBee Routers and ZigBee End Devices. The ZigBee coordinator initializes a network, manages

network nodes and stores network information. The ZigBee router routes messages between paired nodes. The ZigBee End Devices act as a leaf node in the network and can be either an RFD or FFD.

A ZigBee network must have a coordinator, which along with routers must be mains-powered. End nodes can be battery-powered.

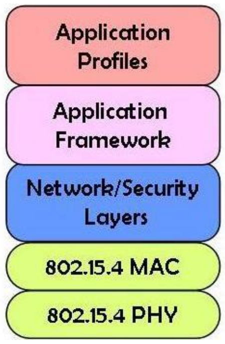


Figure 3.3: The ZigBee Stack

3.2.2 Traffic Types

ZigBee networks support multiple traffic types with their own unique characteristics, which are as follows.

- Periodic Data
  This type of data is usually defined by sensor based applications. These sensors provide data on timely intervals. Typically this type of traffic is handled using the beacon system, where sensor nodes wake up at a set time, check for the beacon, synchronize, exchange data and go back to sleep.

- Intermittent Data
  This type of data is derived from the application or an external event, such as pressing a wireless switch. Data can be handled in a beaconless or disconnected system. In disconnected mode, the attachment to the system is only done when communication is required.

- Repetitive Low Latency Data
  This type of data is handled using guaranteed time slots (GTS). A GTS is a part of contention-free period (CFP) during which a node ensures the transmission without any contention. GTS are managed by MAC. Applications requiring timeliness and critical data passage may include medical alerts and security system.

ZigBee networks are primarily intended for low duty cycle sensor networks. ZigBee applications have the ability to quickly attach, send information, detach and go to sleep; which results in low power consumption and extended battery life.

3.2.3 Addressing

The devices with IEEE 802.15.4-compliant radio have a 64-bit address. This is globally unique address made up of Organizationally Unique Identifier (OUI) and 40 bits assigned by the manufacturer of that module. OUIs are obtained from the IEEE.

When a device joins ZigBee network, it gets a 16-bit address. This address is unique within the network. Either of these, the 64-bit or 16-bit address can be used within the PAN to communicate with the device. The coordinator always has the 16-bit address '0'.

3.3 IPv6 and ZigBee : A comparison of standards

While ZigBee is a complete suite on top of a single PHY and MAC specification, IPv6 is just a layer that theoretically can exist on top of any PHY and MAC. However we can compare the usefulness of implementing IPv6 on sensor networks against an existing and quite mature ZigBee suite. Here is a comparison between both existing technologies.

1. ZigBee is using a small scale and ad-hoc networking model. IPv6 is massively scalable and can provide end to end connectivity to the internet.
2. ZigBee is limited to a single radio standard, i.e. IEEE 802.15.4, whereas IPv6 (or IP) is already existing on various wired and wireless standards, e.g. IEEE 802.3 and IEEE 802.11.
3. ZigBee is a standard of an alliance, which is not open to non-members. IPv6 on the other hand, is an open standardization group.
4. ZigBee has already defined service and device descriptions. No such descriptions are yet available for IPv6 based networks.

An implementation of IPv6 on IEEE 802.15.4 is not a straightforward task and there are limitations of the underlying network. The version of IPv6 to be implemented on IEEE 802.15.4 is named 6LoWPAN, which will be discussed in chapter 4.

Chapter 4

6LoWPAN

6LoWPAN is an acronym for IPv6 over Low power Wireless Personal Area Networks. It is managed by IETF 6lowpan working group. It defines the frame format for the transmission of IPv6 packets over IEEE 802.15.4. Since IPv6 requires support for packet size larger than 802.15.4 can handle, an adaptation layer is defined in this specification. It also defines the header compression and supports fragmentation and reassembly below IP layer[5].

It is assumed that a PAN maps to a specific IPv6 link. A link is a medium over which nodes can communicate at the link layer. Strictly speaking, in wireless networks a link is only single hop - the radio range of one node. In a wired network a link contains all the nodes that share the communication medium to a router. For example let us take three nodes A, B and C. In a wired network if A can communicate to B directly and B can communicate directly to C, A can also directly communicate to C. Therefore A and C are said be on the same link. In wireless network, this is different. If A can communicate to B directly and B can communicate directly to C, it may be possible that A can not communicate directly to C, due to factors such as radio range. In WSN a link of a given node is limited to only one hop. A mesh is formed for multiple hop communication. A 6LoWPAN network can also be seen as a mesh of various wireless links.

A PAN ID can also be used in an IPv6 address. One 6LoWPAN mesh (IPv6 link) maps to one PAN ID.

4.1 IEEE 802.15.4 for IP

IPv6 packets must only be carried on data frames of IEEE 802.15.4, which can optionally request that they may be acknowledged. It is recommended to do so (request acknowledgments) in order to aid link-level recovery.

This specification does not require that IEEE 802.15.4 networks run in beacon enabled mode. It is however, recommended in this specification that beacons to be configured so as to assist association and disassociation events. This specification also requires both source and destination addresses to be included in the IEEE 802.15.4 frame header. The source or destination PAN ID may also be included.

4.2 Addressing Modes

IEEE 802.15.4 defines various addressing modes: it allows the use of either an IEEE 64-bit extended address or a 16-bit short addresses unique within the PAN. These short addresses can only be used after association with a PAN coordinator. This specification supports both addresses. It also assumes that a PAN maps to a specific IPv6 link. IPv6 supports multicast but not broadcast whereas IEEE 802.15.4 does not support multicast natively. Instead multicast is performed on it using link-layer broadcast. In order to do a multicast a node has to do a broadcast in a way that only a specific multicast group is recipient of such messages. This is carried out as below.

1. The destination PAN ID in the frame header must match the PAN ID of the link.
2. The short destination address in frame header must match the broadcast address, i.e. `0xFFFF`.
3. The nodes that receive this message, parse it at higher level and find out if that message was intended for them or not.

A value of `0xFFFF` refers to broadcast and must be accepted by all nodes. Additionally within 6LoWPAN networks, 16-bit addresses are divided into 5 different classes. Only two of these classes are defined as of now, the others are reserved for future use. In following listing 'x' is a place holder for an unspecified address format.

1. `0xxxxxxxxxxxxxxx`
   The first bit (bit 0) shall be zero if 16-bit address is a unicast address. This leaves 15 bits for actual address.
2. `100xxxxxxxxxxxxx`
   Bits 0, 1 and 2 shall follow the pattern above in case of multicast address. Remaining 13 bits will contain the actual multicast address.

4.3 Stateless Address Autoconfiguration

6LoWPAN also provides support for stateless address autoconfiguration, which means that a host can generate its own addresses using a combination of locally available information and information advertised by routers, without making stateful binding with routers. Since every 802.15.4 device has a unique EUI-64 identifier, an IPv6 interface identifier can be obtained from this EUI-64 identifier using stateless autoconfiguration as explained in [13].

Although every device has a EUI-64 address, a 16-bit short address can also be given to such a device after the association with PAN coordinator. These addresses should be unique within a PAN. The specification of such addresses is given as following .

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     16-bit short Address      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

It is also possible to use 16-bit addresses for autoconfiguration. In this case a pseudo 48-bit address is formed by concatenating this 16-bit address with the 16-bit PAN ID (or 16 zero-bits in case PAN is not known) and 16 zero-bits in following way.

```
16_bit_PAN:16_zero_bits:16_bit_short_address
```

Afterwards an interface identifier (64-bit) can be formed from above 48-bits using "IP over Ethernet" specification [13].

The IPv6 link-local addresses are formed by appending the interface identifier as defined above, to the prefix `FE80::/64` as below.

```
 10 bits            54 bits                  64 bits
+----------+-----------------------+----------------------------+
|1111111010|         (zeros)       |    Interface Identifier    |
+----------+-----------------------+----------------------------+
```

4.4 Routing

Network routing defines the ability to send a unit of information from one node to another by determining a single path through the network while meeting certain efficiency criteria. Within a 6LoWPAN header format, routing protocols can be defined at two layers, which are as following.

1. Mesh-under Routing
   The mesh-under routing supports routing under the IP link and is directly based on link-layer standard 802.15.4 standard. It utilizes 64-bit IEEE extended addresses or 16-bit short addresses. This routing can only be used within a 6LoWPAN network.

2. Route-over Routing
   The route-over routing lies at the network layer of 6LoWPAN stack. It utilizes IP addresses for addressing and locating nodes. This routing can be used to access nodes outside the 6LoWPAN as well as inside. This routing is essentially same as used in regular IP networks.

4.5 Constraints

The IPv6 maximum transmission unit (MTU) is 1280 bytes which is more than the largest possible 802.15.4 packet size. Depending on overhead, the 802.15.4 protocol data unit has different sizes. Since maximum physical layer packet size *aMaxPHYPacketSize* is 127 bytes and maximum frame overhead *aMaxFrameOverhead* is 25 bytes, 102 bytes are available for MAC layer. Moreover depending on application requirements link-layer security may or may not be implemented using AES. However it is highly recommended to employ encryption. In case it is implemented it will impose further overhead of 9, 13 or 21 bytes in case AES-CCM-32, AES-CCM-64 or AES-CCM-128 are used respectively. In case AES-CCM-128 is used, only 81 bytes are available for the MAC payload. This is obviously below the minimum IPv6 requirements and therefore a fragmentation and reassembly adaptation layer must be provided at the layer below IP. The IPv6 header is 40 bytes long, resulting in 41 bytes for the actual payload for upper layers, like UDP. Then the UDP layer has a 8 byte header as well, leaving only 33 bytes for actual application data. This induces a huge overhead for a bandwidth constrained medium. As already mentioned there is a need for the fragmentation and the reassembly, in addition there is also need for header compression. This leads to following observations.

1. The adaptation layer must be provided to comply with IPv6 minimum MTU requirements. Although it is not expected from such devices to produce payloads with such large packets.

2. The header size of IPv6 must be reduced or adapted to minimize the overhead.

The above two issues lead to the requirement of an adaptation layer under layer 3.

4.6 6LoWPAN Adaptation Layer

The adaptation layer lies between layer 2 and layer 3, as depicted in Figure 4.1. This layer encapsulates IPv6 datagrams that need to be transmitted, by a header stack. This header is placed at the beginning of the 802.15.4 MAC protocol data unit (PDU), the 6LoWPAN payload (e.g. an IPv6 packet) follows this header.

All 6LoWPAN encapsulated datagrams transported over IEEE 802.15.4 are prefixed by an encapsulation header stack. Each header in the stack starts with a dispatch value indicating the header type. Zero of more headers follow after the dispatch value. A 6LoWPAN encapsulated IPv6 datagram looks like as follows.

```
+----------+------------+---------+
| Dispatch | IPv6 Header | Payload |
+----------+------------+---------+
```

Since this layer does header compression as well, `LOWPAN_IPHC` compressed IPv6 datagram will be written as following.

```
+-----------+---------------------+------------+
| Dispatch  | LOWPAN_IPHC Header   | Payload    |
+-----------+---------------------+------------+
```

`Dispatch` indicates which packet follows dispatch immediately. The different supported types are summarized in Table 4.1.

Table 4.1: Dispatch Value Bit Pattern

Pattern	Header Type
`00 xxxxxx`	NALP - Not a LoWPAN frame
`00 000011`	LOWPPAN_IPHC - LOWPAN_IPHC compressed IPv6 Provisional Value
`01 000001`	IPv6 - Uncompressed IPv6 Addresses
`01 000010`	LOWPPAN_HC1 - LOWPAN_HC1 compressed IPv6
`01 010000`	LOWPPAN_BC0 - LOWPAN_BC0 broadcast
`01 111111`	ESC - Additional Dispatch byte follows
`10 xxxxxx`	MESH - Mesh Header
`11 000xxx`	FRAG1 - Fragmentation Header (first)
`11 100xxx`	FRAGN - Fragmentation Header (subsequent)

Values not mentioned in this table are reserved for future use.

4.6.1 Header Types

The header stack may contain three optional LoWPAN headers summarized as below.

1. Mesh Addressing header
   Frames using mesh networking include a mesh addressing header. The structure is as following.

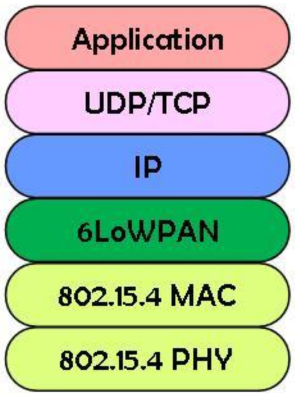


Figure 4.1: 6LoWPAN Stack Overview

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0|V|F|HopsLeft| originator address, final address
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The details of fields are as following.

- V: This value is 0 if the Originator address (very first address) is an IEEE extended address (64-bit) or 1 if it is a 16-bit short address.
- F: This value is 0 if the Final Destination address (very last address) is an IEEE extended address (64-bit) or 1 if it is a 16-bit short address.
- HopsLeft: This 4-bit field will be decremented by each forwarding node before sending this packet to next hop. The packet is not forwarded anymore if this value reaches 0. The value `0xF` is reserved and indicates 8-bit hop limit which will be immediately followed after addresses, hence allowing hops more than just 14.
- Originator Address: This is the link-layer address of the Originator.
- Final Address: This is the link-layer address of the Final Destination within the mesh.

2. Broadcast header
   The broadcast header is used whenever a packet is multicast or broadcast. The structure is as following.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|1|LOWPAN_BC0 |Sequence Number|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Here sequence number is used to differentiate between duplicate packets. This 8-bit field is incremented by the originator whenever it sends a new mesh broadcast or multicast packet.

3. Fragmentation header
If an entire payload fits within single 802.15.4 frame, no fragmentation is required. It is only needed whenever the size of the payload is more than the effective MTU of IEEE 802.15.4. This is done by breaking up the payload into multiple link fragments. Two different types of fragmentation packets are used. One to mark the start of fragmented transmission and others to mark the continuity of the fragmentation with an offset. Link bytes must be a multiples of eight bytes in length. The first header is given below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 0 0 0|    datagram_size    |         datagram_tag          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The remaining headers are contain an offset value.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 0 0|    datagram_size    |         datagram_tag          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|datagram_offset|
+-+-+-+-+-+-+-+-+
```

The fields are described as following.

- `datagram_size`: This 11-bit field encodes the size of an entire IP packet before link-layer fragmentation. This value stays the same for all fragmented packets. Although it is not necessary to contain this value in every packet, doing so will assist in reassembly of the packet on destination in the event that the 2nd or subsequent packet arrives earlier than the first. This will enable buffer space reservation.
- `datagram_tag`: This 16-bit value stays the same for all fragments of a payload. It will be incremented by the sender for every successive fragmented datagrams. This value is wrapped to zero whenever it reaches 65535.
- `datagram_offset`: This field is present only in the second and subsequent fragments. It specifies the offset from the beginning of the payload datagram in unit of 8 bytes. The first packet has an offset of zero.

Upon reception of a link fragment, the receiver starts constructing the original unfragmented packet, whose size is `datagram_size`. It uses `datagram_offset` to determine the position of subsequent fragments within the original unfragmented packet. The size of the reassembly packet is determined by the `datagram_size`.

4.6.2 Header Compression

Along with optional header definitions, the adaptation layer provides stateless header compression which significantly reduces IPv6 header overhead. It is stateless since it does not make binding with any other node and is defined independent of the packet history. Any node operating within the 6LoWPAN can decompress this header.

The header compression changes the packet format, which is indicated by the dispatch value in the encapsulation header preceding the LoWPAN payload. The header compression is done for the IPv6 header and the next layer protocol. The term `LOWPAN_IPHC` is used for the IPv6 encoding and for higher layer protocols the term `LOWPAN_NHC` is used. As of now, `LOWPAN_NHC` is only defined for the UDP header, namely `LOWPAN_UDP`. ICMP and TCP packets are not compressed. The compression details are described as below.

1. `LOWPAN_IPHC` - IPv6 Header Encoding
   `LOWPAN_IPHC` can compress the 40-byte IPv6 header down to 7 bytes (2-byte `LOWPAN_IPHC`, 1-byte Hop Limit, 2-byte Source Address, and 2-byte Destination Address). To do so, following considerations are made.

   - Both Source and Destination Addresses have a link local prefix (FE80::/64) and their last 64 bits can be inferred from link-layer addresses, in case the interface identifiers are autoconfigured.

- The packet length can be inferred from the MAC layer or from `datagram_size`, in case fragmentation is used.
- Next header value is indicated as either ICMP, UDP or TCP.
- `Hop Limit` field is carried uncompressed 1 byte.

`LOWPAN_IPHC` based encoding looks as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    LOWPAN_IPHC encoding      | Non-Compressed fields follow...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

`LOWPAN_IPHC encoding` is a 2-byte value which determines how compression is done and what values are following this header. The structure looks as follows.

```
  0                                       1
  0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| T |VF |NH | HLIM  |     rsv     |  SAM  |  SAC  |  DAM  |  DAC  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

The values are explained in Table A.2.

2. `LOWPAN_UDP` - UDP Header Encoding
   `LOWPAN_NHC` defines the compression scheme for headers following the IPv6 header. In this case, so far compression is only done for UDP headers. `LOWPAN_NHC` is a by that tells which header is next. Bits 0-5 are used to identify the header type and 6-8 are used to set options. The `LOWPAN_UDP` looks as follows.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1 | 1 | 1 | 1 | 1 | 0 | S | D |
+---+---+---+---+---+---+---+---+
```

Table A.3 summarizes the `LOWPAN_UDP` encoding starting from bit 0 to bit 7.

4.7 6LoWPAN Neighbor Discovery

The Neighbor Discovery protocol is a protocol in the Internet Protocol Suite used with IPv6. Nodes use Neighbor Discovery to find other nodes on the link, to determine the link-layer addresses of other on-link nodes, to find available routers and to maintain information on their reachability. If a router or path to a router fails, the node actively searches for a functioning alternatives. The cached values that become invalid are immediately purged.

The Neighbor Discovery protocol for IPv6 defines mechanisms for the following.

- Router Discovery: Nodes can locate routers that reside on an attached link.
- Prefix Discovery: Nodes can discover the set of address prefixes that are used to define their globally unique addresses.
- Address Resolution: Mapping from IP to link layer address. This is not required in 6LoWPAN since there is direct mapping between their globally unique address and the link-layer address.
- Next-hop Determination: Nodes can find next hop routers for a destination.
- Neighbor Unreachability Detection (NUD): Nodes can determine that a neighbor is no longer available on the link.
- Duplicate Address Detection (DAD): Nodes can check whether an address is already in use.
- Address Autoconfiguration: Nodes can automatically configure their address without the use of a stateful configuration protocol, such as Dynamic Host Configuration Protocol for IPv6 (DHCPv6).

Address resolution for neighboring nodes in LoWPAN is not effectively possible. Due to physical mobility of nodes or varying radio strength, a node may often move from one router to another. Moreover characteristics of LoWPAN are different from traditional networks, therefore standard Neighbor Discovery is not suitable and a 6LoWPAN-specific ND is proposed. In this specification, along with standard mechanisms for neighbor discovery, following new entities are introduced.

- Edge Router: An IPv6 router that connects a 6LoWPAN mesh to another IP network.
- Whiteboard: A conceptual data structure which is supported by Edge Routers. The Whiteboard is used for NUD and DAD across the entire 6LoWPAN mesh. The Whiteboard contains information of a node regarding its Owner Interface Identifier, Owner Nonce, IPv6 address, Transaction ID and remaining time of binding.
- Node Registration: A Method in which nodes in LoWPAN register with Edge Routers, therefore creating Whiteboard bindings of their all IPv6 addresses in LoWPAN.

Neighbor Discovery for 6LoWPAN (6LoWPAN-ND) defines additional message formats along with modifying standard ND messages for IPv6. The following message types are used with modifications from [14].

- Router Solicitation: Nodes send this message to request a Router Advertisement immediately.
- Router Advertisement: Routers send this message either periodically or in response to Router Solicitation. Router Advertisements contain various link and Internet parameters as well as prefixes information.
- Neighbor Solicitation: Nodes send this message to determine the link layer address of their neighbors or verify their reachability. Neighbor Solicitations are also used for Duplicate Address Detection. Only used in extended LoWPAN, i.e. an aggregation of multiple 6LoWPANs, interconnected by a backbone link via Edge Routers and forming a single subnet as defined in [15].
- Neighbor Advertisement: Nodes send this message in response to Neighbor Solicitation. They can also send to announce link layer address change. Only used in extended 6LoWPAN. An extended 6LoWPAN is an aggregation of multiple 6LoWPAN interconnected by a backbone link via Edge Routers and farming a single subnet.

The following new message types are defined in Neighbor Discovery for 6LoWPAN.

- Node Registration: A node sends this message along with its IID and requested lifetime to register at an Edge Router. This message can include possible options like the address option, which can include the address the node wants to register.
- Node Confirmation: An Edge Router sends this message to the node in response to its Node Registration. This is used to confirm the binding of the node at the Edge Router.

6LoWPAN-ND messages that are implemented in current assignment are explained in more detail here after.

4.7.1 Node Registration/Confirmation

The Node Registration/Confirmation messages are used for the registration of a node at an Edge Router and the confirmation of the binding. The Node Registration message is sent by a node to the link-local unicast IPv6 address of an on-link Edge Router. When registering for the first time, the sender IP address field contains the unspecified IPv6 address of the sending node. For subsequent NR messages, the IP address field contains the link-local IPv6 address of the sending node. NR/NC messages may be sent over multiple IP hops within a LoWPAN by relaying. When relaying, a new message is created with an updated checksum and a code is used to indicate relaying.

The Node Registration/Confirmation message format is given in Appendix A.4.

Since Node Registration and Node Confirmation are ICMPv6 packets, there is an IP header before them. This IP header contains following values:

- Source Address: The IPv6 address of the source. When registering for the first time, this must be IPv6 unspecified address.
- Destination Address: The link-local unicast IPv6 address of an on-link Edge Router when sent directly by node. The destination IPv6 address of an Edge Router or a well known anycast address 6LOWPAN_ER when relayed by another router. This address will be allocated by IANA. A router is a node that can forward datagrams between arbitrary source-destination pairs by performing IP routing.
- Hop Limit: 255

4.8 Router Solicitation Message

Routers send out their Router Advertisements periodically. However a node can request for an immediate advertisement using Router Solicitation. The message format is given in Appendix A.5.

4.9 Router Advertisement Message

Routers send out their Router Advertisements periodically and upon receiving Router Solicitation. The message format in Appendix A.6.

Along with above mentioned new message types, various new options are defined in 6LoWPAN-ND which are as follows.

4.10 6LoWPAN-ND Message Options

Besides existing message options in [14], new message options were defined in [12]. Which are summarized as below.

- Address Option: This option is used to indicate the address which a node wants to register with the ER in a Node Registration message. Alternatively, this is used in Node Confirmation, sent by the Edge Router, to indicate the success or failure of the binding.
- 6LoWPAN Prefix Information Option: This option carries the prefix information for 6LoWPAN. It also contains the lifetime of the validity of a prefix on a link.

- 6LoWPAN Prefix Summary Option: This option identifies the set of prefix information options by sequence numbers, if there is a change in the sequence number, it denotes that prefix information is likely changed.
- Owner Interface Identifier: This option is for uniquely identifying the binding in question and match it with the Whiteboard entries. This option contains values of Owner Interface Identifier and Owner Nonce.

4.11 6LoWPAN and ZigBee: A comparison

Although both networks operate on IEEE 802.15.4, still we can see some difference.

ZigBee and 6LoWPAN are different in multiple aspects. Firstly the layers are not compatible with each other. 6LoWPAN employs standard IP at the network layer and UDP at the transport layer whereas ZigBee contains its own descriptions for the network and the application frame work layers, which are not included in 6LoWPAN. Application Framework layer is an execution environment for application objects to send and receive data and it is specific to ZigBee devices only.

Secondly, ZigBee requires support from a mains-powered coordinator and routers. In 6LoWPAN network, an Edge Router can also act as pan-coordinator, however it is not necessary that a mains-powered node is a pan-coordinator.

Moreover 6LoWPAN can be connected to other IP based networks using an Edge Router, ZigBee requires a Gateway implementation to do so. Also 6LoWPAN supports a larger network size than ZigBee.

Table 4.2 compares some features of 6LoWPAN and ZigBee stacks[1].

Table 4.2: Stack Comparison as in [1]

	ZigBee	6LoWPAN
Code Size with Mesh	32K to 64K	22K
Code Size w/o Mesh	Not Possible	12K
RAM Requirements	8K	4K
Header overhead	8-16 bytes	2 to 11 bytes
Network Size	65K	2ˆ64
RF Radio Support	802.15.4	802.15.4++
Transport Layer	Within ZigBee	UDP/TCP
Internet Connectivity	ZigBee Gateway	Router/Bridge

Chapter 5

Development Environment

This chapter deals with the development environment, more specifically the operating system, i.e. FreeRTOS and the hardware interfaced during this project.

5.1 FreeRTOS

FreeRTOS is an open source, portable, mini real time kernel. The porting is possible due to several factors. Firstly the code size is small. It is composed of only three core files and an additional port file for specific processor. Secondly most of the code is written in C. Only few lines of assembly code are required to adapt to a specific platform. Finally FreeRTOS is heavily documented on the website [16]. Due to already mentioned factors, it has been already ported to various hardware architectures from 8-bit to 32-bit micro-controllers, including but not limited to, ARM7, ARM9, MSP430, AVR and 8051.

It is licensed under a modified GPL and can be used in commercial applications under this license. The code is freely available on the official website.

The following sections describe FreeRTOS main features utilized in this project.

5.1.1 Scheduler

FreeRTOS features a Round Robin, priority based scheduler. Each task is assigned a priority. The tasks with the same priority share the CPU time in a Round Robin fashion.

Tasks in FreeRTOS can exist in one of the following states.

- Running
  When a task is actually executing, it is said to be in running state. In this state, it is utilizing CPU time.

- Ready
  When a task is able to execute but not currently executing because a different task of equal or higher priority is already in Running State, it is said to be a Ready Task.
- Blocked
  A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event. A task can block waiting for queue and semaphore events. Additionally if we a use delay statement in a task, it will be blocked until the delay period has expired. Tasks in Blocked state always have a 'timeout' period, after which the task will be unblocked. Blocked tasks are not available for scheduling.
- Suspended
  Tasks in Suspended state are also not available for scheduling. Tasks will enter this state only if explicitly commanded to do so through API calls. They will resume via explicit API calls as well. A 'timeout' period can not be specified for Suspended tasks.

Each task is assigned a priority ranging from 0 to the maximal configured priority in FreeRTOS configuration file. Low priority numbers denote low priority tasks, idle task priority is by default defined to be 0.

The scheduler will ensure that a task with higher priority in the Ready or Running queue will always be given processor time in preference to tasks of lower priority that are also in the same queues. In essence the tasks given the processing time will always be the highest priority task available for execution. Only one task can be in the running queue at a given time. If there are no tasks ready to run, FreeRTOS makes a special task running, namely the idle task. The idle task has the lowest priority. This task can be used to perform power management as it is running only when there is no other task available.

The scheduler can be configured as preemptive or collaborative.

5.1.2 Inter-task Communication

FreeRTOS provides various mechanisms for inter-task communication, including message queues and binary semaphores. The Queue mechanism can be used to send messages between tasks, and between interrupts and tasks. A queue is a structured table, used to store and retrieve data, created in the heap. It can be used by two tasks to send data between each other. Tasks and Interrupt Service Routines (ISR) can also communicate through queues. For example a task can send a byte in a queue and this byte can be later accessed by another task. In most cases these queues are used as FIFO (First In First Out) buffers with new data being sent to the back of the queue. Queues contain items with a fixed size. Items are placed into a queue by copy, not by reference.

Binary semaphores in FreeRTOS are used for both mutual exclusion and synchronization purposes. They are actually implemented as a special case of queue, i.e. a queue with one element. On initialization the semaphore queue is full. The first task executing a section of mutual code takes the semaphore. Any other task willing to execute this code will wait until the semaphore is released.

A task trying to receive a byte from an empty queue or willing to send a byte in a full queue will be blocked by the kernel. A task decides the maximum time it allows the kernel to block. If a queue becomes available within that time, the kernel will wake up the and run the task with the highest priority. If the queue stays busy and maximum blocking time elapses, the kernel will get the task ready again and return an error to it.

5.1.3 Memory Management

FreeRTOS provides three memory allocation schemes to allocate memory each time a task, queue or semaphore is created.

- The first scheme is simplest of all schemes, it does not allow memory to be freed once it has been allocated. A huge table in the memory is reserved, namely the heap. Every time the memory allocation system call is used, the pointer for free space is incremented with the allocation size and a pointer is returned to the application. The memory freeing call simply does nothing.
- The second scheme is a more complex memory allocation system. It uses best fit algorithm but it does not combine adjacent free blocks into a single block. It however allows previously allocated blocks to be freed.
- The third scheme is merely a wrapper for standard memory allocation and free functions. This scheme is not deterministic as well and may increase the kernel code size.

In our application, the first memory management scheme was chosen. This was done to avoid the hassle of a garbage collection algorithm and to keep things simple. Moreover, our application does not require memory freeing.

5.2 SAND Node

SAND (Small Autonomous Network Devices) is a low power and small size reconfigurable hardware platform with a CoolFlux DSP [10]. CoolFlux is a 24-bit ultra low-low power, embedded DSP core, originally developed for audio applications with low power requirements.

Programs for CoolFlux are compiled using CHESS Retargetable Compiler by Target Compiler Technologies.

5.3 BSN Node

BSN (Body Sensor Network) Node is a hardware platform with an MSP430 microcontroller with a CC2420 radio chip [8]. It provides a low-powered, miniaturized, stackable platform for the design and development of pervasive health care applications and sensor networks. It may optionally contain physiological sensors such as ECG, EEG, Sp02.

The version of BSN nodes in use are v2 and v3. Like predecessor these nodes are based on an MSP430 microcontroller variant. However this new microcontroller has more resources. It has 10KB RAM and 48KB Flash Memory. It has two built-in 16-bit timers, two universal serial synchronous/asynchronous communication interfaces (USART) and 48 I/O pins.

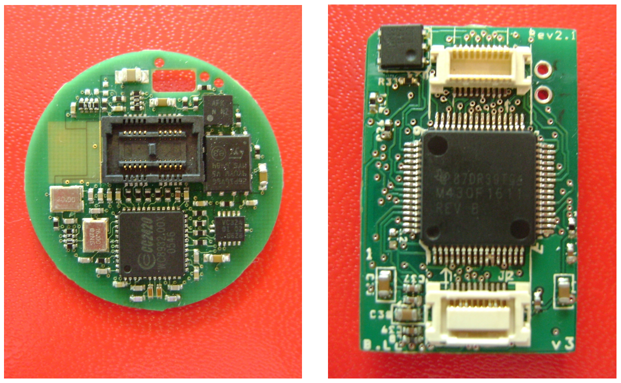


Figure 5.1: SAND Node and BSN Node

5.4 MSPGCC

The source code for the BSN node is compiled using the MSPGCC compiler. MSPGCC is a port of the GNU C and assembly language tool chain to the Texas Instruments MSP430 family of low-power microcontrollers. The MSPGCC port of the GNU C compiler used in our project is based on version 3.2.3 of GNU GCC. A FreeRTOS port for MSPGCC was used in our project. This version of MSPGCC supports optimization for code size reduction. It supports all the current variants of the MSP430 processor, and comes out with a full set of header files for the processors. Signed and unsigned integers of 8, 16, 32 and 64 bit length are supported. Currently only the C language is supported. Within Windows NT/XP environment, MSPGCC was run inside a Cygwin shell. Cygwin is a Linux-like environment for Windows. It consists of two parts:

- A DLL (cygwin1.dll) which acts as a Linux API emulation layer providing substantial Linux API functionality.
- A collection of tools which provide a Linux look and feel

Complete development in this assignment was done using MSPGCC compiler within the Cygwin environment.

5.5 Debugging and Visualization

The debugging and visualization was done using two different techniques, which are as following:

- These nodes can be communicated from PC via Hyperterminal. This can be done using one USART of the MSP430 microcontroller. Data in the form of text strings can be sent to these nodes and vice versa. Hence for debugging purpose a variety of text strings were used which were sent by the node to the user PC and later printed on the Hyperterminal. A unique code is assigned to every error and these codes can be sent in case an error is produced. This debug functionality induces an overhead on the code size but it can be removed from the final application.
- A packet sniffer was used to visualize packets transmitted using the nodes. Chipcon CC2420DK contains this functionality. When a packet is transmitted over a preset radio channel, the sniffer is tuned to this channel and it can visualize the packet and its details. Therefore using this packet sniffer and transmitting packets when needed, visualization of errors was made possible.

Chapter 6

Implementation

Philips Lighting is investigating the possibilities and the impacts of the IP based networking in the lighting networks. Wireless networks have their limitations regarding packet size, power and MTU. Therefore an adaptation was required to implement IP functionality in wireless networks. This was done in the form of a 6LoWPAN stack.

Within Philips Research, a 6LoWPAN stack was developed for the SAND platform[10]. These nodes have more RAM and code memory than the BSN nodes[8]. The stack supports DYMO-LOW [17] routing protocol, which is a mesh-under routing protocol. This means it utilizes link layer addresses for routing. It was developed for FreeRTOS environment and Chipcon MAC. The first part of this assignment dealt with the porting of the code from SAND platform to BSN.

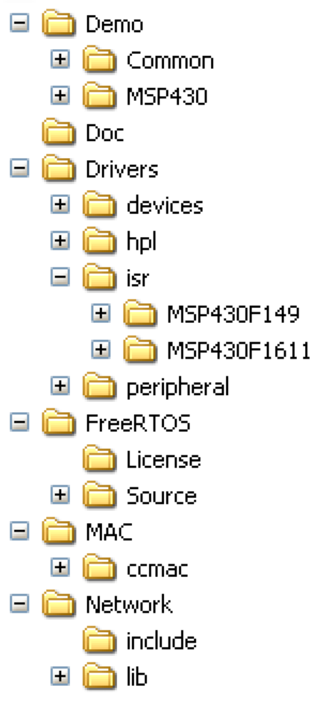
To successfully do the porting of the code to the target platform, first the knowledge of existing architecture was required.

6.1 Pre-porting Analysis

Before porting the code to the BSN node, first an analysis and thorough understanding of the existing code was carried out. The code consists of the following main parts. The directory structure of the code is given in Figure 6.1.

6.1.1 The 6LoWPAN Stack

This section contains the core functionality of 6LoWPAN. It contains all the header and source files required for packet compression, fragmentation and routing according to DYMO-LOW [17]. The stack will be discussed in more detail in section 6.2.



```
Demo
    Common
    MSP430
Doc
Drivers
    devices
    hpl
    isr
        MSP430F149
        MSP430F1611
    peripheral
FreeRTOS
    License
    Source
MAC
    ccmac
Network
    include
    lib
```

Figure 6.1: Directory Structure of the Code

6.1.2 FreeRTOS

This module holds the FreeRTOS header and source files. This module consists of common files used by all ports and port specific files.

6.1.3 Drivers

The drivers section contains the source files for the Timer, SPI and UART drivers.

6.1.4 MAC

The MAC section contains the header files and source files by Chipcon, necessary for MAC communication. It also contains the Hardware Abstraction Layer (HAL). The application and the operating system do not need to know about the hardware. The functionalities of hardware are provided by standard functions through the HAL. HAL is a platform specific layer. The structure of MAC is given in Figure 6.2.

After familiarizing with the code structure, it is necessary to understand the different functionalities that code has to offer.

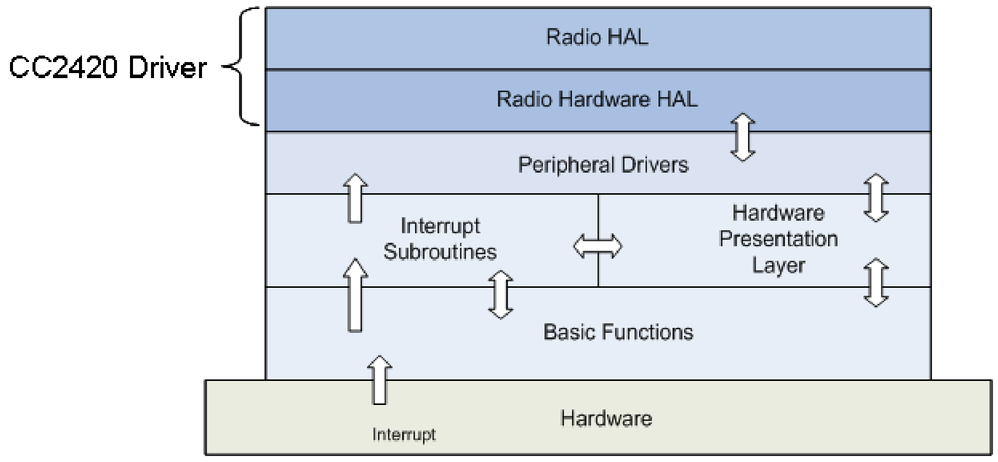


Figure 6.2: Driver Abstraction of CC2420

6.2 The 6LoWPAN Stack

From Figure 6.3 we can identify the main components of the 6LoWPAN stack. Since it is a layered architecture, each layer's functionality is contained in a different file.

Following are the details of different files available in the stack and their functionalities. Files are named according to Figure 6.3.

6.2.1 sixLowPanLayer

This file is named after the 6LoWPAN adaptation layer itself. It contains the core of header compression and adaptation. It contains the functions to perform stateless address auto-configuration. When a node boots up, it calls the function macToIpv6Translation() and passes its MAC address as argument. The function returns a structure containing the link-local IPv6 address with prefix FE80::/64. The remaining 64 bits are created by the PAN identifier and the short MAC address as specified in [13]

On packet transmission this file adds the 6LoWPAN dispatch, `LOWPAN_IPHC` and `LOWPAN_NHC`. Then it sends the packet to DYMO-LOW for routing the packet. On packet reception, depending on the first byte, either this file handles the packet or it is processed by routingUnderIp and routingTable using DYMO-LOW routing protocol. If sixLowPanLayer handles the packet, it processes mesh, fragmentation and `LOWPAN_IPHC`. A full IPv6 datagram is created by this layer and then sent to the IP layer.

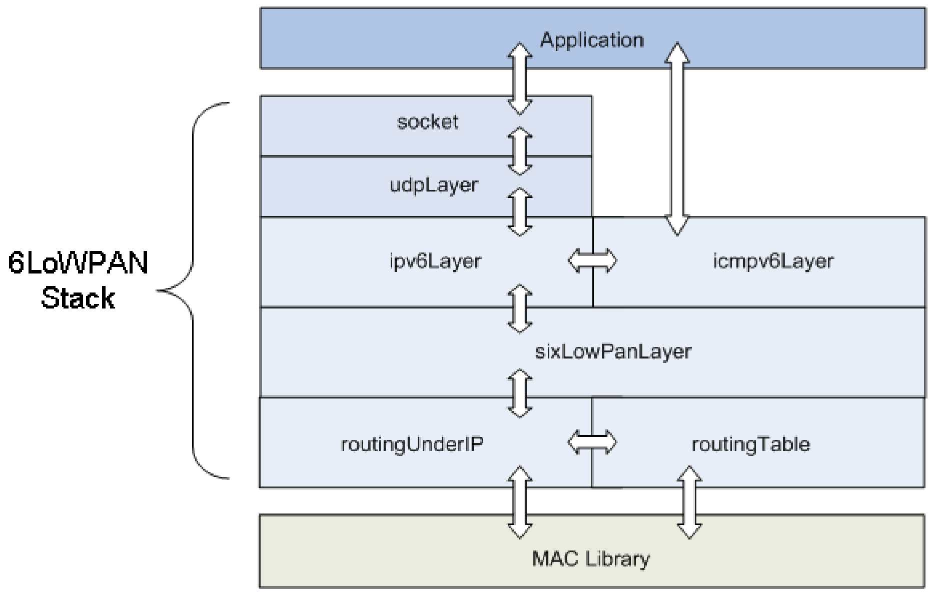


Figure 6.3: The 6LoWPAN Stack

6.2.2 ipv6Layer

This file contains the functions to process the IPv6 header. No functionalities were yet needed at that stage so this file did not contain any extra functionalities.

6.2.3 udpLayer

This file contains the functions to handle the UDP transmission and reception. Upon transmission it compresses the ports according to a specific format [11]. Upon reception it decodes the port values and if they are similar, the payload will be delivered to the right socket where an application will fetch it.

6.2.4 socket

This file contains basic socket functions like *create* and *bind*. With bind applications can bind sockets to their IP address and application port number. This file also contains the code to initiate transmission and reception from the udpLayer.

6.2.5 buffer

This file contains the functionality of creating a buffer, assigning buffers, writing data to them and reading from them. These buffers are used by the socket interface as well as 6LoWPAN adaptation layer to copy incoming packets from the MAC layer or the application layer and process them throughout the stack. It uses the functionality provided by the underlying FreeRTOS kernel for dynamic memory allocation.

6.2.6 routingUnderIp

This file contains the functionality of the packet transmission to the next hop according to the router decisions, i.e. neighbor, not neighbor or even unknown route. If the packet size is greater than the MTU, it breaks them down into fragments and transmits in successive link fragments.

6.2.7 routingTable

This file contains all the functions that are required to maintain the routing table. It contains the core functionality of DYMO-LOW [17]. It handles Route Request and Route Reply messages, updates routing table and checks the addresses in table to initiate Route Discovery.

6.3 Investigation Methodology

After familiarizing with the code structure, the next step was to find out the nature of the problems that might arise. The list of these problems can be broken up into two categories.

1. Structural Issues
   These are the issues that will be produced due to incompatibility of code and the new environment. The code will not effectively compile in this case.

2. Functional Issues
   These are the issues that will be produced due to break up of code. The code will compile but not work on the new platform, i.e. BSN nodes, as effectively as it is supposed to be.

Therefore to continue the investigation, the first aim was to compile the code with MSPGCC without errors. After giving various errors and modifying code structure, code successfully compiled. At this stage, when it was possible to compile with new compiler, next aim was to test the functionality of code against standard benchmark, i.e. a demo application that is shipped with FreeRTOS.

During this phase various minor and major errors were found and handled. More detail of such issues is given in Appendix D.

6.4 6LoWPAN Stack Extension

After the porting of code, the next step was to improve the compliance of the stack with the draft [11]. The original code had limited functionality of the 6LoWPAN adaptation layer. It could not use

128-bit IPv6 addresses, i.e. there was no activity on the IPv6 layer. In this phase of the project, the functionality of the IPv6 layer and the ICMPv6 layer on top of the adaptation layer was improved.

6.4.1 6LoWPAN Adaptation Layer

In the original implementation, the `LOWPAN_IPHC` header always contained the compressed addressing scheme. This limited the transmission of packets to within 6LoWPAN since all 128-bits of the source address and all 128-bits of the destination address were elided. In order to transmit messages outside 6LoWPAN, through the Edge Router, complete address support was required. During this assignment, this support was added. Now in case the destination is outside 6LoWPAN then the full 128-bit address is carried inline. In case if the destination is within 6LoWPAN, IPv6 addresses are elided. Short link layer addresses are used for routing in this case.

The procedure is as follows:

- When a UDP packet is created by the application layer, the destination address is passed as an argument in a function call made to the udpLayer via socket. The udpLayer creates a HC_UDP2 header and after compressing ports and writing into a buffer, it passes the pointer to the buffer to the ipv6Layer. The ipv6Layer in this scheme does not do anything but instead it passes the pointer to the sixLowPanLayer which then checks the destination address. If the destination address is within 6LoWPAN, 128-bit addresses are not used. If the destination address is outside 6LoWPAN, it changes the header format for this packet (`LOWPAN_IPHC`). The IPv6 address of the destination is added to the packet and the value is indicated in the header. Then sixLowPanLayer forwards this packet to the routingUnderIp, the mesh destination address is the short address of the Edge Router.

- Upon reception of a packet, first it is checked whether it is a routing packet or a 6LoWPAN data packet. This is done by comparing the first byte, the dispatch byte with a set of values. If it is 6LoWPAN packet, the pointer is given to sixLowPanLayer. First it is checked that what is value of `LOWPAN_IPHC`. If it denotes that a 128-bit host address is carried, it is extracted from the packet and an IPv6 datagram is created from that packet. If a short address is carried, then the IPv6 address is created from the short address. An IPv6 datagram is constituted from these autoconfigured addresses and this datagram is passed to higher layers.

An example of a UDP packet over 6LoWPAN is depicted in Appendix B.2. The above procedure is also depicted in Figure C.1.

6.4.2 ICMPv6 Layer

In the original implementation, ICMPv6 only contained support for ECHO REQUEST and ECHO REPLY [18]. Neighbor Discovery in IPv6 is done via ICMPv6 messages. Therefore these messages were added and Neighbor Discovery support was added. The procedure of ICMPv6 message parsing is as follows:

- When an ICMPv6 packet is created by icmpv6Layer, it is sent to the ipv6Layer. Creation of ICMPv6 packet is done on a trigger from the application layer, a timer expiration or on start up, depending on the nature of the packet being created. Along with existing support for ECHO REQUEST and ECHO REPLY, support for the following ICMPv6 message types is provided.

  1. Router Solicitation
  2. Router Advertisement
  3. Node Registration
  4. Node Confirmation

  The above mentioned messages are used for Neighbor Discovery. Also support is provided for parsing options which are as follows:

  1. Address Option
  2. Owner Interface Identifier Option
  3. Prefix Summary Option

- When a packet is received at sixLowPanLayer, it is checked whether it is an ICMPv6 packet or a UDP packet. If it is an ICMPv6 packet, it is sent to icmpv6Layer. This layer then processes the packet, starting from the type of message. Upon finding out the type of message, respective actions are preformed on this packet, ranging from the indication to the application layer to silently dropping the packet.

An example of an ICMPv6 packet is given in Appendix B.3.

6.5 UDP Chat Application

The functionality of 6LoWPAN stack was checked with a UDP chat application. This application requires minimum of two nodes, namely client and server. A user sends a message from client node and the message is echoed back from the server application. The resulting message is shown on the screen. The communication with the client node is done via Hyperterminal.

6.5.1 Operation

Initially two nodes (client and forwarder) are powered. The server node is not powered up.
The client node, running TestSixLowPanClient application is connected via UART to a PC. The

application starts with a fixed client and server address. After initialization, it waits for user input via Hyperterminal. When user enters a character string, application copies it into a buffer. When user enters a carriage return (Enter key), the buffer is forwarded to the udpLayer through socket interface for further processing and transmission to the server. The udpLayer adds compressed ports and LOWPAN_UDP header and forwards the packet to the ipv6Layer, which in this case does not add ip addresses as short link layer addresses are used. The packet is then passed to the sixLowPanLayer, which add the LOWPAN_IPHC header and dispatch. It sends the packet to the routingUnderIp layer, which checks if the destination is known or not. In first transmission the destination is not known and through routingTable a Route Request is generated. The packet is meanwhile saved in a structure. After receiving Route Response and therefore a valid route to the destination, the client node transmits the packet. Route Request is done over broadcast, while Route Reply is via unicast.

When server node, running TestSixLowPanServer application receives the packet, it first checks if it is a valid 6LoWPAN packet or not. If it is a 6LoWPAN packet, it is forwarded to sixLowPanLayer. In this layer first dispatch byte is checked. After verifying that the packet is destined to this specific node, an IP datagram is reconstructed. Based on the header values, this layer sets the values of the IP datagram. Afterwards, the buffer is forwarded to ipv6Layer. Which in this particular application, forwards to udpLayer. Compressed ports are extracted from the packet and the payload is copied to the application via the socket interface.

This application can be tested on single hop or multiple hop. For multi-hop chat application, another node is required which runs TestSixLowPanClient. This node receives the packets and just forwards to the destined location. This forwarding is done below sixLowPanLayer.

The procedure can be seen in Appendix C.

6.6 WASP Demonstration

In this demonstration, an application based on the Event-Condition Action (ECA) programming model was to be deployed on the BSN nodes with 6LoWPAN support. These nodes must be able to communicate to the Edge Router. From Edge Router they can communicate to the Server over IPv6 link.

The network consists of an Edge Router, implemented on a PC and equipped with a USB 802.15.4 dongle and several BSN nodes. The devices form a mesh network based at 802.15.4 MAC layer. Within this mesh, the routing is done using DYMO-LOW.

On the BSN node side, the stack contains UDP and ICMP functionality. This stack allows following tasks.

- Router Discovery using Neighbor Discovery Protocol
- Prefix assignment and global address assignment using Neighbor Discovery Protocol

- Ping application using ICMPv6
- UDP communication within 6LoWPAN using short link layer addresses. When addressing outside 6LoWPAN, the stack uses full IPv6 address for the destination.
- Mesh-under routing using DYMO-LOW.

This stack does not support fragmentation for this particular demonstration. The fragmentation support was removed to reduce code size, after assessing that it will not be utilized in this demonstration, since the message size will never exceed MTU. The network and software architecture of this demonstration is shown in Figure 6.4. On the PC side, 6LoWPAN stack is integrated into the

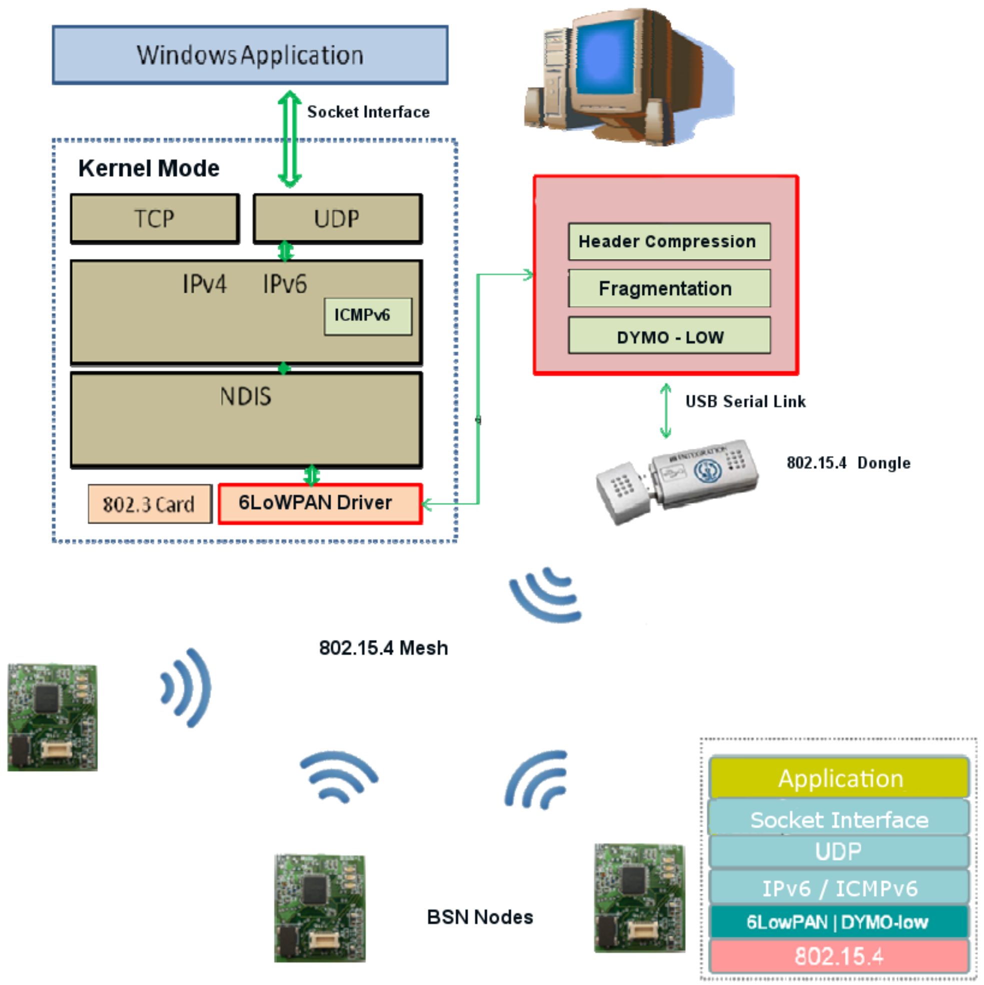


Figure 6.4: Network and Software Architecture of 6LoWPAN

Windows OS through a kernel module implementing a driver that acts as Virtual Network Interface

Card. The communication to the mesh network from the PC is done through an 802.15.4 dongle connected to a USB port.

When a new BSN node boots within mesh, it either waits for a Router Advertisement or sends a Router Solicitation message to an Edge Router. On reception of Router Solicitation, the Edge Router replies with the Router Advertisement, with the prefix option, as explained in Section 4.9. The node receives the Prefix in Router Advertisement, configures it global address and sends Node Registration message, with the address option containing the globally unique address for the node. The Edge Router replies with a Node Confirmation message. After Node Confirmation, the BSN node can communicate with external IP networks using its unique address, through Edge Router. This procedure is shown in Figure 6.5.

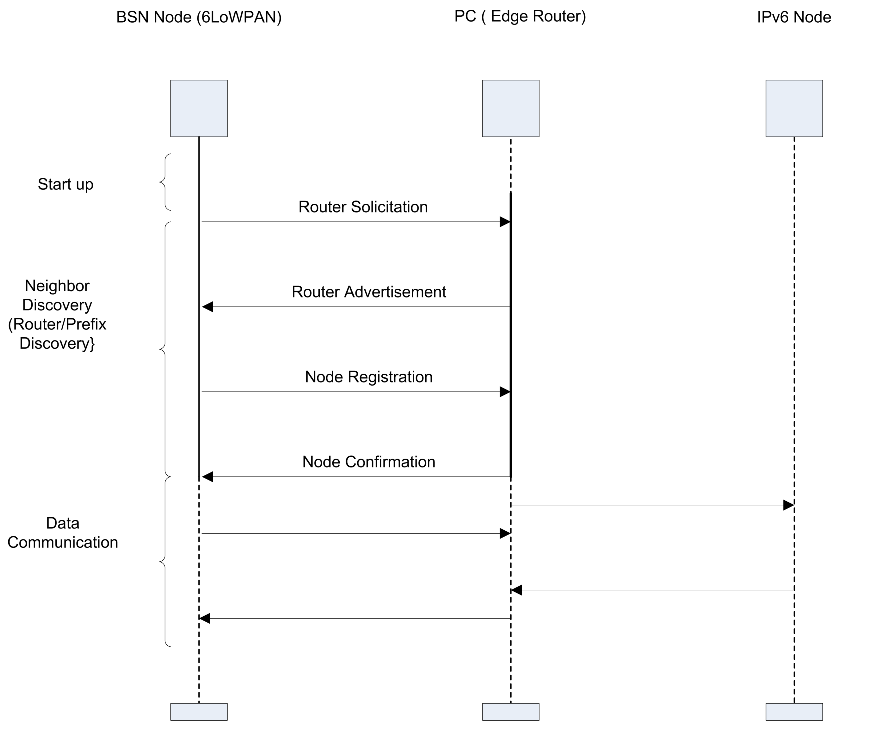


Figure 6.5: BSN Node and PC Interaction

This ECA based application required support for UDP data transfer and Neighbor Discovery. However the size of this application was 13K and there was not enough space available on these nodes. To overcome this problem, the functionality of these nodes was reduced to the requirements of the specific application.

6.6.1 Code Size Reduction

Table 6.1 depicts the code memory utilization of various aspects of the code. If the complete MAC is compiled along with the complete 6LoWPAN stack, there will not be any space left for the application on top of it. Therefore it is a big challenge to provide 6LoWPAN stack support for an application with required functionalities. To do so, a number of design decisions were taken which are as follows.

- A node can have a pre-programmed short address and PAN ID. Therefore PAN coordinator support is not required. Moreover support for PAN association and disassociation is also not required. Therefore this code can be stripped from the MAC code.
- The network is beaconless, so all the MAC code that supports beacons is stripped off as well.
- The size of packets being communicated is almost never more than MTU. Hence support for fragmentation is not required in this implementation. The related code was removed to save almost 1500 bytes.
- Debug support is not required for nodes in the field. Therefore all related debug code is also not included in the build. Almost 900 bytes are saved in this process.

Due to the above modifications, the code size reduction to 35KB was possible.

Table 6.1: Memory Utilization on BSN Node

Code	Size (Bytes)
FreeRTOS, Peripheral Drivers, MAC and Idle task	36740
FreeRTOS, Peripheral Drivers, MAC and MAC test task	36744
FreeRTOS, Peripheral Drivers and Idle Task	13792
FreeRTOS and Idle Task	4330
MAC and HAL (Original Size, with Beacon and PAN association)	22900
6LoWPAN Stack (Original Size, with Fragmentation and debug)	10000
MAC and HAL (without Beacon and PAN association)	16000
6LoWPAN Stack (without Fragmentation)	7300
FreeRTOS, Peripheral Drivers, MAC and 6LoWPAN (Final Version)	35358

Chapter 7

Self-powered Switch and Sensors

A migration from a wired communication system to a wireless communication introduces new and different components. Wireless battery-less self-powered switches and sensors are among these. Such a switch or sensor does not require batteries or any kind of external power. It operates on the technology that converts the pressing action of the switch or sensor into a small amount of energy. This energy can then be used by a radio transceiver to transmit a signal to a destination, carrying commands and data. A piezoelectric conversion mechanism is used in this type of switches or sensor, that when applied with force, produce voltage across the material. The battery free switches can be used to control lights in home and any other on/off devices, including HVAC, security systems, fans and any automation system that require pressing a switch. The wireless nature of these switches allow freedom of mobility and flexibility.

Existing wired building automation systems do not share common physical medium or network protocols. They are vertically isolated, meaning they do not have anything common on any layer. Co-existence of such systems creates complex building networks, which are difficult to install, have expensive cabling layout and are hard to maintain. Introducing a common physical medium and a common network protocol will make the co-existence of such networks easily possible. Just sharing a physical medium will reduce cabling costs and ease installation and maintenance.

Wireless building automation systems offer advantages over their wired counter parts. Firstly these networks are retrofit, that means addition of new components to the existing infrastructure is possible and much easier than wired networks. Secondly they reduce the cabling costs. In wired networks, cabling is an expensive factor. In wireless networks one time commissioning is required. In wired network, new devices require cabling as well, whereas in wireless, just placing new the components within the existing system's range will make connectivity possible. Also wired building automation systems are influenced by the positioning of the cables. Devices can only be placed where cables are laid. In wireless building automation systems, sensors can be placed with more freedom.

IP as a common network protocol offers very promising role in the interoperability of the exiting building automation systems. Seeing the merits of wireless building automation systems, Philips Research is investigating the possibilities regarding IP over wireless networks for building automation systems. This leads to the investigation of 6LoWPAN, i.e. IPv6 over Low Power Wireless Personal Area Networks. Self powered switches are investigated for their role in lighting networks. This assignment led to the investigation of 6LoWPAN for lighting networks employing battery-less switches.

7.1 Requirements

The requirements from a battery-less switch based setup in a lighting network are as follows.

- Luminaries should be able to join multicast groups when they want.
- Switch may be used to turn luminaries on and off.
- Luminaries can be accessed from outside the lighting network through Edge Router.
- Switch and luminaries may have unique IP addresses.

7.2 Procedure

In a 6LoWPAN enabled lighting network, to trigger the luminaries with a battery-less switch requires the switch to be connected to a node. The node must have a radio transceiver and the 6LoWPAN stack. The luminaries should also have the 6LoWPAN stack running on their controllers. For a complete 6LoWPAN based lighting network, presence of an Edge Router is necessary. This Edge Router will provide a prefix to the nodes within the 6LoWPAN mesh for globally unique ID.

When a switch is pressed, it will turn on the attached node for a short duration. In this time, the node configures its link-local address, sends Router Solicitation to an Edge Router and upon receiving Router Advertisement, it registers its globally unique IPv6 address at Edge Router Whiteboard via Node Registration message. The Edge Router replies with the Node Confirmation message.

The luminaries should be able to join a well-known link-local multicast address. The Edge Router should be able to identify the listeners to this multicast group and keep their record. This can be done with MLDv1 [19] and MLDv2 [20] messages. Whenever an Edge Router sends a multicast listener query, luminaries interested in joining the group respond with the multicast listener report.

After configuration of globally unique IPv6 address, the switch can send messages to the luminary multicast group via Edge Router. The message sent to the specific multicast group should be forwarded by the Edge Router to all the luminaries that are subscribed to that address. Luminaries can join or leave a multicast group at any time. The procedure is depicted in Figure 7.1.

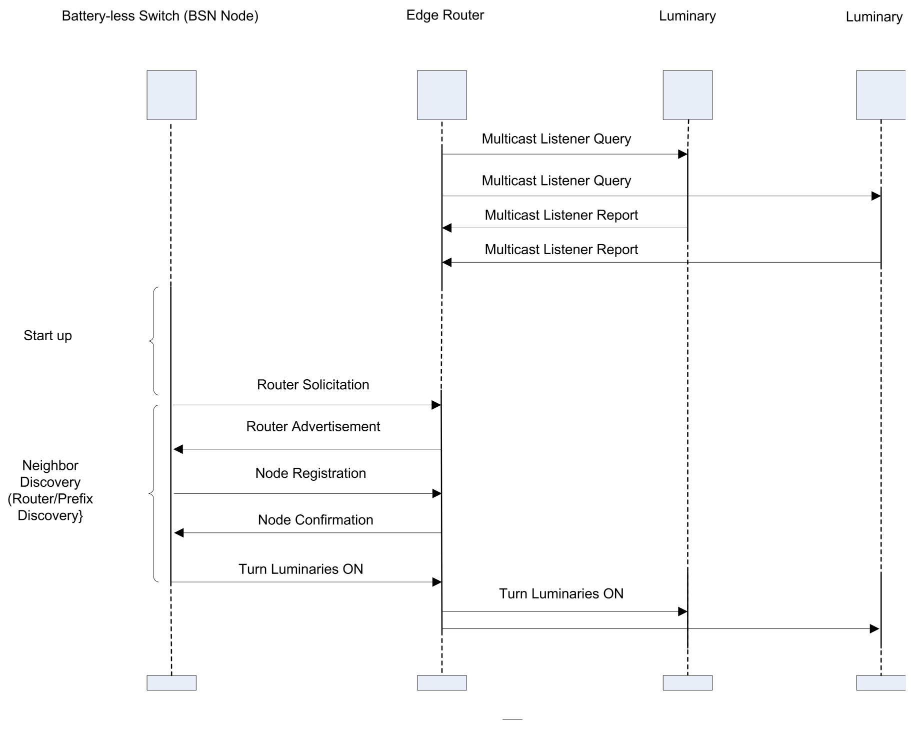


Figure 7.1: Battery-less Switch Based Lighting Network

The above setup is explained for a proposed case when there are no timing constraints due to the limited energy produced by switch. In such case, the switch can save the acquired values to flash memory and turn off. When starting up next time, it will read values from the flash memory and skip the registration process. This is however not possible yet. As we will see there are limitations due to little amount of energy produced by switch.

While within a subnet, where neighborhood discovery is not required, above scenario can be replaced by one given below.

- The switch is pressed.
- The node turns on and sends a Route Request broadcast message for a well known destination.
- Using Route Reply, it finds the route to the luminaries.
- The node then, using the discovered route, transmits the control signal (turn on/off) to the luminary.
- The luminary node acknowledges the reception of control signal.
- In case the acknowledgment is not received, the switch node does multiple retries to transmit that control signal as long as the node stays on.

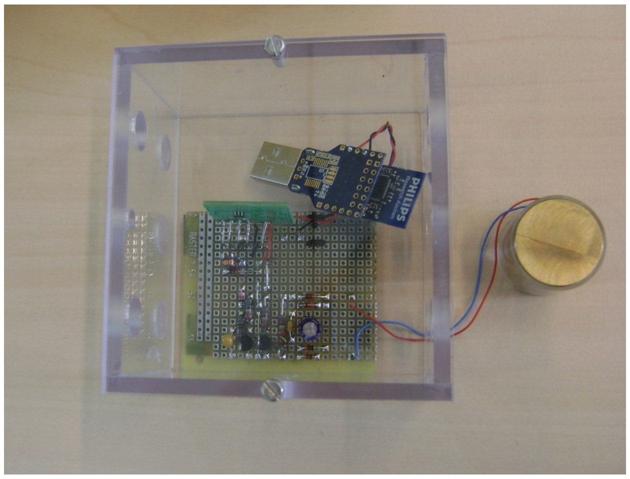


Figure 7.2: Battery-less Switch Circuitry

The luminaries know their existing states. Whenever they receive a message, they toggle their state. e.g. if a luminary is ON and it receives a control signal from the switch, it will turn OFF and vice versa. The above setup is used in this assignment, i.e. Edge Router is not used in lighting network. Therefore Neighbor Discovery is not implemented on switch based nodes due to the energy and timing constraints, which will be discussed later in this chapter.

The battery-less switch is shown in Figure 7.2, where it is connected with AquisGrain 2 module. This switch was also connected with MSP430 nodes for separated testing. An oscilloscope was connected to the ports available on MSP430 and AquisGrain nodes for timing measurements. At various points of interest, these ports were given high or low output. This output was measured using oscilloscope and later analyzed.

7.3 Problems

The main issue with such implementation is that such type of switches provide very little energy. This energy is in the range of millijoules. Therefore a series of packet transmission which is spread over long time is not feasible. In our case the switch was able to provide 30mA @ 3.3V for more or less 60 ms, depending upon how far it was pressed. Which results in total of 5.94 mJ energy.

Booting the BSN node running FreeRTOS and using its radio requires around 45ms. When a node boots up, it does standard reset procedure, which is responsible for following tasks.

- Providing a default vector table
- Providing default interrupt handlers
- Initializing the watchdog timer

- Initializing the .data segment
- Initializing the .bss segment
- Jump to main

A jump is used in the last phase, instead of a call to main, to save space on the stack. Since main is not supposed to return.

Every MSP430 device has its interrupt vectors table located at `0xffe0`. This table looks like as follows.

```
Disassembly of section .vectors:
0000ffe0 <InterruptVectors>:
    ffe0: 3a 40        interrupt service routine at 0x403a
    ffe2: 3a 40        interrupt service routine at 0x403a
    ffe4: 9c 64        interrupt service routine at 0x649c
    ffe6: 42 64        interrupt service routine at 0x6442
    ffe8: e4 bc        interrupt service routine at 0xbce4
         .
         .
    fff8: ba bd        interrupt service routine at 0xbdba
    fffa: 12 bd        interrupt service routine at 0xbd12
    fffc: 3a 40        interrupt service routine at 0x403a
    fffe: 00 40        interrupt service routine at 0x4000
```

The _reset_vector__ is located at address `0x4000`. Whenever execution begins, the PC is loaded with this address.

```
Disassembly of section .text:

00004000 <_reset_vector__>:
    4000: b2 40 80 5a  mov #23168,&0x0120 ;#0x5a80
    4004: 20 01
    .
    .
    4020: 3f 40 7a 11  mov #4474,r15     ;#0x117a
    4024: 3d 40 fe 32  mov #13054,r13     ;#0x32fe
    4028: 0d 9f        cmp r15,r13     ;
    402a: 05 24        jz $+12          ;abs 0x4036
    402c: cf 43 00 00  mov.b #0,0(r15) ;r3 As==00
    4030: 1f 53        inc r15         ;
```

```
        4032: 0f 9d       cmp r13,r15  ;
        4034: fb 2b       jnc $-8          ;abs 0x402c
        4036: 30 40 40 40 br #0x4040     ;

    00004040 <main>:
```

As a first step of the startup, the watchdog timer is initialized. Then the initialized global variables are copied to RAM. After this the uninitialized global variables are cleared, i.e. put 0 as their value. In the above excerpt from the application with an uninitialized data size of `8580` bytes, we can see that all these bytes are initialized with 0.

Firstly the start and the end of the data are stored into two different registers, namely `r15` and `r13` respectively. Then on every address contained in register `r15`, a 0 is copied. Afterwards this address is incremented and compared with the maximum address. As long as current address does not increase than the maximum, this loop continues. When this address increases beyond maximum address, a flag is set and loop terminates. After this a jump to `main` is performed. In above case `main` is located at `0x4040`.

From above we can see that program startup depends heavily on the size of the initialized and the uninitialized data. In our application, the heap size is set to 6000 bytes. This heap is used for dynamic memory allocation for queues, tasks and semaphores. Due to this big size of heap, i.e. uninitialized data, the startup time is increased - since during this phase, this memory is initialized with zeros. A reduction in heap size leads to reduced startup time but it leads to abnormal application behavior.

After startup, Route Discovery takes a considerable amount of time (up to 100-200 ms) depending on the response by different nodes in the network. This does not give us time to do useful transmission of control packets. Therefore it was decided that for the demonstration of such a network, any node capable of broadcasting a control packet when switch is pressed will do the job until the switch energy output and BSN node boot-up time is reduced. The AquisGrain2 was used for above purpose.

This AquisGrain node is not running any RTOS. Instead it runs an application that transmits packets in a specific format, satisfying requirements for UDP transmission over 6LoWPAN. The MAC is provided by Chipcon. The recipient on the other hand, is a BSN node with a functional 6LoWPAN stack, FreeRTOS and Chipcon MAC. It will receive the packet and parse it accordingly. In this case, it will trigger the lights whenever it receives a packet, based on its MAC address. The switch is assigned to a specific luminaries, i.e. the address and PAN ID are fixed. In this demonstration, an LED was soldered to the MSP node, therefore denoting the reception of the packet.

7.4 Acknowledgment Requirements

Originally it was considered that a luminary upon reception of a packet, transmits an acknowledgment to the switch. This can be an application level or a MAC level acknowledgment. An application level acknowledgment can be single hop or multi-hop. A MAC level acknowledgment is only single hop. The acknowledgment is used to indicate successful transmission of packet. Following cases arise when we consider the acknowledgment scenario.

- As of now, there is no way a switch can indicate the successful transmission of a packet to the user. If it is successful, the luminary will turn on. If it is unsuccessful, the luminary will stay off.
- If the transmitting node connected to the switch does not receive the acknowledgment, all it needs to do is to retransmit same packet for a preset number of times.
- The waiting for acknowledgment takes a considerable amount of time, depending on factors like number of hops and whether the MAC layer acknowledgment is sufficient or the application layer acknowledgment is required. Following observations were made in this regard.
  - MAC acknowledgment takes on average 2.7 ms in our set up.
  - An application level acknowledgment takes on average 30 ms for one hop.
- During this waiting time, the switch is consuming precious energy, generated for a short time and in a small quantity.

Considering above, we'll have a look at two scenarios for the transmission of a packet.

- Successful Transmission - the luminary will turn on or off, no further evidence is required by the user.
- Unsuccessful Transmission - the luminary state will remain the same. In this case the switch will wait for the acknowledgment. In case there is no acknowledgment, the switch will retry to send same packet.

It was decided that instead of waiting for acknowledgment, the sending node should send multiple copies of same packet as many times as possible during the time it is powered on. This will ensure the best possible results for transmission of one single packet. The receiver drops copies of same message received within a fixed period of time.

7.5 Timing Results

Due to the timing constraints of the switch, both platforms i.e. MSP430 and AquisGrain were benchmarked for two different applications. The timing measurements were taken using oscilloscope. The results are discussed separately for both platforms.

7.5.1 MSP430 Timing Results

The UDP chat application timing was measured on MSP node. The timing results are depicted in Figure 7.3.

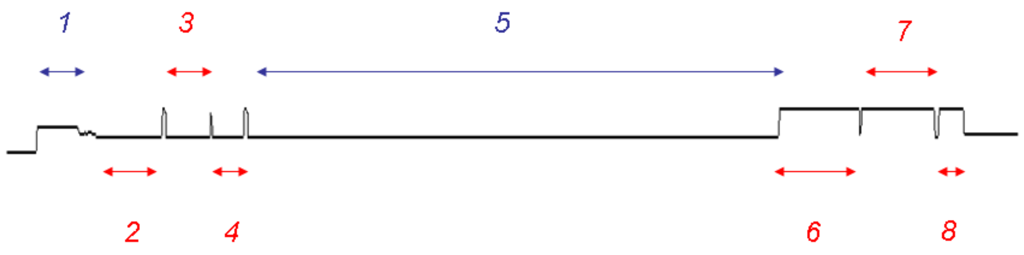


Figure 7.3: MSP430 Timing Results

The details are as follows.

1. FreeRTOS startup and led = 85 ms
2. Route Request and Route Reply = 172 ms
3. Sending Packet = 102 ms
4. Packet Reception wait = 70 ms
5. Delay = 1000 ms
6. Route Request and Route Reply = 162 ms
7. Sending Packet = 138 ms
8. Packet Reception wait = 50 ms

7.5.2 AquisGrain2 Timing Results

The simple application running on AquisGrain does not take too much time to start up. Also the processing does not take much time as well. Detailed timing results for the AquisGrain application for first packet generation are given in Figure 7.4.

1. Button press to Led on = 1900 us
2. MAC_Init() = 174 us
3. Radio_Power_Up() = 1355 us

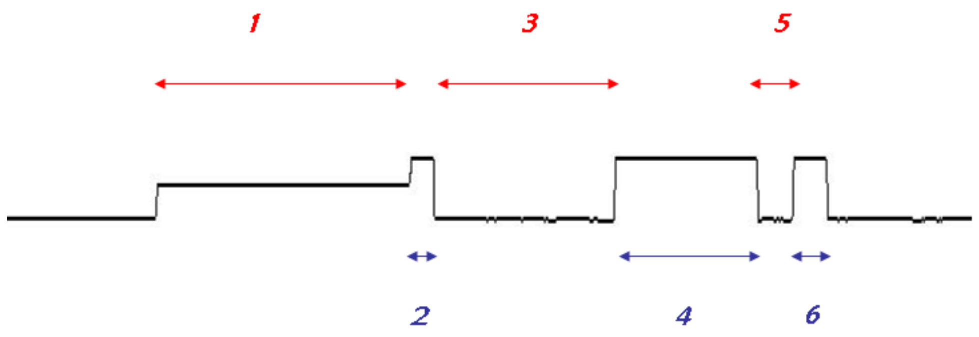


Figure 7.4: AquisGrain 2 Timing Results

4. Reset MAC layer = 1075 us
5. Start as PAN coordinator (set PIB attributes) = 267 us
6. Send one packet = 253.6 us

The total time it takes from pressing the switch till first packet transmission is 5024.6 us.

7.5.3 Timing Analysis

The results from both nodes cannot be compared directly since both applications are different in nature. Still it is evident that AquisGrain startup time and initialization time is quite promising. Absence of FreeRTOS has an impact on these results. In MSP430 the timing results were not exactly same on both occasions. The sending time of packet varied in both recorded tries. This difference can be caused due to unavailability of channel. There is no stack used on AquisGrain. It runs a simple C application on top of Chipcon MAC. The addresses and PAN ID are preset. These factors contribute to the fact that within 5 ms we can transfer a packet to the luminary. In this case switch does not wait for response from the luminaries. In case the response is expected from application layer, it can take up to 50 ms. A link layer acknowledgment on the other hand, only takes 20 ms.

7.6 Conclusion

Due to the limited amount of current produced during the pressing action, it is difficult to operate a fully functional node running an operating system with this energy. The start up time should be minimized to utilize the energy harvested for actual transmissions. This can be achieved by using tailored application for transmission of packets. The application on AquisGrain does not process IPv6 packets. It can receive packets but it is not supposed to perform any action. The switch is not given any processing tasks yet. In practical cases such switch will be utilized for only one way communication to the luminaries. The messages sent from the switch have fixed destination addresses, i.e. luminaries addresses are known in application. Luminaries are running full 6LoWPAN stack, so that they can communicate with switches as well as nodes on external IPv6 network, using Edge Router. So it is possible to monitor and control the luminaries from out side the LoWPAN, since each luminary, when starts up for the first time, configures a link-local and a global address as well. UDP messages sent from the switch are static, the luminaries toggle on every new message. It is the responsibility of the luminaries to save the last state, ignore repetitive messages sent by same switch and toggle only on new messages. The acknowledgments at the MAC level should be sufficient for link level recovery of messages. In this scenario a fully functional 6LoWPAN stack that supports ICMPv6 and the reception of UDP messages is not required. Moreover fragmentation is also not utilized in this simple application. The MAC will not be required to perform power saving as it will be only turned on when the switch is pressed, for a very short duration. Therefore above application sufficed for IPv6 based addressing over wireless. Mesh-under routing suffices for such example as a switch will only be able to access its link-local neighbors. If accessed from outside, the Edge Router can translate from router-over routing to mesh-under routing.

In a real-world example, a switch should not have a fixed destination address since this will limit the design options. The switch should find its neighbors using Neighbor Discovery Protocol via Neighbor Solicitation and Neighbor Advertisement messages. These neighbors should be grouped into multicast addresses based on the services they offer. The switch should address one of these multicast addresses whenever pressed. Another possibility is that a fixed coordinator node is installed in such system. This node keeps track of all the nodes in neighborhood, when a switch node turns on, it detects this node and exchange the addresses of luminaries. These addresses can be used to communicate with one of many luminaries. In order to make the switch address configurable using Neighbor Discovery, the platform needs to be optimized for the harvested energy on pressing action. The coordinator keeps track of all the switches in neighborhood as well as the luminaries.

Chapter 8

Conclusion

IP over Low Power Wireless Personal Area Networks is made possible through 6LoWPAN. It offers new possibilities for networking, especially in Building Automation Systems. As first and second objective of this thesis, the 6LoWPAN stack was ported to MSP430 based BSN nodes. Moreover, additions were made into the existing stack according to the proposed standard draft [11].

The 6LoWPAN stack supports UDP and ICMPv6 messaging. Two different applications for UDP and ICMPv6 were tested for proper functionality. Code size was reduced by stripping off functionalities surplus to our requirements. Changes in the existing stack were done to improve performance in terms of primary memory and code memory usage.

As a third objective Neighbor Discovery was implemented on the nodes as part of ICMPv6 messages. This implementation was done for simple LoWPAN; therefore Neighbor Solicitation and Neighbor Advertisement messages were not included. In essence this implementation does router discovery and prefix assignment. Although it has support for message parsing and options analysis, such functionality is not required in our current configuration as of now.

As a further step, a battery-less switch based demonstration was carried out for lighting networks which operate on IP addresses. Different challenges were faced during this phase, which are explained in chapter 7 and how they were tackled. Finally, support was given for WASP demonstration by providing IPv6 stack integration into WASP network and testing required functionalities.

Although 6LoWPAN has introduced a way to improve flexibility and scalability in communication, it also imposes various restrictions and has pit falls. Points raised during this project were memory usage and code size. As already mentioned in chapter 6, the code size was reduced at the cost of some MAC functionalities. If we have a fully functional MAC, then it is difficult to implement an application utilizing 6LoWPAN stack even on a resource rich MSP430F1611 microcontroller based BSN node. Also memory utilization is another bottleneck. Typical nodes do not have memory specifications as high as nodes in our use. Therefore if we need a 6LoWPAN based network we need nodes with at least 10K RAM. Moreover the standard is still evolving so there will be more additions to the existing stack. Because of that even more resources will be required.

In the end, this will be decided by the network architect whether to use 6LoWPAN connectivity or not. If it is required, more expensive nodes will be used.

Appendix A

Header and Frame Formats

A.1 IPv6 Header

The header format is as below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The details of these fields are given in Table A.1.

Table A.1: IPv6 Header

Field	Meaning
Version	4-bit field is set to be 6
Traffic Class	8-bit field - To identify between different classes and priorities
Flow Label	20-bit field - To identify between normal and special QoS handling of packets
Payload Length	16-bit field - Length of IPv6 payload
Next Header	8-bit field - Identifies type of next header
Hop Limit	8-bit field - Decrements every time a packet is forwarded
Source Address	128-bit address - Originator IPv6 address
Destination Address	128-bit address - Destination IPv6 address

A.2 LOWPAN_IPHC Details

Table A.2: LOWPAN_IPHC Encoding

Subfield	Definition	*BitNo.*	*Values*	Meaning
T	Traffic Class	0	0	Full 8-bits carried in-line
			1	Elided, implicitly 0
VF	Version and Flow Label	1	0	Full 4 bits for Version and 20 bits for Flow Label carried in-line
			1	Elided, Version is 6 and Flow Label is 0
NH	Next Header	2	0	Full 8 bits for next header carried in-line
			1	Next Header compressed using LOWPAN_NHC
HLIM	Hop Limit	3-4	00	All 8-bits of Hop Limit carried in-line
			01	All 8-bits elided, Hop Limit assumed to be 1
			10	All 8-bits elided, Hop Limit assumed to be 64
			11	All 8-bits elided, Hop Limit assumed to be 255
rsv	Reserved	5-7		Reserved for Future use
SAM	Source Address Mode	8-9	00	All 128 bits carried inline
			01	64-bit compressed IPv6 address
			10	16-bit compressed IPv6 address
			11	All 128 bits are elided
SAC	Source Address Context	10-11	00	Source Address is a link-local address
			other values	Reserved
DAM	Destination Address Mode	12-13	00	All 128 bits carried inline
			01	64-bit compressed IPv6 address
			10	16-bit compressed IPv6 address
			11	All 128 bits are elided
DAC	Destination Address Context	14-15	00	Destination Address is a link-local address
			other values	Reserved

A.3 LOWPAN_UDP Details

Table A.3: LOWPAN_UDP Encoding

Subfield	*BitNo.*	*Values*	Meaning
S Source Port	0	0	Not compressed, all 16 bits carried inline
		1	Compressed to 4 bits. The actual source port is calculated as `0xF0B0` + `short_port_value` 4 bit `short_port_value` is carried inline
D Destination Port	2	0	Not compressed, all 16 bits carried inline
		1	Compressed to 4 bits. The actual destination port is calculated as `0xF0B0` + `short_port_value` 4 bit `short_port_value` is carried inline

A.4 Node Registration/Confirmation

The message format is as below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             TID               |     Status    |P|   Reserved  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Lifetime                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                   Owner Interface Identifier                  +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Owner Nonce                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Registration option(s)...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The ICMPv6 packet fields are as follows.

- Type: Type is not defined in the draft yet. However provisional values are set within this project. 200 for NR and 201 for NC.
- Code: Code is 0 if it is a direct message from a node to an Edge Router. It will be set to 1 when the message is relayed. Other values indicate error types.
- Checksum: The ICMP checksum is included in this field.
- TID: A unique Transaction ID for a Node Registration. When a node boots up, TID is set to 0 and increments with every NR message. When it reaches `0xFFFF`, it is reset to 16. Values from 0-15 are only used after first boot.
- Status: Values to be decided, 0 means success.
- P: 1 bit flag, set to denote that the router is the primary router of node.
- Reserved: Unused field, must be initialized with 0 and ignored at receiver's end.
- Lifetime: A 32-bit integer, denoting the amount of time in seconds remaining before the binding expires.

- Owner Interface Identifier: A globally unique identifier for requesting node's interface.
- Owner Nonce: An Owner Nonce is a 32-bit number randomly generated by the node upon booting and generated again each time the node re-boots. This number is used by Edge Routers to detect duplicate Owner Interface Identifiers. It stays same as long as node stays on, and is used to identify rouge or counterfeiting nodes.
- Options: Possible options include Address Option, for each address the node wants to bind.

A.5 Router Solicitation

The message format is as below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Reserved                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+
```

The ICMP fields for this message are as follows.

- Type: Type is set to 133.
- Code: Code is 0.
- Checksum: The ICMP checksum.
- Reserved: Unused field, must be initialized with 0 and ignored at the receiver's end.
- Options: Owner Interface Identifier Option must be included in a Router Solicitation message.

A.6 Router Advertisement

The message format is as below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Cur Hop Limit |M|O|H|Prf|Rsrvd|       Router Lifetime         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Reachable Time                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Retrans Timer                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

The ICMP fields for this message are as follows.

- Type: Type is set to 134.
- Code: Code is 0.
- Checksum: The ICMP checksum.
- M: 1-bit ”Managed address configuration” flag, when set it mentions the availability of the Edge Router. In standard ND, this flag mentions that DHCPv6 is used [14].
- O: 1-bit ”Other configuration” flag, when set it mentions other configuration information is available via DHCPv6.
- H: 1-bit ”Home Agent” flag, when set it indicates that the Edge Router sending this advertisement is also acting as Mobile IPv6 home agent on this link [21].
- Preference: 2-bit signed integer indicating whether to prefer this router over others or not. It should be set to 01 for Edge Routers. More details in [22].
- Reserved: Unused field, must be initialized with 0 and ignored at the receiver’s end.
- Router Lifetime: 16-bit unsigned integer denoting the lifetime associated with default router in unit of seconds.

- Reachable Time: 32-bit unsigned integer denoting the availability of neighbor in milliseconds after having received a reachability confirmation. Used in Neighbor Unreachability Detection algorithm [14].
- Retrans Timer: 32-bit unsigned integer denoting the time in milliseconds between retransmitted neighbor solicitation messages. Zero means unspecified.
- Options: Options may include source link-layer address, .

Appendix B

Packet Details

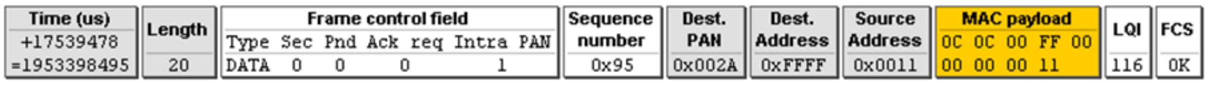
B.1 Route Request/Reply Packet

A Route Request packet looks as in Figure B.1.

The details of the packet field are as following.

- `0x0c` TYPE : set as WASP Route Request, value `0x10` is set is Route Reply
- `0x0c` TTL
- `0x00` Route Cost
- `0xFF` Flags
- `0x00` RREQ_ID
- `0x0000` Destination address for whom Route is requested)
- `0x0011` Source address

Route Request is always sent on broadcast address, i.e. `0xFFFF`.

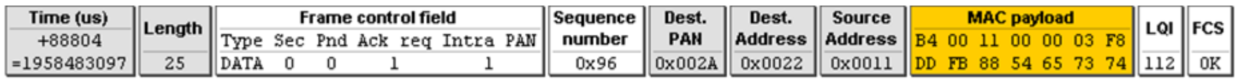


Time (us)	Length	Frame control field					Sequence number	Dest. PAN	Dest. Address	Source Address	MAC payload	LQI	FCS
+17539478		Type	Sec	Pnd	Ack req	Intra PAN					0C 0C 00 FF 00		
=1953398495	20	DATA	0	0	0	1	0x95	0x002A	0xFFFF	0x0011	00 00 00 11	116	OK

Figure B.1: Route Request Message

B.2 UDP Packet

A UDP packet carrying a string as payload looks as in Figure B.2.



Time (us)	Length	Frame control field					Sequence number	Dest. PAN	Dest. Address	Source Address	MAC payload	LQI	FCS
+88804		Type	Sec	Pnd	Ack req	Intra PAN					B4 00 11 00 00 03 F8		
=1958483097	25	DATA	0	0	1	1	0x96	0x002A	0x0022	0x0011	DD FB 88 54 65 73 74	112	OK

Figure B.2: UDP Packet with short addresses

The details of the packet fields are as following.

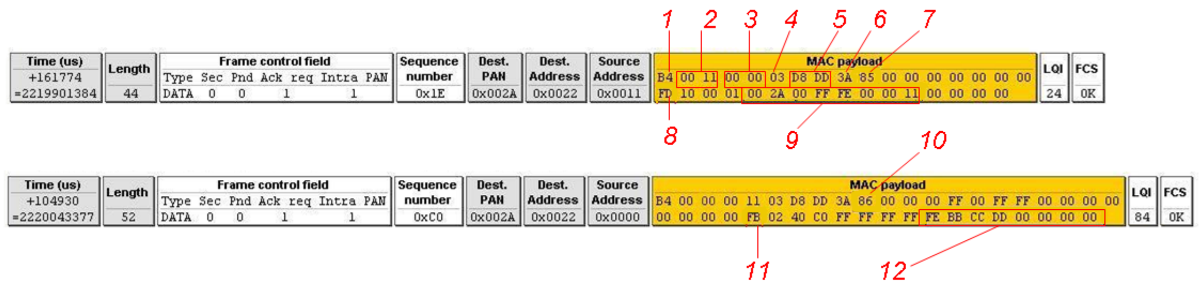
- `0xB4` Mesh header
- `0x0011` - Originator short address
- `0x0000` Final destination short address
- `0x03` IPHC Dispatch
- `0xF8DD` IPHC with next header UDP
- `0xFB88` UDP header with ports carried inline
- `54657374` Text string, "Test"

The values of IPHC and UDP header denote following.

- IPHC Header - `0xF8DD` - Traffic Class is 0, version is IPv6, Flow Label is 0, Next Header is compressed using LOWPAN_NHC, Hop Limit is fixed to 255. Source address and destination addresses are not carried.
- UDP Header - `0xFB88` - `B` denotes both source and destination UDP ports are compressed in one nibble each. Original value of ports can be calculated by P + `short_port`, where P is 61616 (0xF0B0). `Short_port` is carried inline for both source and destination, i.e. `88` denotes, source `short_port` is 8 and destination `short_port` is 8 as well.

B.3 Router Solicitation and Router Advertisement

Router Solicitation sent from a node to an Edge Router and Router Advertisement message sent from an Edge Router to a node look as in Figure B.3.



```
Time (us)    Length  Frame control field                  Sequence  Dest.   Dest.    Source   MAC payload                                           LQI  FCS
+161774              Type Sec Pnd Ack req Intra PAN      number    PAN     Address  Address  B4 00 11 00 00 03 D8 DD 3A 85 00 00 00 00 00 00 00
=2219901384  44      DATA  0   0      1         1        0x1E      0x002A  0x0022   0x0011   FD 10 00 01 00 2A 00 FF FE 00 00 11 00 00 00 00    24   OK

Time (us)    Length  Frame control field                  Sequence  Dest.   Dest.    Source   MAC payload                                                 LQI  FCS
+104930              Type Sec Pnd Ack req Intra PAN      number    PAN     Address  Address  B4 00 00 00 11 03 D8 DD 3A 86 00 00 00 FF 00 FF FF 00 00 00 00
=2220043377  52      DATA  0   0      1         1        0xC0      0x002A  0x0022   0x0000   00 00 00 00 FB 02 40 C0 FF FF FF FF FE BB CC DD 00 00 00 00  84   OK
```

Figure B.3: Router Solicitation and Router Advertisement

The details of the packet fields are as following.

1. `0xB4` - Mesh header
2. `0x0011` - Source short address
3. `0x0000` - Destination short address
4. `0x03` - Dispatch
5. `0xD8DD` - IPHC with ICMP uncompressed next header field
6. `0x3A` - ICMP header
7. `0x85` - Router Solicitation
8. `0xFD` - Owner ID Option
9. `0x002A00FFFE000011` - Owner Address 64 bit comprise of PAN ID and 16-bit short address
10. `0x86` - Router Advertisement
11. `0xFB` - Address Option
12. `0xFEBBCCDD00000000` - 64-bit Prefix

Appendix C

UDP Application

C.1 Client

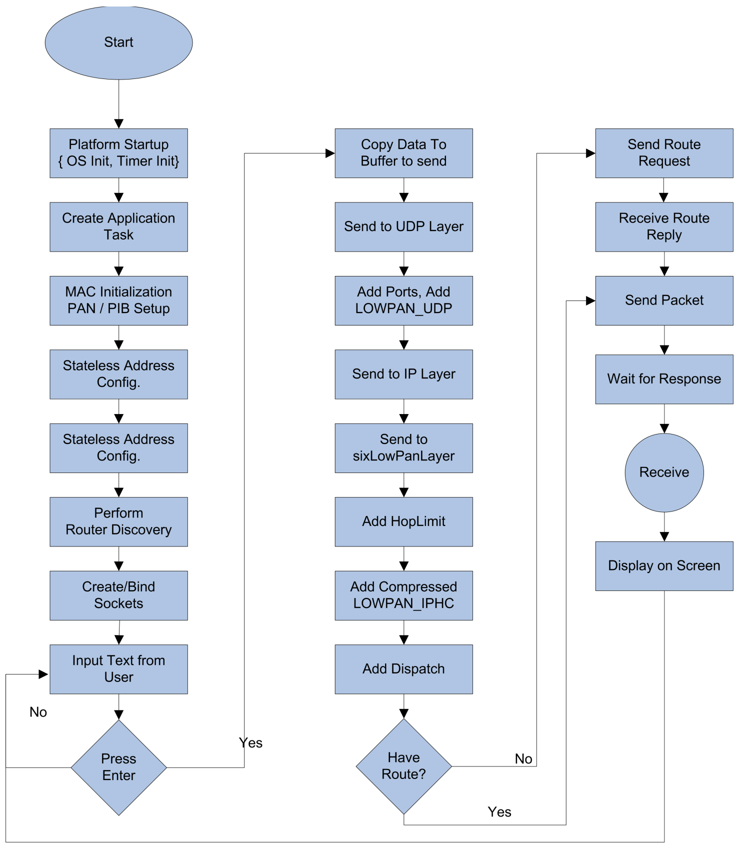


Figure C.1: UDP Client Block Diagram

C.2 Server

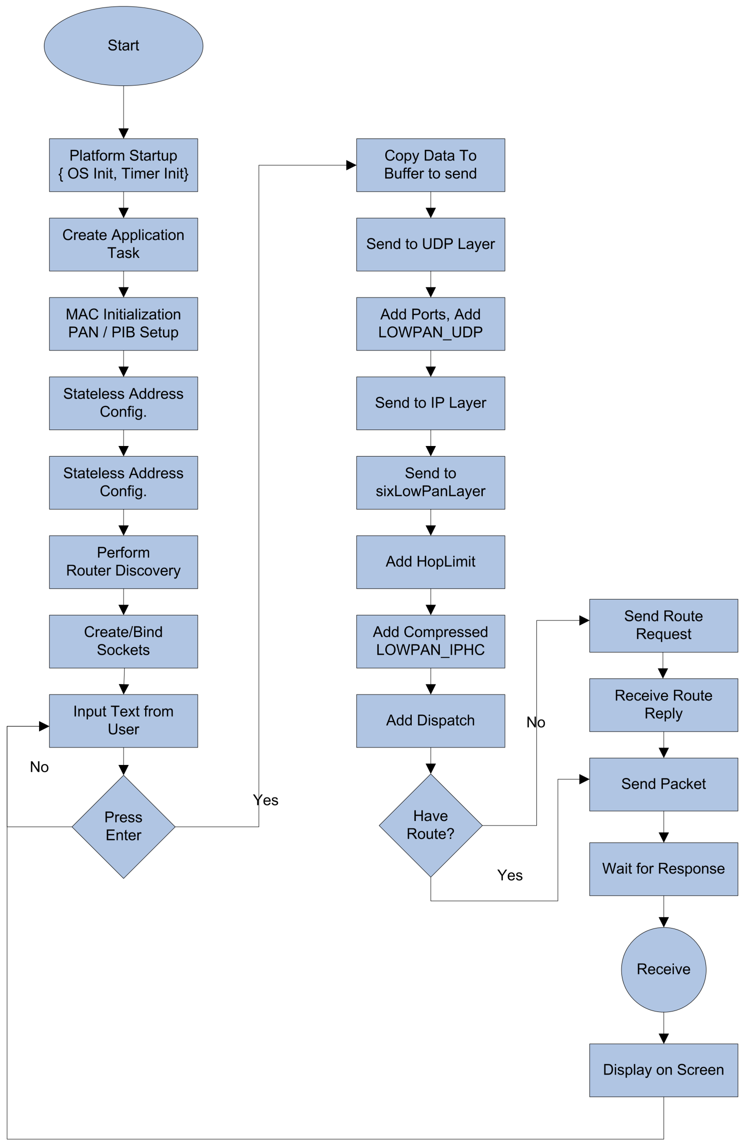


Figure C.2: UDP Server Block Diagram

Appendix D

Error Log

The following are the issues that were faced in the initial phase of porting. Each issue includes a description of problem, its severity, complexity, files involved and solution. Severity denotes the impact of that bug on program execution, being expressed as low, medium, high or critical. Complexity relates with the time consumed to find the bug and solve it, and not just the solution. It is expressed as low, medium or high.

1. **Problem:** *struct in6_addr* declared inside parameter list.

   Description: Original code had irregular use of typedef.

   Severity: Medium

   Complexity: Low

   Files: sixLowPanLayer.c ipv6Layer.c udpLayer.c

   Solution: structure redefined, typedef used at appropriate places to avoid same errors.

2. **Problem:** Request for member in something not a union or structure.

   Description: If pointers are accessed like structures, the C compiler gives this error. Previous code was inconsistent with usage of -¿ and . while calling members of pointer or structure.

   Severity: Medium

   Complexity: Low

   Files: Various

   Solution: Access of pointers and structure elements was made consistent throughout all code.

3. **Problem:** *mactoIpv6Translation()* return value inconsistent.

Description: This function returns a pointer of structure *in6_addr* and not a structure it self. The access was inconsistent through different files.

Severity: Low

Complexity: Low

Files: sixLowPanLayer.c sixLowPanLayer.h Test6LowPanLayerClient.c Test6LowPanLayerServer.c Test6LowPanLayerForwarder.c

Solution: Now function returns a structure instead of pointer. To make it consistent with code. It can also be modified to return pointer though.

4. **Problem:** Variable declaration inconsistent with C99.

   Description: Variables are not allowed to be declared inside case statement.

   Severity: Low

   Complexity: Low

   Files: routingUnderIp.c icmpv6.c

   Solution: Variable declaration was moved to start of function.

5. **Problem:** UART0 inaccessible.

   Description: In port code of FreeRTOS, the UART0 description was not available. The baud rate setting were disabled.

   Severity: Medium

   Complexity: Medium

   Files: All files requiring debugging

   Solution: UART1 was used instead.

6. **Problem:** Variable declaration inconsistent with C99.

   Description: Variables are not allowed to be declared inside loops.

   Severity: Low

   Complexity: Low

   Files: Various

   Solution: Declarations moved earlier in respective functions.

7. **Problem:** Stack size for tasks was not sufficient.

   Description: Minimal stack size for tasks was specified, which was insufficient for task operation.

   Severity: High

   Complexity: High

   Files: Mac.c Test6LowPanLayerClient.c Test6LowPanLayerServer.c Test6LowPanLayerForwarder.c

   Solution: Stack size was increased to next optimal size.

8. **Problem:** Problem with buffer memory allocation.

   Description: When a buffer is declared, using malloc memory should be allocated. However it was not done, resulting in corrupt data at every access.

   Severity: High

   Complexity: Medium

   Files: Buffer.c

   Solution: Memory access was done using malloc where necessary.

9. **Problem:** IPv6ToMacTranslation return type inconsistent.

   Description: Some pointer values were assigned to struct types.

   Severity: Medium

   Complexity: Low

   Files: Various

   Solution: Return type changed. Code modified accordingly.

10. **Problem:**In buffer.c, function writeInBuffer() does not update the value of the pointer to current location after writing new characters into the buffer.

    Description: The buffer pointer should always point to the head of the buffer. It is initialized with the last address of the declared buffer. To explain further, buffer writing procedure is as follows.

    - Declare a buffer of amount MAX_BUFFER_SIZE.
    - The pointer of buffer is pointing at the last address.
    - Whenever there is a value added in buffer, pointer is decremented.

- This pointer in the end will point to the head of message.

This function had a bug that after writing the string, the pointer again pointed to the original value. Therefore at next call, it would overwrites existing data.

Severity: Critical

Complexity: High

Files: buffer.c

Solution: Function was modified as per requirements.

11. **Problem:** Memory allocation for *struct sockeraddr_in6 ** was wrong.

    Description: The memory allocated was not *sizeofstruct sock_addr_in6*.

    Severity: High

    Complexity: Medium

    Files: Test6LowPanLayerClient.c Test6LowPanLayerServer.c Test6LowPanLayerForwarder.c

    Solution: Code modified accordingly. Memory allocated of correct size.

12. **Problem:** MAC Acknowledgment not transmitting.

    Description: Some conditional statements used variables from part of code commented to reduce size. Therefore these conditional terms were never evaluated to *true*.

    Severity: High

    Complexity: High

    Files: mac_rx_engine.c

    Solution: Code modified accordingly so that these conditions become true when ACK is pending.

Bibliography

[1] Geoff Mulligan. The 6lowpan architecture. In *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, pages 78–82, New York, NY, USA, 2007. ACM.

[2] Wikipedia. http://en.wikipedia.org/wiki/dmx512-a, July 2009.

[3] DALI-AG. http://www.dali-ag.org/, July 2009.

[4] Continental Automated Buildings Association. http://www.caba.org/standards-protocols.

[5] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), September 2007.

[6] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks.* John Wiley & Sons, 2005.

[7] Patrick Kinney. Zigbee technology: Wireless control that simply works, October 2003.

[8] Imperial College London. http://vip.doc.ic.ac.uk/bsn/index.php?article=926, September 2006.

[9] WASP. http://www.wasp-project.org/, July 2009.

[10] Martin Ouwerkerk, Frank J. Pasveer, and Nur Engin. Sand: a modular application development platform for miniature wireless sensors. In *BSN*, pages 166–170, 2006.

[11] J. Hui and D. Culler. Compression format for ipv6 datagrams in 6lowpan networks. draft-hui-6lowpan-hc-01, 2008.

[12] Ed. Z. Shelby, P. Thubert, J. Hui, and S. Chakrabarti. 6lowpan neighbor discovery. draft-ietf-6lowpan-nd-04, 2009.

[13] M. Crawford. Transmission of IPv6 Packets over Ethernet Networks. RFC 2464 (Proposed Standard), December 1998.

[14] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007.

[15] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational), August 2007.

[16] Richard Barry and FreeRTOS Team. http://www.freertos.org/, January 2006.

[17] Ed. K. Kim, G. Montenegro, Ed. S. Park, I. Chakeres, and C. Perkins. Dynamic manet on-demand for 6lowpan (dymo-low) routing. draft-montenegro-6lowpan-dymo-low-routing-03, 2007.

[18] A. Conta and S. Deering. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 2463 (Draft Standard), December 1998. Obsoleted by RFC 4443.

[19] S. Deering, W. Fenner, and B. Haberman. Multicast Listener Discovery (MLD) for IPv6. RFC 2710 (Proposed Standard), October 1999.

[20] R. Vida and L. Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810 (Proposed Standard), June 2004.

[21] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775 (Proposed Standard), June 2004.

[22] R. Draves and D. Thaler. Default Router Preferences and More-Specific Routes. RFC 4191 (Proposed Standard), November 2005.