A
PROJECT REPORT
ON

Generic ZigBee Cluster Library

AT
Volansys Technology PVT. LTD.

In the partial fulfillment of the
Requirements for the award of the degree

of
BACHELOR OF TECHNOLOGY
in
ELECTRONICS & COMMUNICATION
Submitted by

Rishabh Sheth
Roll No EC-96, ID 13ECUOG055
B.Tech. SEM. VIII

Under the guidance of

Faculty Supervisor
Dr. Vinay Thumar
Faculty-In-Charge

External
Mr. Kewal Agola
Embedded Engineer

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
FACULTY OF TECHNOLOGY,
DHARMSINH DESAI UNIVERSITY, NADIAD-387001.

APRIL 2017

CERTIFICATE

This is to certify that Mr. Rishabh Sanjiv Sheth (13ECUOG055) Roll number EC-96 studying in B.Tech. Sem. VIII (Electronics & Communication) of Faculty of Technology, D. D. University has satisfactorily delivered seminars and has satisfactorily completed term work in the subject of Industrial Training / Project.

Date: 6th April, 2017

Dr. Vinay Thumar
(Faculty Supervisor)

Dr. Nikhil Kothari
(Head of Department)

ACKNOWLEDGMENT

It has been great honour and privilege to undergo training at Volansys Tech Pvt. Ltd, Ahmedabad.

I am very much thankful to Mr. Kewal Agola for providing all facilities and support to meet my project requirements. I would like to take opportunity to express my humble gratitude to Mr. Kewal Agola, under whom I executed this project. Their constant guidance and willingness to share their vast knowledge made me understand this project and its manifestations in great depths and helped me to complete the assigned tasks.

I am extremely thankful and pay my gratitude to my faculty Dr. Vinay Thumar for his valuable guidance and support on completion of this project in its presently. I extend my gratitude to Dharamsinh Desai University for giving me this opportunity. I also acknowledge with a deep sense of reverence, my gratitude towards my parents and member of my family, who has always supported me morally as well as economically. At last but not least gratitude goes to all of my friends who directly or indirectly helped me to complete this project report. Any omission in this brief acknowledgement does not mean lack of gratitude.

Thanking You,
Rishabh Sheth
13ECUOG055

COMPANY PROFILE

Volansys Technology PVT. LTD. is one stop solution enabler to realize your ideas into reality quickly. Volansys was founded in 2008, by a team of experienced engineers to provide. technology solutions and services along with exceptional business value. Aimed to offer best products and services along with bleeding-edge technologies, company provide high quality solutions and services from initial concept stage, to prototyping through production – catering to the complete product development life cycle with less time to market. Company's team of designers & engineers specialize in the design, development and validation of Software solutions and Electronic Products.

Electronic Product Engineering Services : Manufacturing Ready Hardware Prototypes, Schematics Design, Layout Design, Placement & Routing, Design Validation Testing, Manufacturing Diagnostics & Sustenance Engineering, Product Re-engineering, Board Support Packages, Firmware Development, System/ Product Validation, QA Automation, Pre-silicon Validation, Post-silicon Validation, field of Connectivity, Communication, Audio-Video, IoT, Wearable and Medical Devices.

Mobility & Cloud Services : Saas & PaaS Development & Validation, IoT Server Development & Validation, Mobility Clients (iOS, Android & Windows ME), SAP Mobility, Enterprise Mobility, Android OS Customization & Porting, Android Hardening, Framework Customization, UX Design Services, Samsung Gear, Google Glass, Chromium, Validation services, GUI validation, Interop across devices & versions.

Software Solutions & Services : Enterprise Applications, Web Based application, Content Management, Big Data, business process automation, user experience, maintenance of application, desktop as well as web based applications, industrial automation, hardware interfacing, Complete QA cycle, QA Automation, agile processes.

SYNOPSIS

ZigBee is commonly preferred wireless technology in IoT because of its advantages over WiFi and Bluetooth like low power consumption, low cost, ability to work in mesh network etc. ZigBee architecture is divided into four main layer MAC layer, Physical layer , Network layer and Application layer. MAC and Physical layer follows IEEE 802.15.4 standard where Network layer is well defined by ZigBee Alliance. ZigBee Alliance has also given guidance for developing Application layer and it leaves designing part for manufacturing company. ZigBee Cluster Library known as ZCL is used to communicate between Application and Network layer. Because of the lack of standard protocol every manufacturing company has its own version of ZCL. This creates compatibility issues with various IoT products and it is difficult for application developer to make a common solution. Generic ZCL project tries to address this problem by making a universal library which ideally supports all ZigBee platform, which will help application developer to make platform independent solution. Apart from this project I have gone through training of two and half month which covers topics like advanced C programming, Linux essential, Modern operating system and Data structure. This training helped lot during development of this project.

TABLE OF CONTENTS

LIST OF FIGURES

Part - I : Industrial Project

LIST OF TABLE

Part - I Industrial Project

1. Problem Definition & Design Specification

Implement Generic ZigBee Cluster Library and to demonstrate APIs provided by generic ZCL also implement Network layer, Application support Sublayer, ZigBee Device Object and End application.

1.1 What is IOT?

The Internet of Things (IOT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.
the Internet of Things refers to the rapidly growing network of connected objects that are able to collect and exchange data using embedded sensors. Thermostats, cars, lights, refrigerators, and more appliances can all be connected to the IoT.

1.1.1 Applications of IOT

1. Smart Home: The smart home is likely the most popular IoT application at the moment because it is the one that is most affordable and readily available to consumers. From the Amazon Echo to the Nest Thermostat, there are hundreds of products on the market that users can control with their voices to make their lives more connected than ever.

2. Wearable: Watches are no longer just for telling time. The Apple Watch and other smartwatches on the market have turned our wrists into smart phone holsters by enabling text messaging, phone calls, and more. And devices such as Fitbit and Jawbone have helped revolutionize the fitness world by giving people more data about their workouts.

3. Smart Cities: The IoT has the potential to transform entire cities by solving real problems citizens face each day. With the proper connections and data, the Internet of Things can solve traffic congestion issues and reduce noise, crime, and pollution.

4. **Connected Car:** These vehicles are equipped with Internet access and can share that access with others, just like connecting to a wireless network in a home or office. More vehicles are starting to come equipped with this functionality, so prepare to see more apps included in future cars.

1.2 What is ZigBee?

ZigBee is a low-cost, low-power, wireless mesh network standard targeted at the wide development of long battery life devices in wireless control and monitoring applications. ZigBee devices have low latency, which further reduces average current. ZigBee chips are typically integrated with radios and with microcontrollers that have between 60-256 KB of flash memory. ZigBee operates in the

industrial, scientific and medical (ISM) radio bands: 2.4 GHz in most jurisdictions worldwide; 784 MHz in China, 868 MHz in Europe and 915 MHz in the USA and Australia. Data rates vary from 20 kbit/s (868 MHz band) to 250 Kbit/s (2.4 GHz band).

The ZigBee network layer natively supports both star and tree networks, and generic mesh networking. Every network must have one coordinator device, tasked with its creation, the control of its parameters and basic maintenance. Within star networks, the coordinator must be the central node. Both trees and meshes allow the use of ZigBee routers to extend communication at the network level. ZigBee builds on the physical layer and media access control defined in IEEE standard 802.15.4 for low-rate WPANs. The specification includes four additional key components: network layer, application layer, ZigBee device objects (ZDOs) and manufacturer-defined application objects. ZDOs are responsible for some tasks, including keeping track of device roles, managing requests to join a network, as well as device discovery and security.

ZigBee is one of the global standards of communication protocol formulated by the significant task force under the IEEE 802.15 working group. The fourth in the series, WPAN Low Rate/ZigBee is the newest and provides specifications for devices that have low data rates, consume very low power and are thus characterized by long battery life. Other standards like Bluetooth and IrDA address high data rate applications such as voice, video and LAN communications.

1.2.2 ZigBee architecture

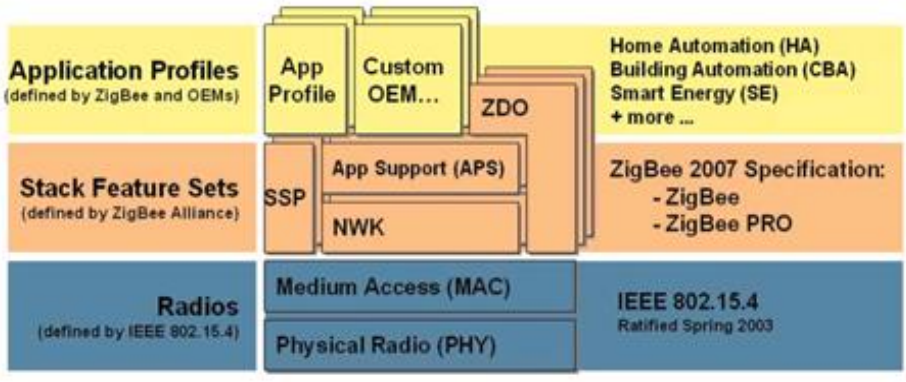

Figure 1.1 ZigBee Architecture [1]

The ZigBee architecture is separated by layers. Each layer performs a specific service set for the layer above. A data entity provides a data transmission service and a management entity provides all

other services. Each service entity exposes an interface to the upper layer through a SAP (service access point), and each SAP supports a number of service primitives to achieve the required functionality. The IEEE 802.15.4 defines the two lower layers. First, the PHY(Physical) layer and the MAC (medium access control) sub-layer. The ZigBee Alliance builds on this foundation by providing the network layer and the framework for the application layer. The application layer framework consists of the APS (application support sublayer) and the ZDO (ZigBee device objects). Application objects defined by the manufacturer use the framework and share APS and security with the ZDO. IEEE 802.15.4 has two PHY layers that operate in two separate frequency ranges, 868/915 MHz and 2.4 GHz. The lower frequency PHY layer covers both the 868 MHz European band and the 915 MHz band, used in countries such as the US and Australia. The higher frequency PHY layer is used virtually worldwide. The IEEE 802.15.4 MAC controls access to the radio channel using a CSMA-CA mechanism. Its responsibilities may also include transmitting beacon frames, synchronization, and providing a reliable transmission mechanism.

1.2.3 Advantages of ZigBee

- Low power consumption
- Low cost
- Long battery life
- Support multiple network topologies
- Ideal for WPAN and mesh networking

1.3 What is Generic ZigBee cluster library?

Up to now all the manufacturer made their own cluster library which is manufacture dependent. It means the layers which are going to use the ZigBee Cluster Library[2] wants some particular functionality to make system works. But in our design we make the generic ZigBee cluster library which is independent from manufacture. Means any ZigBee product can use this library without doing any major hardware changes.

2. Overview of ZigBee Specification

2.1 PHY, MAC and NWK layer in ZigBee

The ZigBee stack architecture includes a number of layered components including the IEEE 802.15.4-2003 Medium Access Control (MAC) layer, Physical (PHY) layer, and the ZigBee Network (NWK) layer. Each component provides an application with its own set of services and capabilities.

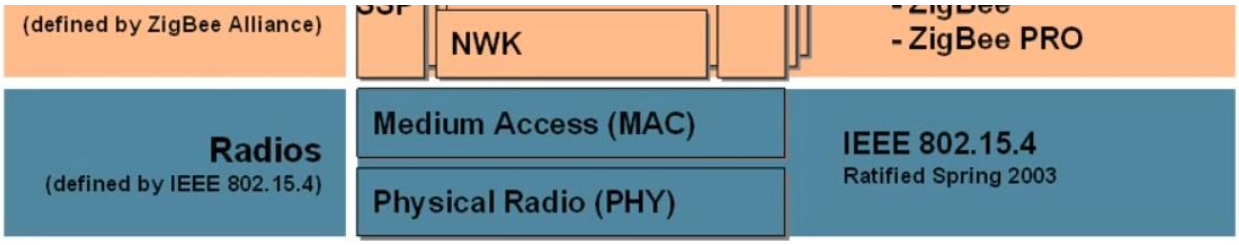

Figure 2.1 PHY, MAC and NWK layer in ZigBee

2.1.1 IEEE 802.15.4 standard Overview

IEEE 802.15.4 is a technical standard which defines the operation of low-rate wireless personal area networks (LR-WPANs). It specifies the physical layer and media access control for LR-WPANs, and is maintained by the IEEE 802.15 working group, which defined the standard in 2003. It is the basis for the ZigBee, ISA100.11a, WirelessHART, MiWi, SNAP, and Thread specifications, each of which further extends the standard by developing the upper layers which are not defined in IEEE 802.15.4. Alternatively, it can be used with 6LoWPAN, the technology used to deliver the IPv6 version of the Internet Protocol (IP) over WPANs, to define the upper layers.

IEEE standard 802.15.4 intends to offer the fundamental lower network layers of a type of wireless personal area network (WPAN) which focuses on low-cost, low-speed ubiquitous communication between devices. It can be contrasted with other approaches, such as Wi-Fi, which offer more bandwidth and require more power. The emphasis is on very low cost communication of nearby devices with little to no underlying infrastructure, intending to exploit this to lower power consumption even more.

The basic framework conceives a 10-meter communications range with a transfer rate of 250 kbit/s. Tradeoffs are possible to favor more radically embedded devices with even lower power requirements, through the definition of not one, but several physical layers. Lower transfer rates of 20 and 40 kbit/s were initially defined, with the 100 kbit/s rate being added in the current revision.
Even lower rates can be considered with the resulting effect on power consumption. As already mentioned, the main identifying feature of IEEE 802.15.4 among WPANs is the importance of achieving extremely low manufacturing and operation costs and technological simplicity, without sacrificing flexibility or generality.

Important features include real-time suitability by reservation of guaranteed time slots, collision avoidance through CSMA/CA and integrated support for secure communications. Devices also include power management functions such as link quality and energy detection.
IEEE 802.15.4-conformant devices may use one of three possible frequency bands for operation (868/915/2450 MHz).

2.1.2 The physical layer (PHY)

The physical layer is the initial layer in the OSI reference model used worldwide. The *physical layer* (PHY) ultimately provides the data transmission service, as well as the interface to the *physical layer management entity*, which offers access to every layer management function and maintains a database of information on related personal area networks. Thus, the PHY manages the physical RF transceiver and performs channel selection and energy and signal management functions. It operates on one of three possible unlicensed frequency bands:

- 868.0–868.6 MHz: Europe, allows one communication channel (2003, 2006, 2011)
- 902–928 MHz: North America, up to ten channels (2003), extended to thirty (2006)
- 2400–2483.5 MHz: worldwide use, up to sixteen channels (2003, 2006)

2.1.3 The Medium Access Control (MAC) layer

The medium access control (MAC) enables the transmission of MAC frames through the use of the physical channel. Besides the data service, it offers a management interface and itself manages access to the physical channel and network beaconing. It also controls frame validation, guarantees time slots and handles node associations. Finally, it offers hook points for secure services.

2.1.3.1 Topologies

Networks can be built as either peer-to-peer or star networks. However, every network needs at least one FFD to work as the coordinator of the network. Networks are thus formed by groups of devices separated by suitable distances. Each device has a unique 64-bit identifier, and if some conditions are met short 16-bit identifiers can be used within a restricted environment. Namely, within each PAN domain, communications will probably use short identifiers.

Peer-to-peer (or point-to-point) networks can form arbitrary patterns of connections, and their extension is only limited by the distance between each pair of nodes. They are meant to serve as the basis for ad hoc networks capable of performing self-management and organization. Since the standard does not define a network layer, routing is not directly supported, but such an additional layer can add support for multihop communications. Further topological restrictions may be added; the standard mentions the cluster tree as a structure which exploits the fact that an RFD may only be associated with one FFD at a time to form a network where RFDs are exclusively leaves of a tree, and most of the nodes are FFDs. The structure can be extended as a generic mesh network whose nodes are cluster tree networks with a local coordinator for each cluster, in addition to the global coordinator.

A more structured star pattern is also supported, where the coordinator of the network will necessarily be the central node. Such a network can originate when an FFD decides to create its own PAN and declare itself its coordinator, after choosing a unique PAN identifier. After that, other devices can join the network, which is fully independent from all other star networks.

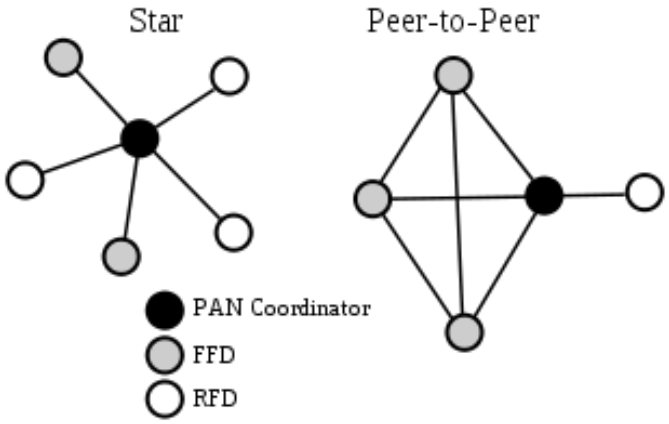

Figure 2.2 IEEE 802.15.4 Star and Peer-to-Peer connection

2.1.4 Network (NWK) Layer Overview

The network layer is required to provide functionality to ensure correct operation of the IEEE 802.15.4-2003 MAC sub-layer and to provide a suitable service interface to the application layer. To interface with the application layer, the network layer conceptually includes two service entities that provide the necessary functionality. These service entities are the data service and the management service. The NWK layer data entity (NLDE) provides the data transmission service via its associated SAP, the NLDE-SAP, and the NWK layer management entity (NLME) provides the management service via its associated SAP, the NLME-SAP. The NLME utilizes the NLDE to achieve some of its management tasks and it also maintains a database of managed objects known as the network information base (NIB).

2.1.4.1 Network Layer Data Entity (NLDE)

The NLDE shall provide a data service to allow an application to transport application protocol data units (APDU) between two or more devices. The devices themselves must be located on the same network.

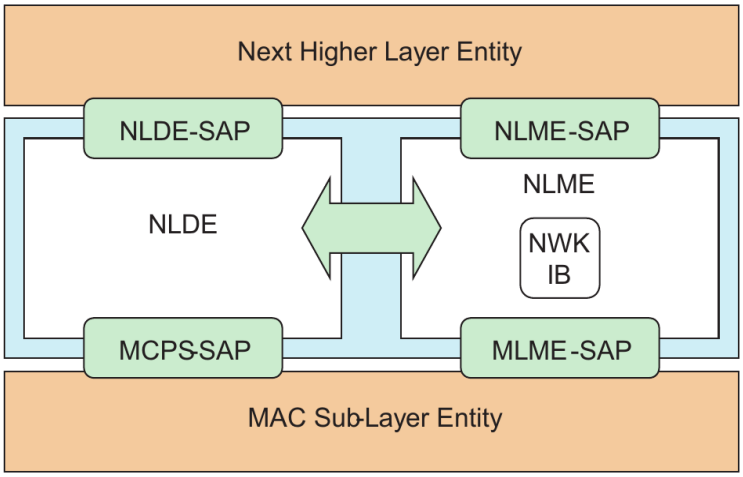

Figure 2.3 The NWK Layer Reference Model

The NLDE will provide the following services

- Generation of the Network level PDU (NPDU) : The NLDE shall be capable of generating an NPDU from an application support sublayer PDU through the addition of an appropriate protocol header.

- Topology-specific routing : The NLDE shall be able to transmit an NPDU to an appropriate device that is either the final destination of the communication or the next step toward the final destination in the communication chain.
- Security : The ability to ensure both the authenticity and confidentiality of a transmission.

2.1.4.2 Network Layer Management Entity (NLME)

The NLME shall provide a management service to allow an application to interact with the stack.
The NLME shall provide the following services

- Configuring a new device : this is the ability to sufficiently configure the stack for operation as required. Configuration options include beginning an operation as a ZigBee coordinator or joining an existing network.
- Starting a network : this is the ability to establish a new network.
- Joining, rejoining and leaving a network : this is the ability to join, rejoin or leave a network as well as the ability of a ZigBee coordinator or ZigBee router to request that a device leave the network.
- Addressing : this is the ability of ZigBee coordinators and routers to assign addresses to devices joining the network.
- Neighbour discovery : this is the ability to discover, record, and report information pertaining to the one-hop neighbours of a device.
- Route discovery : this is the ability to discover and record paths through the network, whereby messages may be efficiently routed.
- Reception control : this is the ability for a device to control when the receiver is activated and for how long, enabling MAC sub-layer synchronization or direct reception.
- Routing : this is the ability to use different routing mechanisms such as unicast, broadcast, multicast or many to one to efficiently exchange data in the network.

2.2 APS layer in ZigBee

The application support sublayer (APS) provides an interface between the network layer (NWK) and the application layer (APL) through a general set of services that are used by both the ZDO and the manufacturer-defined application objects.

2.2.1 Service Specification

In the application layer services are provided by two entities :

- The APS data entity (APSDE) through the APSDE service access point
- The APS management entity (APSME) through the APSME service access point

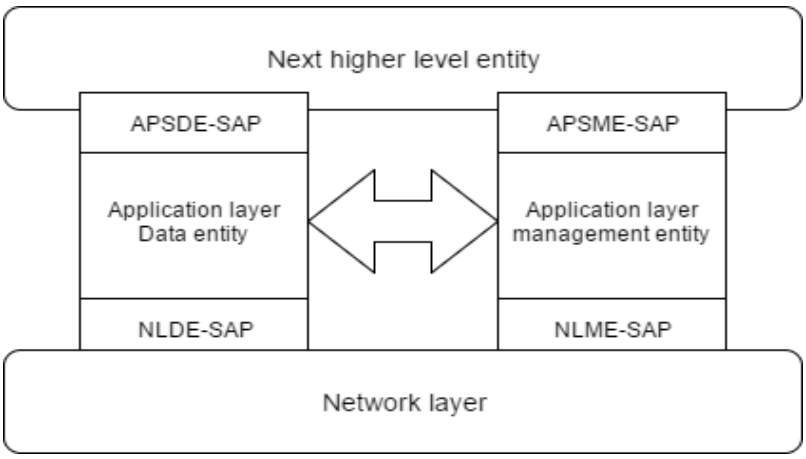

Figure 2.4 Interface of application layer

2.2.2 Application Support Sub-Layer Data Entity (APSDE)

The APSDE shall provide a data service to the network layer and both ZDO and application objects to enable the transport of application PDUs between two or more devices. The devices themselves must be located on the same network.

The APSDE should provide the following services :

- Generation of the application level PDU (APDU) : the APSDE shall take an application PDU and generate an APS PDU by adding the appropriate protocol overhead. When any data request came from the application layer, it will make the frame for that request and forward the packet to the network layer.

2.2.3 Application Support Sublayer Management Entity(APSME)

The APSME shall provide a management service to allow an application to interact with the stack.

The APSME should provide the following services :

- Generation of the device announce PDU : Once the device join in network, ZigBee device object of the devices is send the request of device announce, APSME shall take a ZDO

PDU and generate an APS PDU by adding the appropriate protocol overhead and forward the frame to the network layer.

- Device remove information : When device remove from the network.APS will inform that to the ZigBee device object of the coordinator to remove entry of that device from the device table.

2.2.4 Service primitives

- APSDE_DATA_Request: APSDE_DATA.Request This primitive requests the transfer of a NHLE PDU (ASDU) from the local NHLE to one or more peer NHLE entities.
- APSDE_DATA_Indication: This primitive indicates the transfer of a data PDU (ASDU) from the APS sub-layer to the local application entity.

2.2.5 Frame format

Octets :1	**0/1**	**0/2**	**0/2**	**0/2**	**0/1**	**1**	**0/ Variable**	**Variable**
Frame control	Destin-Ation endpoint	Group address	Cluster identifier	Profile Identifier	Source endpoint	APS Counter	Extended Header	Frame payload
APS header								APS payload

Table 2.1 APSE General Frame Format

2.2.5.1 Frame Control Field

The frame control field is 8-bits in length and contains information defining the frame type, addressing fields, and other control flags.

Bits: 0-1	2-3	4	5	6	7
Frame type	Delivery mode	Ack.Format	Security	Ack. request	Extended header present

Table 2.2 Format of the Frame Control Field

2.2.5.1.1 Frame Type Subfield

The frame type sub-field is two bits in length and shall be set to one of the non- reserved values listed in Table.

Frame Type Value b 1 b 0	Frame Type Name
00	Data
01	Command
10	Acknowledgement
11	Reserved

Table 2.3 Values of the Frame Type Subfield

2.2.5.1.2 Delivery Mode Sub-Field

The delivery mode sub-field is two bits in length and shall be set to one of the non-reserved values from Table.

Delivery Mode Value b 1 b 0	Delivery Mode Name
00	Normal unicast delivery

01	Indirect
10	Broadcast
11	Group addressing

Table 2.4 Values of the Delivery Mode Sub-Field

2.2.5.1.3 Ack Format Field

This bit indicates if the destination endpoint, cluster identifier, profile identifier and source endpoint fields shall be present in the acknowledgement frame

Ack format field value	Frame name
0	data frame acknowledgement
1	command frame acknowledgement.

Table 2.5 Values of the Ack format field value

2.2.5.1.4 Acknowledgement Request Sub-Field:

The acknowledgement request subfield is one bit in length and specifies whether the current transmission requires an acknowledgement frame to be sent to the originator on receipt of the frame. If this subfield is set to 1, the recipient shall construct and send an acknowledgement frame back to the originator after determining that the frame is valid. If this subfield is set to 0, the recipient shall not send an acknowledgement frame back to the originator. This sub-field shall be set to 0 for all frames that are broadcast or multicast.

2.2.5.1.5 Extended Header Present

The extended header present sub-field is one bit in length and specifies whether the extended header shall be included in the frame. If this subfield is set to 1, then the extended header shall be included in the frame. Otherwise, it shall not be included in the frame.

2.2.5.2 Destination Endpoint Field

The destination endpoint field is 8-bits in length and specifies the endpoint of the final recipient of the frame. This field shall be included in the frame only if the delivery mode sub-field of the frame control field is set to 0b00 (normal unicast delivery), 0b01 (indirect delivery where the indirect address mode sub-field of the frame control field is also set to 0), or 0b10 (broadcast delivery). In the case of broadcast delivery, the frame shall be delivered to the destination endpoint specified within the range 0x01-0xf0 or to all active endpoints if specified as 0xff.

2.2.5.3 Group Address Field

The group address field is 16 bits in length and will only be present if the delivery mode sub-field of the frame control has a value of 0b11. In this case, the destination endpoint shall not be present. If the APS header of a frame contains a group address field, the frame will be delivered to all endpoints for which the group table in the device contains an association between that endpoint and the group identified by the contents of the group address field. Devices where nwk Use Multicast is set to TRUE shall never set the group address field of an outgoing frame.

2.2.5.4 Cluster Identifier Field

The cluster identifier field is 16 bits in length and specifies the identifier of the cluster to which the frame relates and which shall be made available for filtering and interpretation of messages at each device that takes delivery of the frame. This field shall be present only for data or acknowledgement frames.

2.2.5.5 Profile Identifier Field

The profile identifier is two octets in length and specifies the ZigBee profile identifier for which the frame is intended and shall be used during the filtering of messages at each device that takes delivery of the frame. This field shall be present only for data or acknowledgement frames.

2.2.5.6 Source Endpoint Field

The source endpoint field is eight-bits in length and specifies the endpoint of the initial originator of the frame. A source endpoint value of 0x00 indicates that the frame originated from the ZigBee device object (ZDO) resident in each device. A source endpoint value of 0x01-0xf0 indicates that the frame originated from an application operating on that endpoint. All other endpoints (0xf1-0xff) are reserved.

2.2.5.7 APS Counter

This field is eight bits in length and is used to prevent the reception of duplicate frames. This value shall be incremented by one for each new transmission.

2.2.5.8 Extended Header Sub-Frame

The extended header sub-frame contains further sub-fields and shall be formatted as illustrated in Figure

Octets: 1	0/1	0/1
Extended frame control	Block number	ACK bit field

Table 2.6 Extended header sun-frame format

2.2.5.9 Frame Payload Field

The frame payload field has a variable length and contains information specific to individual frame types.

2.3 ZCL in ZigBee

The ZCL is a repository for cluster functionality that is developed by the ZigBee Alliance, and, as a consequence, it will be a working library with regular updates as new functionality is added.

A developer constructing a new application profile should use the ZCL to find relevant cluster functionality that can be incorporated into the new profile. Correspondingly, new clusters that are defined for application profiles should be considered for inclusion in the ZCL.

The ZCL consists of the ZCL Foundation, a set of elements that apply across the entire library (such as frame structures, attribute access commands and data types), and a number of sets of clusters. Clusters that are generally useful across many application domains are included in the General set.

Clusters that are intended for use mainly in specific application domains are grouped together in domain oriented sets.

2.3.1 Command Frame Formats

The ZCL frame format is composed of a ZCL header and a ZCL payload. The general ZCL frame shall be formatted as illustrated in Table 2.7.

Bits: 8	**0/16**	**8**	**8**	**Variable**
Frame control	Manufacturer code	Transaction sequence number	Command identifier	Frame payload
ZCL header				ZCL Payload

Table 2.7 Format of the General ZCL Frame

2.3.1.1 Frame Control Field

The frame control field is 8 bits in length and contains information defining the command type and other control flags. The frame control field shall be formatted as shown in Table.2.8 Bits 5-7 are reserved for future use and shall be set to 0.

2.3.1.1.1 Frame Type Sub-field

The frame type sub-field is 2 bits in length and shall be set to one of the non-reserved values listed Table 2.9.

Bits: 0-1	**2**	**3**	**4**	**5-7**
Frame type	Manufacturer specific	Direction	Disable default response	Reserved

Table 2.8 Format of the Frame Control Field

2.3.1.1.2 Manufacturer Specific Sub-field

The manufacturer specific sub-field is 1 bit in length and specifies whether this command refers to a manufacturer specific extension to a profile. If this value is set to 1, the manufacturer code field shall be present in the ZCL frame. If this value is set to 0, the manufacturer code field shall not be included in the ZCL Frame.

Frame Type value **b1 b0**	**Description**
00	Command acts across the entire profile
01	Command is specific to a cluster
10-11	Reserved

Table 2.9 Values of the Frame Type Sub-field

2.3.1.1.3 Direction Sub-field

The direction sub-field specifies the client/server direction for this command. If this value is set to 1, the command is being sent from the server side of a cluster to the client side of a cluster. If this value is set to 0, the command is being sent from the client side of a cluster to the server side of a cluster.

2.3.1.1.4 Disable Default Response Sub-field

The disable default response sub-field is 1 bit in length. If it is set to 0, the Default response command will be returned. If it is set to 1, the Default response command will only be returned if there is an error.

2.3.1.2 Manufacturer Code Field

The manufacturer code field is 16 bits in length and specifies the ZigBee assigned manufacturer code for proprietary extensions to a profile. This field shall only be included in the ZCL frame if the manufacturer specific sub-field of the frame control field is set to 1.

2.3.1.3 Transaction Sequence Number

The transaction sequence number field is 8 bits in length and specifies an identification number for the transaction so that a response-style command frame can be related to a request-style command

frame. The application object itself shall maintain an 8-bit counter that is copied into this field and incremented by one for each command sent. When a value of 0xff is reached, the next command shall re-start the counter with a value of 0x00. The transaction sequence number field can be used by a controlling device, which may have issued multiple commands, so that it can match the incoming responses to the relevant command.

2.3.1.4 Command Identifier Field

The command identifier field is 8 bits in length and specifies the cluster command being used. If the frame type sub-field of the frame control field is set to 0b00, the command identifier corresponds to one of the non-reserved values. If the frame type sub-field of the frame control field is set to 0b01, the command identifier corresponds to a cluster specific command. The cluster specific command identifiers can be found in each individual document describing the clusters.

2.3.2 General Commands Frames

General command frames are used for manipulating attributes and other general tasks that are not specific to an individual cluster. All clusters (server and client) shall support generation, reception and execution of the Default response command.

Command Identifier Field Value	**Description**
0x00	Read attributes
0x01	Read attributes response
0x02	Write attributes
0x04	Write attributes response

Table 2.10 ZCL Command Frames

2.3.2.1 Read Attributes Command

2.3.2.1.1 Read Attributes Command Frame Format

The read attributes command frame shall be formatted as illustrated in Table 2.11.

Octets: Variable	2
ZCL header	Attribute identifier

Table 2.11 Format of the Read Attributes Command Frame

ZCL Header Fields

The frame control field shall be specified as follows. The frame type sub-field shall be set to indicate a profile wide command (0b00). The manufacturer specific sub-field shall be set to 0 if this command is being used to read attributes defined for any cluster in the ZCL or 1 if this command is being used to read manufacturer specific attributes.

Attribute Identifier Field

The attribute identifier field is 16 bits in length and shall contain the identifier of the attribute that is to be read.

2.3.2.1.2 When Generated

The read attributes command is generated when a device wishes to determine the values of one or more attributes located on another device. Each attribute identifier field shall contain the identifier of the attribute to be read.

2.3.2.1.3 Effect on Receipt

On receipt of this command, the device shall process each specified attribute identifier and generate a read attributes response command.

2.3.2.2 Read Attributes Response Command

2.3.2.2.1 Read Attributes Response Command Frame Format

The read attributes response command frame shall be formatted as illustrated in Table 2.12.

Octets: Variable	Variable
ZCL header	Read attribute status record

Table 2.12 Format of Read Attributes Response Command Frame

Read attribute status record shall be formatted as illustrated in Table 2.13.

Octets: 2	1	0/1	0 / Variable
Attribute identifier	Status	Attribute data type	Attribute value

Table 2.13 Format of the Read Attributes Status Record Field

ZCL Header Fields

The frame control field shall be specified as follows. The frame type subfield shall be set to indicate a profile wide command (0b00). The manufacturer specific sub-field shall be set to 0 if this command is being used as a response to reading attributes defined for any cluster in the ZCL or 1 if this command is being used as a response to reading manufacturer specific attributes.

Attribute Status Record

- **Attribute Identifier Field**

  The attribute identifier field is 16 bits in length and shall contain the identifier of the attribute that has been read (or of which an element has been read). This field shall contain the same value that was included in the corresponding attribute identifier field of the original read attributes or read attributes structured command.

- **Status Field**

  The status field is 8 bits in length and specifies the status of the read operation on this attribute. This field shall be set to SUCCESS, if the operation was successful, or an error code.

- **Attribute Data Type Field**

  The attribute data type field shall contain the data type of the attribute in the same read attributes status record. This field shall only be included if the associated status field contains a value of SUCCESS.

- **Attribute Value Field**

  The attribute value field is variable in length and shall contain the current value of this attribute. This field shall only be included if the associated status field contains a value of SUCCESS.

2.3.2.2.2 When Generated

The read attributes response command is generated in response to a read attributes or read attributes structured command.

2.3.2.2.3 Effect on Receipt

On receipt of this command, the originator is notified of the results of its original read attributes attempt and, for each successful request, the value of the requested attribute.

2.3.2.3 Write Attributes Command

2.3.2.3.1 Write Attributes Command Frame Format

The write attributes command frame shall be formatted as illustrated in Table 2.14.

Octets: Variable	Variable
ZCL header	Write attribute record

Table 2.14 Format of the Write Attributes Command Frame

Write attribute record shall be formatted as illustrated in Table2.15.

Octets: 2	1	Variable
Attribute identifier	Attribute data type	Attribute data

Table 2.15 Format of the Write Attributes Record Field

ZCL Header Fields

The frame control field shall be specified as follows. The frame type sub-field shall be set to indicate a profile wide command (0b00). The manufacturer specific sub-field shall be set to 0 if this command is being used to write attributes defined for any cluster in the ZCL or 1 if this command is being used to write manufacturer specific attributes.

Write attribute record

- **Attribute Identifier Field**
  The attribute identifier field is 16 bits in length and shall contain the identifier of the attribute that is to be written.
- **Attribute Data Type Field**
  The attribute data type field shall contain the data type of the attribute that is to be written.
- **Attribute Data Field**
  The attribute data field is variable in length and shall contain the actual value of the attribute that is to be written.

2.3.2.3.2 When Generated

The write attributes command is generated when a device wishes to change the values of one or more attributes located on another device. Each write attribute record shall contain the identifier and the actual value of the attribute to be written.

2.3.2.3.3 Effect on Receipt

On receipt of this command, the device shall attempt to process each specified write attribute record and shall construct a write attribute response command.

2.3.2.4 Write Attributes Response Command

2.3.2.4.1 Write Attributes Response Command Frame Format

The write attributes response command frame shall be formatted as illustrated in Table 2.16.

Octets: Variable	3
ZCL header	Write attribute status Record

Table 2.16 Format of Write Attributes Response Command Frame

write attribute status record shall be formatted as illustrated in Table2.17.

Octets: 2	1
Attribute identifier	Status

Table 2.17 Format of the Write Attribute Status Record Field

ZCL Header Fields

The frame control field shall be specified as follows. The frame type sub-field shall be set to indicate a profile wide command (0b00). The manufacturer specific sub-field shall be set to 0 if this command is being used as a response to writing attributes defined for any cluster in the ZCL or 1 if this command is being used as a response to writing manufacturer specific attributes.

Write Attribute Status Record

- **Attribute identifier**

  The attribute identifier field is 16 bits in length and shall contain the identifier of the attribute on which the write operation was attempted.

- **Status**

  The status field is 8 bits in length and specifies the status of the write operation, attempted on the attribute specified by *attributeId*. If Attribute value modified successfully, write attribute status record shall be included in the command, with the status field set to SUCCESS.

2.3.3 Cluster

2.3.2.1 ON/OFF Cluster

2.3.2.1.1 Overview

Attributes and commands for switching devices between ‘On’ and ‘Off’ states.

2.3.2.1.2 Dependencies

There is no dependency.

2.3.2.1.3 Attributes

The server supports the attributes shown in Table 2.18.

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	*OnOff*	Boolean	0x00 – 0x01	Read only	0x00	M

Table 2.18 Attributes of the On/Off Cluster

2.3.2.1.4 Commands Received

The command IDs for the On/Off cluster are listed in Table 2.19.

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Off	M
0x01	On	M
0x02	Toggle	M
0x03 - 0xff	Reserved	-

Table 2.19 Command IDs for the On/Off Cluster

- **Off Command :** This command does not have a payload. On receipt of this command, a device shall enter its 'Off' state. This state is device dependent, but it is recommended that it is used for power off or similar functions.
- **On Command :** This command does not have a payload. On receipt of this command, a device shall enter its 'On' state. This state is device dependent, but it is recommended that it is used for power on or similar functions.
- **Toggle Command :** This command does not have a payload. On receipt of this command, if a device is in its ‘Off’ state it shall enter its 'On' state. Otherwise, if it is in its ‘On’ state it shall enter its 'Off' state.
- **Commands Generated :**The server generates no commands.

2.3.2.2 Level Control Cluster

2.3.2.2.1 Overview

This cluster provides an interface for controlling a characteristic of a device that can be set to a level, for example the brightness of a light, the degree of closure of a door, or the power output of a heater.

2.3.2.2.2 Dependencies

For many applications, a close relationship between this cluster and the OnOff cluster is needed. This section describes the dependencies that are required when an endpoint that implements the Level Control server cluster also implements the On/Off server cluster.

The OnOff attribute of the On/Off cluster and the CurrentLevel attribute of the Level Control cluster are intrinsically independent variables, as they are on different clusters. However, when both clusters are implemented on the same endpoint, dependencies may be introduced between them. Facilities are provided to introduce dependencies if required.

2.3.2.2.3 Effect of On/Off Commands on the CurrentLevel Attribute

The attribute *OnLevel* determines whether commands of the On/Off cluster have a permanent effect on the *CurrentLevel* attribute or not. If this attribute is defined (i.e. implemented and not 0xff) they do have a permanent effect, otherwise they do not. There is always a temporary effect, due to fading up / down. The effect on the Level Control cluster on receipt of the various commands of the On/Off

cluster are as detailed in following Table 2.20. In this table, and throughout this cluster specification, 'level' means the value of the *CurrentLevel* attribute.

Command	Action On Receipt
On	Temporarily store *CurrentLevel* Set *CurrentLevel* to the minimum level allowed for the device. Move *CurrentLevel* to *OnLevel*, or to the stored level if *OnLevel* is not defined, over the time period *OnOffTransitionTime*.
Off	Temporarily store *CurrentLevel* Move *CurrentLevel* to the minimum level allowed for the device over the time period *OnOffTransitionTime*. If *OnLevel* is not defined, set the *CurrentLevel* to the stored level.
Toggle	If the *OnOff* attribute has the value Off, proceed as for the On command. Otherwise proceed as for the Off command.

Table 2.20 Actions on Receipt for On/Off Commands, when Associated with Level Control

2.3.2.2.4 Effect of Level Control Commands on the OnOff Attribute

There are two sets of commands provided in the Level Control cluster. These are identical, except that the first set (Move to Level, Move and Step) shall not affect the *OnOff* attribute, whereas the second set ('with On/Off' variants) shall. The first set is used to maintain independence between the *CurrentLevel* and *OnOff* attributes, so changing *CurrentLevel* has no effect on the *OnOff* attribute. As examples, this represents the behavior of a volume control with a mute button, or a 'turn to set level and press to turn on/off' light dimmer.

The second set is used to link the *CurrentLevel* and *OnOff* attributes. When the level is reduced to its minimum the *OnOff* attribute is automatically turned to Off, and when the level is increased above its minimum the *OnOff* attribute is automatically turned to On. As an example, this represents the behavior of a light dimmer with no independent on/off switch.

2.3.2.2.5 Attributes

The attributes of the Level Control server cluster are summarized in Table 2.21.

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	CurrentLevel	Unsigned 8-bit integer	0x00 – 0xff	Read Only	0x00	M
0x0001	RemainingTime	Unsigned 16-bit integer	0x0000 – 0xffff	Read Only	0x0000	M
0x0002	OnOffTransitionTime	Unsigned 16-bit integer	0x0000 – 0xffff	Read / Write	0x0000	M
0x0003	OnLevel	Unsigned 8-bit Integer	0x00 – 0xfe	Read / Write	0xfe	M

Table 2.21 Attributes of the Level control Cluster

- **CurrentLevel Attribute :** The CurrentLevel attribute represents the current level of this device. The meaning of 'level' is device dependent.
- ***RemainingTime* Attribute :** The *RemainingTime* attribute represents the time remaining until the current command is complete, it is specified in 1/10ths of a second.
- ***OnOffTransitionTime* Attribute :** The *OnOffTransitionTime* attribute represents the time taken to move to or from the target level when On of Off commands are received by an On/Off cluster on the same endpoint. It is specified in 1/10ths of a second. The actual time taken should be as close to *OnOffTransitionTime* as the device is able.
- ***OnLevel* Attribute** The *OnLevel* attribute determines the value that the *CurrentLevel* attribute is set to when the *OnOff* attribute of an On/Off cluster on the same endpoint is set to On. If the *OnLevel* attribute is not implemented, or is set to 0xff, it has no effect.

2.3.2.2.5 Commands

The command IDs specific to the Level Control cluster are listed in Table 2.22.

Command Identifier Field Value	Description	Mandatory / Optional
0x00	Move to Level	M
0x04	Move to Level (with On/Off)	M

Table 2.22 Command IDs for the Level Control Cluster

Move to Level Command

Payload Format

The Move to Level command payload shall be formatted as illustrated in Figure 2.23.

Octet	1	2
Data Type	Unsigned 8-bit Integer	Unsigned 16-bit Integer
Field Name	Level	Transition time

Table 2.23 Format of the Move to Level Command Payload

2.3.2.2.6 Effect on Receipt

On receipt of this command, a device shall move from its current level to the value given in the Level field. The meaning of ‘level’ is device dependent – e.g. for a light it may mean brightness level.

The movement shall be as continuous as technically practical, i.e. not a step function, and the time taken to move to the new level shall be equal to the value of the Transition time field, in tenths of a second, or as close to this as the device is able.

If the Transition time field takes the value *0xffff* then the time taken to move to the new level shall instead be determined by the *OnOffTransitionTime* attribute. If *OnOffTransitionTime* , which is an optional attribute, is not present, the device shall move to its new level as fast as it is able. If the device is not able to move at a variable rate, the Transition time field may be disregarded.

2.4 ZDO layer in ZigBee

2.4.1 ZigBee Coordinator

ZigBee coordinator will send simple descriptor request whenever any new device announce itself and in response that end device will have to send the response to the coordinator. On reception of simple descriptor response ZDO of coordinator will check clusters that are supported by that device and if that supported cluster matches with supported cluster list maintain by coordinator then only that node can be served by ZigBee coordinator. If cluster matches then that device will be added into the available device list so end application can control that node.

2.4.1.1 Simple Descriptor Request

The Simple_Desc_req command (ClusterID=0x0004) shall be formatted as illustrated in Table 2.24.

Octets : 2	**1**
Network Address of Interest	Endpoint

Table 2.24 Format of the Simple_Desc_req Command Frame

Name	**Type**	**Valid Range**	**Description**
Network Address of Interest	Device address	16 bit nwk address	Network address for the request
Endpoint	8 bits	1-240	The endpoint on the destination

Table 2.25 Fields of the simple descriptor request

The Simple_Desc_req command is generated from a local device(ZigBee coordinator) wishing to inquire as to the simple descriptor of a remote device on a specified endpoint. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device. The local device shall generate the Simple_Desc_req command using the format illustrated in Table 2. The Network Address of Interest field shall contain the network address of the remote device for which the simple descriptor is required and the endpoint field shall contain the endpoint identifier from which to obtain the required simple descriptor.

Upon receipt of this command, the recipient device shall process the command and generate a Simple_Desc_rsp command in response.

2.4.2 ZigBee End Device

Whenever End device join the ZigBee network at coordinator side end application has no information related to that device to make end application aware about newly joined device. Device itself will send announce command to coordinator Zigbee Device object. And on reception of this command ZigBee coordinator ZDO will send request for Simple descriptor. And by reception of this request end device ZDO will form response and send it to ZDO.

2.4.2.1 Device_annce

The Device_annce command (ClusterID=0x0013) shall be formatted as illustrated in Table 2.26.

Octets: 2	**8**	**1**
Network Address	IEEEAddr	Capability

Table 2.26 Format of the Device_annce Command Frame

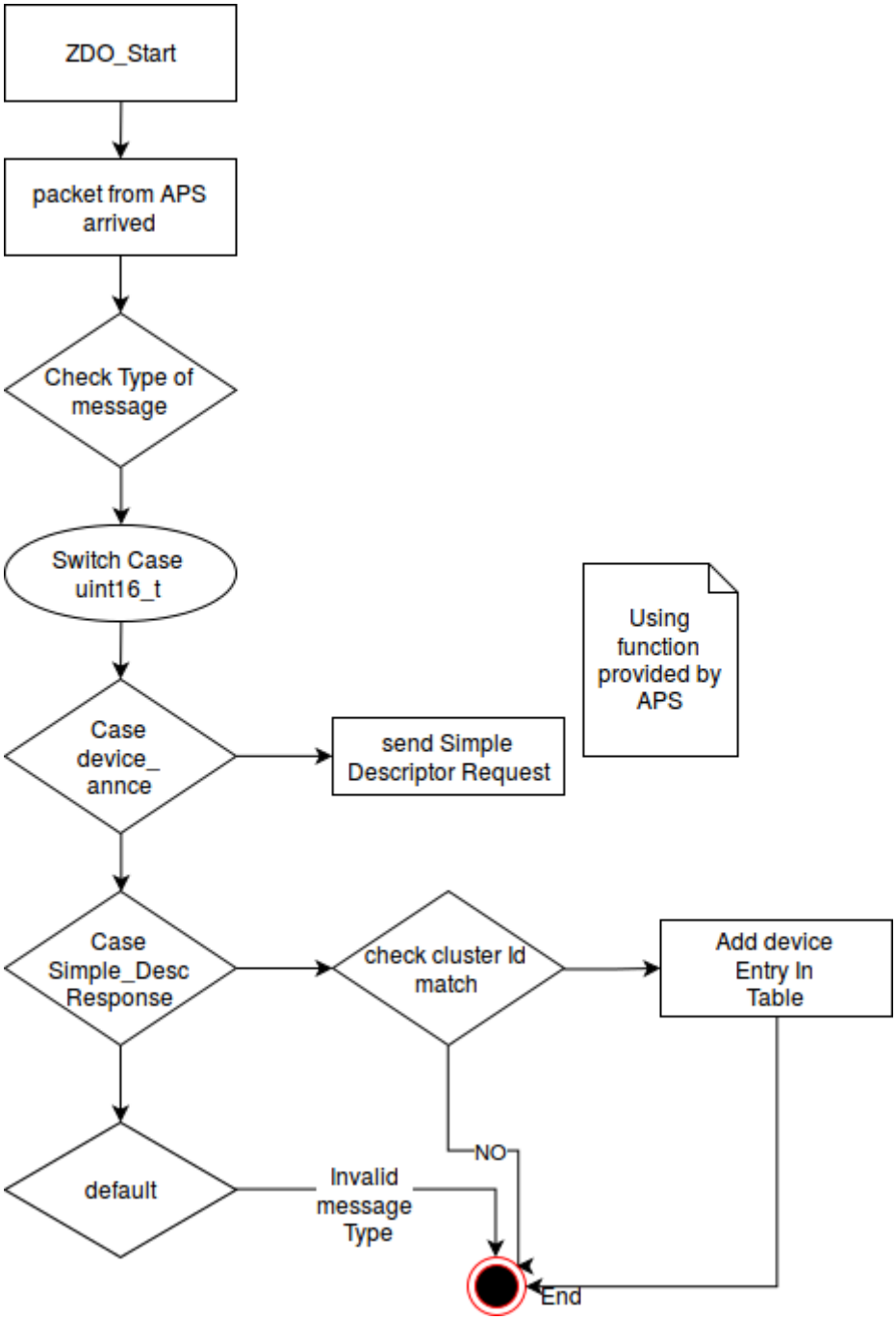

Figure 2.5 Overview of ZigBee Coordinator ZDO

Name	Type	Valid Range	Description
Network Address	Device Address	16-bit NWK address	NWK address for the Local Device

IEEEAddr	Device Address	64-bit IEEE address	IEEE address for the Local Device
Capability	Bitmap	See Table 5	Capability of the local device

Table 2.27 Fields of the Device_annce

Bits : 0	**1**	**2**	**3**	**4-5**	**6**	**7**
Alternate PAN coordinator	Device type	Power Source	Receiver on when idle	Reserved	Security capability	Allocate address

Table 2.28 Capability Flags Field

2.4.2.2 Simple Descriptor Response

The Simple_Desc_rsp command (ClusterID=0x8004) shall be formatted as illustrated in table 2.29.

Octet : 1	**2**	**1**	**variable**
Status	Network Address of Interest	Length	Simple descriptor

Table 2.29 Format of the Simple_Desc_rsp Command Frame

The Simple_Desc_rsp is generated by a remote device in response to a Simple_Desc_req directed to the remote device. This command shall be unicast to the originator of the Simple_Desc_req command. The remote device shall generate the Simple_Desc_rsp command using the format illustrated in Table 6. The Network Address of Interest field shall match that specified in the original Simple_Desc_req command. If the endpoint field specified in the original Simple_Desc_req

command does not fall within the correct range specified in Table 30, the remote device shall set the Status field to INVALID_EP, set the Length field to 0 and not include the SimpleDescriptor Field.

If the Network Address of Interest field matches the network address of the remote device, it shall determine whether the endpoint field specifies the identifier of an active endpoint on the device. If the endpoint field corresponds to an active endpoint, the remote device shall set the Status field to SUCCESS, set the Length field to the length of the simple descriptor on that endpoint, and include the simple descriptor for that endpoint in the SimpleDescriptor field.

If the endpoint field does not correspond to an active endpoint, the remote device shall set the Status field to NOT_ACTIVE, set the Length field to 0, and not include the SimpleDescriptor field. If the Network Address of Interest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE, set the Length field to 0, and not include the SimpleDescriptor field.

If the Network Address of Interest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the Network Address of Interest field matches the network address of one of its children. If the Network Address of Interest field does not match the network address of one of the children of the remote device, it shall set the Status field to DEVICE_NOT_FOUND, set the Length field to 0, and not include the SimpleDescriptor field. If the Network Address of Interest matches the network address of one of the children of the remote device, it shall determine whether a simple descriptor for that device and on the requested endpoint is available.

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INVALID_EP, NOT_ACTIVE, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the Simple_Desc_req command.
Network Address of	Device	16-bit NWK address	NWK address for the

Interest	Address		request.
Length	Integer	0x00-0xff	Length in bytes of the Simple Descriptor to follow.
Simple Descriptor	Simple Descriptor		See the Simple Descriptor format in sub clause(5.2.2.1) .This field shall only be included in the frame if the status field is equal to SUCCESS.

Table 2.30 Fields of the simple descriptor response

If a simple descriptor is not available on the requested endpoint of the child indicated by the Network Address of Interest field, the remote device shall set the Status field to NO_DESCRIPTOR, set the Length field to 0, and not include the SimpleDescriptor field. If a simple descriptor is available on the requested endpoint of the child indicated by the Network Address of Interest field, the remote device shall set the Status field to SUCCESS, set the Length field to the length of the simple descriptor on that endpoint, and include the simple descriptor for that endpoint of the matching child device in the SimpleDescriptor field.

2.4.2.2.1 Simple Descriptor

The simple descriptor contains information specific to each endpoint contained in this node. The simple descriptor is mandatory for each endpoint present in the node. The fields of the simple descriptor are shown in Table 2.31.

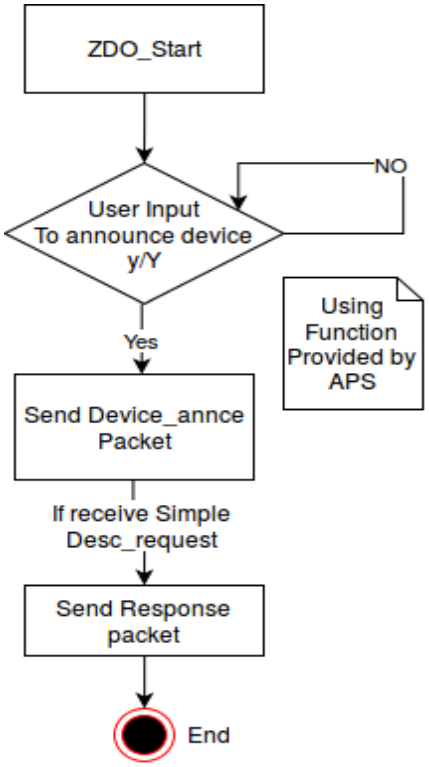

Figure 2.6 Overview of ZigBee End Node ZDO

- **Endpoint Field :** The endpoint field of the simple descriptor is eight bits in length and specifies the endpoint within the node to which this description refers. Applications shall only use endpoints 1-240.
- **Application Profile Identifier Field :** The application profile identifier field of the simple descriptor is sixteen bits in length and specifies the profile that is supported on this endpoint. Profile identifiers shall be obtained from the ZigBee Alliance.

Field Name	Length (Bits)
Endpoint	8
Application profile identifier	16
Application device identifier	16
Application device version	4
Reserved	4

Application input cluster count	8
Application input cluster list	16*i (where i is the value of the application input cluster count)

Table 2.31 Fields of the Simple Descriptor

- **Application Device Identifier Field :** The application device identifier field of the simple descriptor is sixteen bits in length and specifies the device description supported on this endpoint. Device description identifiers shall be obtained from the ZigBee Alliance.
- **Application Device Version Field :** The application device version field of the simple descriptor is four bits in length and specifies the version of the device description supported on this endpoint.
- **Application Input Cluster Count Field :** The application input cluster count field of the simple descriptor is eight bits in length and specifies the number of input clusters, supported on this endpoint, that will appear in the application input cluster list field. If the value of this field is zero, the application input cluster list field shall not be included.
- **Application Input Cluster List :** The application input cluster list of the simple descriptor is 16*i bits in length, where i is the value of the application input cluster count field. This field specifies the list of input clusters supported on this endpoint, for use during the service discovery and binding procedures. The application input cluster list field shall be included only if the value of the application input cluster count field is greater than zero.

3. Design & Implementation

3.1 Network Layer

Our project is to make Generic ZigBee Cluster Library but to test our Library implementation along with ZCL we have to make NWK, APS layer and also End user application. As we are implementing our project on PC not on actual ZigBee device (because of time constraints and feasibility) MAC , PHY and NWK layer are different on PC (LAN & IPv4 socket which we used in used in our project). So here network layer implementation don't follow ZigBee Alliance specification but we tried to make network layer in way that when we shift our whole project on actual ZigBee device we just need to change some part in network layer and upper layer should remain unchanged.

3.1.1 Design overview

We used socket programming concepts to build / simulate ZigBee like network layer. The network layer is required to provide functionality to ensure correct operation of passing message of next higher layer to peer device's next higher layer. It receives data from higher layer (in our case Application support sublayer - APS) then as per the requirement it should add it's own header and other security fe.. Then pass packet to lower layer (i.e. MAC) on other side network should remove its header and check parameters like checksum etc. After that should pass to above layer .

There are two types of Internet Protocol (IP) traffic. They are TCP or Transmission Control Protocol and UDP or User Datagram Protocol. TCP is connection oriented – once a connection is established, data can be sent bidirectional. UDP is a simpler, connectionless Internet protocol. Multiple messages are sent as packets in chunks using UDP. Table 3.1 compares various aspect of TCP and UDP .

	TCP	UDP
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Function	As a message makes its way across the internet from one computer to	UDP is also a protocol used in message transport or transfer. This

	another. This is connection based.	is not connection based which means that one program can send a load of packets to another and that would be the end of the relationship.
Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other. If ordering is required, it has to be managed by the application layer.
Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes
Reliability	There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.	There is no guarantee that the messages or packets sent would reach at all.
Speed of transfer	The speed for TCP is slower than UDP	UDP is faster because error recovery is not attempted. It is a "best effort" protocol

Table 3.1 TCP vs. UDP

Despite TCP is slower and has big header size compared to UDP it is more reliable and has capability like error correction , data packets in the order specified way etc. TCP is our choice for

this project . Figure 3.1. Present TCP client - server interaction which. Each block represent system call which have following meaning

- System Calls for socket programming
- getaddrinfo() — Prepare to launch!
- socket() — Get the File Descriptor!
- bind() — What port am I on?
- connect() — Hey, you!
- listen() — Will somebody please call me?
- accept() — “Thank you for calling port 3490.”
- send() and recv() —Talk to me!
- close() and shutdown() — Goodbye !

3.1.2 Summary of NWK layer design

- TCP / IP is used for network connection
- Start topology is used to communicate between Coordinator and End device
- Number of end device can be connected to coordinator is limited to Five
- As requested from upper layer network should add or remove device
- Network layer should manage a Device table which contains following attributes
    - Device ID
    - Socket Number
    - IP address of that device
- Network should provide APIs to upper for following purpose
    - To network initialization
    - For sending message from APS peer APS
    - For passing received message from NWK to APS

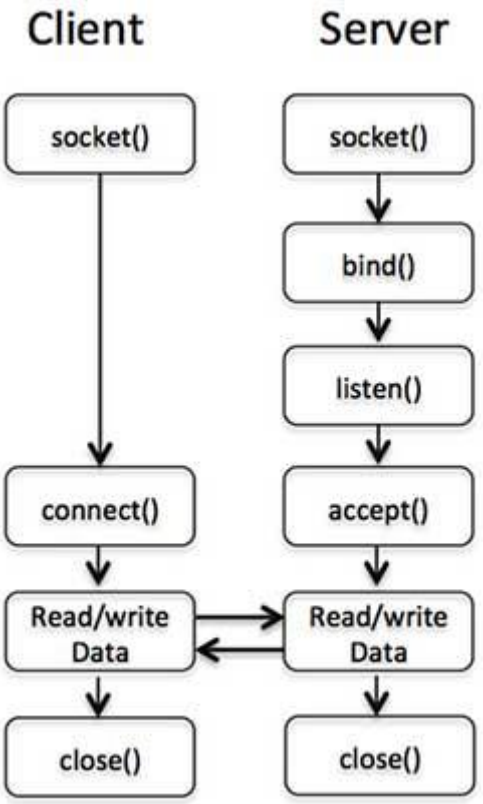

Figure 3.1 A TCP Server - Client Interaction

3.1.3 Implementation of NWK layer

3.1.3.1 Used Structures

- **NwkMsg_t** : This structure is used to pass payload from APS to NWK to other devices NWK to APS layer.

```
typedef struct
{
        device_id;
        packet_length;
        packet[500];
}NwkMsg_t;
```

Name	Type	Description
device_id	uint16_t	Network will provide unique Device ID from available pool of IDs to every new incoming connection and upper layer only knows device id and all data communication and management in upper layer is based on this ID
packet_length	uint16_t	Length of packet which we want to send to other device
Packet	uint8_t	Maximum length of payload / packet is kept to 500 bytes

Table 3.2 Structure NwkMsg_t

- **DeviceList :** This structure is used by NWK layer to maintain list of connected device it is most important component on NWK layer because all communication happen based on this database.

```
typedef struct list
{
        int client_sid;
        uint16_t  device_id;
        char client_ip[ MAXSIZE ];
        struct list *next;
} DeviceList;
```

Name	Type	Description
client_sid	Int	It is used to store number of client socket descriptor
device_id	uint16_t	Network will provide unique Device ID from available pool of IDs to every new incoming connection and upper layer only knows device id and all data communication and management in upper layer is based on this ID
Client_ip[MAXSIZE]	char *	This variable is used to store IP address of client
Next	Struct list*	It points to next element in list.

Table 3.3 Structure DeviceList

3.1.3.2 APIs

API overview

API	Description
int NWK_Init(int type)	This function is used to initialize network layer it will create required thread and set various global parameter for future reference.
int APDE_NWK(NwkMsg_t * msg)	This function is used by APS to pass packet to network layer.
void NW_RegisterSapHandlers(void (*MCPS_ZCL_SapHandler)(NwkMsg_t))	This function is used to call back when some message is received by receiving thread of network layer.

Table 3.4 APIs in NWK layer

APDE_NWK

This function is used by APS to pass packet to network layer. To pass message APS should provide pointer of filled structure name NWKMsg_t when it call this API. We are using TCP/IP connection and star topology so client don't have to give device ID when it want to send to server / coordinator. While server side network should take care to send message to right client by searching in its list of connected devices.

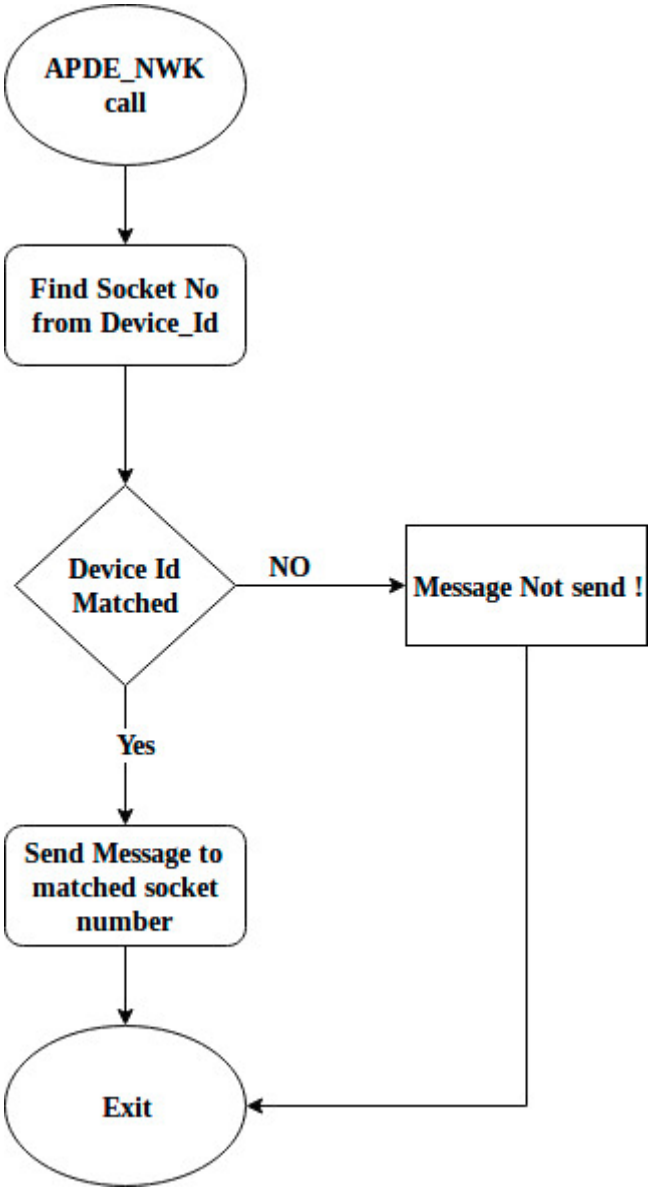

Figure 3.2 APS to NWK call

NWK_Init

This function is used to initialize network layer it will create required thread and set various global parameter for future reference. As parameter we have to give 1 or 2 which represent which type of layer you want to initialize. Here type = 1 is for server and type = 2 is for client. According to type it will act entirely differently. It will first ask socket descriptor number and in case of client IP of server then it will start required thread and initialize global variable.

- **NWK initialization as server**

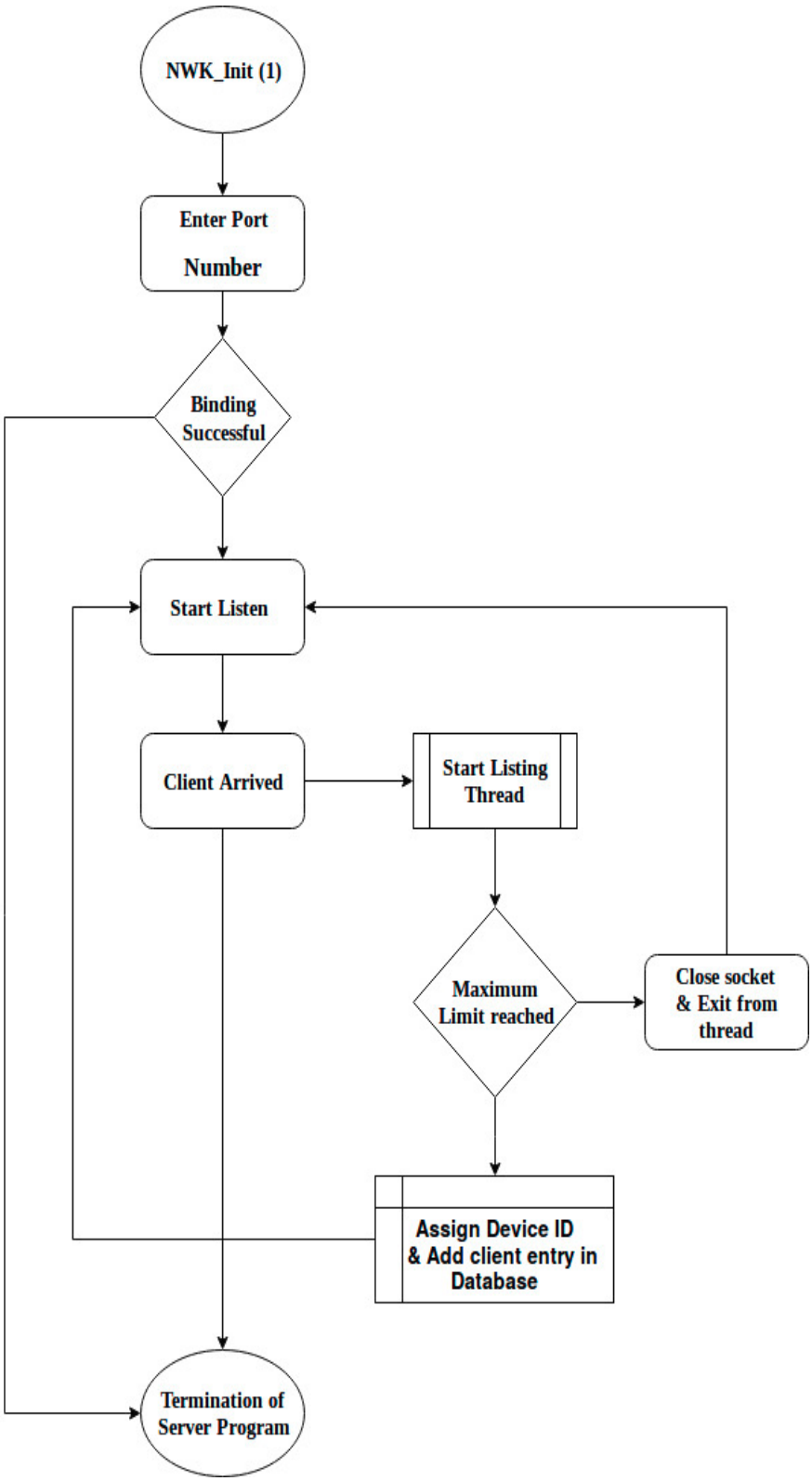

Figure 3.3 NWK init as server

- **NWK initialization as client**

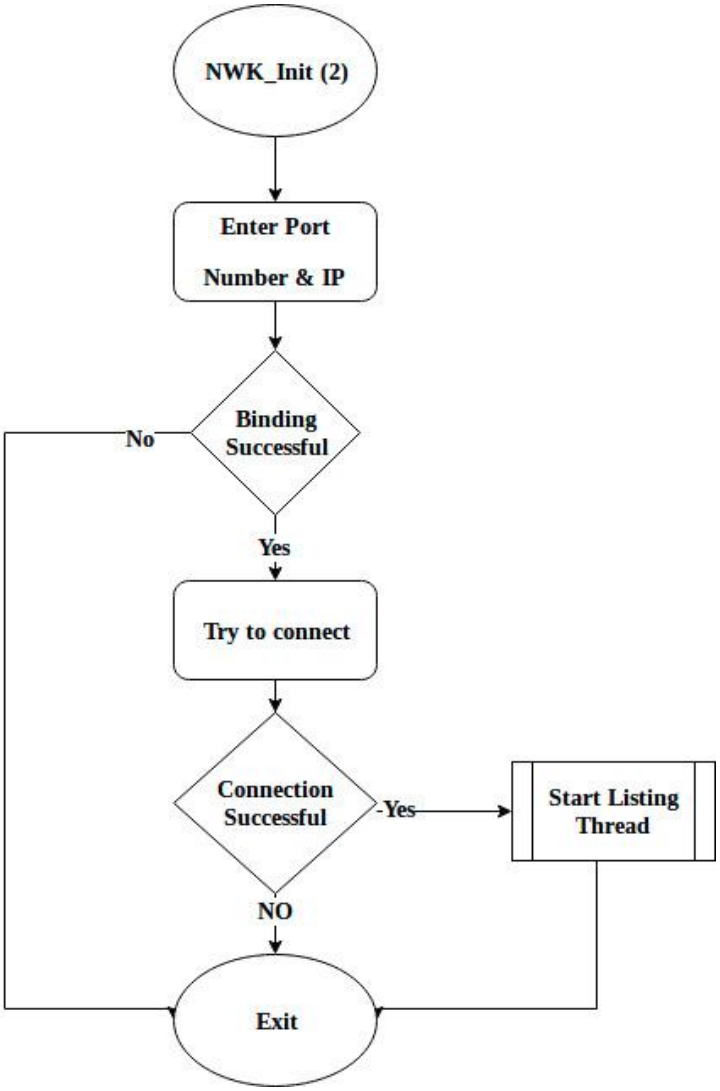

Figure 3.4 NWK init as client

NW_RegisterSapHandlers

This function is used to call back when some message is received by receiving thread of network layer. Again because we are using TCP/IP at client side NWK layer just have to pass received message to APS's callback function where server side NWK have to add device ID for that received message and then pass to APS.

3.2 Application Support Sublayer

3.2.1 Design specification

Application Support Sublayer implemented as per ZigBee Alliance specification though it is not clearly written in specification that what should be next upper layer of APS so as per our design APS will interact with Application and ZDO.

3.2.2 Used Structures

3.2.2.1 Data Request Structure

struct DATA_Request{

DstAddrMode;
DstAddress;
DstEndpoint;
ProfileId;
ClusterId;
SrcEndpoint;
ADSULength;
ADSU[100];
TxOptions;
RadiusCounter
}

Name	Type	Description
DstAddrMode	Integer	The addressing mode for the destination address used in this primitive and of the APDU to be transferred. This parameter can take one of the non-reserved values from the following list: 0x00 = DstAddress and DstEndpoint not Present 0x01 = 16-bit group address for DstAddress; DstEndpoint not present

		0x02 = 16-bit address for DstAddress and DstEndpoint present 0x03 = 64-bit extended address for DstAddress and DstEndpoint present 0x04 – 0xff = reserved
DstAddress	Address	The individual device address or group address of the entity to which the ASDU is being transferred.
DstEndpoint	Integer	This parameter shall be present if, and only if, the DstAddrMode parameter has a value of 0x02 or 0x03 and, if present, shall be either the number of the individual endpoint of the entity to which the ASDU is being transferred or the broadcast endpoint (0xff).
ProfileId	Integer	The identifier of the profile for which this frame is intended.
ClusterId	Integer	The identifier of the object for which this frame is intended.
SrcEndpoint	Integer	The individual endpoint of the entity from which the ASDU is being transferred.
ADSULength	Integer	The number of octets comprising the ASDU to be transferred. The maximum length of an individual APS frame payload is given as NsduLength - apscMinHeaderOverhead.
ADSU	Set of Octets	The set of octets comprising the ASDU to be transferred.
TxOptions	Bitmap	The transmission options for the ASDU to be transferred. These are a bitwise OR of one or more of the following: 0x01 = Security enabled transmission 0x02 = Use NWK key

		0x04 = Acknowledged transmission 0x08 = Fragmentation permitted
Radius	Unsigned Integer	The distance, in hops, that a transmitted frame will be allowed to travel through the network.

Table 3.5 APSDE_DATA_Request Parameters

3.2.2.2 Data Indication Structure

```
Struct DataIndication_t     {
                            DstAddrMode;
                            DstAddress;
                            DstEndpoint;
                            SrcEndpoint;
                            SrcAddress;
                            ProfileId;
                            ClusterId;
                            ADSULength;
                            ADSU[100];
                            Status;
                            LinkQuality;
                            RxTime;
                            }
```

Name	**Type**	**Description**
DstAddrMode	Integer	The addressing mode for the destination address used in this primitive and of the APDU to be transferred. This parameter can take one of the non-reserved values from the following list: 0x00 = DstAddress and DstEndpoint not Present 0x01 = 16-bit group address for DstAddress;

		DstEndpoint not present 0x02 = 16-bit address for DstAddress and DstEndpoint present 0x03 = 64-bit extended address for DstAddress and DstEndpoint present 0x04 – 0xff = reserved
DstAddress	Address	The individual device address or group address of the entity to which the ASDU is being transferred.
DstEndpoint	Integer	This parameter shall be present if, and only if, the DstAddrMode parameter has a value of 0x02 or 0x03 and, if present, shall be either the number of the individual endpoint of the entity to which the ASDU is being transferred or the broadcast endpoint (0xff).
SrcEndpoint	Integer	The individual endpoint of the entity from which the ASDU is being transferred.
SrcAddress	Address	The individual device address of the entity from which the ASDU has been received.
ProfileId	Integer	The identifier of the profile for which this frame is intended.
ClusterId	Integer	The identifier of the object for which this frame is intended.
ADSULength	Integer	The number of octets comprising the ASDU to be transferred. The maximum length of an individual APS frame payload is given as NsduLength - apscMinHeaderOverhead.
ADSU	Set of octets	The set of octets comprising the ASDU to be transferred.

Status	Enumeration	The status of the incoming frame Processing.it will contain any value from following list: SUCCESS, DEFRAG_UNSUPPORTED, DEFRAG_DEFERRED.
SecurityStatus	Enumeration	UNSECURED if the ASDU was received without any security. SECURED_NWK_KEY if the received ASDU was secured with the NWK key. SECURED_LINK_KEY if the ASDU was secured with a link key.
LinkQuality	Integer	The link quality indication delivered by the NLDE.not used in our case.
RxTime	Integer	A time indication for the received packet based on the local clock, as provided by the NWK layer.

Table 3.6 APSDE_DATA_indication Parameters

3.2.2.3 APS Frame Format Structure

```
struct APS_Frame    {
                    ControlField;
                    u8DestinationEndPoint;
                    u16GroupAddress;
                    u16ClusterIdentifier;
                    u16ProfileIdentifier;
                    u8SourceEndPoint;
                    u8APSCounter;
                    Packet[100];
                    }
```

Name	Type	Description
ControlField	Bitmap	ControlField contains information defining the frame type, addressing fields, and other control flags.
u8DestinationEndPoint	Integer	This parameter shall be present if, and only if, the DstAddrMode parameter has a value of 0x02 or 0x03 and, if present, shall be either the number of the individual endpoint of the entity to which the ASDU is being transferred or the broadcast endpoint (0xff).
u16GroupAddress	Address	If frame contains a group address field, the frame will be delivered to all endpoints for which the group table in the device contains an association between that endpoint and the group identified by the contents of the group address field.
u16ClusterIdentifier	Integer	The identifier of the object for which this frame is intended.
u16ProfileIdentifier	Integer	The identifier of the profile for which this frame is intended.
u8SourceEndPoint	Integer	The individual endpoint of the entity from which the ASDU is being transferred.
u8APSCounter	Integer	This field is eight bits in length and is used to prevent the reception of duplicate frames. This value shall be incremented by one for each new transmission.
Packet	Integer	field has a variable length and contains information specific to individual frame types.

Table 3.7 APS_Frame Parameters

3.2.3 APIs

3.2.3.1 Public APIs

There are five main functions which are exposed to other layer. This functions are handle all data request and all management request.

API	Description
void APS_init();	APS initialization. function will register the callback function of network layer.
void APSDE_DATA_Request(struct DATA_Request *req);	Function will use by application layer and ZDO for data request.
void MCPS_ZCL_SapHandler(NwkMsg_t msg);	Callback function of APS.network layer will use this function when it will receive some packet.
void DeviceRemove(uint16_t device_ID);	callback Function of APS.network layer will call this function when any device will disconnect.
void APS_RegisterSapHandlers(void (*APME_ZDO_SapHandler)(DataIndication_t MsgDecode), void (*APDE_ZCL_SapHandler)(DataIndication_t MsgDecode), void (*DeviceRemove)(uint16_t ID));	APS callback function register handler. This function will be used by upper layer to register callback function of it.

Table 3.8 APS public APIs

3.2.3.2 Private APIs

API	Description
void *frame_decoder(void *packet);	Thread handler.thread will be created when APS receive frame from the network..
void APSDE_DATA_Indication(frame_t MsgDecode, int len,uint16_t devId);	Send data indication to the application layer or ZDO

Table 3.9 APS private APIs

3.2.4 Data Flow Diagram:

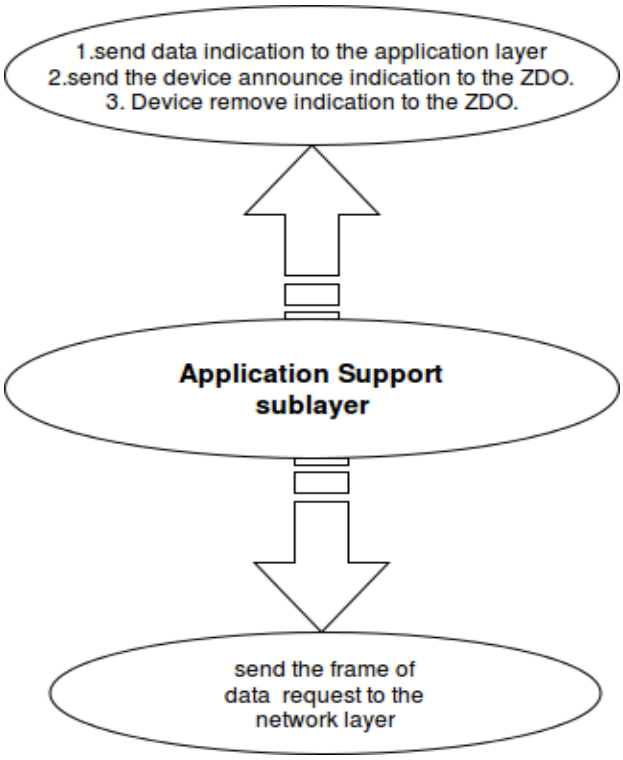

Figure 3.5 APS data flow diagram

3.2.5 APIs Detailed Description:

3.2.5.1 APS_init

This function is used for initialization of application layer.it is register the callback function of APS.when network layer receive packet from peer entity, it will handover that packet to the APS by using this callback function.

3.2.5.2 APSDE_DATA_Request

This function is used by the application layer or ZDO for the data request. When application wants some data from the other device it will send the request to the APS by using this function. This function will make the frame according to the data request and then dump that frame into the network packet. After making packet it will handover that packet to the network layer

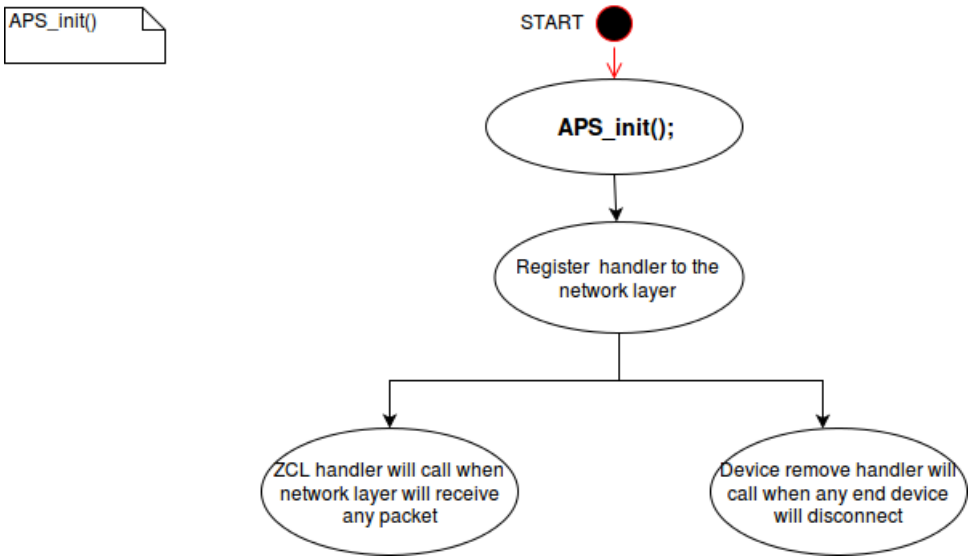

Figure 3.6 Data flow diagram of APS_init

3.2.5.3 MCPS_ZCL_SapHandler

Callback function of the APS. When network receive any packet it will call this function for handover the packet to the APS. This function will take the frame and create one thread for decode that frame. Thread will decode the frame and call the function APSDE_DATA_Indication.

APSDE_DATA_Indication: This function will fill the structure of data indication according to the frame received. And callback the application layer or ZDO according to the Destination End point.

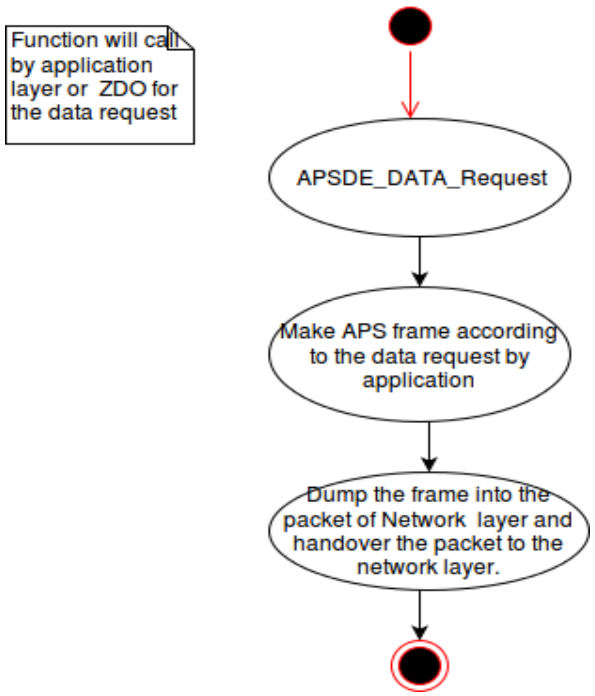

Figure 3.7 Data Flow diagram of APS data request function

3.2.5.4 Device Remove

Callback function of APS.network layer will callback to the APS when any device will disconnect from the network.

3.2.5.5 APS_RegisterSapHandlers

This function register the function pointer of the callback function of ZDO and APP.

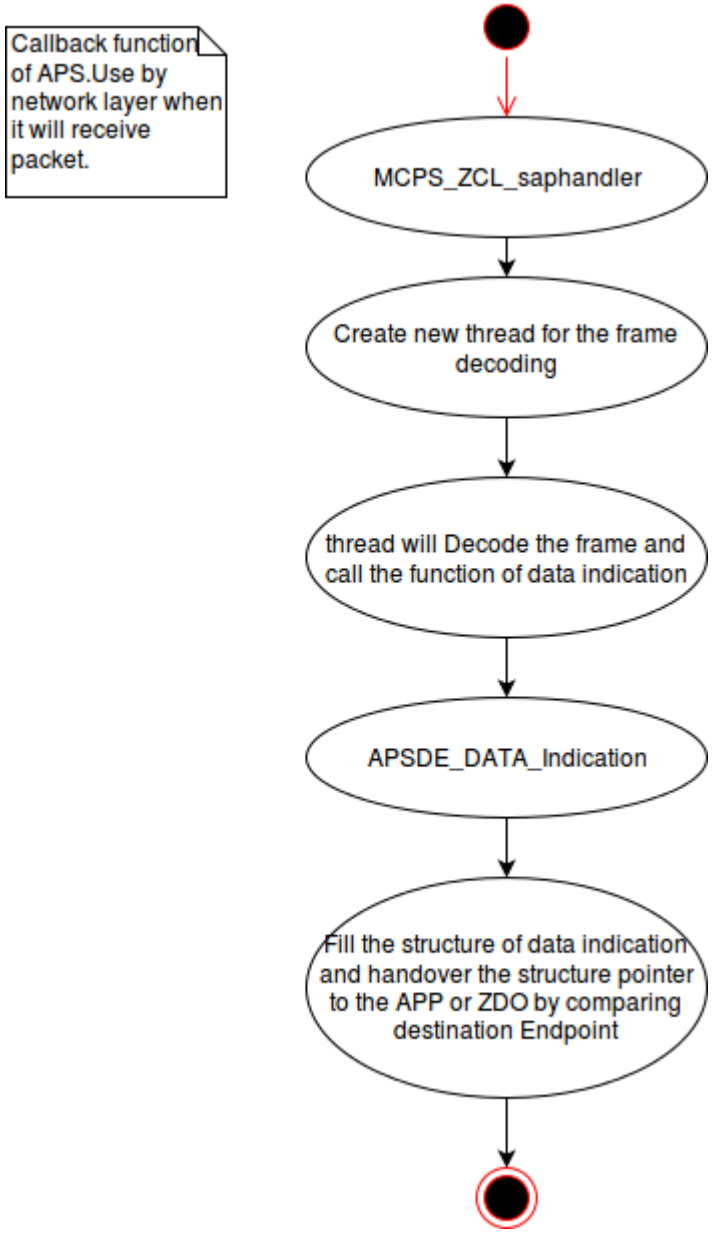

Figure 3.8 Data flow diagram of APS handler

3.3 ZigBee Cluster library

3.3.1 Structures

3.3.1.1 Struct command

```
struct command {
        clusterId;
        clusterSpecific;          // 1 for yes, 0 for not
        commandId;
        attributeId;
        data;
        dataLength;
    };
```

Any communication between app and zcl is done on the basis of this structure.

Name	Type	Description
clusterId	uint16_t	This field is of 16 bits in length and it holds the clusterId of the cluster on which the command will act.
clusterSpecific	bool	It indicates that the command is specific to a given cluster or a general command.
commandId	uint8_t	This field is of 8 bits in length and it holds the id of the command which is to be performed.
attributeId	uint16_t	This field is of 16 bits in length and contains the id of the attribute which is to be kept under consideration while performing the command specified by *commandId*.
Data	void*	This field has variable length and contains the data specific to the packet, when a write request is

		made it will hold the data to be written when read request response is being made it will hold the data which was stored into the attribute specified by the attribute.
dataLength	uint16_t	As data field has variable length its length is specified in this field which will aid the ZCL decoder of other device to decode the packet. dataLength is of 16 bit in length .

Table 3.10 Struct command

Note : At any point if there is no use of a particular field then it must be kept to NULL.
For eg, if read request is being made than the *data* and *dataLength* field must be kept NULL.

3.3.2 Tree and Flow diagram

3.3.2.2 Overview of Packet formatting

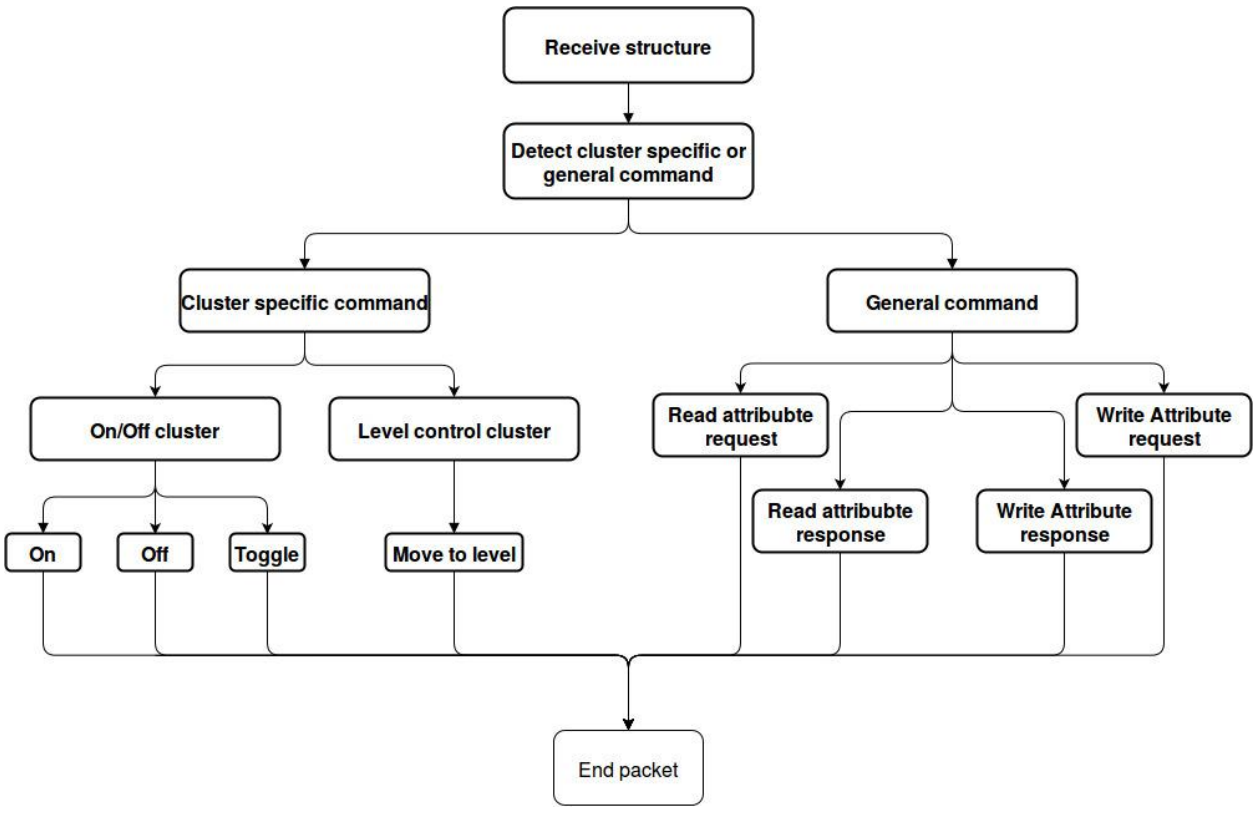

Figure 3.9 Overview of Packet formatting

Explanation

When a need for formation of ZCL packet comes in, than ZCL encoder is called by the obligated application and the details that are required to be kept in the packet are transferred in a structure and passed on to the ZCL encoder function. On acquisition of the structure, ZCL encoder will follow the above tree to narrow down to what kind of packet has to be made and after getting to the end node it will make the packet . Along with the structure an address of a void pointer is passed to the ZCL encoder, so that it can reply the address where packet is stored.

3.3.2.3 Overview of Packet decoding

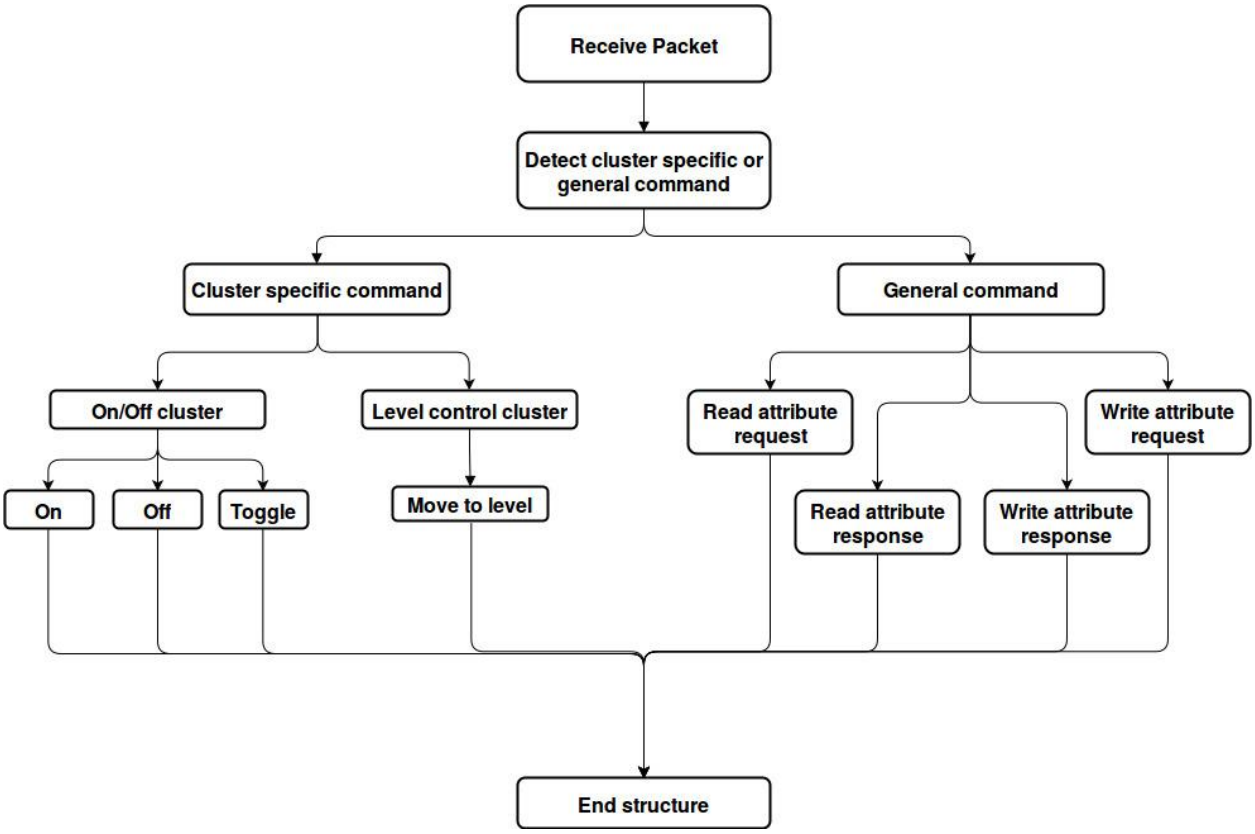

Figure 3.10 Overview of Packet decoding

Explanation

When a ZCL packet is received by an application it calls ZCL decoder to decode the packet. ZCL decoder tree diagram is shown in the above tree image. ZCL decoder decides whether the packet has cluster specific command or a general command by the clusterSpecific bit in the frame frame control byte of the frame header. Once the information about the type of packet is received it moves ahead accordingly and detects the exact command which was received in the packet, along with decoding

the packet ZCL decoder starts filling up command structure which will be used by application to perform the job which was specified in the zcl packet.

3.3.2.4 Activity diagram of ZCL Encoder

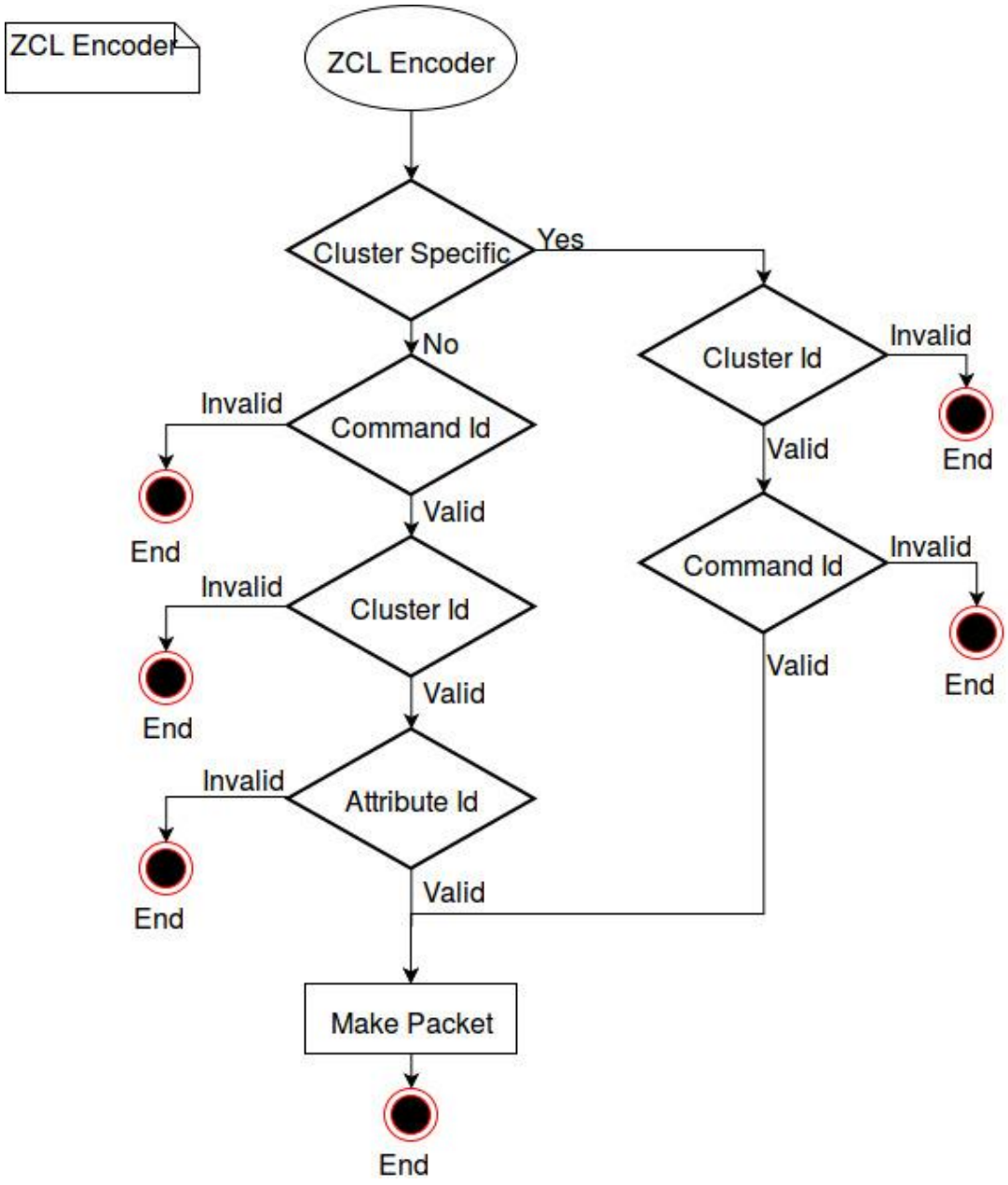

Figure 3.11 Activity diagram of ZCL Encoder

Explanation

On receiving the structure first thing checked is whether the command is cluster specific or not if not than it will move on to make a general packet else it will go for cluster specific command. If it is cluster specific command it will check the cluster id for verification and after that it will check command id if all these inputs are correct then it than packet will be formed. If any of these inputs

are wrong it will return UNSUCCESSFUL. If it is not a cluster specific command then command id will checked followed by that cluster id and attribute id are validated if everything is valid than only packet will be formed.

3.3.4 Packet Formation

The frame header has first byte as frame control, from the information given by the APP in the structure the frame type bits will be set accordingly (which is lowest 2 bit of the 8bit frame control). After that manufacturer code which is 3rd bit, in our case as the simulation is done on a PCs there is no manufacturer code and hence this bit is set to 0. Direction which is the 4th bit is set according to the direction of the packet(which is hardwired as we are emulating star network and thus a device can only be co-ordinator or end device and not the both). For trial purpose the 5th bit is set to one which disables the default response. After Frame control byte comes transmission number which specifies the part of which frame it is and for our case we are working on one attribute at a time it won't exceed max packet length and thus it is set to zero. After transmission number comes the commandId, this 8 bits are field using commandId given to zcl in the structure.

Now according to the fields *clusterSpecific*, *clusterId* and *commandId* zcl will know that data field holds some data which is to incorporated into the packet and accordingly it will keep that data into the packet. After this the end packet will be formed and will be handed over to the caller.

3.4 ZigBee Device Object

As actually using ZigBee device is not used its impact on ZDO implementation can't be ignored though design is kept in such way that it can easily transferred as per specification.

3.4.1 Design overview

3.4.2 Structures

```
Struct device {
        deviceId;
        clusterId;
        deviceType;
        next;
}
```

Variables used In structure device		
Name	Datatype	Description
deviceId	uint16_t	Device Id which is provided by network layer.
clusterId	uint16_t	Cluster Id that is supported on that device.
deviceType	Char *	Device type based on its ClusterId
Next	struct device *	Points to next element of the device table

Table 3.11 Struct device

```
Struct annce {
        devId;
        iEEEAdd;
        capability;
}
```

Variables used In structure annce		
Name	Datatype	Description
devId	uint16_t	Network address for the Local Device.
iEEEAdd	uint64_t	IEEE address for the Local Device.
Capability	uint8_t	Capability of the local device.

Table 3.12 Struct annce

```
struct dataZDA {
        u16Type;
        u16Length;
        payLoad;
}
```

Variables used In structure dataZDA		
Name	Datatype	Description
u16Type;	uint16_t	Message Type of command.
u16Length	uint16_t	Length of payLoad
payLoad	struct annce	Structure to transmit as payload of announce frame

Table 3.13 Struct dataZDA

```
Struct  _SimpleDescriptor{
        u16TargetAddress;
        u8Endpoint;
}
```

Variables used In structure _SimpleDescriptor		
Name	Datatype	Description
u16TargetAddress	uint16_t	Network Address of local Device.
u8Endpoint	uint8_t	The endpoint on the destination.

Table 3.14 Struct _SimpleDescriptor

```
Struct descriptorReq {
        u16MsgType;
        u16Length;
        payLoad;
}
```

Variables used In structure descriptorReq		
Name	Datatype	Description
u16MsgType;	uint16_t	Message type of descriptor request command
u16Length	uint16_t	Length of payload.
payLoad	struct _SimpleDescriptor	Structure to transmit as payload of request frame

Table 3.15 Struct descriptorReq

```
Struct descRsp {
        shortAddr;
        clusterId;
}
```

Variables used In structure descRsp		
Name	Datatype	Description
shortAddr	uint16_t	Network Address of local Device.
clusterId	uint16_t	Cluster Id supported by that device.

Table 3.16 Struct descRsp

```
struct descResponse {
        u16MsgType;
        u16Length;
        payLoad;
}
```

Variables used In structure descResponse		
Name	Datatype	Description
u16MsgType;	uint16_t	Message type of descriptor response command
u16Length	uint16_t	Length of payload.
payLoad	struct descRsp	Structure to transmit as payload of response frame

Table 3.17 Struct descResponse

3.4.3 APIs

3.4.3.1 Public APIs

API	**Description**
void ZDO_Start(int node);	Provided to end application to decide whether ZDO_Start as coordinator or as end node
void APME_Deviceremove_Saphandler(uint16_t device_ID);	Callback function used by APS layer to inform ZDO about disconnection of any end Node
void APME_ZDO_Saphandler(DataIndication_t IndicationData);	Callback function Used by APS layer whenever packet related to ZDO received.
deviceList *getDevice();	Provided to end Application so that it can have updated device list maintained by ZDO.

Table 3.18 ZDO public APIs

3.4.3.2 Private APIs

API	Description
void *cordHandler(void *arg);	Thread routine handles coordinator functionalities.
void *clientThread(void *arg);	Thread routine handles end node functionalities.
void deviceAnnce();	Announce the device. Send announce packet using APS function.
void simpleDescReq(uint16_t add);	On reception of deviceAnnce command from end node. coordinator will Form packet for simple descriptor request.
void simpleDescRsp(uint16_t addr);	On reception of SimpleDescReq command from ZigBee coordinator. End node will form packet for simple descriptor response.
void maintainTable(uint16_t shortAddr, uint16_t clusterId);	On Reception of SimpleDescRsp this function will Update table of connected device if that device can be served by Coordinator.
void removeDevice(uint16_t remId);	Whenever APS use callback to inform ZDO that device is disconnected. This function will remove that disconnected device from table maintained by ZDO.

Table 3.19 ZDO private APIs

APIs detailed Description

- **void ZDO_Start(int node) :** This API will initialize ZDO as coordinator or else end node. If integer node has value equal to 1 then ZDO will be initialized as Coordinator and if integer node has value equal to 2 then ZDO will be initialized as end node device.
- **void APME_Deviceremove_Saphandler(uint16_t device_ID) :** This API will be used to

remove entry of device from device table by whenever any device get disconnected from the network.flow diagram mention in figure 3.12.

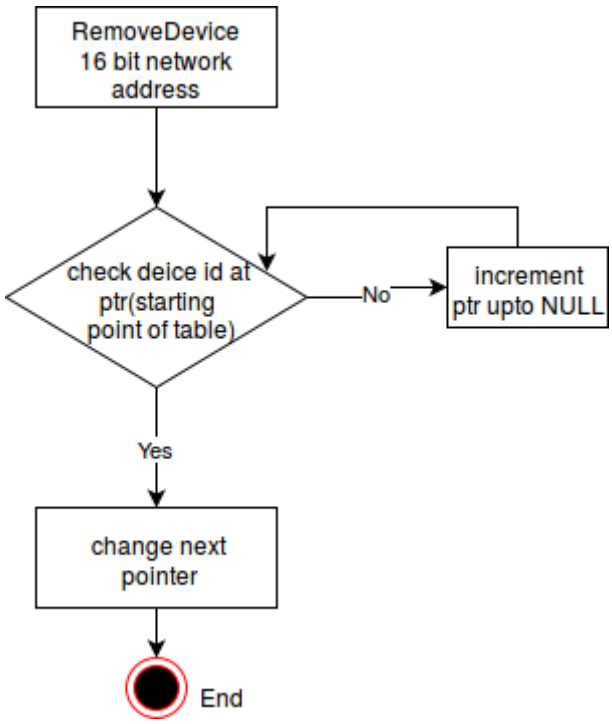

Figure 3.12 Flow diagram of APME_Deviceremove_Saphandler

- **void APME_ZDO_Saphandler(DataIndication_t IndicationData) :** This API will provide structure that is provided by APS layer. and decode packet that is stored in Indication data ADSU.
- Based on type of message further step will be taken. Flow diagram mention in figure 4.
- **deviceList *getDevice() :** This API will provide starting point of device table to end application so it can check available devices. And controlling option for that device.
- **void *cordHandler(void *arg) :** Thread routine for coordinator ZDO.
- **void *clientThread(void *arg) :** Thread routine for client(end node) from which client can send announce request.
- **void deviceAnnce() :** This API will form packet to announce local device in network.packet will be send using function APSDE_DATA_Request provided by APS. And on receiving this command frame coordinator will form request for simple descriptor.
- **void simpleDescReq(uint16_t add) :** on reception of device announce command coordinator will form packet for simple descriptor request using this API. packet will be send using function APSDE_DATA_Request provided by APS. On receiving this request end node will

send simple descriptor response to coordinator.

- **void simpleDescRsp(uint16_t addr) :** On reception of simple descriptor request from coordinator. End node device will form simple descriptor response packet using this API. packet will be send using function APSDE_DATA_Request provided by APS. On receiving this command coordinator will check cluster Id from the packet and if that device can be supported by coordinator then it will be added in device table by calling maintainTable function:
- **void maintainTable(uint16_t shortAddr, uint16_t clusterId) :** This function will manage device table. Device table will contain information in form of device Id vs cluster Id vs device name. Device name will be provided based on cluster id.

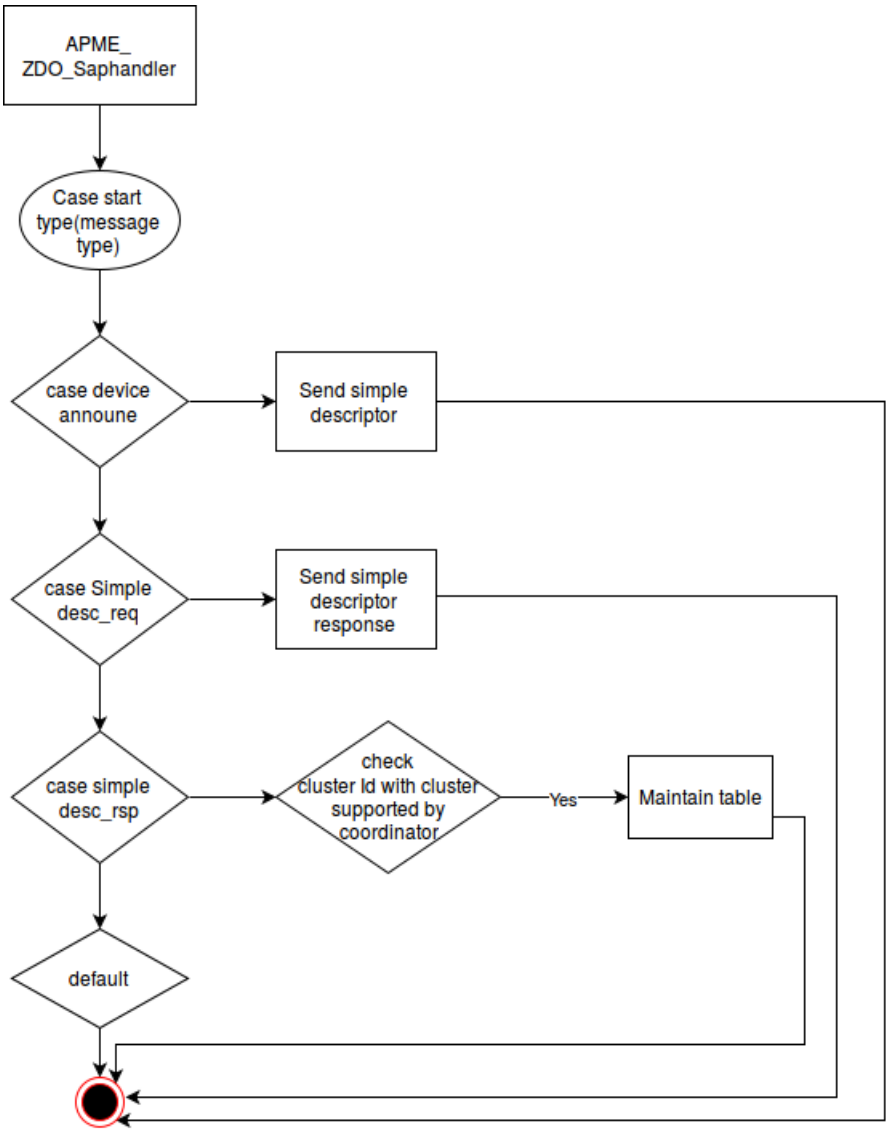

Figure 3.13 Flow diagram of APME_ZDO_Saphandler

3.5 End User Application

3.5.1 Coordinator's End User Application

Application layer at ZigBee coordinator side should have functionality to control endpoints that are connected in ZigBee network. To control various device it should support following APIs:

- **displayDeviceList(D) :** Display device or endpoint that are connected in network. This API will read link list and display connected device with their device ID.
- **operateDevice(O) :** This API should ask user which device user wants to operate or for which device user wants to check status. And based on user selected device. It should be able call other API.
- **exit(E) :** This API should be able to close ZigBee Coordinator.

On any invalid selection error message will be displayed. And user will be ask to enter their choice again. Any type of invalid input won't make coordinator disconnection with end device.

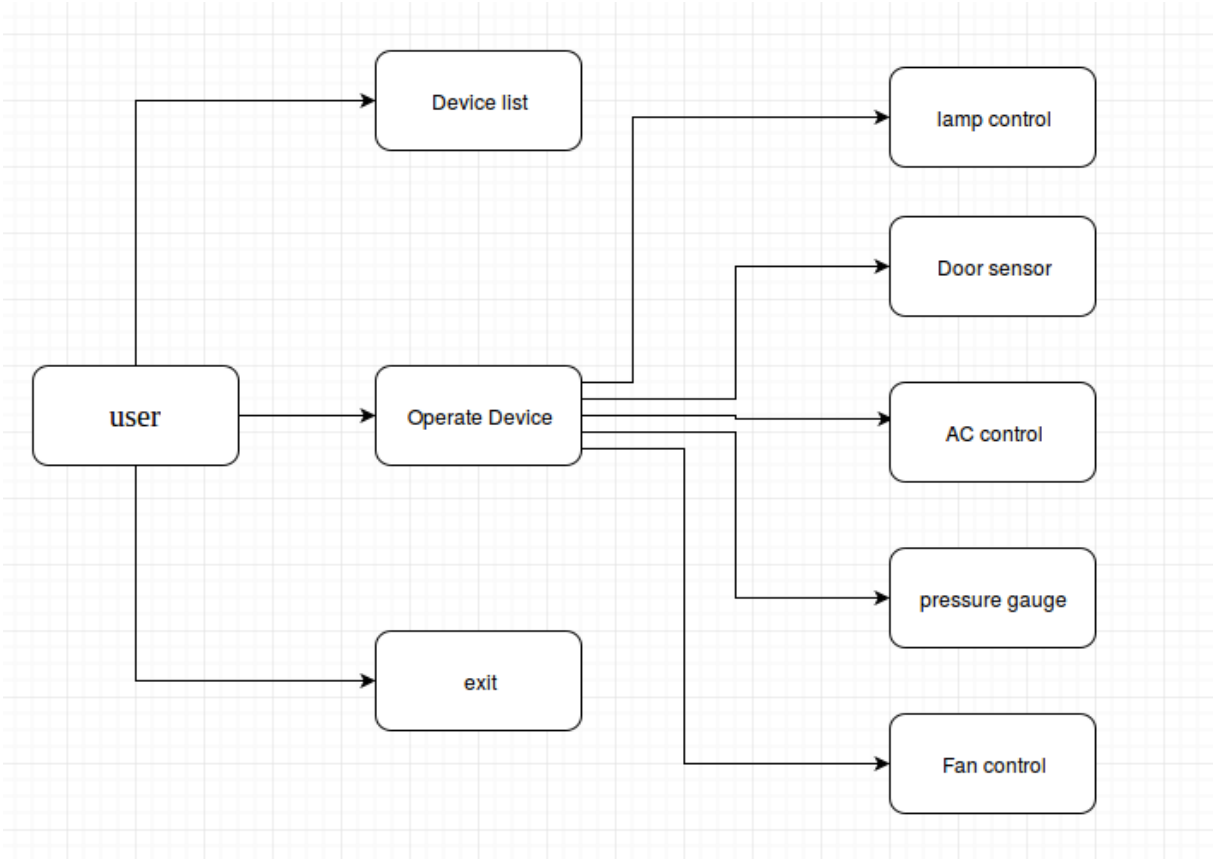

Figure 3.14 coordinator application layer overview

Lamp control:

This API should provide options to operate lamp and also to check status of the lamp.

Operation that should be supported :

- ON(O)/OFF(E) : Command end device to on or off the bulb.
- BRIGHTNESS_CONTROL(B) : Change illuminance level
- STATUS(C) : Check whether the bulb is on or not and if on then measured value of illuminance should be available at coordinator end.
- EXIT(Q) : This command should make user exit from lamp operation and once again option for display, operate and exit should appear.

3.5.2 End Device Application

Lamp App

This application will take commands from coordinator. As per the command end application will respond to coordinator. To operate Lamp it support following commands :

- ON/OFF : Command to turn on/off Lamp
- STATUS : Command to check status of Lamp
- BRIGHT : Command to set brightness level of Lamp
- QUIT : Command to disconnect Lamp from network

4. Testing

Testing was successfully done for each individual layers on LAN connected PC with IPv4 socket. As we are five people team, work is distributed by each layer and the major challenge was to integrate all layer and make user friendly end application. Integration was done successfully but NWK , APS, ZDO and ZCL comes after end user application it works behind the screen and at first glance user can't get feel of those layer. Major goal of this project is not to make IoT end user application so print statements added in each layer only for debug purpose. Actually end user application does not have this type of debugging statements. Process in each layer is printed with different color so it can be understand easily.

4.1 Simulation on PC

Whole project is divided in two application, first is server-coordinator and second is client-end device. Following images represent various test cases of both applications. For demo purpose both application where ran on single PC on internal network. IP address of internal network is 127.0.0.1.

Figure 4.1 Starting Server with Port Number 8888

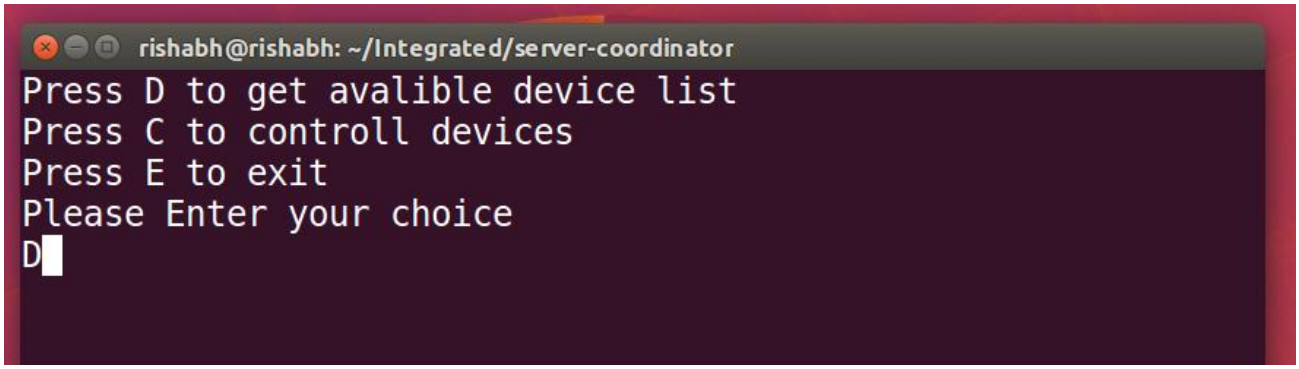

Figure 4.2 Server Main Menu

Figure 4.3 Scenario When no Device is Connected

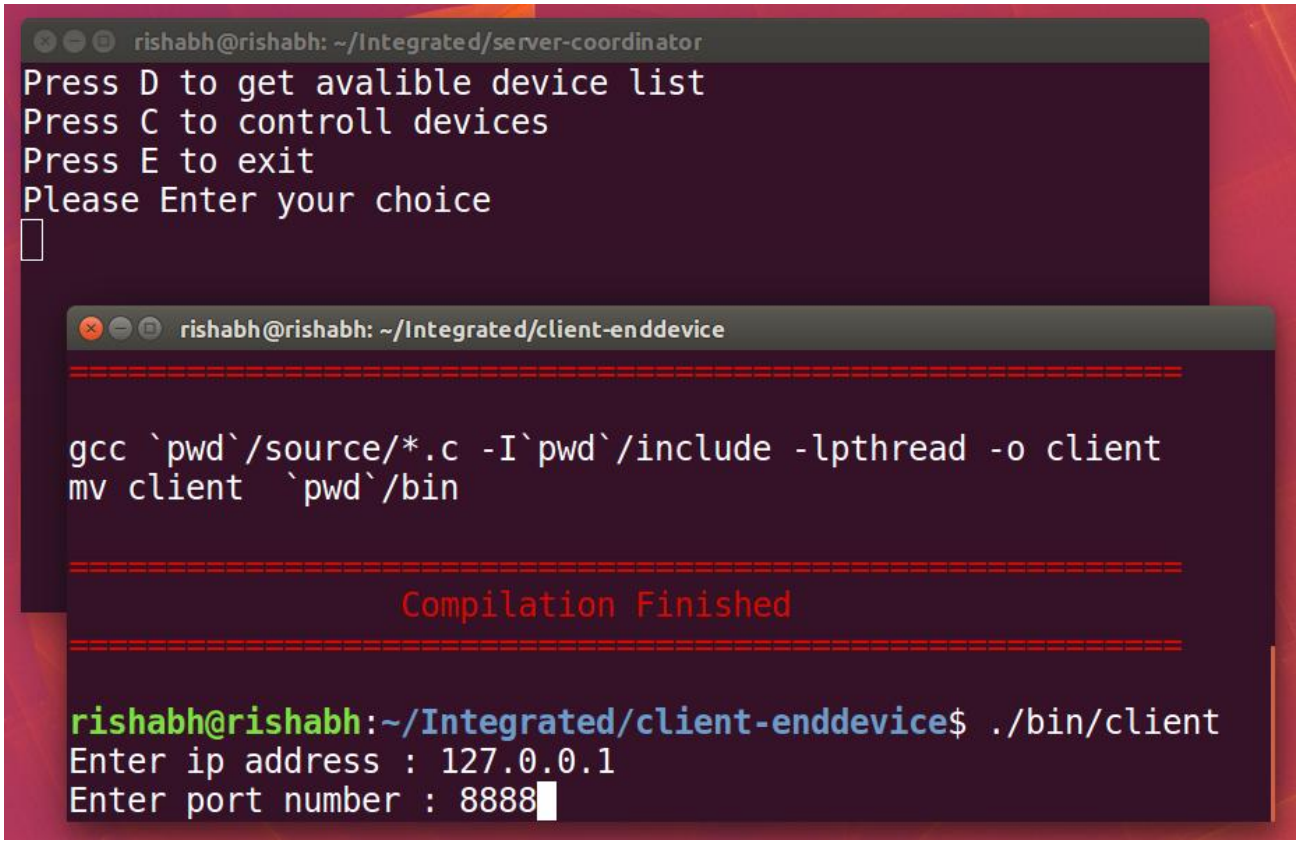

Figure 4.4 Starting Client

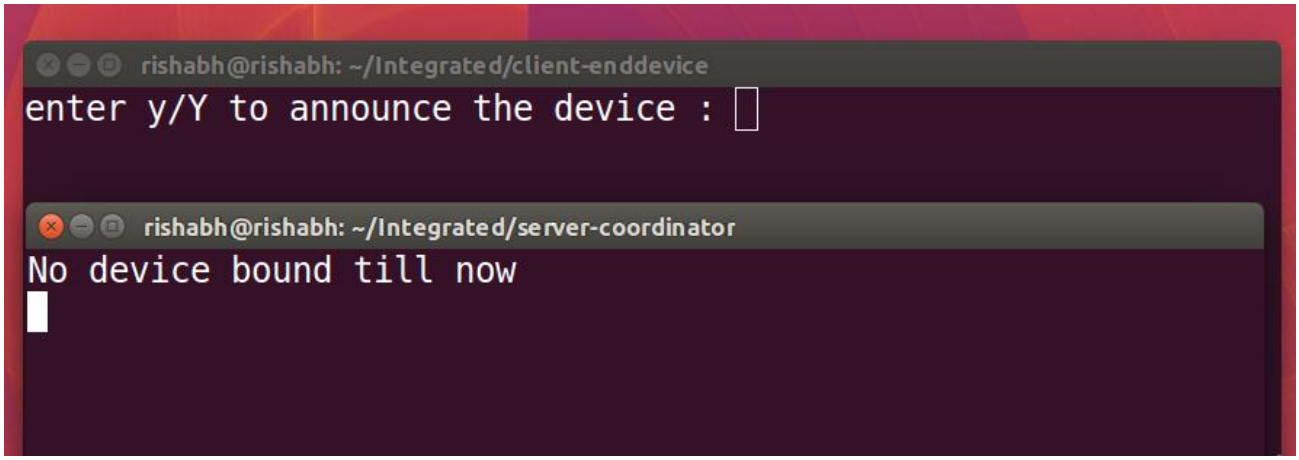

Figure 4.5 Client is connected but device announce remains

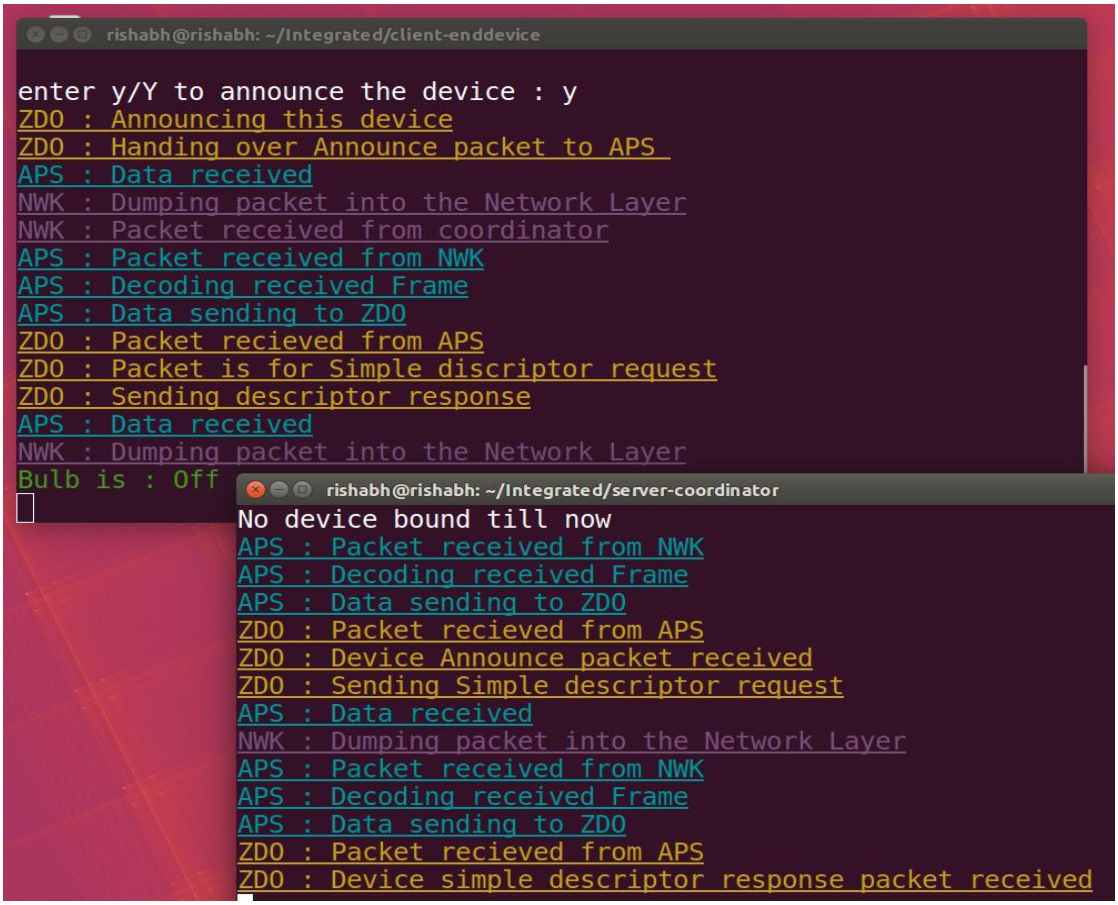

Figure 4.6 Client Announced its identity, actions of respected layers

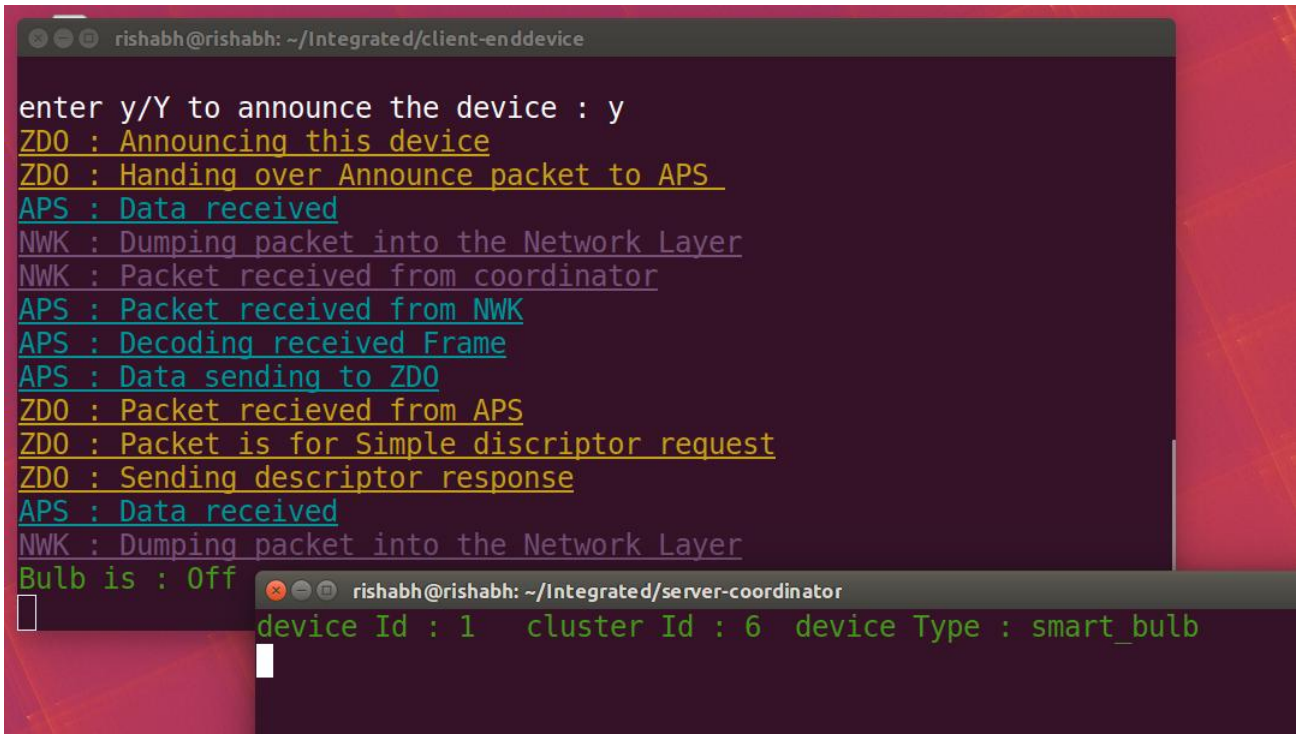

Figure 4.7 List in option D at server side

Figure 4.8 List in option C at server side

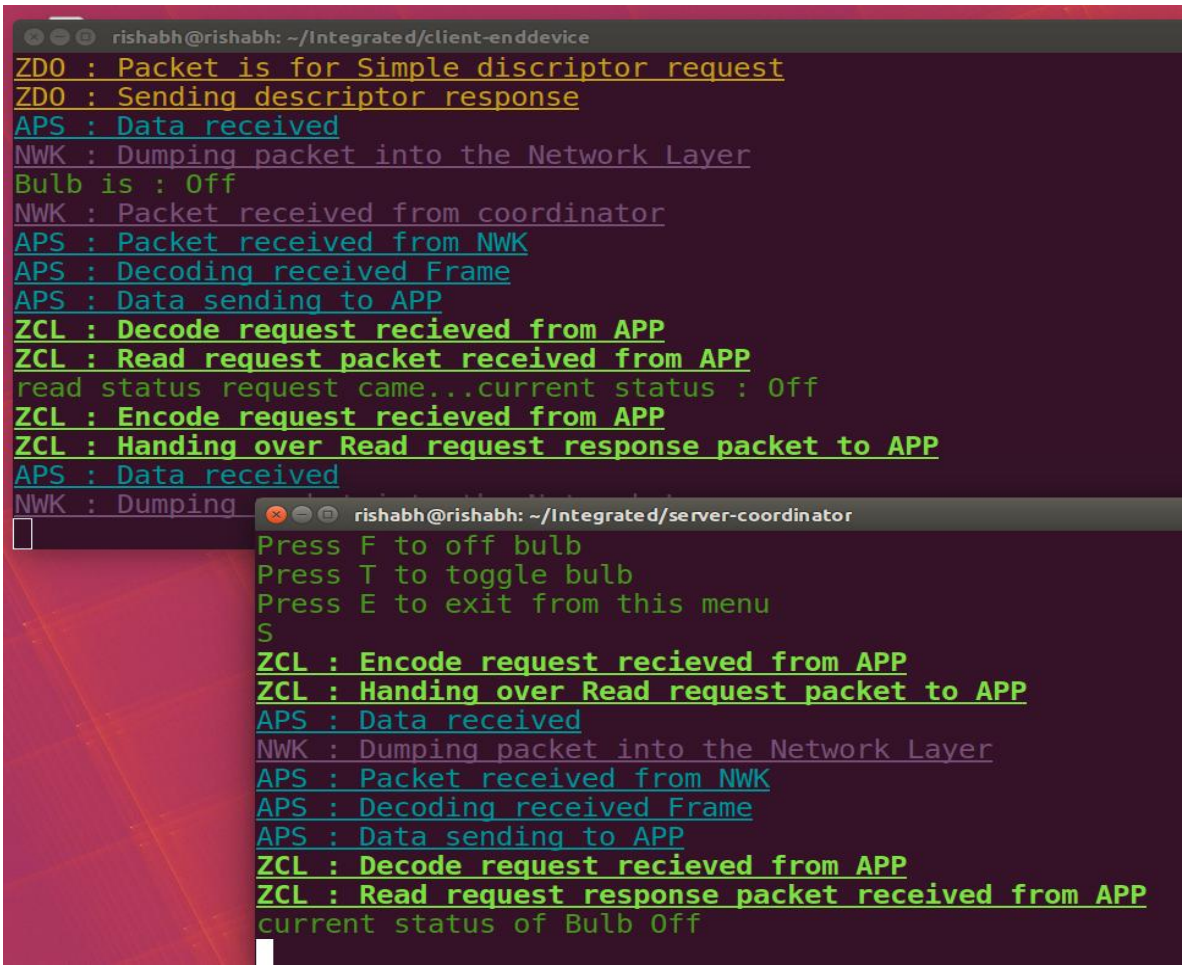

Figure 4.9 Initial status read by server

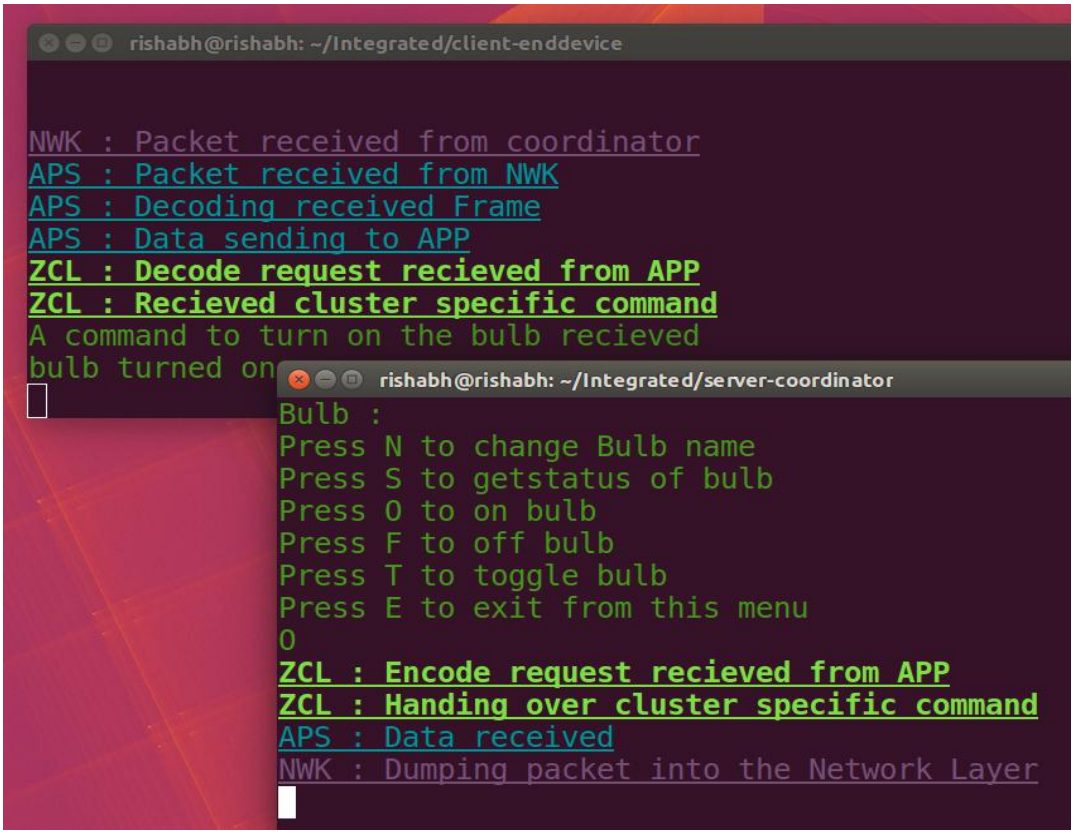

Figure 4.10 Server side command to on the bulb was pressed, client turned on bulb

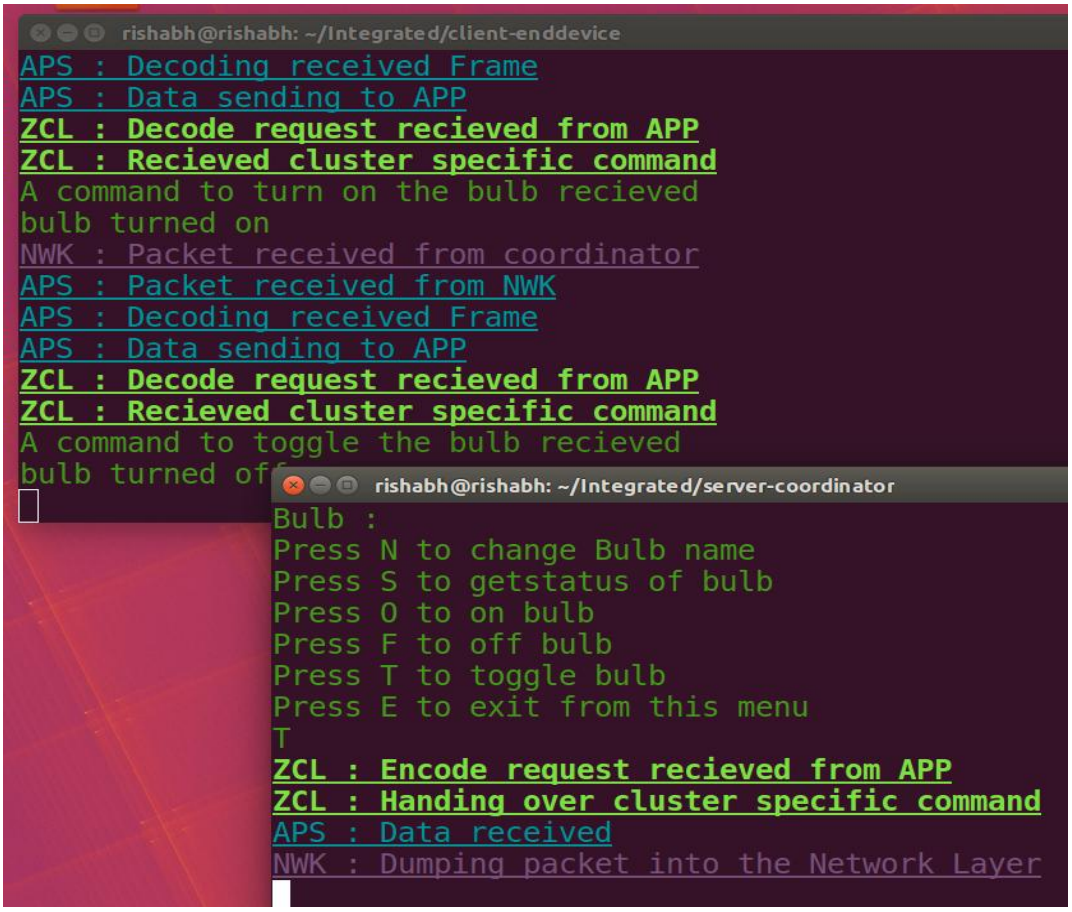

Figure 4.11 Server side command to toggle the bulb was pressed, client toggled the bulb

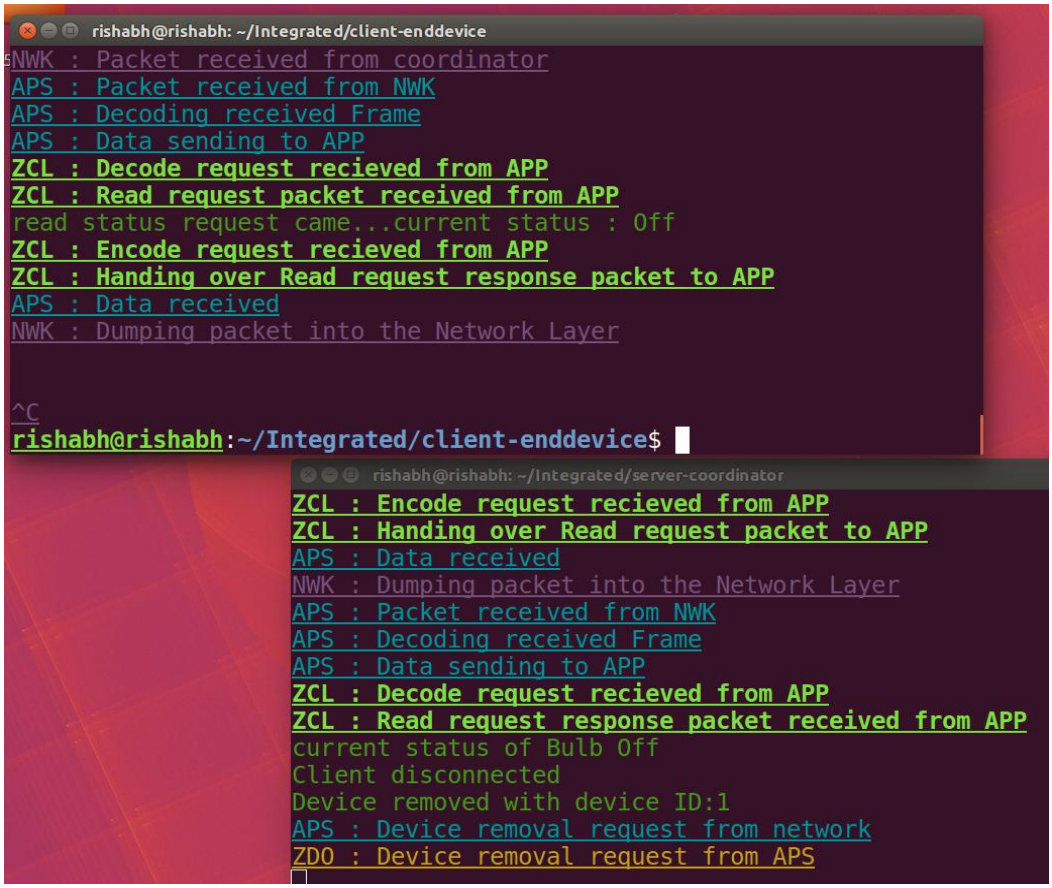

Figure 4.12 Server side command to off the bulb was pressed, client turned on bulb

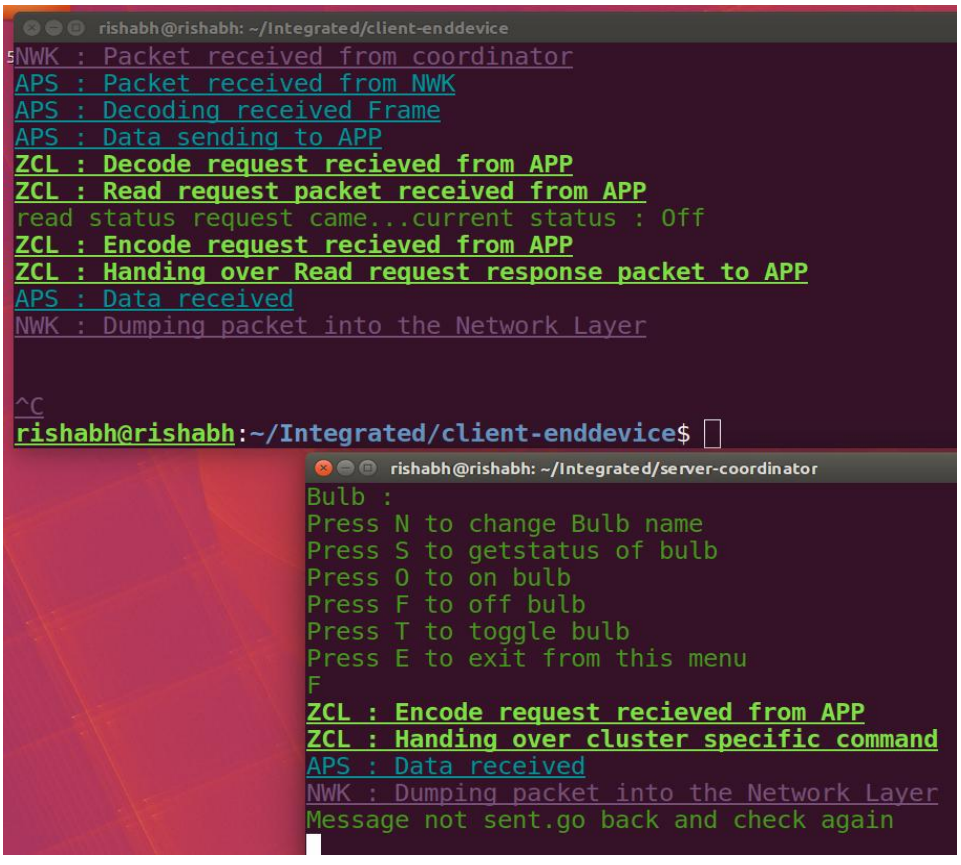

Figure 4.13 Client disconnected

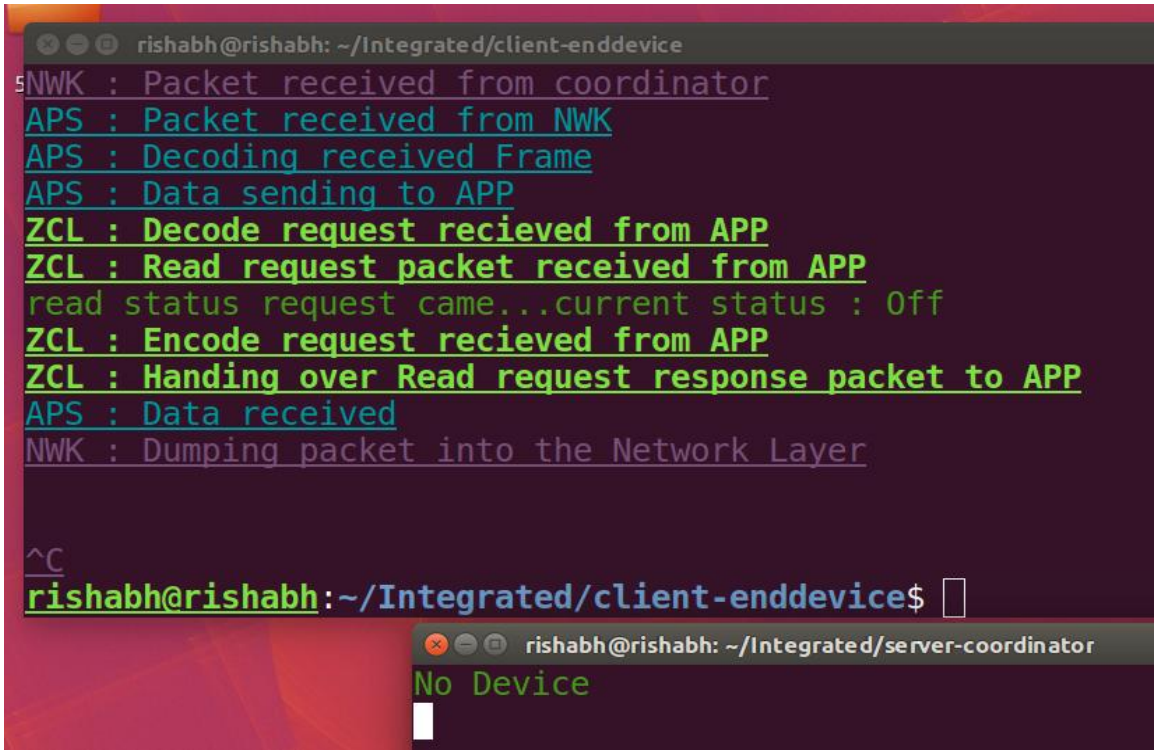

Figure 4.14 Updated device list

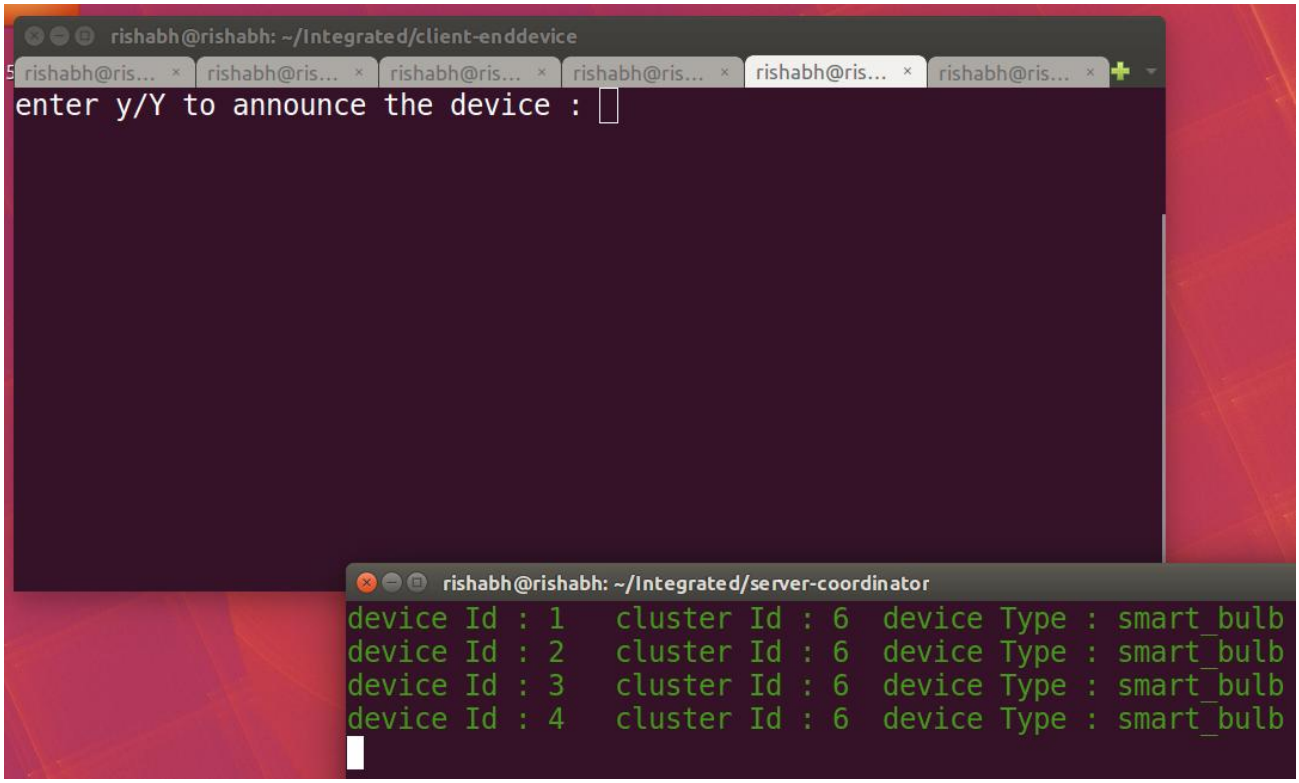

Figure 4.15 Five client connected, Four announced its ID

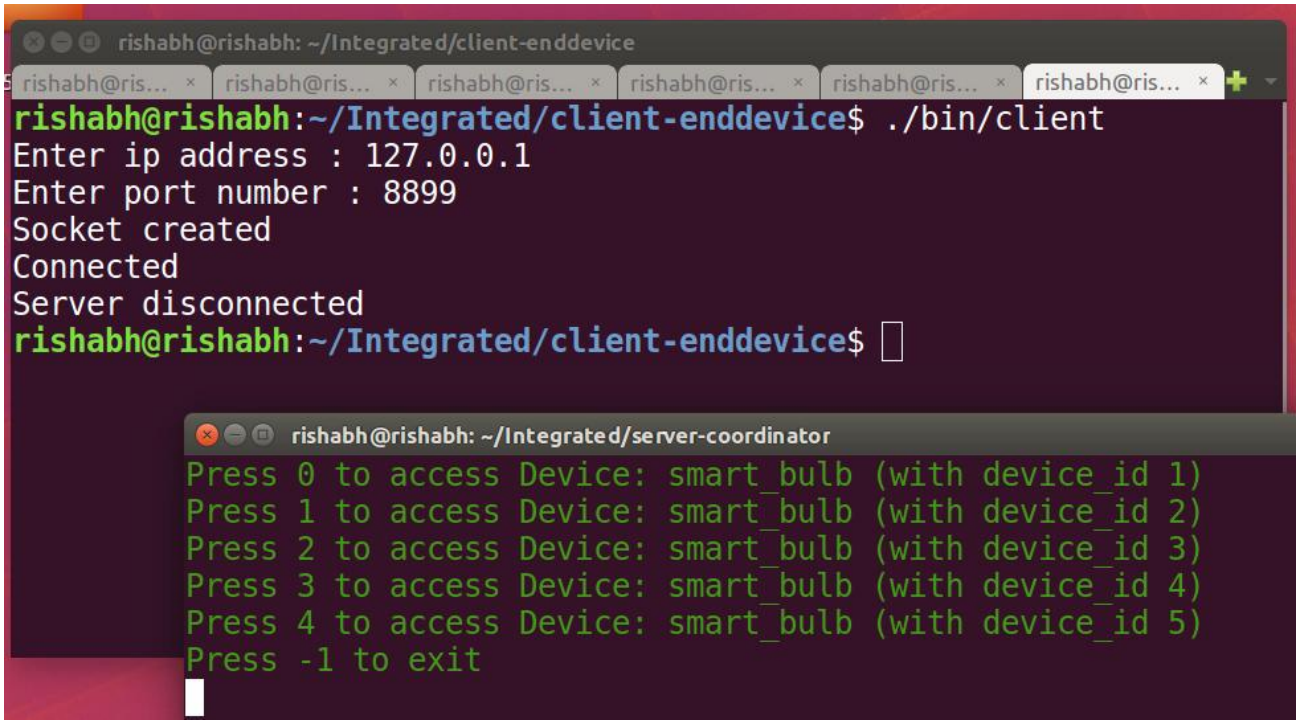

Figure 4.16 Sixth client trying to connect

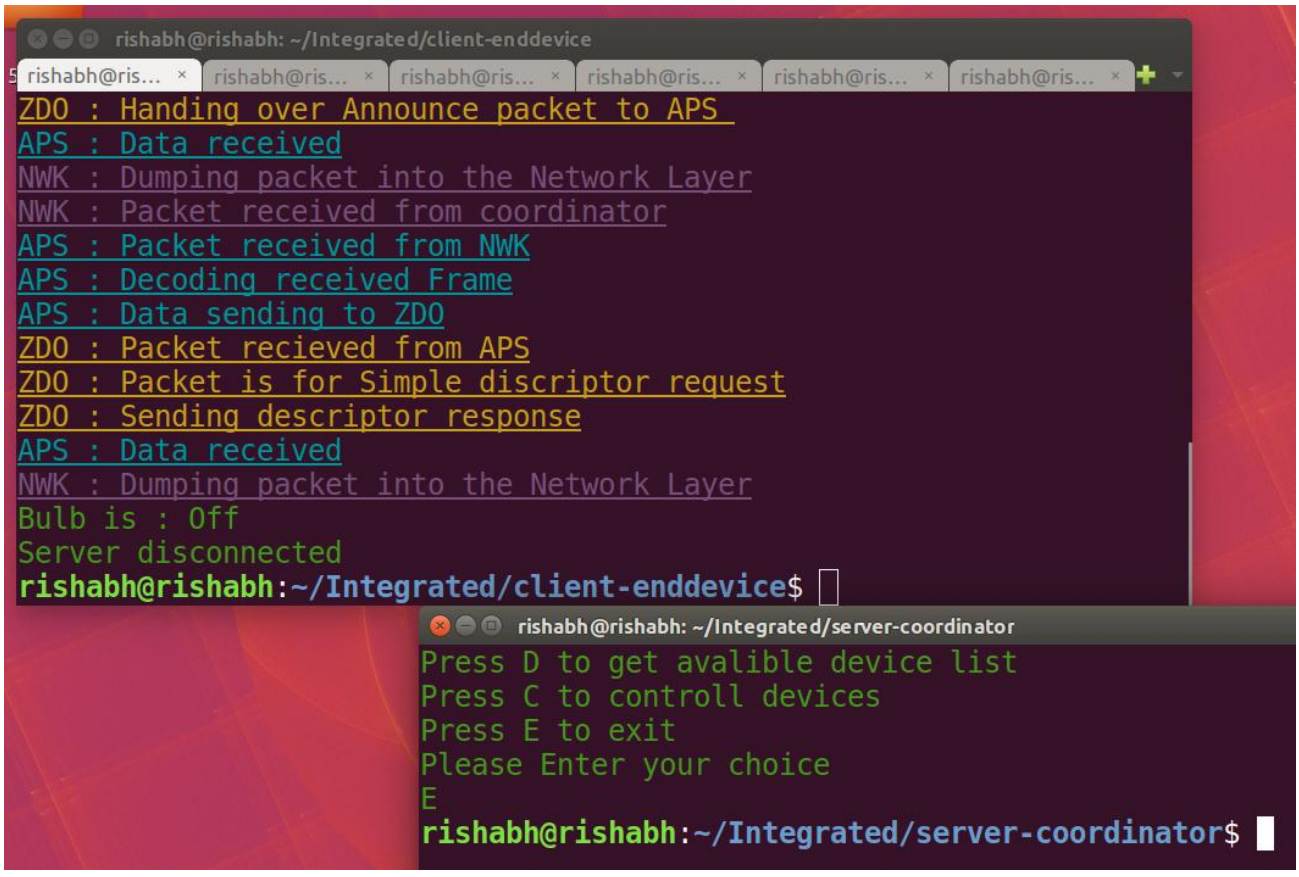

Figure 4.17 Operating 5 device from one coordinator

4.2 Hardware Integration

To make project idea more understandable I expanded project by implementing it in hardware for this purpose I used BeagleBone Black. BeagleBone Black is single board 1GHz , ARM based Linux computer. In following project BeagleBone Black act as client or end device which is bulb in our case. To run above code in ARM board arm-gcc toolchain is used to cross-compile plus Device Tree Overlay for LED is add on board kernel. Following image represents two case when LED is off and on.

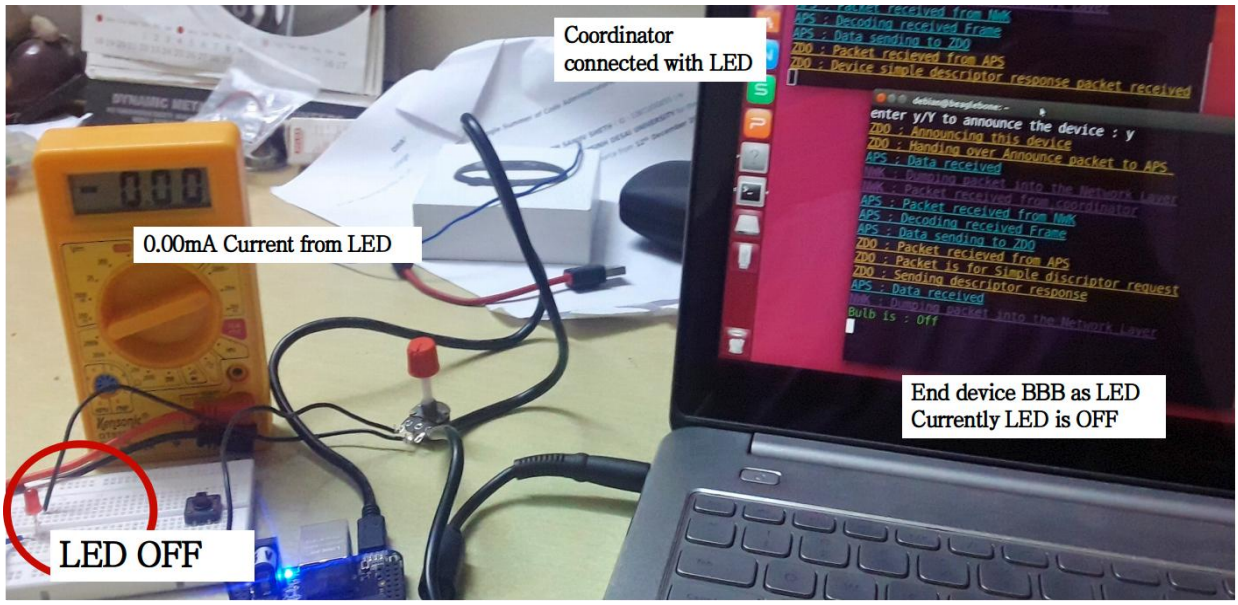

Figure 4.18 Client connected with server & LED is off at starting

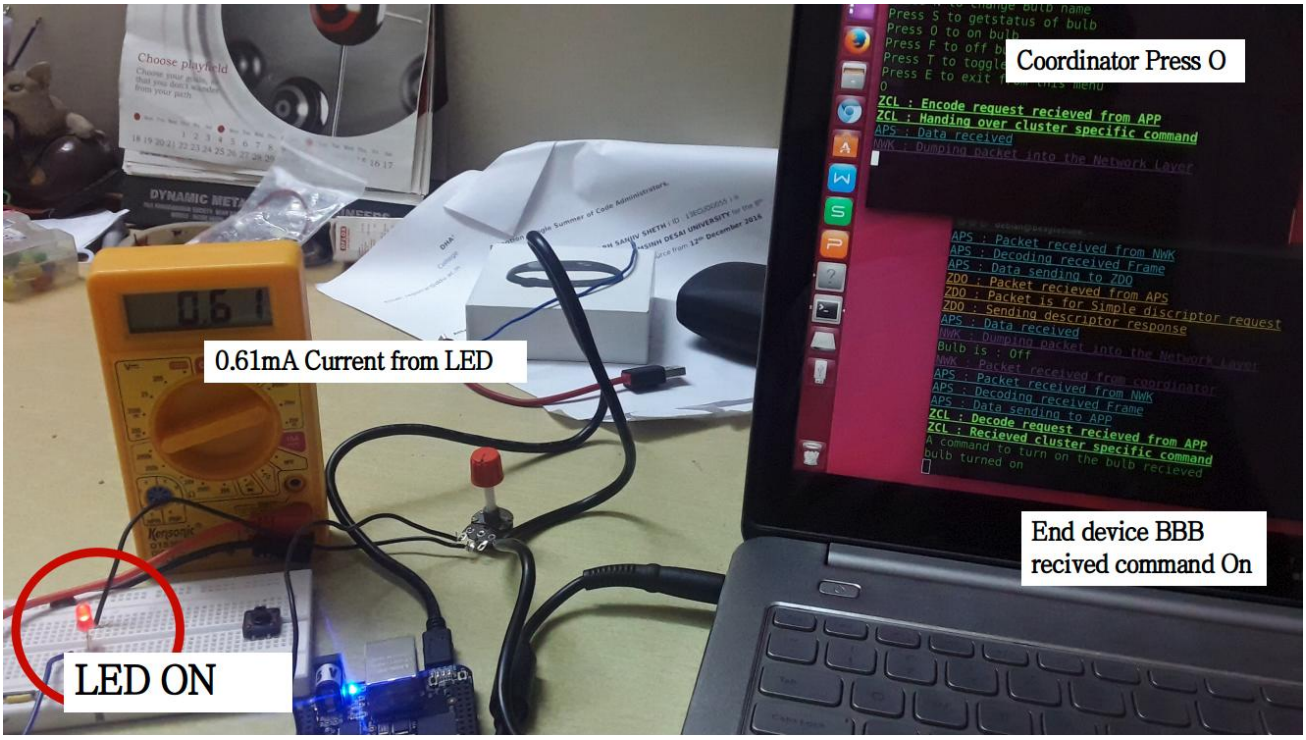

Figure 4.19 Server issue on command so LED is on client side

5. Conclusion

Currently ZigBee is one of the best choice available in market for IoT application because its design for specific controlling application . Which makes it low cost, less power consuming. Though ZigBee has good documentation for each layer but it clearly not define relation between two layers which makes it ambiguous for developing software and it create compatibility issue. Idea of making Generic ZigBee Cluster Library is really exiting and after successful implementation hole library surly it will solve major problem in industry. We successfully made part of ZigBee Cluster library because time and resources are major constrain here also we have make other layer in ZigBee to make our project fully functional. Currently project is in simulation form so for the industrial use it must be fist check on actual hardware plus to make ZigBee device truly generic some change on hardware needs to be done. By leaping into this project we witnessed the importance of layered architecture. How layered architecture provides flexibility when it comes to updating of a specific part of entire system.

Part - II Industrial Training

1.1 Overview

Before the industrial project all inters have to go through predefine two and half month of training in Volansys. This training covers topics like advance C programming, Linux essential, Data structure, Morden Operating System. The aim of training is to improve knowledge about embedded system and provide bridge between theoretical knowledge and industrial requirement. Following section provides brief introduction about what I learn and project I did as part of this training.

1.2 Project - Railway Ticket Reservation System

Aim of this project was to implement all concepts we learn during course of Advance C training. The scope of this software to simulate railway ticket reservation system in which admin can add train and also can edit them. User can see train details and can book or cancel ticket. All data must be stored in disk and displayed when needed.

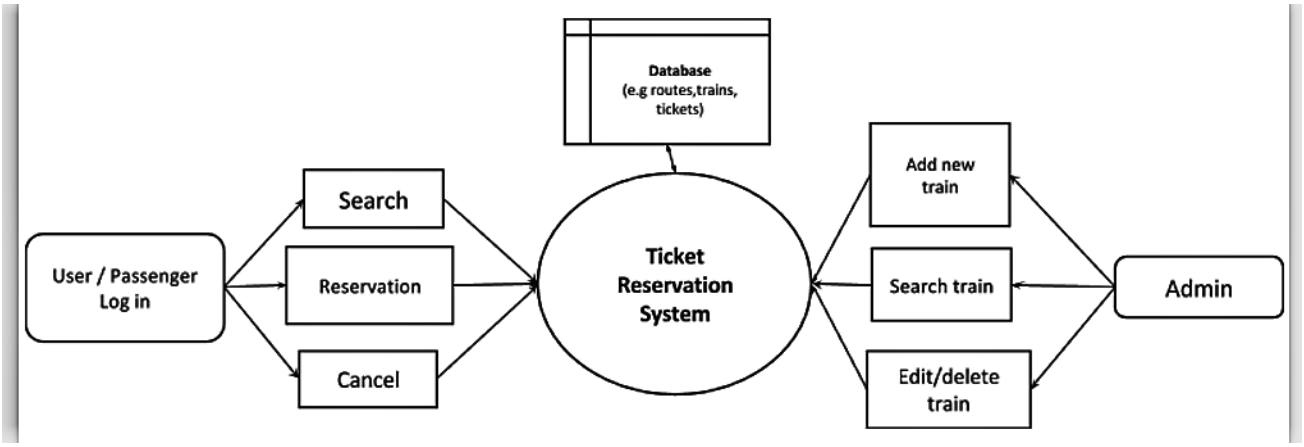

Figure 1.1 Overall view of railway Ticket Reservation system

The given problem is divided into following sub modules.

- Module 1 : Initialization
- Module 2 : Search
- Module 3 : Login
- Module 4 : Add new train
- Module 5 : Edit/Delete train
- Module 6 : Book ticket
- Module 7 : Cancel ticket

Each modules have Its own importance and it is high level representation of solution. Each modules may have other sub modules or its own components, these all are discussed in details in following section.

Top level algorithm of each module

Module 1 : Initialization

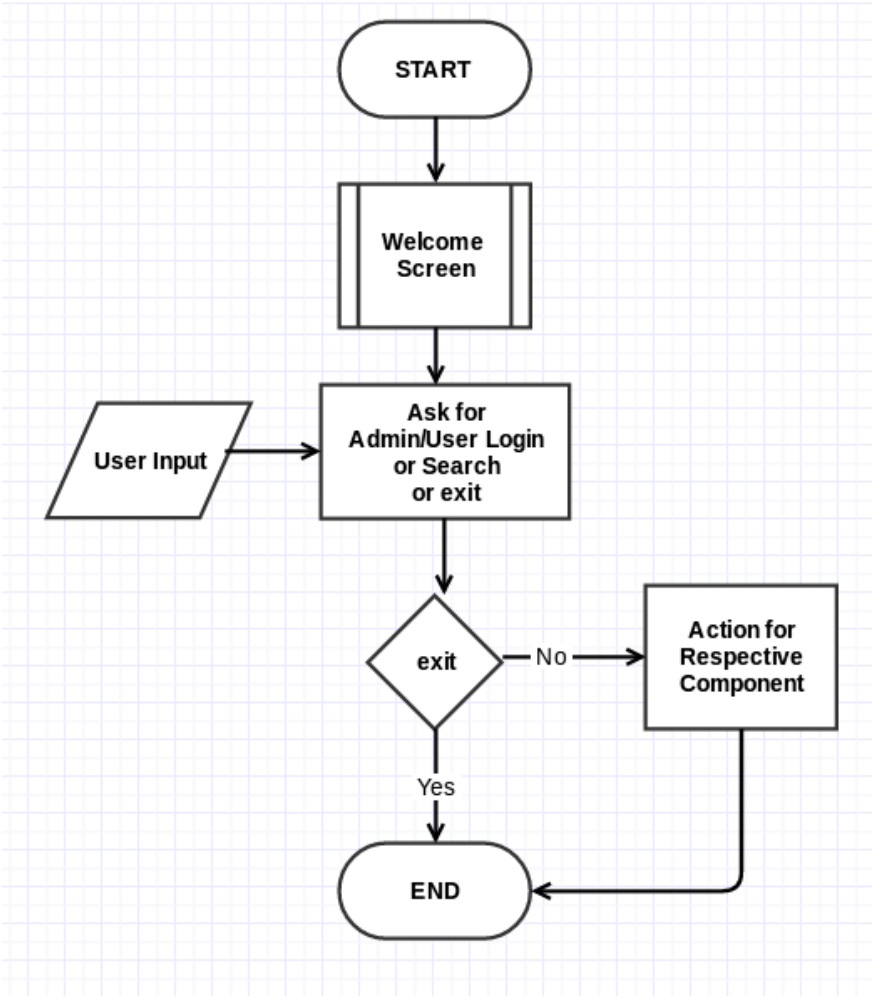

Figure 1.2 Initialization

Module 2 : Search

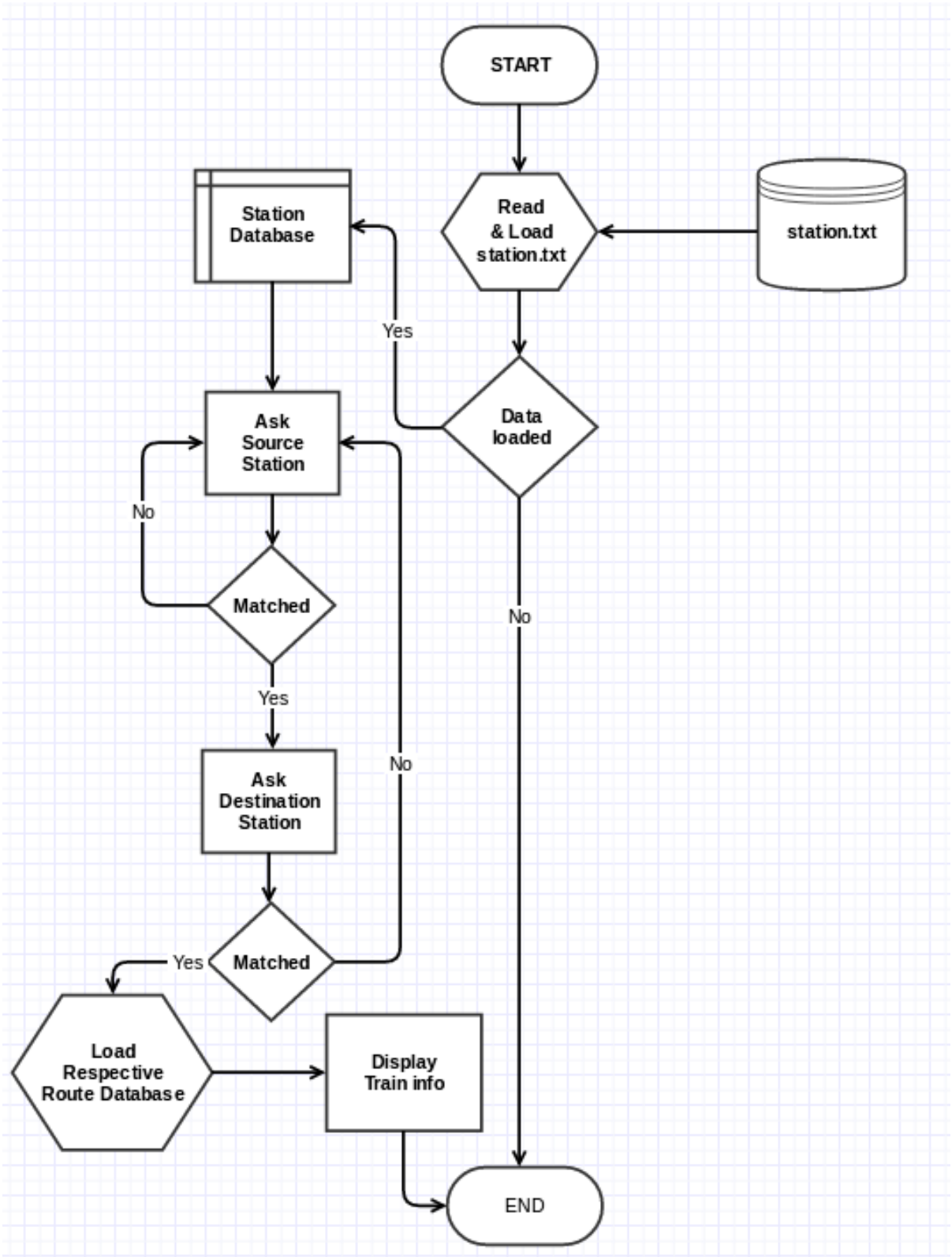

Figure 1.3 Search

Module 3 : Login

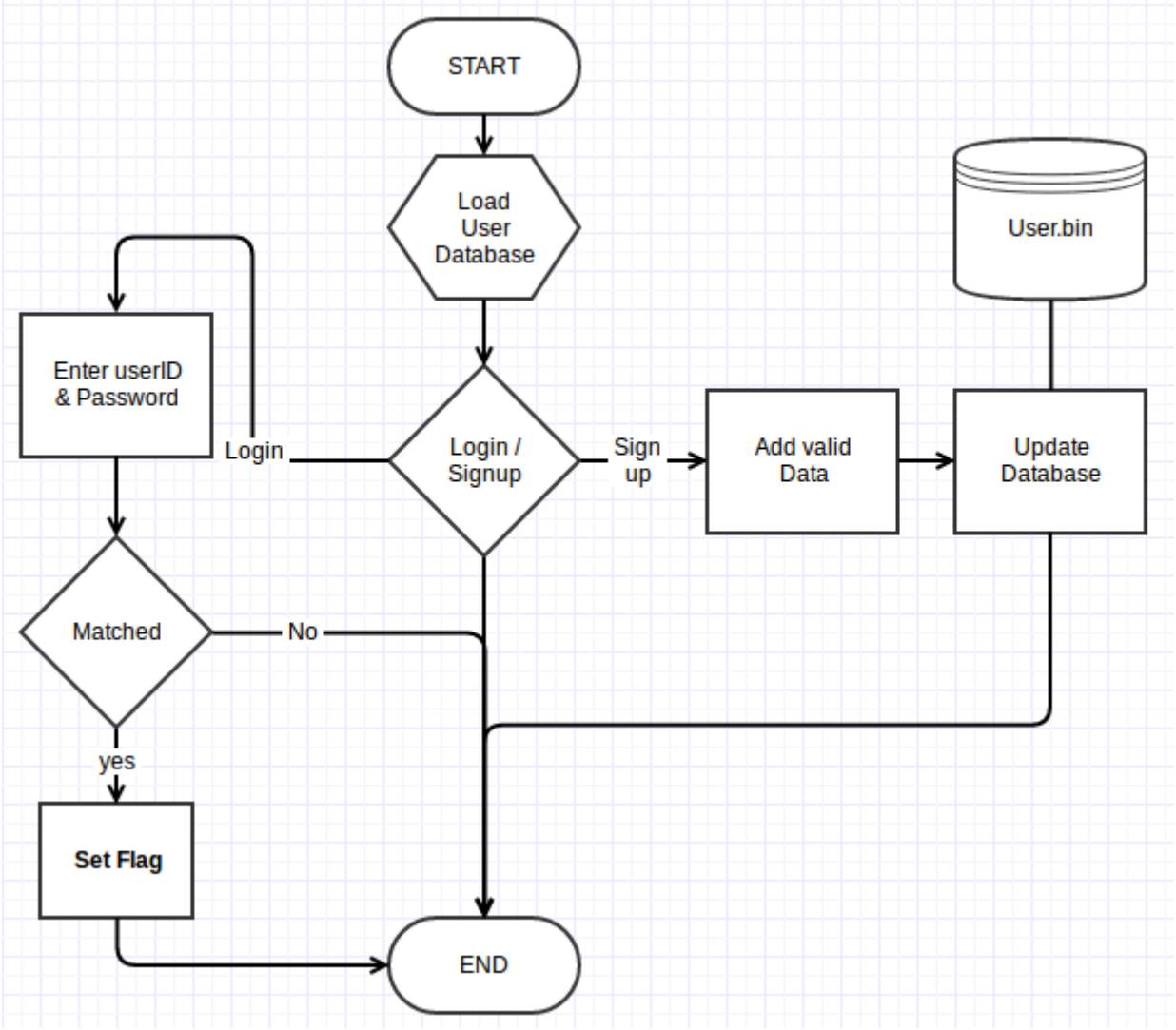

Figure 1.4 Login

Module 4 : Add new train

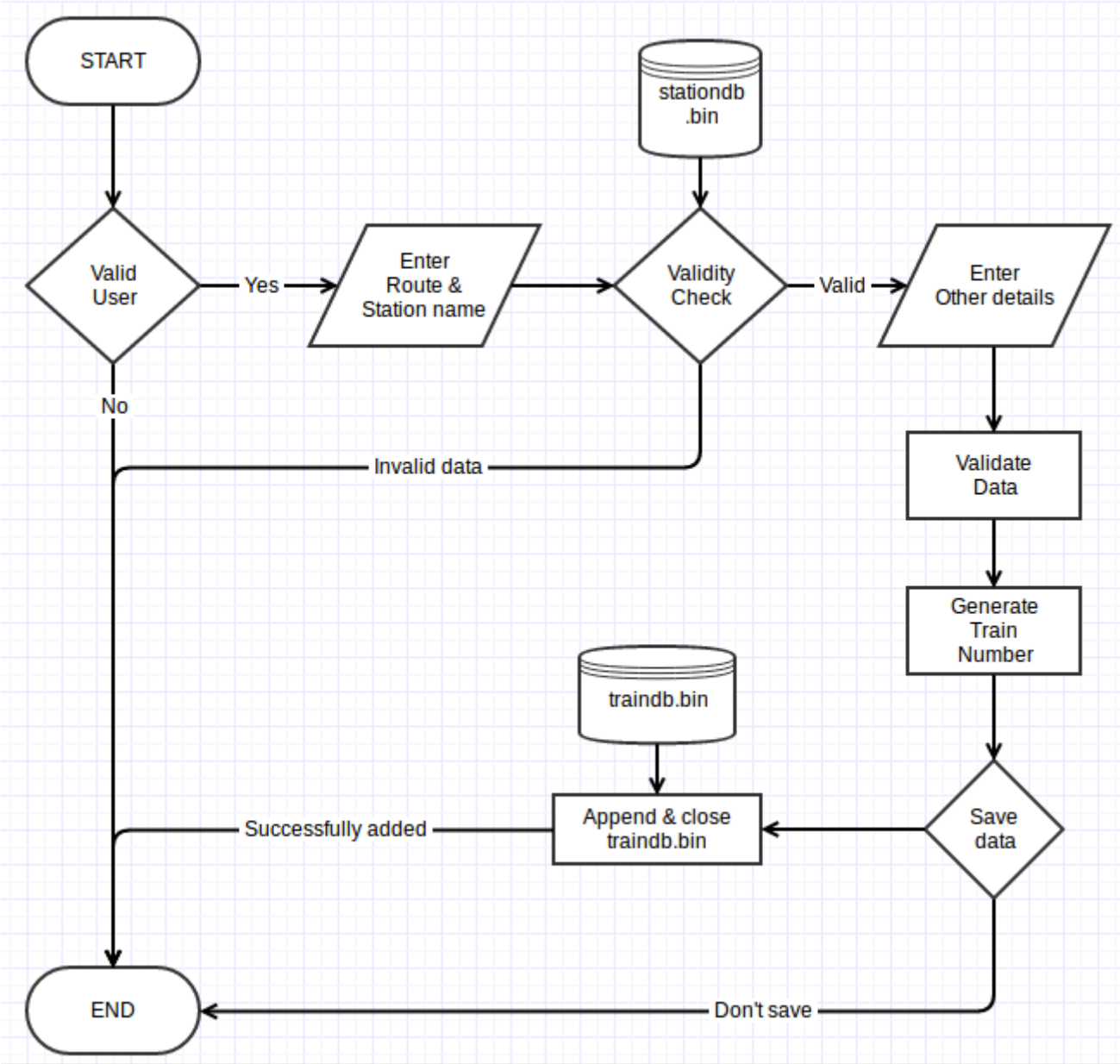

Figure 1.5 Add new train

Module 5 : Edit/Delete train

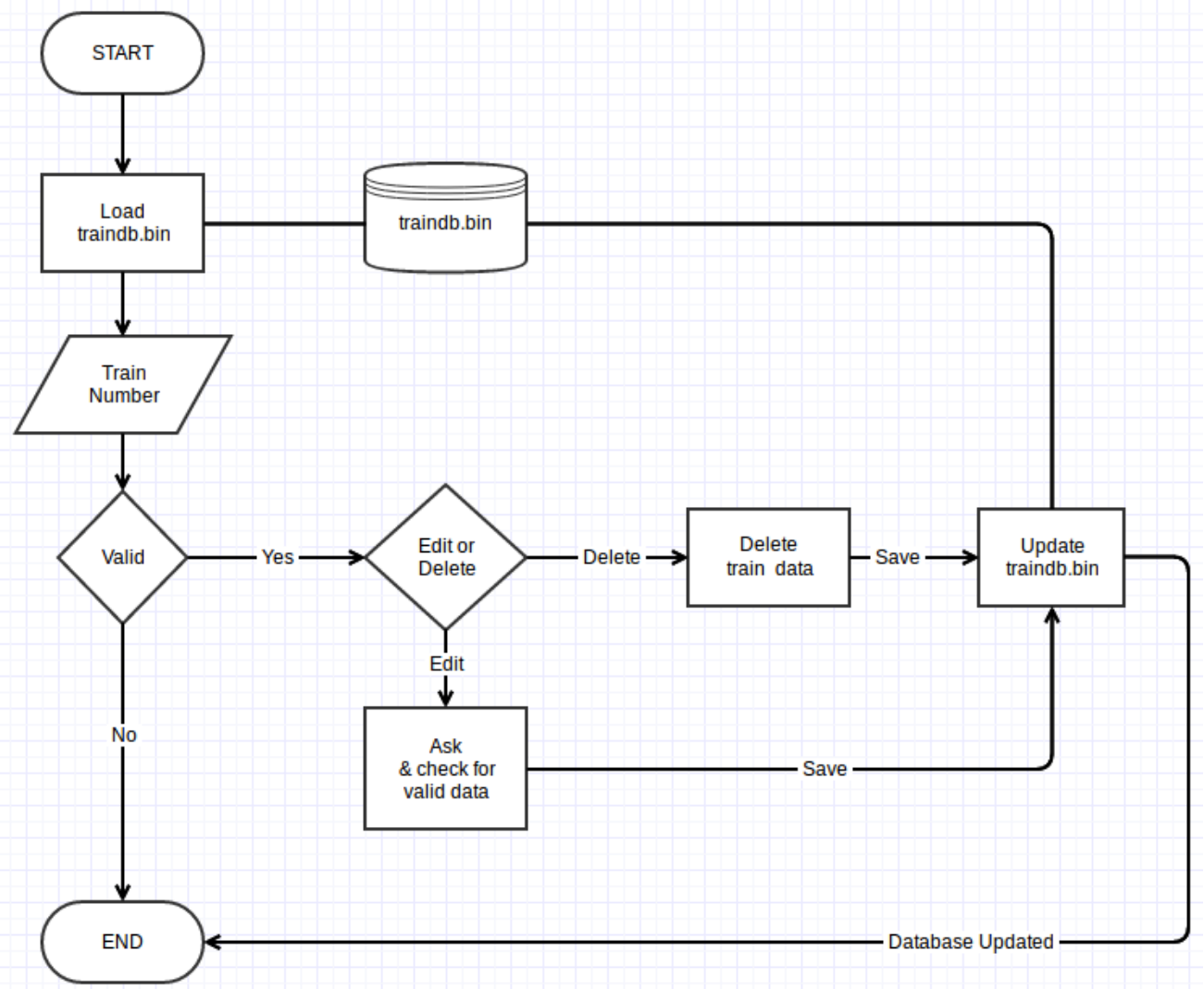

Figure 1.6 Edit/Delete train

Module 6 : Book ticket

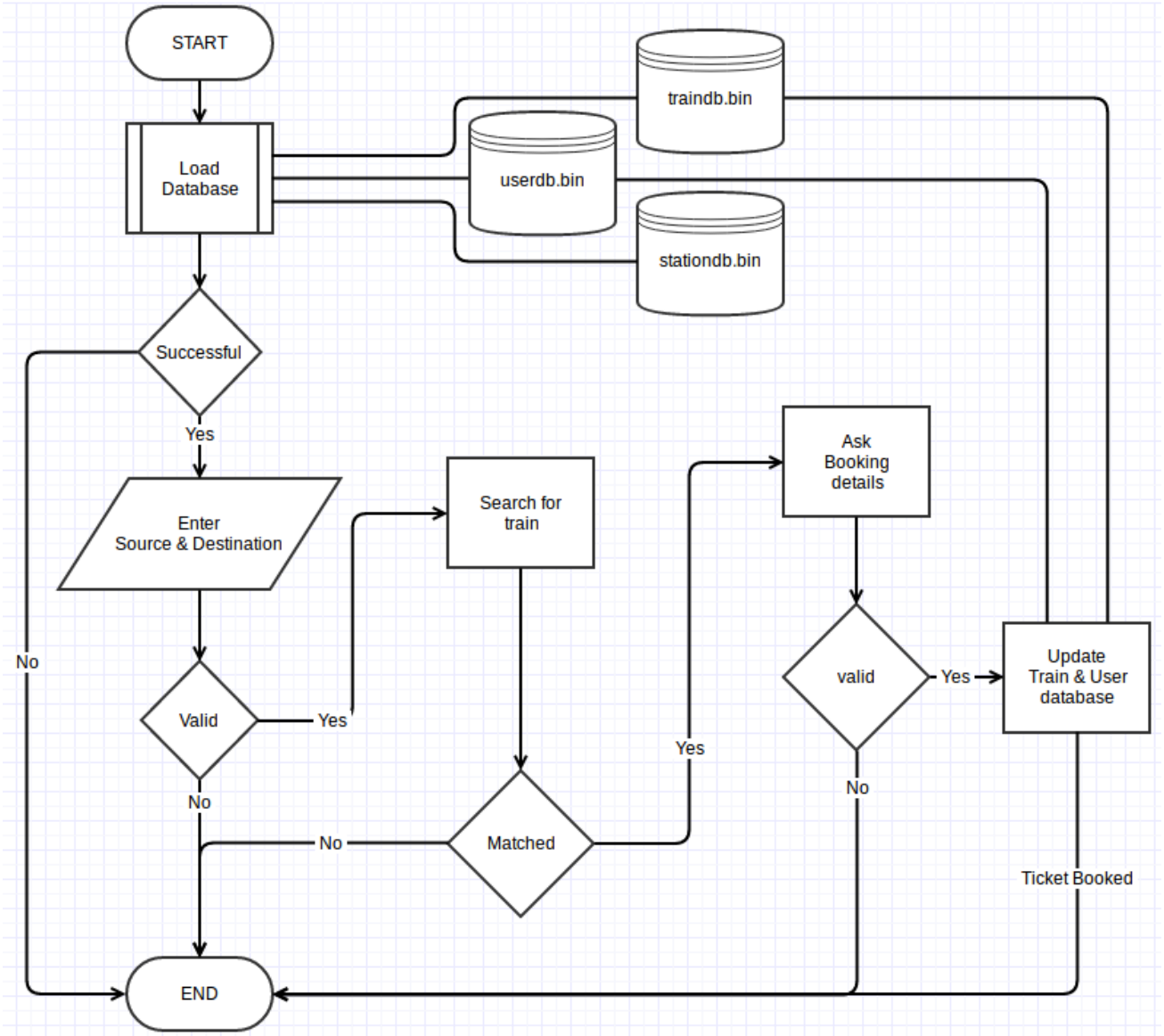

Figure 1.7 Book ticket

Module 7 : Cancel ticket

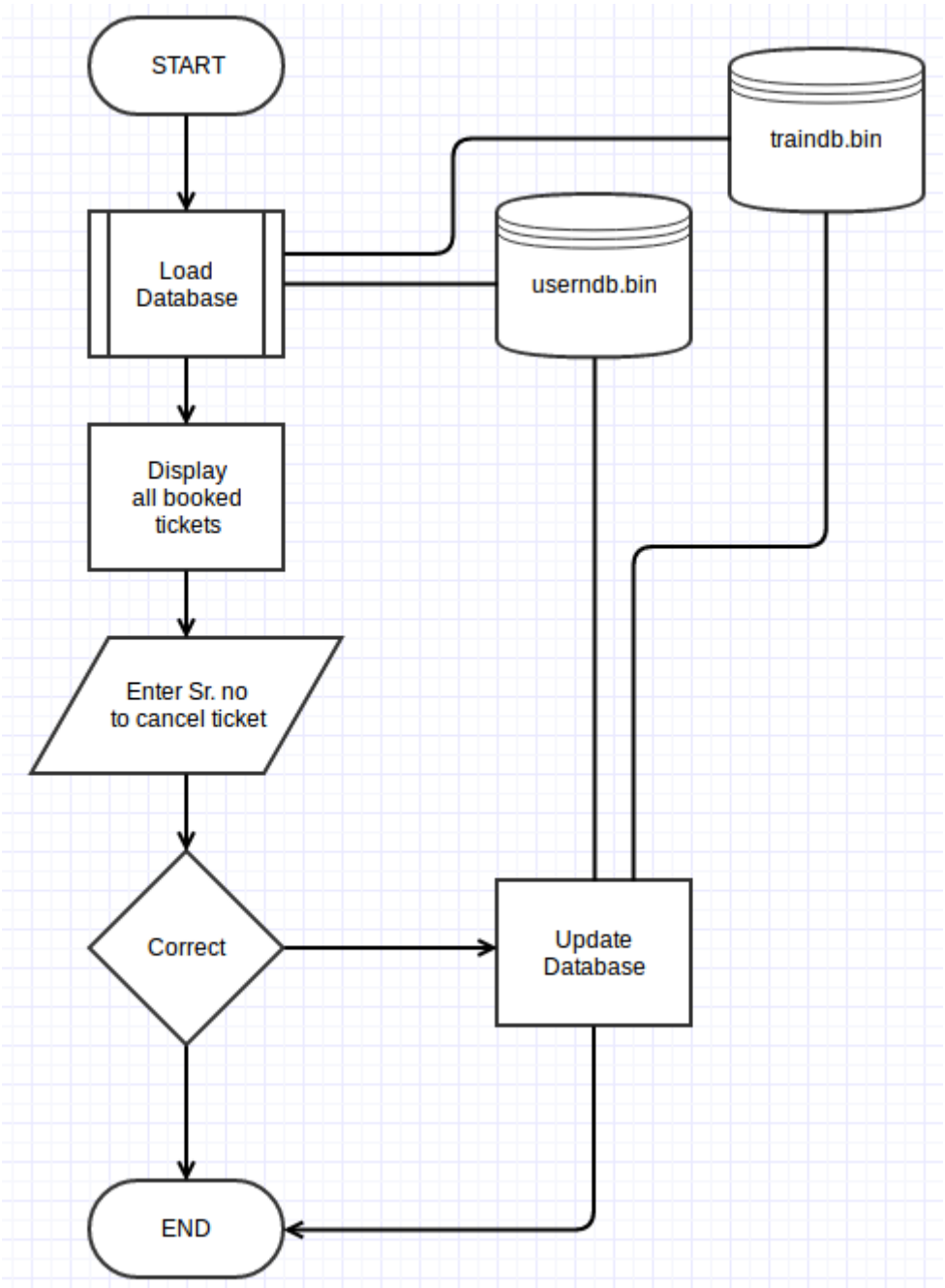

Figure 1.8 Cancel ticket

1.3 Data Structure

Data Structure training provides vast knowledge of different types of algorithms and how it can be implemented in our project. It is essential part of any software or program which defiantly impact on performance.

Searching Algorithms

A search algorithm is an algorithm that retrieves information stored within some data structure. Data structures can include linked lists, arrays, search trees, hash tables, or various other storage methods. Search algorithms can be classified based on their mechanism of searching. Linear search algorithms check every record for the one associated with a target key in a linear fashion. Binary, or half interval searches, repeatedly target the centre of the search structure and divide the search space in half.

Linear Search Algorithm

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

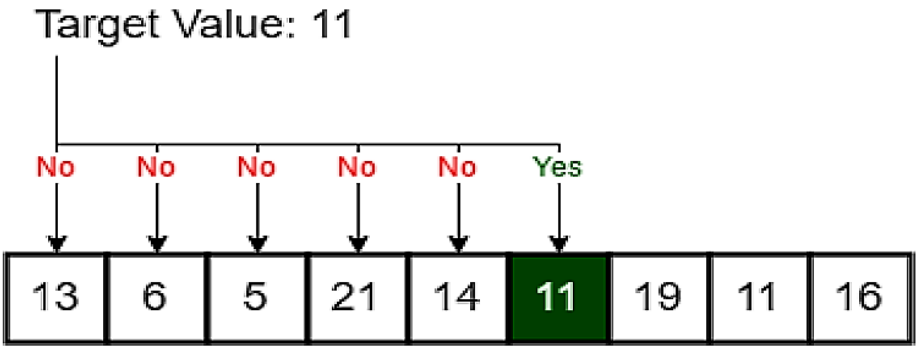

Figure 1.9 Linear Search Algorithm

Binary Search Algorithm

Binary search is a fast search algorithm with run-time complexity of O(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in

the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the sub array reduces to zero.

Binary Tree

Tree represents the nodes connected by edges. We will discuss binary tree or binary search tree specifically. Binary Tree is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.

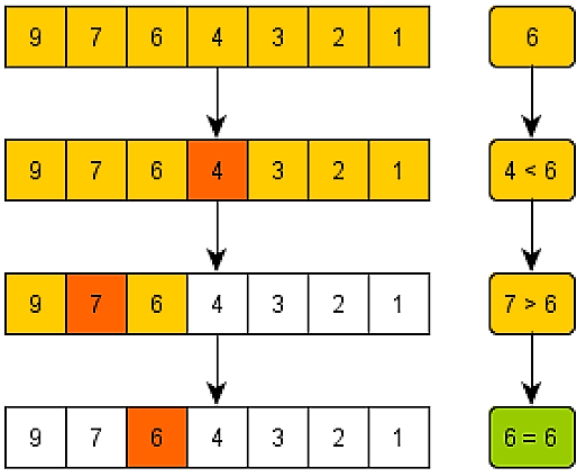

Figure 1.10 Binary Search Algorithm

Binary Search Tree Representation

Binary Search tree exhibits a special behaviour. A node's left child must have a value less than its parent's value and the node's right child must have a value greater than its parent value.

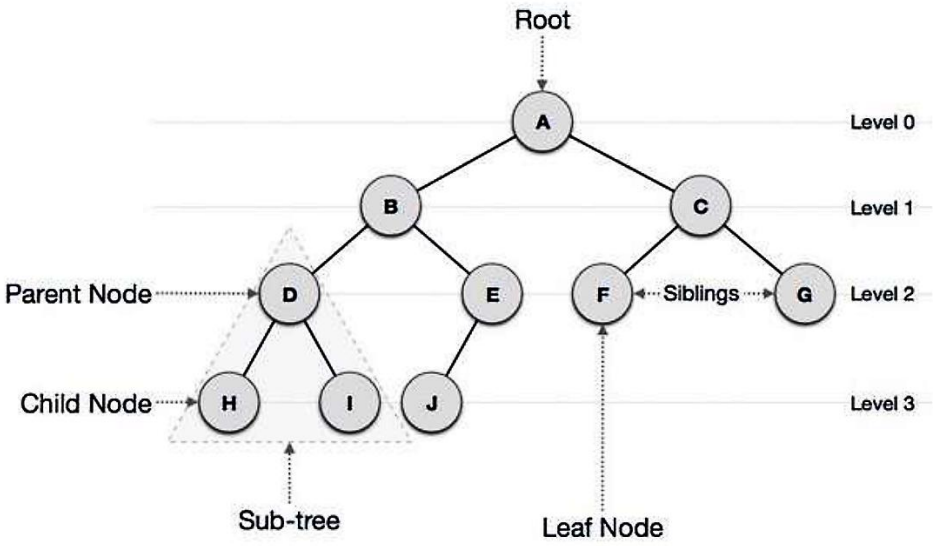

Figure 1.11 Binary Tree Algorithm

Stack

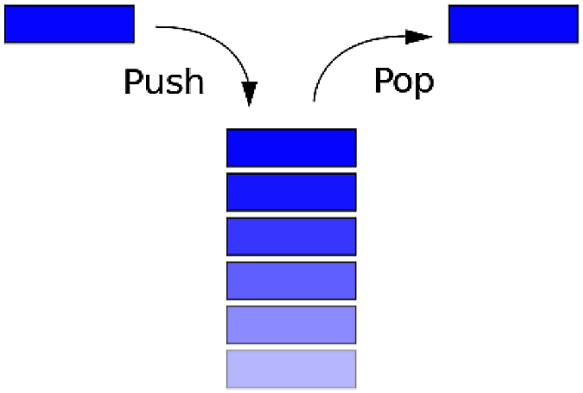

Figure 1.12 Stack

A stack is a basic data structure that can be logically thought as linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack. The basic implementation of a stack is also called a LIFO (Last In First Out) to demonstrate the way it accesses data. There are basically three operations that can be performed on stacks . They are 1) inserting an item into a stack (push). 2) deleting an item from the stack (pop). 3) displaying the contents of the stack(pip).

Applications

The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack. Another application is an "undo" mechanism in text editors; this operation is accomplished by keeping all text changes in a stack

Queue

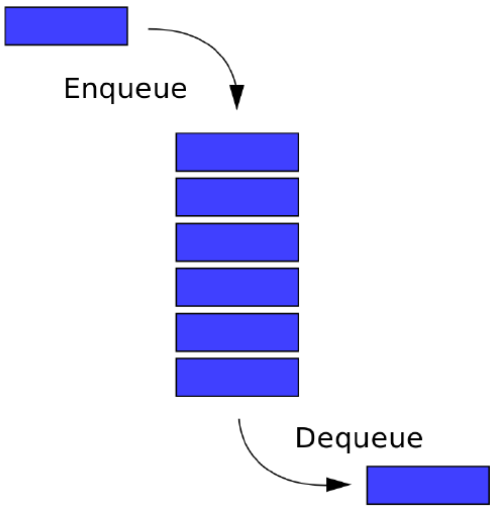

Figure 1.13 **Queue**

A queue is a basic data structure that is used throughout programming. You can think of it as a line in a grocery store. The first one in the line is the first one to be served. Just like a queue. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Applications

When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

1.4 Conclusion

In this two month training period we learn so many thing in the field of embedded technologies. We also learn to use Linux operating system and its benefits. In Linux OS we learn vim editor to write C programs. In train reservation system project we have increased our knowledge for file handling, pointers, and Structures in C programming. We also learn how to make doxygen documents of any C code. In data structure training we learn implementation of queue and stack, different sorting algorithms and Searching algorithms. This training helped us lot during development of our final project.

References

[1]. ZigBee Device Specifiation
https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/kjb79_ajm232/pmeter/ZigBee%20Specification.pdf

[2]. ZigBee Cluster Library
https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/kjb79_ajm232/pmeter/ZigBee%20Cluster%20Library.pdf

[3]. NXP ZigBee Cluster Library User Guide
http://www.nxp.com/documents/user_manual/JN-UG-3077.pdf

Appendix - A

Acronyms and Abbreviations

For the purposes of this standard, the following acronyms and abbreviations apply:

AIB	Application support layer information base
AF	Application framework
APDU	Application support sublayer protocol data unit
APL	Application layer
ASL HDR	Application support sub-layer Header
MAC	Medium access control
NIB	Network layer information base
NLDE	Network layer data entity
NWK	Network
PAN	Personal area network
PHY	Physical layer
ZDO	ZigBee device object

Definitions

Application object: This is a component of the top portion of the application layer defined by the manufacturer that actually implements the application.

Application profile: This is a collection of device descriptions, which together form a cooperative application. For instance, a thermostat on one node communicates with a furnace on another node. Together, they cooperatively form a heating application profile.

Attribute: This is a data entity which represents a physical quantity or state. This data is communicated to other devices using commands.

Binding: This is the creation of a unidirectional logical link between a source endpoint/cluster identifier pair and a destination endpoint, which may exist on one or more devices.

Cluster: This is an application message, which may be a container for one or more attributes. As an example, the ZigBee Device Profile defines commands and responses. These are contained in Clusters with the cluster identifiers enumerated for each command and response. Each ZigBee Device Profile message is then defined as a cluster. Alternatively, an application profile may create sub-types within the cluster known as attributes. In this case, the cluster is a collection of attributes specified to accompany a specific cluster identifier (subtype messages.)

Cluster identifier: This is a reference to an enumeration of clusters within a specific application profile or collection of application profiles. The cluster identifier is a 16-bit number unique within the scope of each application profile and identifies a specific cluster. Conventions may be established across application profiles for common definitions of cluster identifiers whereby each application profile defines a set of cluster identifiers identically. Cluster identifiers are designated as inputs or outputs in the simple descriptor for use in creating a binding table.

Coordinator: This is an IEEE 802.15.4-2003 device responsible for associating and disassociating devices into its PAN. A coordinator must be a full function device (FFD).

PAN coordinator: the principal controller of an IEEE 802.15.4-2003-based network that is responsible for network formation. The PAN coordinator must be a full function device (FFD).

ZigBee device object: the portion of the application layer responsible for defining the role of the device within the network (e.g., ZigBee coordinator or end device), initiating and/or responding to binding and discovery requests, and establishing a secure relationship between network devices.