2-2019

A Comparative study of Wireless Star Networks Implemented with Current Wireless Protocols

Sizen Neupane
Grand Valley State University

A Comparative study of Wireless Star Networks Implemented with Current Wireless Protocols

Sizen Neupane

A Thesis Submitted to the Graduate Faculty of

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

Masters of Science in Engineering

Padnos College of Engineering and Computing

December 2018

Acknowledgments

I would like to express the deepest appreciation to my committee chair, Dr. Robert Bossemeyer for constantly guiding and supporting me to complete this thesis. I would like to thank my committee members, Dr. Nabeeh Kandalaft and Dr. Bruce Dunne for providing necessary feedback and guidance to improve my thesis. I am very thankful to Dr. Bossemeyer and Dr. Jiao for the Embedded System Interface class, where I learned about MSP432 and nRF24L01+, including SPI and different design techniques. I would like to thank Mr. Neil Kolban for providing the library for ESP32 BLE implementation. Lastly, I would like to express my sincere gratitude to my graduate advisor Dr. Shabbir Choudhuri, for guiding me throughout my master's program.

Abstract

Wireless communication is one of the most advanced technological developments of this era. Wireless technology enables both short-range and long-range services. Today, there are several different wireless communication technologies in existence. Each has its characteristics different from another one. This thesis will implement three short-range wireless technologies in star connection and compare the performance in the wireless network.

For this thesis, the performance of three different RF protocols - a proprietary packet protocol called Enhanced ShockBurst in nRF24L01+, Bluetooth Low Energy, and a special Wi-Fi protocol ESP-Now was compared. The general concept was to establish a star network for these protocols consisting of each module as a central hub while the others as end nodes, where all modules were configured as transceivers. The wireless star network for the proprietary radio frequency protocol Enhanced ShockBurst Feature was implemented using a transceiver device built by a Norwegian company Nordic Systems called the nRF24L01+. Similar wireless networks were also implemented for ESP-Now and BLE in an ESP32 development board. ESP-Now is a proprietary radio frequency protocol developed by a Chinese company called Espressif that allows multiple devices to connect over 2.4 GHz channels using elements of a Wi-Fi protocol without requiring a router to form a network, while Bluetooth Low Energy is a wireless personal network designed by the Bluetooth Special Interest Group (Bluetooth SIG).

Different performance metrics such as throughput (kbps), range (ft), current consumption (mA) and network routing recovery time (s) were measured in the network. From the implementations tested, it was found that the nRF24L01+ has maximum throughput when transmitting large payloads (344% higher than ESP-Now in 32 bytes payload size, BLE

throughput was 50.50 bps in 32 bytes payload size), least current consumption (ESP-Now consumes 714% more current and BLE consumes 409.7% more current), and shortest network recovery time (nRF24L01+ took 335us, ESP-Now took 31ms and BLE took 1.3s on average), while the Wi-Fi based ESP-Now has a maximum range (30.53% better than BLE and 120% better than nRF24L01+).

Table of Contents

List of Tables

List of Figures

Abbreviations

ACK	:	Acknowledgment
ADC	:	Analog to Digital Converter
AES	:	Advanced Encryption Standard
AP	:	Access Point
ATT	:	Attribute Protocol
BLE	:	Bluetooth Low Energy
CE	:	Chip Enable
CRC	:	Cyclic Redundancy Check
CS	:	Chip Select
DAC	:	Digital to Analog Converter
dBm	:	decibels relative to one milliwatt
FIFO	:	First-in, First-out
GAP	:	Generic Access Profile
GATT	:	Generic Attribute
GFSK	:	Gaussian Frequency-Shift Keying
HCI	:	Host Controller Interface
I2C	:	Inter-IC
I2S	:	Inter-IC Sound
IDF	:	IoT Development Framework
IoT	:	Internet of Things
ISM	:	Industrial, Scientific and Medical
Kbps	:	Kilobits per second

LAN	:	Local Area Network
LL	:	Link Layer
MAC	:	Media Access Control
MCU	:	Micro Controller Unit
Mbps	:	Megabits per Second
PID	:	Packet Identification
PRX	:	Primary Receiver
PTX	:	Primary Transmitter
PWM	:	Pulse Width Modulator
QoS	:	Quality of Service
RF	:	Radio Frequency
ROM	:	Read only Memory
RSSI	:	Received Signal Strength
RTC	:	Real Time Clock
SMP	:	Security Manager Protocol
SPI	:	Serial Peripheral Interface
SRAM	:	Static Random-Access Memory
SSID	:	Service Set Identifier
STA	:	Station Mode
UART	:	Universal Asynchronous Receiver-Transmitter
UUID	:	Universal Unique Identifier
WPA2	:	Wi-Fi Protected Access 2

1 Introduction

1.1 Background

In the past decades, wireless technologies have made significant progress allowing transmission of high –speed data with advanced smart devices. Wireless communication is a type of data communication where there is no physical wired connection between transmitter and receiver, but rather the communication path instead is established and connected by radio waves or microwaves to maintain communication. The communication distance between transmitter and receiver can range from few meters to thousands of kilometers range.

Wireless technologies are used mostly in those situations where mobility is essential, and wires are not practical. This thesis will compare the efficiency and performance of a proprietary communication protocol running on a device built by Nordic Systems called the nRF24L01+, the standard protocol Bluetooth Low Energy (BLE), and a proprietary protocol designed for point to point wireless communications called the ESP-Now, designed for devices that normally connect to router using Wi-Fi protocol. These three technologies have limited range but can be used in conjunction with other devices such as wireless gateways and routers to extend communications, especially in the Internet of Things (IoT) applications. The IoT is a network that consists of the web enabled devices which can collect, send and process the data that they acquire from surrounding environments. The IoT network consists of usually embedded sensors, processors and communication hardware. Many IoT devices require the use of low power and low cost wireless technology when communicating between them or connecting to the Internet; this is where these three protocols come in handy. They can be used to transmit or receive data collected from nearby sensors to gateways and routers that connect to the Internet. Since they are

low power and have low cost, they can be used as effective communication tools in an IoT network.

1.2 Terminology

1.2.1 Network Metrics

1.2.1.1 Throughput

Usually, throughput means the maximum rate of production or the maximum rate at which something can be processed. While in a communication network, throughput is the maximum number of successful transmissions over the channel. It is controlled by the available bandwidth, signal-to-noise ratio and also hardware limitations. Network throughput is usually represented as an average and measured in bits per second or sometimes data packets per second. It is an important characteristic of a network and indicates the performance and quality of a network connection. Bandwidth and throughput, at first glance, might seem similar to each other, but they are quite different. Bandwidth refers to the theoretical size of the pipe while on the other hand throughput is the actual number of data packets that get transmitted.

In this thesis, to calculate the transmission time in ESP-Now and BLE, two data packets will be transmitted consecutively and the time difference between the arrivals of the two packets in the receiver side will be measured. Whereas, in nRF24L01+ to calculate the transmission time, the time difference between the transmission of the packet and the arrival of acknowledgment signal from receiver will be measured. The measured transmission times along with the number of bits sent in a packet are used to obtain the throughput of the system. This process will be repeated numerous times to calculate the average throughput.

1.2.1.2 Transmission Range

It is the maximum distance between two nodes such that the data transmitted from one node reaches the next node successfully without any error. The range at which data can be transferred depends upon the manufacturer, also varies with the environment.

1.2.1.3 Network Recovery Time

Network Recovery Time is the time required for processing and resorting normal working operations in a network. Network Recovery allows the network administrators to regain and restore operations after a network goes offline, disconnects or any other event that stops normal network operations.

In this thesis, the network recovery time will be measured in two ways. In one way, the power of the module will be removed and the time required for the module to reconnect to the network will be measured. In another way, the module will be taken out of transmission range and similarly, the time required for reconnection to the network will be calculated.

1.2.1.4 Bandwidth and Spectral Efficiency

Bandwidth is the range of frequencies associated with signal that can pass through a medium. Bandwidth is treated as a finite resource in a communication system. Bandwidth is always shared among the communication technologies operating in the same frequency spectrum. In any communication system it is always desired to have a high bandwidth in order to accommodate more signals. But in the restricted frequency bands technologies the low bandwidth is preferred over high bandwidth. As the bandwidth of a channel increases in a restricted frequency band, the number of the devices that can make error free communication in the available frequency band decreases. And also, the high bandwidth only matters if it is needed. Any bandwidth over the

required size of data is left unused. Beside this, the lower bandwidth technologies are less susceptible to the noise compared to the high bandwidth technologies. Thus, all other parameters being same, the technologies that use the least amount of bandwidth are preferred in the restricted frequency band communication systems. The comparison of bandwidth between three protocols is given in section 5.5.

The spectral efficiency or the bandwidth efficiency of any network is the amount of the information rate that can be transmitted over a given bandwidth. It is generally expressed in the format of *bits per second per hertz*. The spectral efficiency can be calculated using Equation 1:

$$\text{Spectral Efficiency} = \frac{(\text{Information Rate})}{\text{Bandwidth}} \quad (1)$$

The comparison of spectral efficiency between three protocols is given in section 5.5.

1.2.1.5 Security

Security in any wireless communication is the process of preventing unauthorized users from accessing communication. The wireless networks are more vulnerable than the wired communications. That is why security has been always considered an important factor in the wireless communication. There are different wireless security protocols that can be used. But the most commons are Wired Equivalent Privacy (WEP) and Wi-Fi Protected Access (WPA). The ESP32 implementations of ESP-Now and BLE include encryption of the data sent over the wireless connection as part of their standard protocols. The ESP32 supports RTC and cryptographic hardware accelerators that are used to encrypt the data during transmission. However, no encryption of the nRF24L01+ signals is part of the Enhanced ShockBurst protocol. Encryption of data would be possible using hardware accelerated encryption within the MSP432 that is part of the wireless communication network using the nRF24L01+ devices. The security

feature available in three protocols is not being considered during the comparison in this thesis, though a brief description of security feature available in each protocol is mentioned in section 5.7.

1.2.2 Network Topology

1.2.2.1 Star Network

A star network is a local area network (LAN) in which all nodes are connected to a central network device, like a hub or computer. The Central device acts as a server while the rest of the nodes act as clients. All the clients can communicate with each other through the central hub. Figure 1 shows an example of a star network.

Figure 1: Star Network

In the star network, all the communication occurs through the central hub. If the peripheral nodes want to send the data to each other, then at first the data is sent to the central hub, and after that central hub sends the data to the intended receiver. The connections can be both wired and wireless links. The advantages of this network are:

- Even if one device (except the central hub) fails, the rest of the network continues to function normally.
- Devices can be added easily in the network.
- The range is doubled between nodes.

Some of the disadvantages are:

- If the central hub fails, the whole network goes down.
- In case of wired connection, this network can be expensive layout due to the number and length of cables required to connect each device to the central hub.
- The overall throughput goes down in the star network due to the overhead of packet routing through the central node.

1.2.3 Wireless Technology

1.2.3.1 Wi-Fi Direct

Wi-Fi Direct is a technology that allows Wi-Fi devices to connect directly to one another. This technology makes the connection easy and convenient to do things like print, share, display, etc. The devices which support Wi-Fi Direct do not need traditional hotspot network or router to make a connection with each other. Wi-Fi Direct can use different standards to establish communications, which are explained below:

- Wi-Fi: Wi-Fi Direct can use the same Wi-Fi technology to communicate with each other. A Wi-Fi Direct device can act as an access point, and other Wi-Fi enabled devices can directly connect to it. In this standard, Wi-Fi direct can also use different Wi-Fi attributes such as different modes configuration, Wi-Fi channel range, the creation of SSID, etc. These attributes make the set up, and discovery features easy. For example, in this thesis, the ESP32 modules are configured in the station or soft-AP modes to connect with each other. Soft-AP mode is a Software enabled Access Point mode, which when used turns the ESP32 into an access point to which a connection can be established as any other Wi-Fi device, whereas in a station mode, the device can connect to a Wi-Fi network. The Wi-

Fi direct devices use Wi-Fi set configuration functions to configure the devices. For example, in ESP-Now *esp_wifi_set_config()* function of the Espressif implementations of Wi-Fi standards can be used to set the device in either soft-AP or Station mode. The Wi-Fi channel is used to make the connection.

- Wi-Fi Direct Device and Service Discovery: This protocol allows the Wi-Fi direct devices a way to discover the available devices and services before connecting to each other. For example, Wi-Fi direct devices can see all the compatible devices in the surrounding and narrow down the list to only devices that support the printing before displaying them as Wi-Fi direct enabled printers.
- Wi-Fi Protected Setup: When the two devices make the connection, they automatically connect through Wi-Fi Protected Setup to support a secure connection. Wi-Fi Protected Setup is a network security standard created to secure the wireless home network. The security set up generally starts after the discovery of devices has been done or if the client and master setup has been configured.
- WPA2: Wi-Fi direct devices use more secure encryption WPA2 encryption to make the connection.

1.3 Description of Components

1.3.1 Hardware Description

1.3.1.1 MSP432 Launchpad

The MSP432P401R Launchpad development kit allows the development of high-performance low-power applications. This Launchpad contains ARM 32-bit Cortex-M4F microcontroller and onboard debug probe needed for programming, debugging and energy measurement [1]. Figure 2

shows that the Launchpad consists of a micro-USB port, a debugger, crystal, MSP432P401R chip, energy trace software, LEDs and other components.

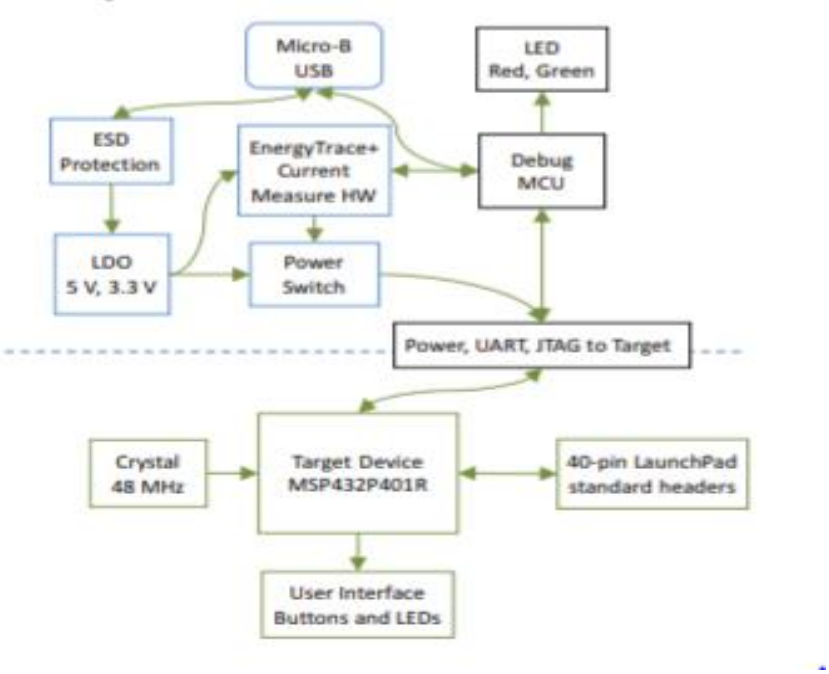

Figure 2: Block Diagram of MSP432 Launchpad [1]

The MSP432R Launchpad development kit is compatible with both 40-pin and 20-pinout standards. The Launchpad also has 2 push buttons and 2 LEDs for user interaction.

1.3.1.1.1 MSP432P401R

The MSP432P401R Launchpad features MSP432P401R which includes a 48MHz ARM® Cortex®-M4F, 80uA/MHz active power and 660nA RTC operation, 14-bit 1MSPS differential SAR ADC and AES256 accelerator [1]. The RTC operation in MSP 432 uses a computer clock to keep track of the current time. This feature can also be used to calibrate the clock, initialize the dates in Calendar mode and enable the interrupts for the RTC modules. AES256 accelerator module performs the encryption and decryption according to the AES256 encryption standard. This feature provides the security for the communication data in the MSP432. Generally, the encryption takes place between 128-bit data and a 128-bit key. For this thesis, both RTC and

AES features have not been used. The Launchpad development kit includes different clock resources; among them the High-frequency oscillator was used in this thesis to generate 48 MHz clock frequency. Figure 3 shows the functional block diagram of MSP432 Launchpad.

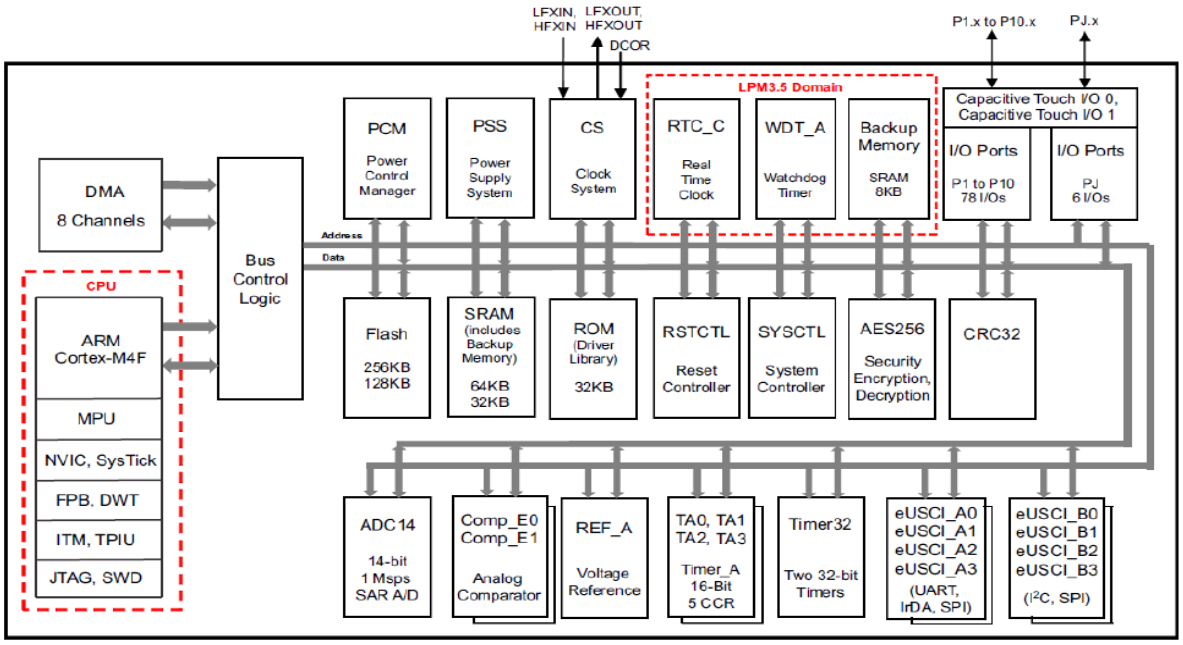

Figure 3: Functional Block Diagram of MSP432 Launchpad [2]

The CPU and all the peripherals interact with each other through a common AHB matrix. AHB stands for Advanced High-performance Bus that enables parallel access between multiple masters and slaves in a system. In this system, each master is connected to the slave devices using an interconnection matrix [2]. In this system, the bus masters are assigned priorities which are used to resolve the access when multiple masters request access to the same slave device. MSP432 supports up to eight serial communication channels (I^2C, Serial Peripheral Interface (SPI), UART, and IrDA) [2]. Out of these eight communication capabilities, only two capabilities – UART and SPI were used in this thesis. The UART and SPI capabilities of MSP432 were used to realize the communication for the nRF24L01+ based networks. The SPI

protocol establishes the communication between the MSP432 board and nRF24L01+ module while UART communication protocol takes the user input data from a terminal window to MSP432 or vice -versa. For UART or SPI communications, MSP432 offers two different modules; eUSCI_A modules which support both UART and SPI, and eUSCI_B modules that support SPI and I2C protocols. In this thesis, eUSCI_A module was used since it supports both SPI and UART communications. Besides this, internal clock source known as digitally-controlled oscillator clock source was used to generate a high-frequency clock signal of 48MHz. Besides these features, System Tick Time (SysTick) feature was also used for the time measurement task. This timer generates an interrupt request on a regular basis and based on those interrupt requests the time can be measured.

1.3.1.2 nRF24L01+

The nRF24L01+ is a highly integrated, ultra low power transceiver that communicates in 2.4 GHz ISM band. The industrial, scientific and medical (ISM) radio bands are the portions of the radio spectrum reserved – unlicensed communications allocated for industrial, scientific and medical research purposes other than telecommunications. The nRF14L01+ transceiver IC supports 250kbps, 1Mbps, and 2 Mbps on air data rates and four different power settings.
The nRF24L01+ integrates a complete 2.4GHz RF transceiver, RF synthesizer, and baseband logic including the Enhanced ShockBurst hardware protocol. The nRF24L01+ radio chip supports the high speed serial peripheral interface (SPI) protocol which allows it to be operated with any microcontrollers that support SPI communication. The nRF24L01+ always requires an external microcontroller to operate, due to which the circuit becomes a bulky unit, sometimes not ideal for the wireless sensor network.

The nRF24L01+ has four operational modes. They are Power down, Standby, TX and RX modes. In power down mode, nRF24L01+ is disabled by using minimum current consumption. In this mode, the register values are still available, and the configurations of the module can be changed. If nRF24L01+ is not transmitting or receiving any packet, then it is recommended to put nRF24L01+ in the standby mode. The current consumption in the standby mode is less than the one in TX/RX mode.

The TX mode is an active mode where the transmission of packets is carried out. To put the nRF24L01+ in TX mode, the PRIM_RX bit should set low, CE should be set high for more than 10us, and there should be a payload in TX FIFO. The module will continue transmitting the packets as long as TX FIFO is refilled.

Similarly, the RX mode is an active mode where radio acts as a receiver. To enter RX mode from standby mode, PRIM_RX and CE bits are set high. When the receiver receives a packet then at first, it checks packet validity using CRC and matching address. If the packet is valid, then the received packet is sent to a vacant slot in the RX FIFOs. But if the RX FIFOs are full or the packet is invalid, the receiver discards the received packet.

The nRF24L01+ can operate on frequencies from 2.400 GHz to 2.525 GHz, and programming resolution of the RF channel frequency setting is 1 MHz [3]. The channel bandwidth in nRf24L01+ depends upon the data rates. At 250 Kbps, the channel bandwidth is between 700 to 1000 kHz, at 1 Mbps the channel bandwidth can be between 900 to 1000 kHz, and at 2 Mbps the channel bandwidth is between 1800 kHz to 2000 kHz [3]. At 2 Mbps operation, there is channel overlapping between consecutive channels as the bandwidth is greater than the resolution of RF channel frequency setting. So, at 2 Mbps when using multiple channels, channel spacing must be

2 MHz or more than that to avoid overlapping between the channels. The RF channel frequency is set by *RF_CH* register using Equation 2 [3]:

$$F_0 = 2400 + RF\_CH(MH_Z) \quad (2)$$

Similarly, the spectral efficiency of nRf24L01+ also varies with the data rates. Using the Equation 1 the spectral efficiency at 250 Kbps is calculated to be between 0.25 bits/s/Hz to 0.35 bits/s/Hz, at 1Mbps the efficiency is between 1 bits/s/Hz to 1.111 bits/s/Hz and at 2 Mbps the efficiency is between 1 bit/s/Hz to 1.111 bits/s/Hz.

The nRF24L01+ module can work on any one of a number of channels (126) at one time, and the transmitter and the receiver must use the same channel to establish communication. In the same frequency channel, up to six different data pipes can be connected. But the communication can be done through only one data pipe at a time. It is not possible to have communication in more than one data pipes at the same time in the same frequency channel. The nRF24L01+ does not provide any channel noise reducing methods. If two different nRF24L01+ networks try to communicate on the same channel there will an error during the transmission. It is recommended to use auto acknowledgment feature of nRF24L01+ so that the status of transmission can be monitored and thus lost packets can be retransmitted. It is also recommended to use a simple channel scanner to scan the channels in the environment to find a free channel before the communication is established. Hence, in the case of nRF24L01+, there is no effective noise reducing techniques. To prevent the interference from the outside environment, it is suggested to use the highest 25 channels because other 2.4 GHz radio waves like Wi-Fi uses most of the lower channels [4].

For this thesis, a radio module supporting the operation of the nRF24L01+ IC is used including a clock crystal, antenna, and matching circuit. This module allows easy interface of the nRF24L01+ chip with a microcontroller SPI port. Some of the features of nRF24L01+ are:

- Low cost, a single chip that works in 2.4 GHz ISM band.
- Supports GFSK modulation. In GFSK modulation different frequencies are assigned to the carrier depending upon the bit that is transmitted. Let's assume that the data consists of only two symbols 0 and 1. In GFSK, the information is conveyed by assigning a fixed carrier frequency for the duration of a 0 symbol and assigning another carrier frequency for the duration of a 1 symbol. In GFSK modulation, the waveform is passed through a Gaussian filter before the waveform goes into the modulator. This is done to smooth the transition between the values of impulses. The Gaussian filter limits the spectral width of the message signal and reduces the possibility of intersymbol interference in the signal.
- Supports Multiceiver feature in which six different modules can be configured as receivers at the same time.
- It has 126 channels points; thus, it can meet multipoint communications and channel hopping requirements. These channels points range from 0 to 125, and any channel among these 126 channels can be used for communication.
- Since the link layer is fully integrated on the IC – writing code with instructions that control the nRF24L01+ is easy. The link layer in the nRF24L01+ handles the functions like data framing, physical addressing, error control and access control for the user.

Figure 4 is the functional block diagram of nRF24L01+. It shows that a typical nRF24L01+ chip consists of RF transmitter and receiver, filters, modulator, demodulator, TX and RX FIFOs and Enhanced ShockBurst Baseband Engine.

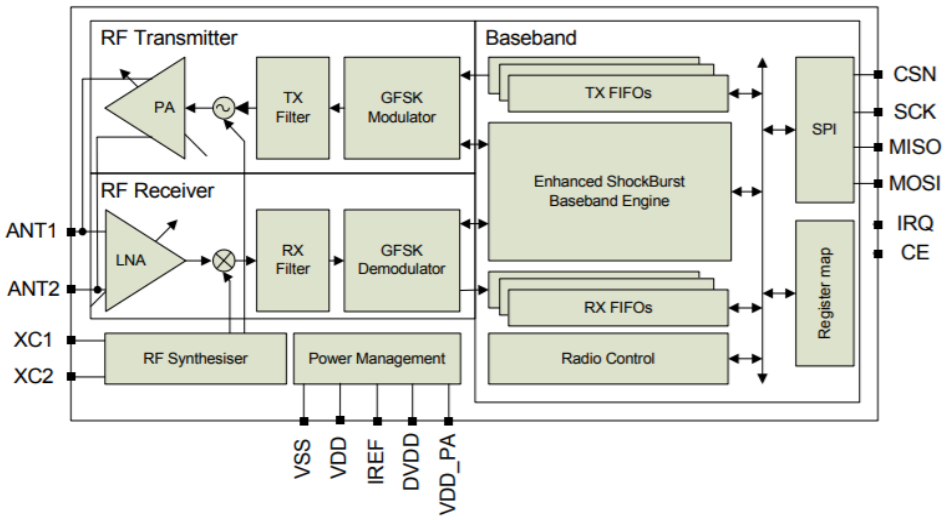

Figure 4: Functional Block Diagram of nRF24L01+ [3]

The digital signals are converted in to radio waves using GFSK modulation. The data in nRF24L01+ is always put in packet format in the form of 0s and 1s. An embedded baseband protocol engine handles the packet communication between two modules. A bit in the configuration register is used to set the module in either transmit or receive mode. The Chip Enable (CE) bit is used to activate the transmitter or put the radio in receiving mode. If the RF module is configured as a receiver, then CE bit is set high to monitor the air and receive the packets. While if it is configured as a transmitter, then the CE bit is set high for about 10us to initiate the transmission and after that, it is set back to low. In this thesis, the transmitter receives data from MSP432 through SPI communication, formats it into proper packets, and transmits it to the receiver. While in the receiver side the received data is downloaded from FIFO, the error checking mechanism is performed, and at last, the user sent data or payload is extracted from the packet, and transmitted to MSP432 through SPI communication.

1.3.1.2.1 Enhanced ShockBurst Protocol

Enhanced ShockBurst is a packet based data link layer that features automatic packet assembly and timing, automatic acknowledgement and retransmissions of packets [3]. This feature enables the implementation of ultra-low power and high-performance communication. This feature improves the power efficiency for both bi-directional and uni-directional systems significantly, without adding any complexity. Some important features of Enhanced ShockBurst are given below:

- It supports a star network topology with one Primary Receiver (PRX) and up to 6 Primary Transmitters (PTXs).
- Provides both static and dynamic payload length features.
- In dynamic payload length feature, a packet from 1 to 32 bytes can be transmitted to the receiver.
- Supports bi directional data transfer between each PTX and the PRX.
- Supports Packet acknowledgment (ACK) and automatic packet re transmission.
- Have individual transmit (TX) and receive (RX) FIFOs for every pipe.

An Enhanced ShockBurst packet transaction is initiated when a packet is transmitted by the Primary Transmitter (PTX) and the transaction is completed when it receives an acknowledgment packet (ACK packet) from the Primary Receiver (PRX). The PRX also has an option of transmitting a payload to PTX along with ACK packet, this feature is known as Acknowledgement with Payload. If the PTX does not receive the acknowledgment after initial transmission, it retransmits the packet until it receives back the acknowledgment from the receiver. The total number of retransmissions attempts and the delay between the attempts can be configured in the register values. The receiver checks the validity of the received packet with the

help of a packet ID (PID) and cyclic redundancy check (CRC) fields. The PID is a 2-bit field which is used to detect if the received packet is a new or old one, while CRC is an error detection mechanism in the packet. The CRC is automatically calculated based on the packet content with the predefined polynomials. After receiving the packet, Enhanced ShockBurst performs CRC and checks its validity. If the CRC is valid, the received PID is compared with the previous received PID. If both PID fields are different, then the packet is considered a new one. But if the PID fields are same, then the receiver checks the CRC field. If the received CRC field is equal to the previous one, the newly received packet is defined as the previous packet and packet is discarded. But if the CRC field of the new packet is different than the previous one, the packet is considered a new one and is accepted.

Figure 5 is a basic auto acknowledgment communication timing diagram for the nRF24L01+. Figure 5 shows that after the packet is transmitted by the PTX and received by the PRX the ACK packet is transmitted from PRX to PTX. At first the PTX uploads the packet in the FIFO. Then, the packet is transmitted to the PRX and after 130us PTX changes its state to receiving state to receive the acknowledgement (ACK) packet from PRX. Similarly, when the PRX receives the packet, it downloads the packet and checks for error. If the received packet is a new one, then it sends the acknowledgment back to PTX. When PTX receives an acknowledgment, an interrupt is generated in MCU to indicate the successful transmission of data.

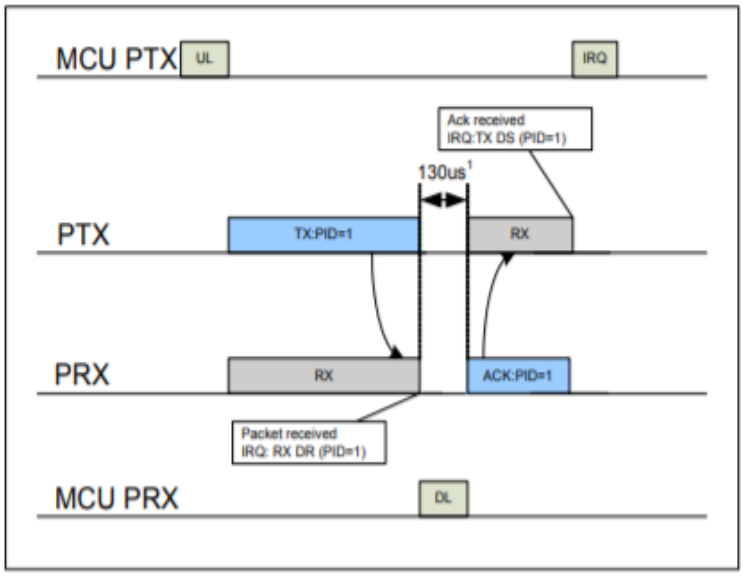

Figure 5: Data Transfer Timing Diagram in ShockBurst [3]

Static and Dynamic Payload Length

Enhanced ShockBurst provides two different alternatives to handle payload length. The payload in a whole transmitted packet is the actual user sent data. In normal condition, the nRF24L01+ transmits the data using static payload length feature. In static payload length configuration, all packets transmitted between a transmitter and a receiver always has the same payload length that is 32 bytes. But with dynamic payload length feature, the transmitter can send packets with variable payload lengths to the receiver. So that means that for a system with different payload lengths it is not necessary to scale the payload length to the longest payload size (32 bytes), thus in dynamic payload length feature, the length of the payload is equal to the length of the actual user sent data. The dynamic payload length decreases the transmission time in the nRF24L01+.

Auto Acknowledgment

Auto acknowledgment is a function that allows the receiver to automatically send the acknowledgment (ACK) packet to the transmitter after it has received and verified the data. The auto acknowledgment function reduces the load of the system MCU and also reduces cost and

average current consumption. An ACK packet from the receiver can also contain payload from PRX to PTX. This feature is known as acknowledgment with payload feature. To use this feature, the Dynamic Payload Length should be enabled. The MCU on the PRX side uploads the payload by clocking it into the TX FIFO. This payload stays in the TX FIFO (PRX) until a new packet is received from PTX. After a new packet is received this payload is transmitted with the new acknowledgment packet. Figure 6 shows the TX FIFO in receiver side when the acknowledgment with payload feature is enabled. From Figure 6 it can be observed that at a time, nRF24L01+ can have up to three ACK payloads pending in the TX FIFO.

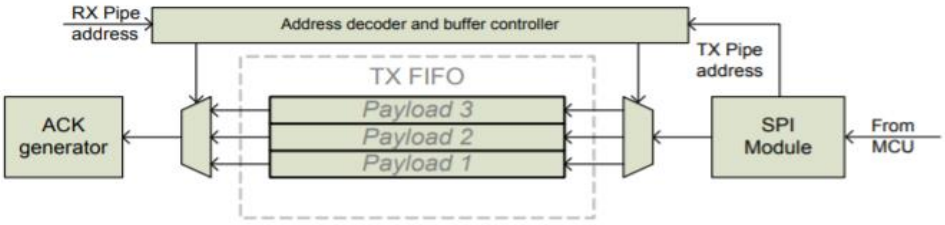

Figure 6: TX FIFO (PRX) with Pending Payloads [3]

Multiceiver

Multiceiver is a feature used in RX mode that allows six different TX to connect with it using a set of six parallel data pipes with unique addresses. This feature allows nRF24L01+ to implement the Star Network protocol. A data pipe is a logical channel in the physical RF channel and has its own physical address. All data pipes addresses are searched at the same time, but only one data pipe can receive a packet a time. Figure 7 shows a Multiceiver network in nRF24L01+.

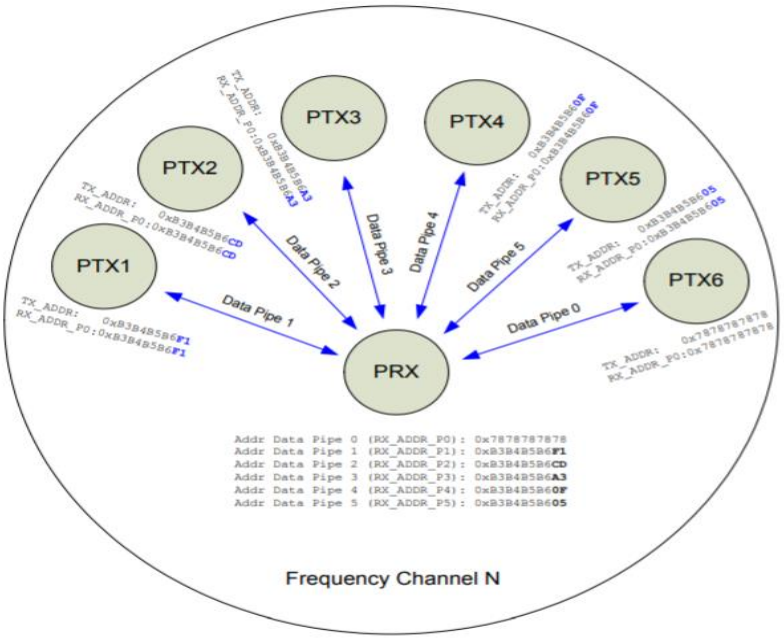

Figure 7: Data Pipe Addressing in Multiceiver [3]

The main characteristic of Multiceiver capability is having up to 6 radio communication channels open in a receiving mode simultaneously. The first task is to open six reading pipes in the primary receiver hub (PRX). This PRX acts as the central hub in the network. All the data communication is done through this hub. Now each PTX or peripheral node links to one of these opened pipes. These PTX nodes transmit and receive the packets using this pipe. So, when the data is being transmitted from PRX to PTX, the writing address of PRX should match the address of PTX it wants to transfer. The data pipe addressing in Multiceiver network has some specific rules. Data pipe 0 always has a unique 5-byte address while data pipe 1-5 share the four most significant address bytes. And also, the write address on the PRX side should match the PTX address. Another one important thing to remember is that when transmitting from the PRX to PTX, the write address should always be opened in pipe 0. For example, let us suppose that there are 3 PTXs, and they are opened in pipe 0, 1 and 2 in PRX respectively. Now if the data has to be transmitted from the PTX 1 to PTX 3, then the first transmission is done from the

PTX1 to PRX through data pipe 0. During the second transmission from the PRX to PTX 2, the write address from central hub (PRX) should be opened in pipe 0 even though the PTX 2 was opened initially in pipe 2 in PRX.

1.3.1.3 ESP32

ESP32 is a single 2.4 GHz Wi-Fi and Bluetooth combo chip designed with the TSMC ultra-lower-power 40nm technology [5]. It is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs. This device is specifically designed for mobile, wearable electronics and the IoT applications. Figure 8 shows the general block diagram of ESP32. The clock frequency of ESP32 can be up to 240 MHz. There are different modules of ESP32 available in the market. In this thesis, the ESP-WROOM-32 module on the ESP-DevKitC board [5] is used to realize the BLE and ESP-Now protocols. ESP32 supports classical Bluetooth, BLE and Wi-Fi standard wireless protocols. Beside these standard protocols, ESP32 also supports ESP-Now, which is a nonstandard wireless protocol used for communicating Wi-Fi enabled devices without the use of a router. The ESP32 also supports RTC and cryptographic hardware accelerators. None of these securities and RTC features has been studied in this thesis.

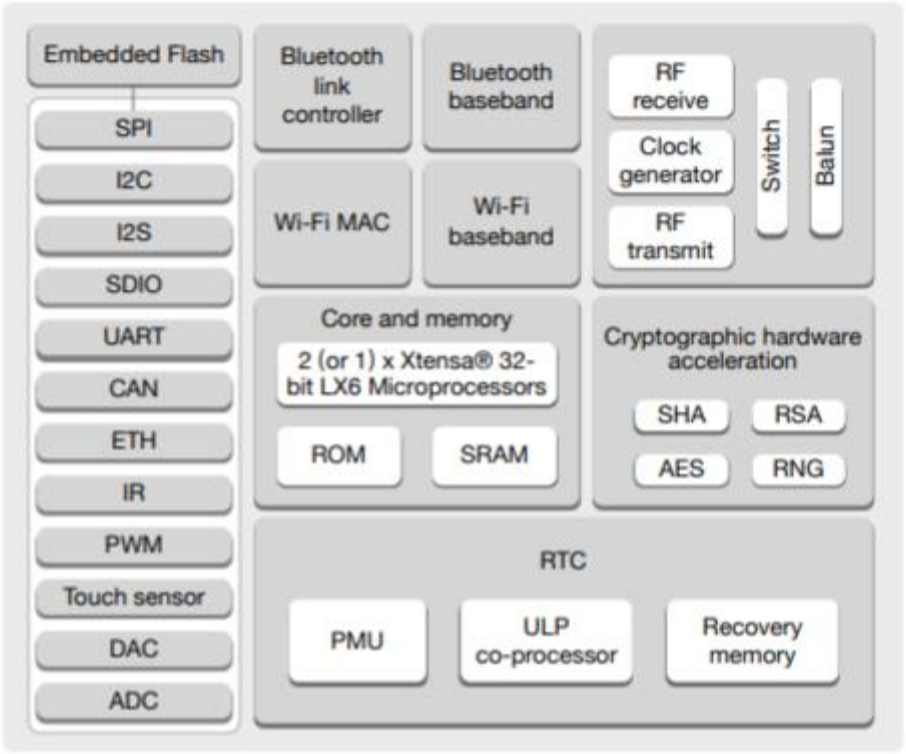

Figure 8: Functional Block Diagram of ESP32 [5]

1.3.1.3.1 Bluetooth Low Energy (BLE)

Bluetooth Low energy is a wireless personal area network technology designed by the Bluetooth Special Interest Group. BLE is an updated version of classical Bluetooth aimed at reducing the power consumption and cost while maintaining similar communication range. The main difference between the regular (classical) Bluetooth and BLE is the power consumption and message size. The classical or regular Bluetooth was designed to handle large data size at high power consumption. But BLE is used for those applications which do not need to exchange large amounts of data, thus have low power consumption. The application throughput and communication set up time for regular Bluetooth is higher compared to the BLE. Thus, BLE is generally suited for those applications which require the periodic transfer of data and not the

continuous data stream, while classical Bluetooth is suitable for later one. Also, the classical Bluetooth has 79 channels in the frequency band whereas BLE consists of only 40 channels in the frequency band.

The Bluetooth Low Energy consists of 40 channels from 2.402 GHz to 2.480 GHz. Among 40 channels, three are used as advertising channels, and 37 are used as data channels. These channels have center frequencies at $2402 + k \times 2$ MHz, where k = 0 to 39 [6]. The three channels from 37 to 39 are used for advertising packets whereas the remaining channels are used to exchange data packets in the connections. Each channel has a bandwidth of 2 MHz.

Figure 9 shows the BLE frequency channels. The first channel, 37, is assigned at 2402 MHz, while the last one, the 39 is centered at 2480 MHz. The maximum data rate of BLE in ESP32 is up to 700 kbps [7] . So, using Equation 1 the maximum spectral efficiency that can be achieved in ESP32 BLE is 0.35 bits/s/Hz.

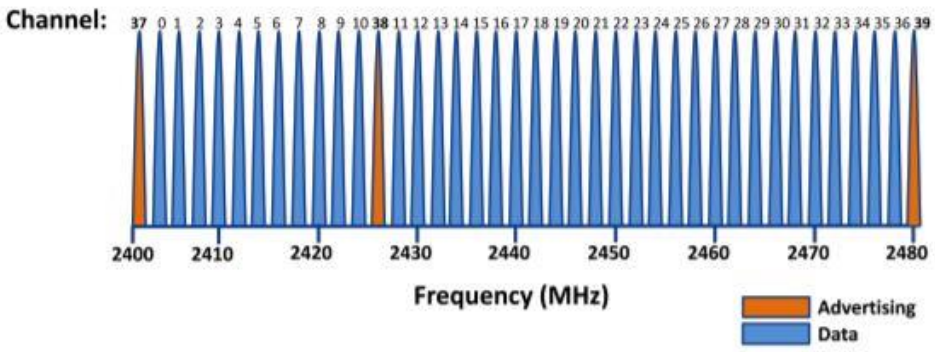

Figure 9: BLE Frequency Channels [6]

In the BLE, two devices use a shared physical channel for communication. In BLE specification, for devices to communicate with each other they are tuned to same RF frequency at a time [8]. If there are multiple BLE devices tuned into the same frequency, then there is a possibility of

channel collision among devices. To mitigate this issue, BLE uses a randomly generated Access Address to identify the link between the devices.

The data channel of BLE is characterized by a pseudo-random sequence of PHY channel as shown in Figure 9**.** In addition to this, it is also characterized by the three additional parameters provided by master. A master is the BLE device in the network that initiates the connection and handles almost every aspects of the connection. Either a server or client can act as master after the connection is established. One of the parameters is the channel map that indicates the set of physical channels used. The channel map is a sequence of 40 bits where three most significant bits are reserved for advertising channels and rest 37 bits corresponding to the data channels. If in a BLE network, channel x is used, then the bit corresponding to that data channel in channel map is set to 1 [9]. The next parameter is a hop, whose value ranges from 5 to 16 and is set when the connection is created. The data channel in BLE is divided into connection events where each event corresponds to a physical frequency. The connection event in the ESP32 BLE occurs when a client is connected to the server. The data exchanging occurs between BLE devices over these dedicated frequency hopping data channels [10]. The connection will stay on the same channel during the interval of connection events. During the new connection event, the master switches to a new channel.

The maximum number of slaves that can connect to a master depends upon the software and hardware characteristics, such as the amount of memory on BLE IC, the type of communications between master and slave, and some connection parameters such as *connInterval* [6]. If a master wants to connect to a new slave, then it has to reset its RF frequency since different slaves belong to different piconets with a master in common. For this reason, the *connInterval* parameter influences the number of slaves. This parameter is the time between the beginnings of

two consecutive connection events. It is always the multiple of 1.25ms in the range of 7.5ms to 4.0s [6]. This interval can be set in Generic Access Profile (GAP) parameters. The master needs time to switch between the different slaves without overlapping their respective connection events. Thus, it is possible to have two overlapping communications in BLE if connection interval is not properly set. In this case, there will be a loss of data during communication. Hence, if a master needs to connect with a lot of slaves, then the connection event of each connection has to be decreased. At present, theoretically, a maximum number of 7 slaves can connect to the master in the latest version of ESP32 BLE [11].

An adaptive frequency mechanism is used in BLE to reduce the interference from other technologies. The hopping pattern in BLE is unique and is derived from the clock and BLE device address of master. Adaptive frequency hopping is the type of hopping process in which channel conditions are constantly monitored to identify the occupied or low quality channels. This allows the Bluetooth to identify the fixed sources of interference and exclude them from the list of available channels. Common metrics that are used to determine the quality on RX channel are Received Signal Strength Indication, Bit Error Rate, and Signal to Noise Ratio [12].

A BLE device can communicate to another device in two ways: Broadcasting and Connection.

Broadcasting

Broadcasting is the method in which a device sends the data out to all listening devices. The broadcaster sends the non-connectable advertising packets periodically to any listening device, while the observer continuously scans the surrounding to receive the packet. Non- connectable advertising packets mean that the central device cannot connect to the peripheral device. The non-connectable advertising packet is used only for broadcasting, and the device transmits the internally stored information in this type. This advertisement type is used when the device does

not want to connect to any devices but wants only to broadcast the internally stored information to other peripheral devices. In this method, the observer passively listens to BLE devices in its surrounding. After it receives the advertising packets from the broadcaster, it processes the data from the received packets. A device can transmit data to more than one peer at a time by only using non-connectable advertisings [13]. Figure 10 shows a network where a device is broadcasting the non-connectable advertising packets to the observer devices.

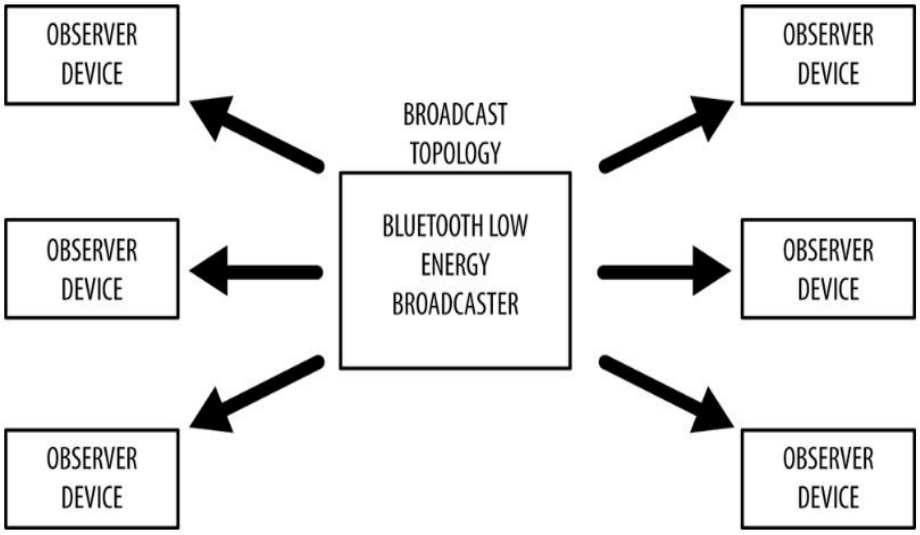

Figure 10: BLE Broadcast Topology [13]

Connection

In the connection method, the link is permanent that means a connection is established between two devices and periodical data exchanges occur between them unless they are disconnected. In this thesis, the BLE devices are communicating with each other using this method. The master scans the area for connectable advertising packets. The packets are connectable or non-connectable based on the value of the packet data unit or PDU field in the BLE advertising packets. The connectable advertising packets type allows the device to both transmit and receive the information. This advertising packet can be both directed and undirected. The direct

method is used when a device needs to connect quickly with another device. The directed advertisements have an intended receiving device, while undirected advertisements do not have a specific intended receiver.

The central device initiates the connection in this topology. It commands almost every aspect of the connection. It acts as an initiator in the SMP level, and thus commands the beginning of all SMP procedures [13]. The SMP stands for Security Manager Protocol and is responsible for functions like device authentication, authorization, integrity, etc. In this method, the pairing device is not allowed to make any decisions regarding the connection or pairing. The pairing device sits there advertising the packets and accepts all the incoming configurations about the connection from the central device. The peripheral always follows the center's timing and commands to establish the connection. Figure 11 shows a connectable advertising packets network where the central device is establishing the connections with the peripheral devices which are advertising the connectable advertising packets in the network. Unlike in broadcasting, Figure 11 shows that communication is both ways in connectable advertising packet method.

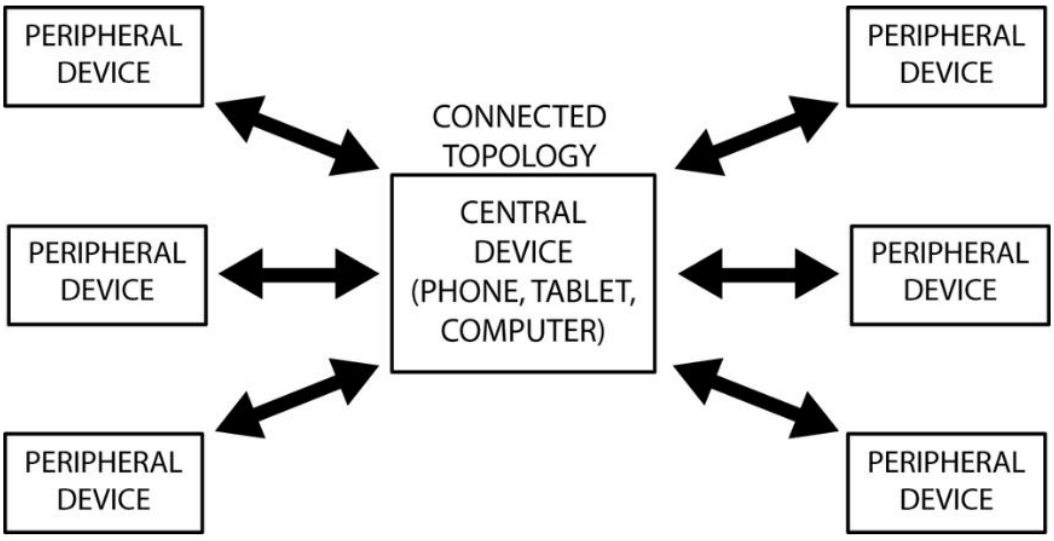

Figure 11: BLE Connected Topology [13]

BLE Architecture

BLE, like many other wireless technologies has some protocol layers, and each layer has its purpose and plays a significant role in BLE communication. The main three building blocks of a BLE device are Application, Host, and Controller. Figure 12 shows the general architecture of BLE.

Application

The application layer is the top most layer and is responsible for user interface and data handling logic. The architecture of this layer highly depends upon the project developed with BLE.

Host

The host layer in BLE provides information resources, services, and applications to the user or other nodes. It consists of the upper layers of the Bluetooth protocol stack. This layer consists of many sub layers as shown in Figure 12.

- Generic Access Profile (GAP)

  This layer controls the advertising and connections between the devices. It specifies how devices will implement control procedures such as discovery, connection, etc. The discovery procedure handles the discovery of services, characteristics, and attributes on the remote GATT server on the connected device. The remote GATT server means GATT layer of the connected server device. The main focus of this layer is to control the roles and interaction between devices.

- Generic Attribute Profile (GATT)

  This layer defines how the data is organized and exchanged between devices. The data in GATT is organized in Services, which contains one or more characteristics. Each

characteristic is a combination of user data along with the descriptive information about the characteristic itself. GATT services are organized in GATT profiles. Each service and characteristic has their 16-bit UUID which distinguish it from other services and characteristics respectively.

- Security Manager Protocol (SMP): It defines the procedures and behaviors to manage pairing, authentication, and encryption between the devices.
- Logical Link Control Adaptation Protocol (L2CAP): It is mainly responsible for two tasks:
  - It takes multiple data from upper layers and encapsulates them into the standard BLE packet format and vice versa.
  - It also breaks the large packets from upper layers into chunks that fit into the maximum payload size and also recombines the fragmented chunk of received data into a single large packet.

Controller

- Host Controller Interface (HCI): It implements commands, event, and data interface that allow upper layers such as GAP to access link layer.
- Link Layer (LL): This layer manages the physical BLE connection between devices.
- Physical Layer (PHY): This layer contains the circuitry that manages the modulating and demodulating analog signals and transforming into digital symbols. For this BLE uses a technique known as frequency hopping spread spectrum. In frequency hopping spread spectrum, the carrier frequency is rapidly switched among many frequency channels. The available frequency channel is sub divided into many sub-frequencies. Signals rapidly hop among these in a predetermined order to transmit the radio signals [14].

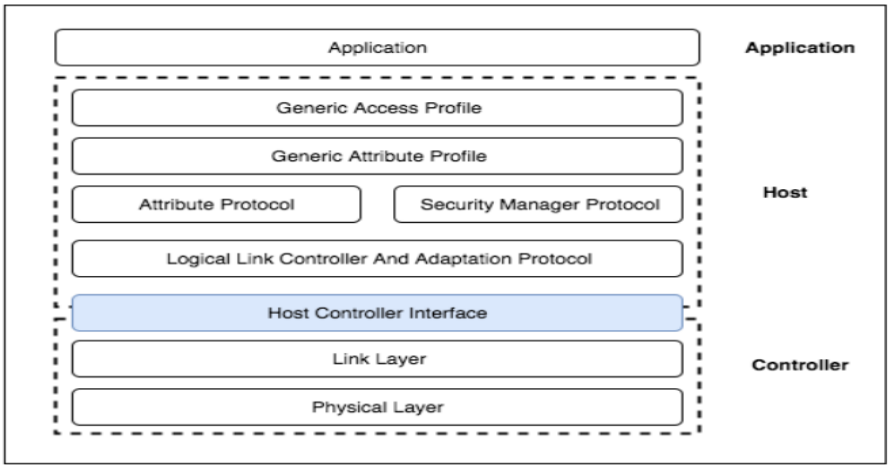

Figure 12: BLE Architecture [14]

The Generic Access Profile (GAP) and Generic Attribute (GAAT) are the most two important layers of BLE Architecture when establishing the communication between client and server in ESP32 BLE, so more detailed description of these two layers is given below:

Generic Access Profile (GAP)

The GAP is responsible for the discovery process, device management and the establishment of device connection between BLE devices. The BLE GAP is implemented in the form of API calls and Event returns [11]. Figure 13 shows the state transitions among GAP roles. The four main responsibilities of GAP for a BLE device are:

- Broadcaster: A broadcaster is a device which sends the advertising packets so that it can be discovered by the observers. This device is only responsible for broadcasting not connecting.
- Observer: An observer is a device that scans for broadcasters. This device can only send scan requests.

- Peripheral: It is a device that advertises the connecting advertising packets. This device becomes a slave once the connection is made. In this thesis, the client was configured in this role.
- Central: A central is a device that initiates connections to peripherals and becomes master once the connection is made. In this thesis, the server was configured as central.

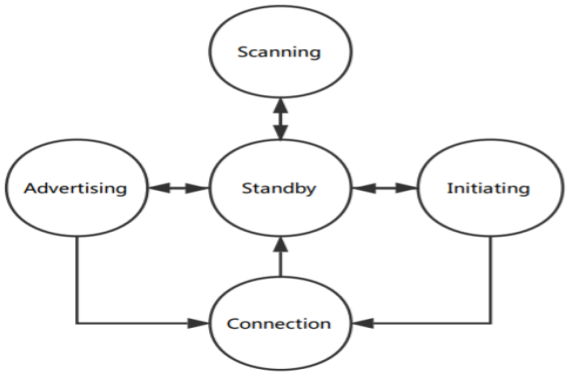

Figure 13: Status Transitions among GAPs [11]

BLE Modes

- Connectable Scannable Undirected Mode:

  In Connectable Scannable Undirected mode, a device can be discovered by and connected to any device. In this mode, the local device replies with a scan response, when a peer device sends a scan request. The packet format for this mode contains 6 bytes of the broadcast address and 0 – 31 bytes of broadcast packet data. In this thesis, ESP32 modules were configured in this mode.
- High Duty Cycle Directed Mode and Connectable Low Duty Cycle Directed Mode: In this mode, the broadcaster can only connect to the designated devices. The packet

contains 6 bytes of the broadcasting device's address and 6 bytes of the receiving device's address.

- Scannable Undirected Mode: In the Scannable Undirected mode, a device can be discovered by any other device, but it cannot get connected to it. The packet format for this mode is similar to Connectable Scannable Undirected Mode.
- Non-connectable Undirected Mode: In this mode, a device can be discovered by any device, but neither can be connected to or scanned by any other device. An unscannable device is a device that will not reply with any scan response to the peer device. The packet contains 6 bytes of broadcast address and 0 – 31 bytes of broadcast packet data.

Generic Attribute [GATT]

The data inside the BLE architecture are in the form of Attributes which consist of four basic elements:

- Attribute Handler: This helps us to locate any attribute, like using an address to locate data in the memory.
- Attribute Type: This element identifies the type of information contained by the data. A 16-bit or 128-bit number, known as UUID is used to identify the type of attribute.
- Attribute value: The attribute value is the key information of each attribute. The length of the attribute types can be fixed or variable.
- Attribute Permission: This element determines whether the information contained by the attribute can only be read or written.

A server can send data to the client in two ways. The server can send data by using an Indication or Notification, and a client can also obtain data from the server by initiating a Read Request. A client sends data to a server by writing the data to the characteristic of the server. The Write

Request and Write Command are used to do write operation in the server. However, a Write Response is only promoted when a Write Request is used [11]. Indication is similar to Write Request and Notification is similar to Write Command. Figure 14 shows the common operations that happen between a Server and a Client during data exchange.

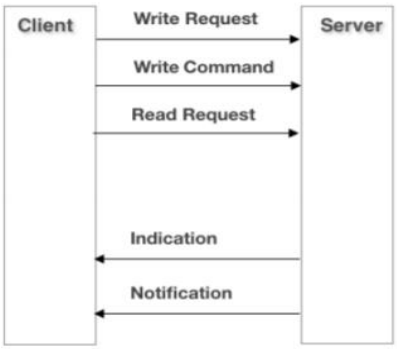

Figure 14: Common Operations between a Server and a Client [11]

The Attribute gives the minimum data storage unit in the BLE architecture, while the GATT defines how to represent the data set using its attributes and descriptors, how to aggregate similar data into a service, and how to find out what services and data a peer device owns [11]. GATT introduces the concept of Characteristics about the information. A characteristic can be categorized into three basic parts:

- Characteristic Declaration: It is the beginning of the characteristics and informs the peer device that the content followed by it is the characteristic value. The write and read properties of a data set are also included in the declaration.
- Characteristic Value: It is the main part of a characteristic which contains the most important information of a characteristic.
- Descriptor: Descriptors further describe the characteristic. A characteristic can have single, multiple or no descriptors.

In BLE, the similar functions are grouped together in the services. Figure 15 shows the way services, characteristics and descriptors are defined in the attribute table. Figure 15 indicates that a service can have one or more characteristics and each characteristic includes zero or multiple descriptors.

Attribute Handle	Attribute Type
0x0001	Service 1
0x0002	Characteristic Declaration 1
0x0003	Characteristic Value 1
0x0004	Descriptor 1
0x0005	Characteristic Declaration 2
0x0006	Characteristic Value 2
0x0007	Descriptor 2
0x0008	Descriptor 3
0x0009	Service 2
.....	.....

Figure 15: The Definition Table of service [11]

1.3.1.3.2 ESP-Now

ESP- Now is a proprietary wireless technology developed by Espressif systems that can be implemented in ESP32 development board. ESP-Now is a Wi-Fi direct application. Like in Wi-Fi direct, the AP or client roles are dynamic in the ESP-Now. ESP-Now does not require access point or router to establish the connection. It uses the concept of soft-AP to negotiate the roles of the pair to pair devices when they discover each other.

ESP-Now uses the same frequency and channels as Wi-Fi does. On the 2.4 GHz band in North America, there are 11 channels of 22 MHz present in a Wi-Fi frequency band [15]. Figure 16 shows the Wi-Fi channels on the frequency band. Some or all channels from 12 to 14 are allowed in some other countries. The center frequencies of channels from 1-13 differ by only 5 MHz.

Hence, there are only three non-overlapping channels in a Wi-Fi frequency band. The bandwidth of the single channel in Wi-Fi is 22 MHz. The maximum throughput that can be achieved in ESP32 Wi-Fi channel is 30 Mbit/s [16]. Thus, using Equation 1 the spectral efficiency of ESP-Now is 1.36 bits/s/Hz.

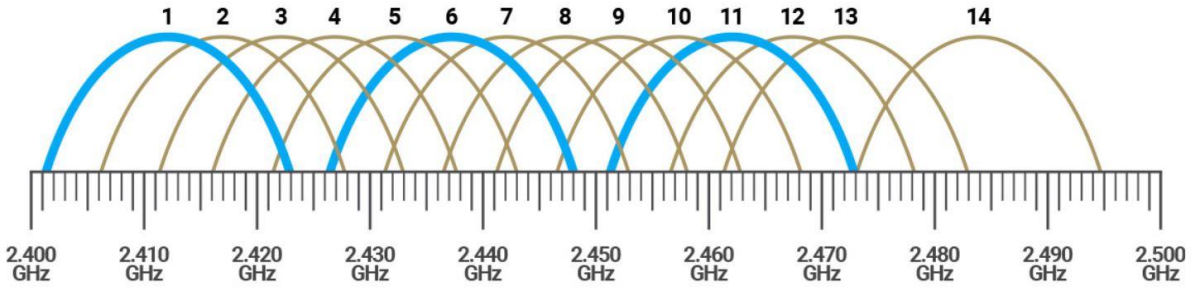

Figure 16: Wi-Fi Channels on the 2.4 GHz Frequency band [15]

ESP32 Wi-Fi driver supports IEEE-802.11b and IEEE-802.11g standards to configure the protocol mode [16]. IEEE-802.11b/g operate in the 2.4 GHz frequency band. This frequency band contains only three non-overlapping channels for the communication, and it is recommended to use these channels for communication. But nowadays, the density of nodes in the available frequency band is very high, and there is always some risk of interference from other radio networks.

IEEE 802.11 has provided two operating modes: Distributed Coordination Function (DCF) and Point Coordination Function (PCF) to coordinate communication in the network [17]. In the DCF operating mode, the channel claim process begins when a station senses the channel to determine whether it is free or not. For this DCF utilizes two techniques: DCF *Inter-frame Space* (DIFS) and the backoff algorithm [18]. DIFS is the time period where a channel should be idle before transmission from the station can begin. If the medium is sensed busy for that time period, then the transmission is delayed until the channel is idle again. In this case, a backoff interval

(BI) is randomly selected. The BI interval is in between a minimum contention period and a maximum contention period. The difference between these periods is known as the contention window (CW) [18]. There still is a chance of collision happening if two or more stations select the same backoff interval for the channel. This problem is mitigated by doubling CW every time a collision occurs [18]. When a station enters the backoff, the station monitors the channel as before. This backoff timer is decreased as long as the channel is idle. When the transmission is in progress, this timer is stopped and reactivated when the channel becomes idle again. Point Coordination Function (PCF) uses Access Point to coordinate the communication within the network. The AP waits for the PCF *Inter-frame Space* (PIFS) duration to grasp the channel. PIFS duration is less than DIFS duration. Thus this method always has the priority to access the channel. PCF is not that common and is only implemented in some hardwares since it is not part of Wi-Fi Alliance's interoperability standard.

There are mainly two kinds of interference that can occur in a Wi-Fi channel: co-channel Interference (CCI) and adjacent channel interference (ACI). The CCI occurs when transmission occurs on the same frequency in the same area. The ACI occurs when transmissions are sent on the adjacent or partially overlapping channels. These interferences are decreased by different operating modes of IEEE 802.11 MAC procedure. Besides these modes, the IEEE 802.11 standard has also defined the transmit spectrum mask intended to limit the energy of the transmitted signal. This method is used to limit the invasion of the signal on adjacent channels around the central frequency. The IEEE 802.11 standard also says that bandpass filters should be applied in both transmitter and receiver, to isolate the desired signal before transmission and reception [17]. This method decreases the energy received from another adjacent channel. In ESP32 this filter can be enabled or disabled by using the *channel_filter_en* parameter.

One of the limitations of ESP32 is that it is limited to only one channel at a time. So, when there are soft-AP and station mode, the soft-AP device always adjusts its channel to the channel of ESP32 station device.

A wireless router or any other access point can handle up to 250 devices at once. This is just a theoretical value, practically it is not feasible to connect that many devices to a single access point at once. If there are many clients accessing the same access point, then there always is a risk of data collision and interference. Thus, normally many routers are configured in such a way that the maximum number of devices that can be connected is 50 [19]. Whereas, the maximum number of peers that can be supported by ESP-Now is less than 20; this includes both encrypted and unencrypted peers [16].

ESP-Now requires an initial pairing between the devices. Once pairing is established the connection is persistent peer-to-peer. In an ESP-Now communication, the wireless module acting as Master or Controller uses MAC address to pair with the slave device. A MAC address is a unique identifier of a network hardware interface which is used for communicating on the physical network. The MAC address of the slave can be set in the program itself using ESP32's Wi-Fi attributes. Further details on the MAC address are described in section 3.2.3**.** Up to 20 peer devices can be as paired devices.

Once the devices are paired with each other, the data can be sent from master to slave or vice versa. ESP-Now uses callback functions to indicate the transmission and reception of the data. A callback function is a function that is passed into another function as an argument, which is then invoked inside the outer function when some event or routine occurs. The call back functions used in ESP-Now for transmission and reception are *OnDataSent()* and *OnDataRecv().* These functions are passed as an argument into register functions – *esp_now_register_send_cb()* and

esp_now-register_rec_cb() respectively. The callback functions are defined as pointer type in ESP-Now, and they are linked or registered with another pointer inside the register functions. When ESP-Now is initialized, the Wi-Fi task starts running. In an RTOS environment of ESP32, each task like Wi-Fi task is allocated its stack memory and priority. These stacks are added to the scheduler, which runs one task at a time. The scheduler runs these tasks along with Wi-Fi task based on their priority. The task with the highest priority runs first. Each task is provided a certain time slice to run. If the task is not completed within the time slice, the scheduler saves the context of the task into the stack and schedules the next task in the queue. In this way, RTOS of ESP32 has a scheduler which continuously runs all the tasks including Wi-Fi task one at a time. When the transmitter sends data to the receiver, the *OnDataSent()* is called using the linked pointer that was registered inside the *esp_now_register_send_cb()* function. Similarly, in case of the reception of new data, the pointer to the *OnDataRecv()* is called using the linked pointer that was registered inside the *esp_now-register_rec_cb()* function. The arguments that are received form the *OnDataSent()* call back function are MAC address and status of the transmission. Whereas in the case of *OnDataRecv(),* the arguments are MAC address, the pointer to the received data buffer and the length of received data in terms of bytes. These arguments are used to retrieve the status of transmission in the transmitter and the received data in the receiver.

1.3.2 Software Description

Two different IDEs were used in this thesis, Code Composer Studio was used to implement the ShockBurst Feature in nRG24L01+, and for ESP32 Arduino IDE was used.

1.3.2.1 Code Composer Studio

Code Composer Studio is an integrated development environment (IDE) that supports TI's Microcontroller and Processors portfolio [20]. Code Composer consists of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features [20]. It is designed for embedded project design and low level JTAG based debugging.

EnergyTrace

EnergyTrace Technology is an energy based code analysis tool that measures and displays the application's energy profile and helps to optimize it for ultra-low power consumption [21] . This technology implements a new method to measure the MCU current consumption. Traditionally power is measured by amplifying the signal and measuring the current consumption and voltage drop over a shunt resistor. But in EnergyTrace debugger generates target power supply by a software controlled DC-DC converter. The energy consumption of the target microcontroller is equal to the time density of the DC-DC converter charge pulses. A built-in calibration circuit gives the energy equivalent for a single charge pulse. The width of each pulse remains constant. So, the debugger can just count the number of pulses over a period of time and calculate the average current consumption. The time periods with a small number of charge pulses indicate low energy consumption while the higher one indicates high energy consumption. The main advantage of using continuous sampling is that even the shortest device activity that consumes

energy contributes to overall recorded energy. This is not possible using the traditional shunt based measurement system.

In this thesis, this feature is used to measure the current consumption by nRF24L01+. At first, the current consumption of the only MSP432 was measured using EnergyTrace. After that, the nRF24L01+ was connected and the communication channel was established. This feature gives us three different current values – average, minimum and maximum values. Now when nRF24L01+ was transmitting or receiving packets continuously, the EnergyTrace feature was run for certain duration of time, and current consumed by the device was measured. As nRF24L01+ consumes maximum current in TX or RX mode, the maximum current value from EnergyTrace was taken and subtracted from the measured current value by only MSP432 to get the final current consumption value of nRF24L01+ during transmission and reception.

1.3.2.2 Arduino IDE

Arduino IDE is a cross platform application written in JAVA that supports the languages C and C++ using special rules of code structuring. The user written code in Arduino requires two functions, one is to start the sketch, and another one is the main program loop. These functions are compiled and linked with a program sub *main()* into an executable program with the GNU tool chain .

Generally, Arduino IDE is only used for Arduino Boards. To make it work with ESP32, an Espressif folder should be added and cloned into the Arduino sketchbook directory. Espressif has its own IDE which can be used to program the ESP32. But with Espressif IDE the programming is complex with all glue codes needed for event handling and processing. Whereas with Arduino IDE, there already exits a library which contains all the classes in C++ that handle those complex event handling and processing necessary for BLE and ESP-Now implementation. Also, more

work has been done in Arduino IDE implementation of BLE and ESP-Now in ESP32 than in Espressif IDE. Thus, in thesis Arduino IDE was chosen over Espressif IDE because of simplicity and availability of more resources.

1.4 Purpose

The purpose of thesis is to implement nRF24L01+ ShockBurst feature, Bluetooth Low Energy and ESP-Now in a star network configuration, and then measure and compare the performance metrics of these protocols in the wireless network. The performance metrics that will be compared are Throughput, Transmission Range, Current Consumption and Network Recovery Time. The main goal of this comparison is to find the advantages and disadvantages of these three wireless protocols based on the four network performance metrics. The comparison will also be done within nRf24L01+ only, by measuring the Throughput, Range and Current Consumption in different data rates and power settings. The nRF24L01+ will be interfaced with MSP432 microcontroller, while BLE and ESP-Now will be implemented in ESP32 development board to realize the star connection.

1.5 Thesis Organization

This thesis is divided into total six sections. The section 1 has provided background for understanding the problem studied in this thesis. It has introduced the wireless devices, and protocols that are used to implement the networks compared in this thesis. This section also gives the information about the hardware and software tools used in this thesis, and also defines the research goals. Section 2 provides the review of Literatures which have compared the network metrics of different wireless technologies with each other. Section 3 explains the procedures involved in the design and implementation of the system. In this section the implementation of

all three protocols in one to one and star network is described. Section 4 provides the results obtained from the experiments while the analysis of the results and comparison of network metrics between the protocols is done in Section 5. Section 6 outlines the conclusion made from the experiments and also provides insight into the probable future work.

2 Review of Literature

Various technologies communicate in different ways. The difference lies in the quality of service and some considerations related to the application and its environment. Past research has examined the performance analysis of wireless communication protocols based on network metrics like transmission time, transmission power, range and energy consumption as seen in Saad, Mostafa, Cheikh and Abderrahmane's research study [22]. The authors compared and studied the network metrics of Bluetooth, Wi-Fi, ZigBee and GPRS protocols. The transmission time for the GPRS was found to be longer than other protocols in this study. Similarly, the authors found that the range of ZigBee, Wi-Fi and Bluetooth was almost same, and greater than that of GPRS. While in terms of energy consumption the authors noted that the ZigBee and Bluetooth have lower power consumption compared to that of Wi-Fi and GPRS. Saha, Mandal, Mitra and Banerjee [23] also did the comparative performance analysis of mesh recovery time, and energy consumption between nRF24L01+ and XBEE ZB. They found that the nRF24L01+ had better power consumption and mesh recovery time compared to XBEE ZB.

The throughput of the Wi-Fi was found to be better than Bluetooth and ZigBee by Lee, Su and Shen in their research study [24]. Besides throughput, the authors also studied power consumption and network capacity among these three protocols. They observed that the Bluetooth and ZigBee have low power consumption capability than Wi-Fi, and the network size of ZigBee in a star network is greater than Wi-Fi and Bluetooth protocols. The same outcome for throughput and energy consumption between Wi-Fi, ZigBee and Bluetooth was also found by Arefin, Ali and Haque [25] , and they also suggest that the Packet Delivery Ratio (PDR) of Wi-Fi is better than Bluetooth and ZigBee, and PDR decreases as the number of nodes increases in

all three protocols. The network metric Packet Delivery Ratio is not being considered during the comparison in this thesis. After an overview of the research studies that have been done, the network metrics: throughput, range, power consumption and network recovery time were chosen for comparison. The brief description of these network metrics is given in section 1.2.1.

In present world, there are different wireless communication modules and technologies available in the market. Among this vast spectrum of modules, the nRF24L01+ transceiver developed by Nordic Semiconductor and ESP32 development board developed by Espressif System are gaining popularity in local wireless sensor network. The nRF24L01+ chip has been immensely popular in the application of local wireless sensor network system due to the factors like low cost, low power, ease of use and availability. In addition to these factors, the proprietary communication protocol Enhanced ShockBurst Feature provides some useful features that help in establishing more reliable wireless communication. These features of Enhanced ShockBurst Protocol are discussed in detail in section 1.3.1.2.1. Chen, Hu and Liao [26] studied the performance of nRF24L01+ in a Wireless Body Area Network (WBAN) system. The authors developed a non-standard wireless communication protocol for WBAN system using nRF24L01+ transceivers in a star topology using some features from Enhanced ShockBurst Feature protocol. From the implementation the authors concluded that then nRF24L01+ is a good fit for WBAN system applications due to higher data rate, low power and low cost factors. The nRF24L01+ has also been used along with MSP430 to design a Wireless Sensor Node (WSN) by Sonavane, Patil and Kumar [27] in their research study. In the study, the authors have described the design procedures involved in WSN node, and have calculated the power consumption, range and packet loss of the designed WSN node. From their implementation, the authors concluded that the nRF24L01+ is good choice for a local wireless sensor network due to

its low cost, low power and multiple data rates features. Besides this, the authors found out that the packet loss of designed WSN node was about 0.1 % and it increased with the increment in the range. Also, they concluded that the power consumption of the node dependent upon the power and data rate settings in nRF24L01+ transmitter. The nRF24L01+ transceiver has also been compared with other transmission modules such as CC1101 and Wi-Fi module by Wang, Chu and Ren [28]. The comparison was done based on the power consumption and Data Error Rate (DER) properties to find out the cost-effective modules in different applications. Through the experiments and analysis of the wireless transmission modules, the authors concluded that the nRF24L01+ and CC1101 are more appropriate for low power and low cost applications. Also, the authors noted that the DER of CC1101 wireless transmission module is superior to the nRF24L01+ and Wi-Fi modules.

ESP32 is a new development board which is especially designed for the IoT application. It was released in late 2016, and since its release, it has become immensely popular in the IoT applications. Maier, Sharp and Vagapov [29] did the comparative analysis of the ESP32 with ESP8266, CC32 and XBEE in their research paper. The authors did the comparison of different features of these wireless modules based on the different features and functions provided on the technical data sheet of each module. Unlike other studies, this comparison was based on the theoretical values provided on the technical data sheets. Also, this comparison was more focused on the technical details of the modules such as CPU, SRAM, I/O, and other details rather than the network properties of the communication protocols in each module. Since the ESP32 was released recently, much research on the properties of communication protocols available in ESP32 has not been done yet. ESP32 device supports standard communication protocols like Wi-Fi, Bluetooth, and Bluetooth Low Energy. Besides these standard communication protocols, this

device also supports another nonstandard communication protocol known as ESP-Now. The nonstandard communication protocol ESP-Now is an application of Wi-Fi direct technology that allows the Wi-Fi enabled devices to connect with each other without the requirements of router or access point. Due to this feature nowadays, many IoT applications are using Wi-Fi direct technology over Wi-Fi for local wireless communication. Also, recently BLE has been replacing the classical Bluetooth in many wireless sensor network applications due to low cost and low power features. Therefore, ESP-Now and BLE were chosen over Wi-Fi and Classical Bluetooth for performance comparison in ESP32.

During the review of literature, it was found that no significant comparison of the network constraints such as throughput, power consumption, network recovery time and range between the ESP-Now and BLE implemented in ESP32 has been done yet. The published throughput values, range values and current consumption values in a datasheet or manual for different protocols are theoretical values. In reality, there are different factors which affect the actual value of a network metric. Thus, the measurement of actual values for these four network metrics and comparing three technologies was also one of the motivations behind this thesis.

3 Design and Implementation of the System

3.1 Experimental Setup

A simple connection between nRF24L01+ and MSP432 was made to establish the wireless connection. Figure 17 shows the connection between two devices along with the pin configuration.

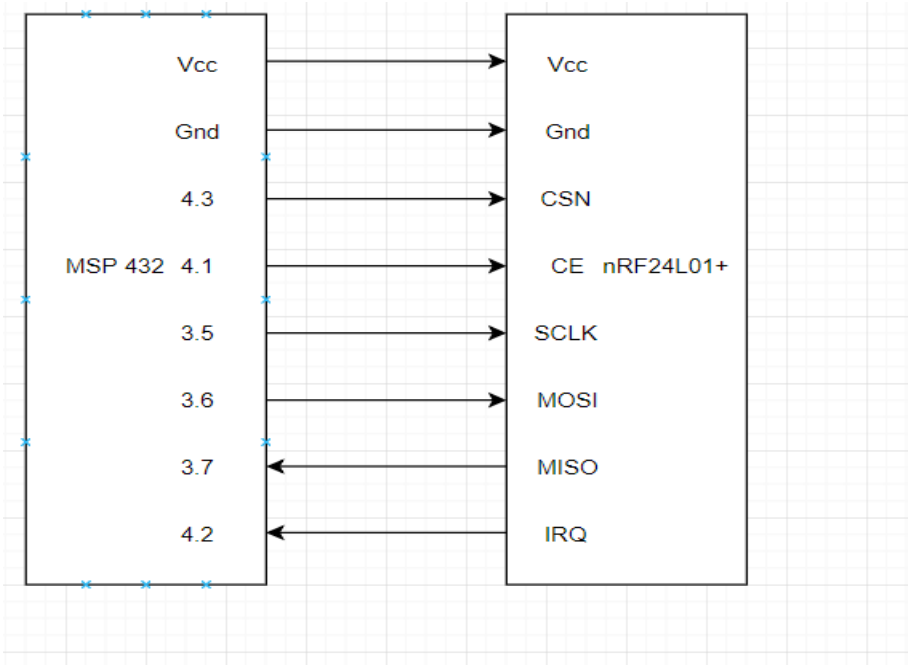

Figure 17: MSP432 Connection with nRF24L01+

The nRF24L01+ communicates with the MSP432 through the SPI protocol. Serial Peripheral Interface (SPI) is a synchronous serial communication interface that is commonly used to send data between microcontrollers and sensors, modules, etc. It uses a separate clock signal and data lines for communication. For example, in Figure 17, the signals MISO and MOSI are the dedicated data lines used for communication between MSP432 and nRF24L01+, and SCLK is the clock signal. MOSI is Master Out Slave In signal used to send the data from MSP432 to

nRF24L01+ and MISO is Master In Slave Out used for transmitting data from nRF24L01+ to MSP432. The MSP432 is a master and nRf24L01+ is a slave in the above design configuration. For that, GPIO pin 3.5, 3.6 and 3.7 were used as SCLK, MOSI and MISO respectively to establish the SPI connection. Besides this, pin 4.2 was used as the interrupt pin for nRF24L01+, and pins 4.3 and 4.1 were used as Chip Select Signal (CSN) and Chip Enable Signal (CE). Unlike nRF24L01+, ESP32 was used in a stand alone condition. Thus the only connection from ESP32 was to the laptop through a USB connection.

3.2 Software Flow

3.2.1 nRF24L01+

3.2.1.1 One to One Connection

The nRF24L01+s were configured as transmitter and receiver, and communication between them was established in this network configuration.

3.2.1.1.1 Transmitter

Figure 18 shows the program flow in transmitter side in nRF24L01+. The first task was to initialize the nRF24L01+ as a transmitter. For that initially, the CRC bit is enabled, the address width of the channel, the power and speed, and the channel are set up. During initialization, SPI transfer was enabled, pin 4.2 of MSP432 was configured as interrupt pin, and the auto re-transmit feature was enabled. Then the size of the data pipe was set up. For the static payload the packet size is always 32, but for dynamic payload, the packet size was initialized as zero, as there is no fixed packet size in dynamic payload feature.

After that, an address of 5-byte length was assigned to the pipe. One thing to remember is that the address in both PTX and PRX should be the same. All of these set up fall under the initialization of nRF24L01+.

After the initialization, the nRF24L01+ was put in standby condition unless the Chip Enable signal was enabled, and data was made available in TX FIFO. For that, the terminal Tera Term was used to provide the data to the chip. MSP432 serial port eUSCI_Ax was used to receive the data from the keyboard through the terminal. When the user entered the data in the terminal, then an Interrupt handler was generated, and the data was transmitted to the TX FIFO. After that, the Chip Enable signal was enabled for around 10 us. Once the required conditions for the initialization of TX were achieved, the TX was activated, and data was transmitted to the receiver. Now after the transmission of data, TX went into a receiving state to receive the acknowledgment packet from the receiver. Once the acknowledgment was received, an interrupt was triggered which indicated the successful transmission of data. Then, at last, the interrupt was cleared and nRF24L01+ went back to standby condition.

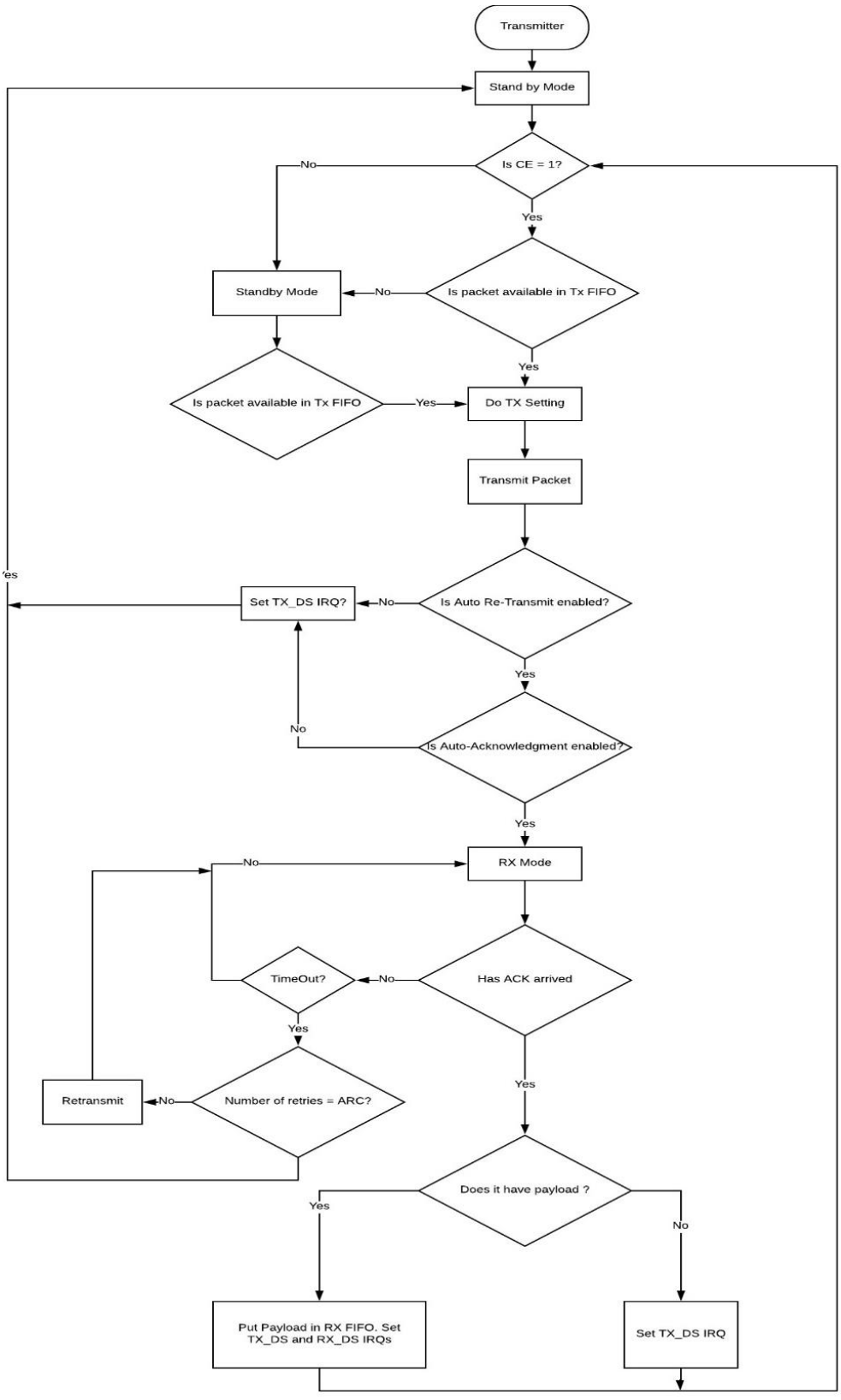

Figure 18: Transmitter Side in nRF24L01+

3.2.1.1.2 Receiver

Figure 19 shows the program flow in the receiving side of nRF24L01+ in Enhanced ShockBurst Protocol. The initialization process of the receiver was similar to the transmitter. The only significant difference was that the PRIM_RX is high for the receiver while it is low for the transmitter. The initialization parameters such as address width, speed, power, channel, pipe number, pipe size, and address were the same for both receiver and transmitter. As shown in Figure 19, the receiver stayed in standby mode after the initialization until the chip enable (CE) signal was activated. When the CE signal was activated, the receiver checked whether its FIFO was full or not. If the FIFO was empty, then the receiver started listening for the packets from the transmitter. When a packet was received, at first the PRX checked whether the auto acknowledgment feature was enabled or not. If it was enabled, PRX performed an error checking mechanism. If no error was found, the received packet was made available in RX FIFO. At last, the receiver transmitted the acknowledgment back to the transmitter and went back to receiving standby mode.

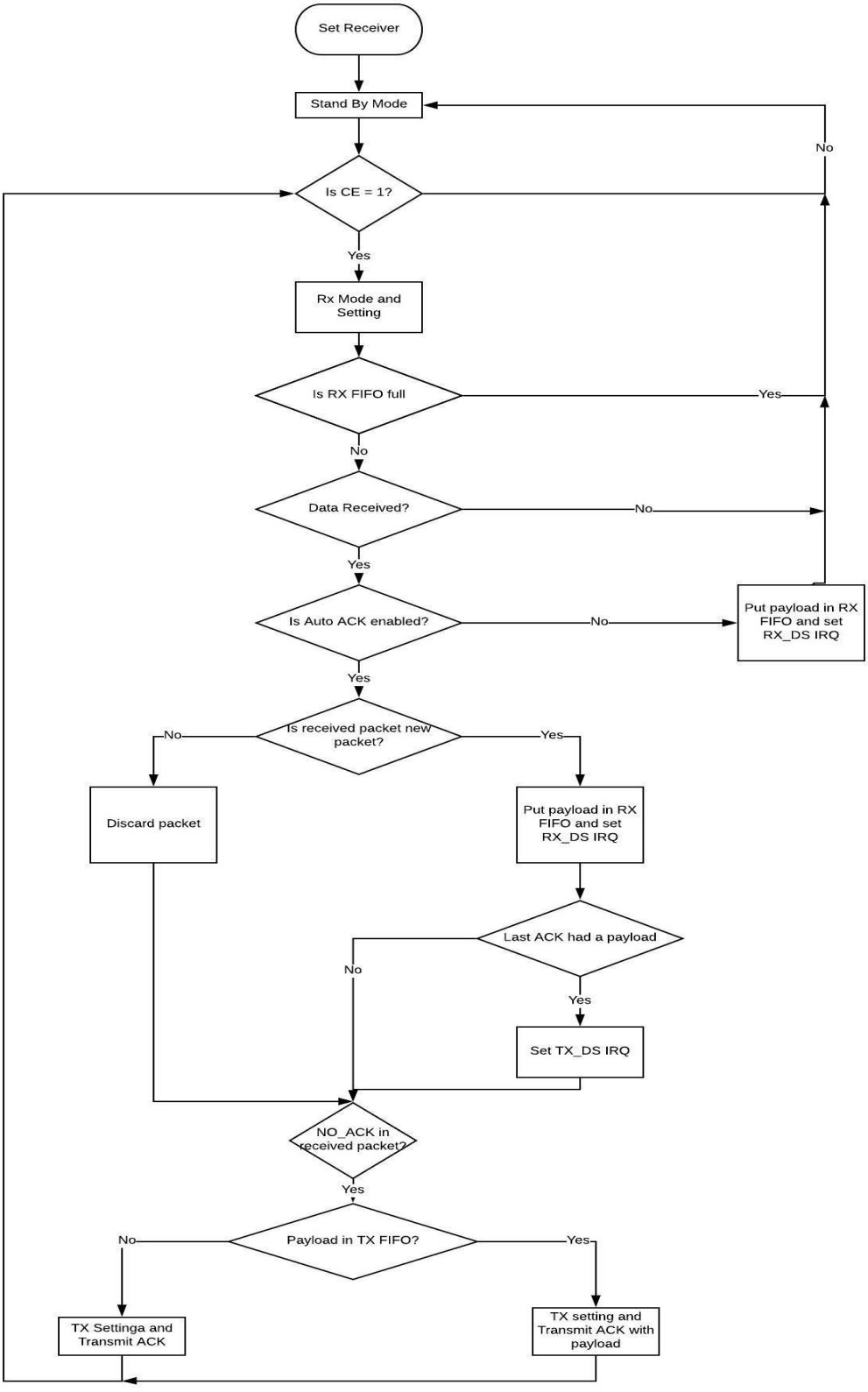

Figure 19: Receiver Side in nRF24L01+

3.2.1.2 Star Connection

Unlike one to one connection, where a radio module was acting either as a transmitter or receiver, every nRF24L01+ (both hub and nodes) was working as a transceiver. Star Network mainly consists of two parts: central hub and peripheral nodes.

3.2.1.2.1 Central Hub

The task of the central hub in the star network was to receive the data from a node and transmit it to the intended end node. Figure 20 shows the general program flow in nRF24L01+ in the central hub of a star network. The initial task was to initialize the nRF24L01+ as the receiver. During the initialization, all the pipes were opened in a similar configuration. And, the addressing of each pipe was done following the rules of Multiceiver as mentioned in section 1.3.1.2.1. After the initialization was completed, the hub was in the receiving state and started listening for the data from available nodes. Once the data arrived from a node, the next task for the central hub was to determine the intended end node receiver.

The standard nRF24L01+ packet does not provide any information about the intended receiving node or pipe to the central hub in a star network. So, in this thesis, the user was told to write pre-assigned device ID of the end node at the beginning of actual data before transmitting it. For example, let's assume that there are three nodes and the data need to be transmitted from node 1 to node 3. Now when the data arrives in central hub form node 1, the hub does not have the information regarding the intended end receiving node, in this case node 3. In this thesis, the pre-assigned devices IDs are numeric values, so device IDs for node one is 1; node two is 2 and node three is 3. Hence when the user wanted to send the data to node three, the user had to enter 3 at the beginning of the data.

When the data was received by the central hub, at first parsing of the received data was done to find the intended end node. To determine the end nod, the first character of payload was compared with the pre-defined device ID. Once the end node was located, the hub was reinitialized as transmitter and data was sent to the intended end node. When the hub received the acknowledgment back from the end node, the hub was reinitialized as a receiver and it continued listening for the data.

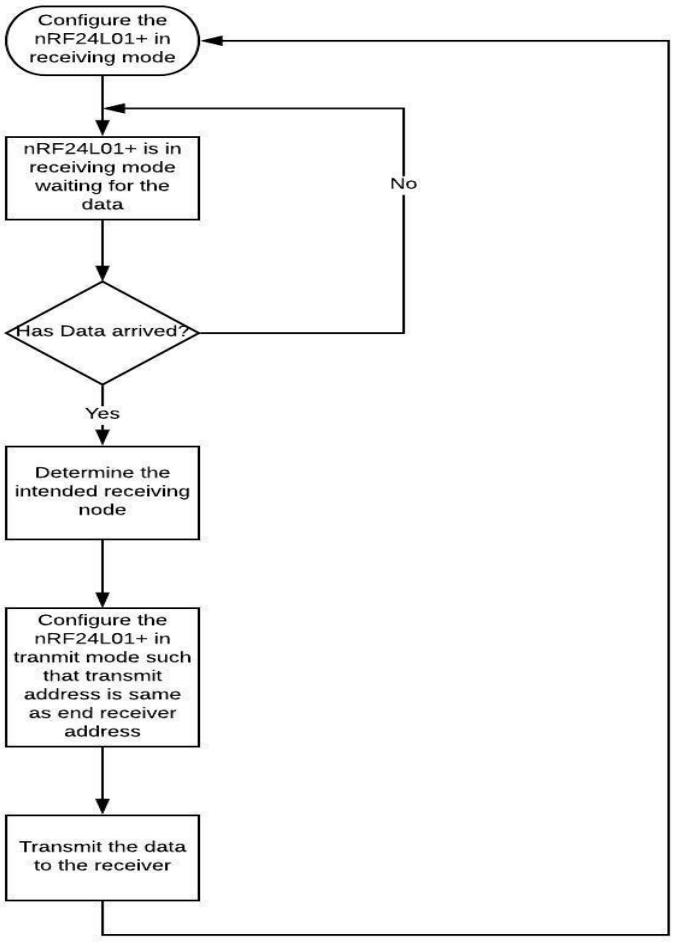

Figure 20: Central Hub in Star Connection

3.2.1.2.2 Peripheral Node

Figure 21 shows the program flow in peripheral node side in nRF24L01+ in star connection. Similar to the central hub, the nRF24L01+ in a peripheral mode was configured as a receiver at the beginning. It stayed in the receiver mode until an interrupt was generated from the Terminal. The interrupt from the Terminal indicated the node to reinitialize itself as a transmitter. This interrupt from the Terminal is not a part of a star network. This configuration was done only for testing of the communication between the nodes. Through the Terminal, the transmission and reception of data can be monitored with much more reliability. When the user entered the data along with the device ID in Terminal, the nRF24L01+ was initialized as a transmitter, and data was transmitted to a central hub. When it received an acknowledgment back from hub, it was reinitialized as a receiver and continued listening for the data from a central hub.

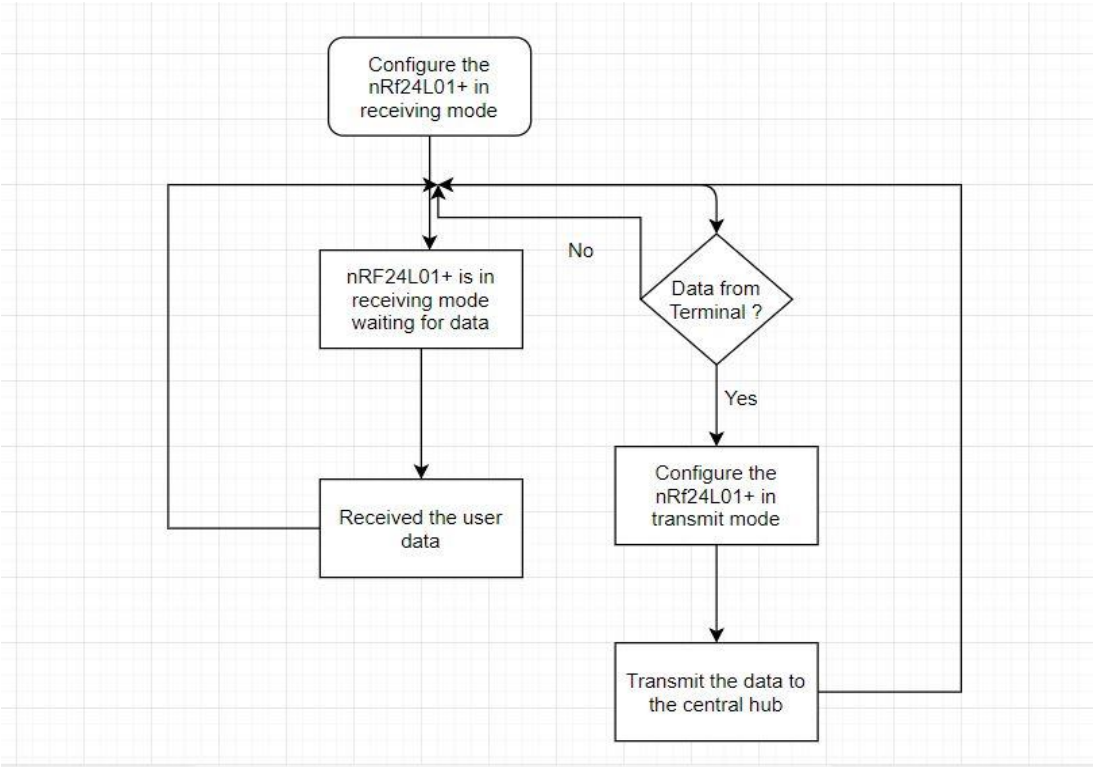

Figure 21: Peripheral node in nRF24L01+ in Star Connection

3.2.2 ESP32 BLE

The Bluetooth in ESP32 module was enabled using the Espressif software library. For that, Espressif IoT Development Framework (ESP-IDF) was downloaded from the GitHub. Next, the repository of ESP-IDF was cloned to the computer, and a path to IDF was entered. After correctly setting up ESP-IDF in the PC, menu- driven configuration feature of IDF was used to configure the ESP32 module. This ESP tool can be used to configure the serial COM port, baud rate and many other features. Not only BLE, other features of ESP32 like Wi-Fi, Ethernet can also be set using this tool.In the *menuconfig* tool component *config* was selected, and the Bluetooth option was enabled. The next step was to specify the size of the Bluetooth stack and select the Bluedroid Bluetooth stack to be enabled. These processes enabled the BLE in ESP32 board.

In this thesis, a library provided for implementation of BLE in Arduino was used. ESP32 BLE APIs require different complex and glue code necessary for event handling and processing of BLE requests. This library consists of different classes which handle those events handling process so that BLE workflow can be efficiently processed.

3.2.2.1 One to One Connection

The BLE implementation in one to one connection can be distinguished into two parts: BLE server and BLE client.

3.2.2.1.1 Server

Figure 22 shows the program in BLE server. In the beginning, the 128-bit long Service and Characteristic UUIDs were defined for the server. UUID stands for Universally Unique ID that is used to identify the services and characteristics. These UUIDs were used by the client side to

recognize the advertised services and characteristics of the server. The next task was to give a device name. During the implementation, it was found that the device name should not be more than five characters; otherwise the server could not connect to the client. Now after the above processes were done, the service and characteristics for the server were created by using the respective UUIDs. The characteristic can have read, write, notify or indicate properties. In this thesis, during the creation, the characteristic was initialized to have read and write features. After the creation was completed, the service was added into the advertisement payload, and finally, the advertisement was started.

The o*nRead* function was used to write the value in the server while *onWrite* was used to write the new value to the client. The *onRead* function was called upon when the client made a read request. This function enabled the client to read the server's characteristic value, whereas *onWrite* function was called upon when the client made a write request. This function allowed the server to write the value in the client side.

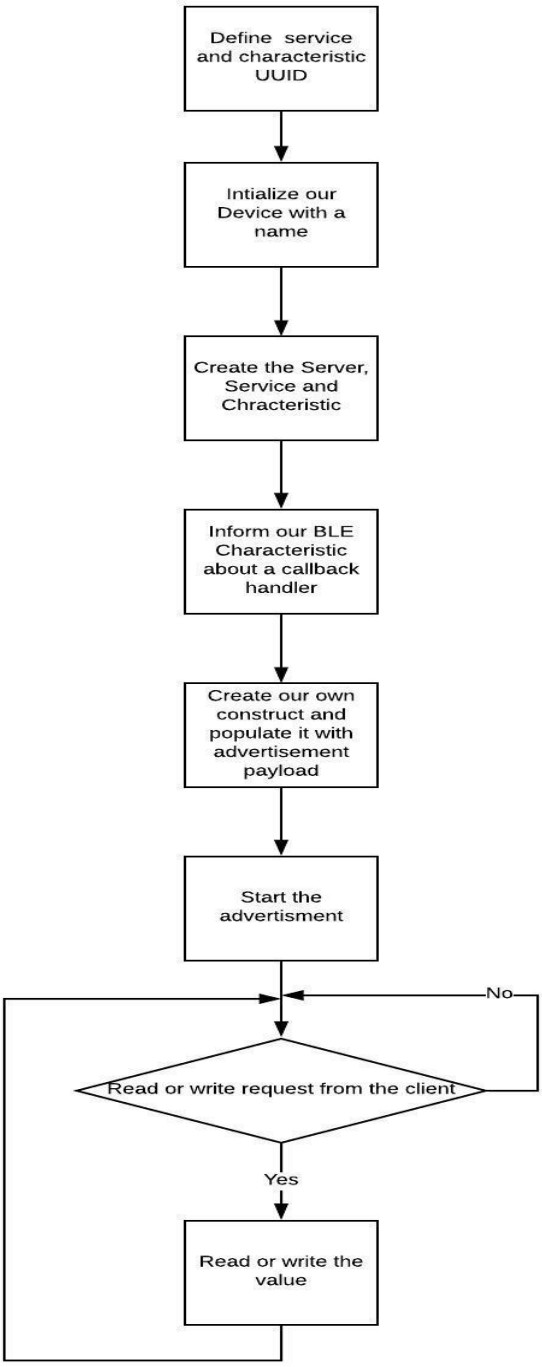

Figure 22: BLE Server

3.2.2.1.2 Client

The client operation can be broken down into two parts: Scanning and Interaction. Every BLE device has a unique address that can be used to identify it, and thus establish the connection. The address has a length of 6 bytes. This address can be hard coded into the code, but this is not the recommended way. Instead, a scanning procedure was performed to listen to the servers which are advertising. From scanning performance, the address of the server was obtained. The first task in client side was to define the UUIDs of the service and characteristic of the server; that the client was trying to connect. After that, the scanning was performed, and a callback function was called for each device that was found. Then the client was created, and a connection to the detected device was established. Once the connection was formed, the UUIDs of service and characteristic of the connected server were compared with the desired ones. If the UUIDs were found to be same, then either a read or write operation was performed. If the UUIDs were different, the client continued to search for the desired service and characteristic. Figure 23 shows the general flow of the program in the client side of BLE.

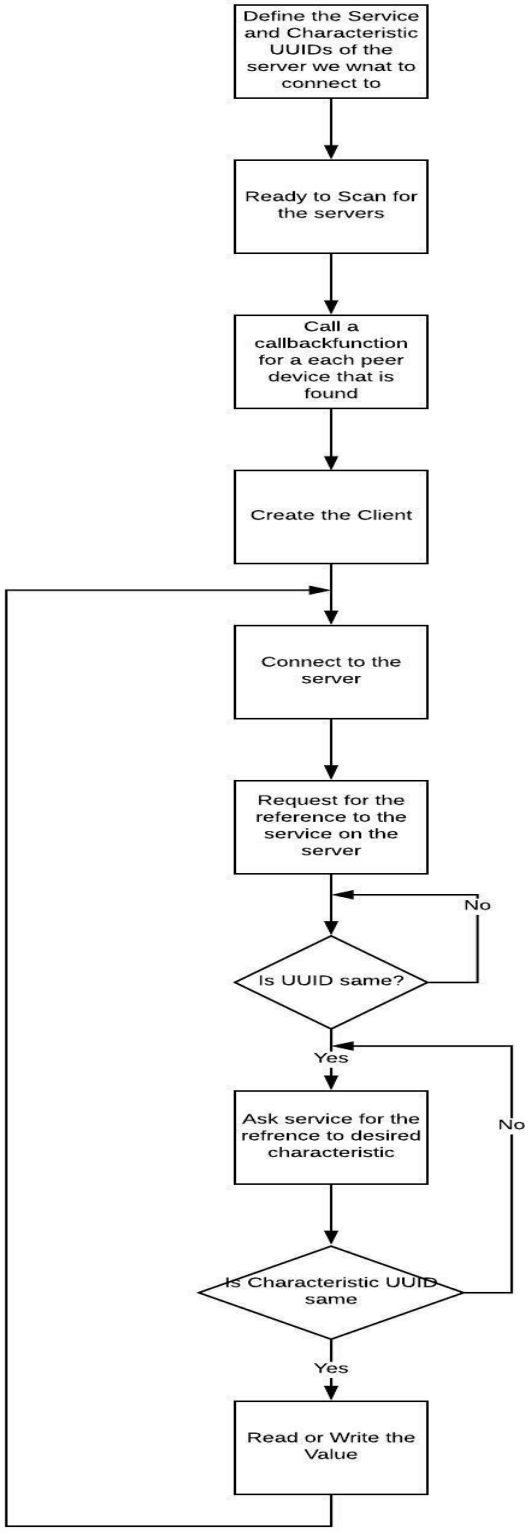

Figure 23: BLE Client

3.2.2.2 Star Connection

Like that in nRF24L01+, the ESP32 BLE in star topology also consisted of a central hub and peripheral nodes, and both nodes and hub were acting as a transceiver.

3.2.2.2.1 Central Hub

Figure 24 shows the program flow of BLE in the central hub in star connection. Initially, the central hub in the network was configured as a server. The current BLE library in Arduino does not support more than one client connection at a time. The multi-client connection in BLE in ESP32 was enabled only in the latest 30.2 version. This version was released on August 2018. So, the ESP32 library for BLE is still in the building phase for multi-client connection.

For this thesis, a multi-client connection feature was necessary to realize the star connection. In a star network, the central hub is connected to multiple peripheral nodes so that it can receive data from any one off them and transmit the received data to any peripheral node. Hence, to form a multi-client connection, the advertisement was restarted in the hub once there was a connection between a client and server as shown in Figure 24. This process did not work all the time, but most of the time multiple clients were able to connect with the hub. In the hub, only one service was created, but inside that numerous characteristics were created. The number of characteristics in the hub was dependent upon the number of peripheral nodes. Each characteristic was assigned a separate UUID. For example, if there were two peripheral nodes, two different characteristics were created.

At first, the callback functions for all characteristics were called in such a way that the central hub was acting as the receiver and was listening for a write request from all nodes. Now when the server received a write request from a node, the new value was written to the correct

characteristic. For example, if the data was received from node one, the new value was written to the characteristic of node 1.

The next task was to determine the intended end receiver or node. The processing for determining the end node is the same for all protocols. A pre-defined device ID was assigned to each node; the received value was compared to the pre-defined ID to determine the end node. When the central hub received data from the node, first a function was run to determine whether the received data was number or not. Since the pre-assigned device IDs are numbers in this thesis, the checking was done to differentiate between actual data and device ID. If it was found to be a number, then ESP32 entered a new state. In this state, the server was still acting as receiver and waited for the actual data from the node. In this thesis, the node was sending two different characters right one after another. This was done to calculate the throughput of the network. When the central hub received all the data from the node, then it went into a new state where it determined the end receiving node. Once the correct end node was figured out, then the central callback function for only that node was called, and data was written to the correct end node.

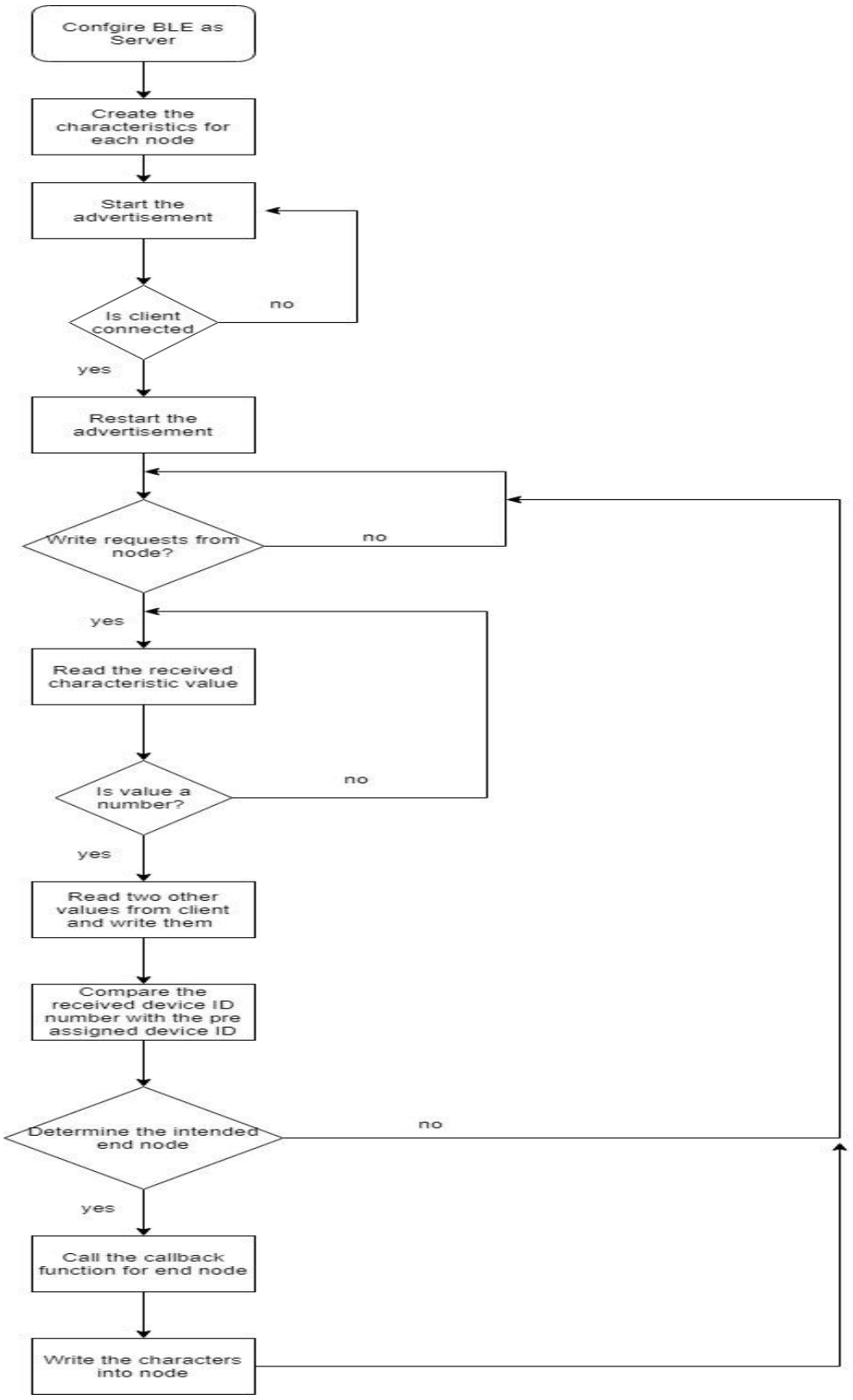

Figure 24: Central Hub in BLE in Star Connection

3.2.2.2.2 Peripheral Node

Figure 25 shows the program flow of BLE in a peripheral node in star connection. The first task was to configure the node as a client, after that scanning was performed to connect with the hub. Once the connection to the hub was established the node went to receiving state. In this state, receiver made a constant read request to read the data from hub. In the star connection, the communication is always initiated by the peripheral node. So, in the receiving state, there was another condition where the node was continually listening to the serial monitor for the data from the user. If the node received any value from the serial monitor or the user, the node changed its state from receiver to transmitter and made a write request to the hub. Once the write request was made, the new value was written to hub. Thus, in the receiving state ESP32 was performing two different tasks at the same time. As it was making a read request to the server, it was also listening to see if any data was coming from the serial monitor or not.

The data obtained from the serial monitor is the device ID of the intended end node. After writing operation of device ID was completed user defined data were written to the hub.

Once the all the data were written to the central hub the node came back to the receiving state.

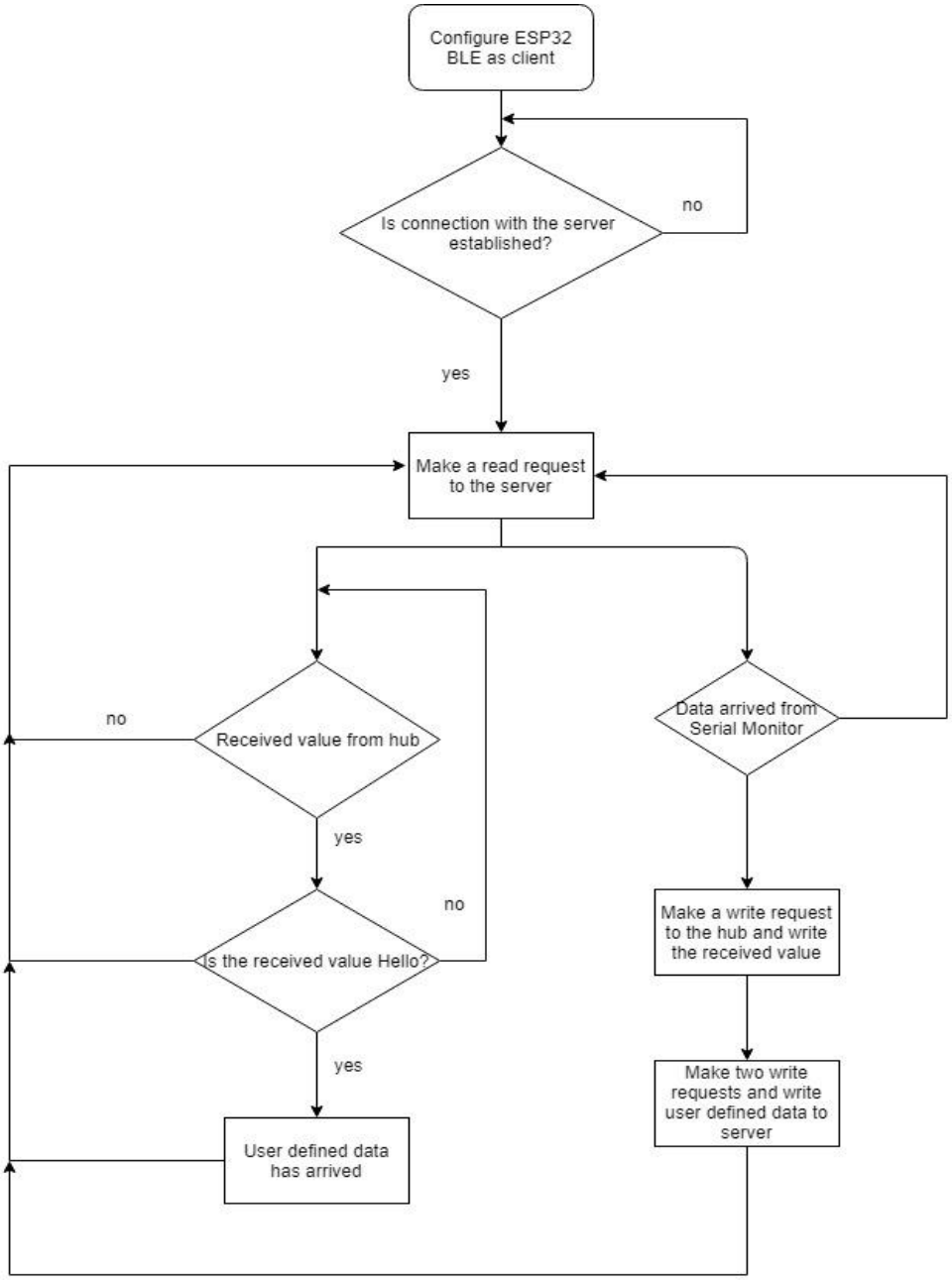

Figure 25: Peripheral Node in BLE in Star

3.2.3 ESP-Now

ESP-Now in this thesis is implemented using Wi-Fi protected setup standard. So, the Wi-Fi library for ESP32 was used to perform the ESP-Now wireless connection between two devices. This library provides different Wi-Fi functions which are used to initialize the ESP32 in correct configuration modes and make a successful connection.

3.2.3.1 One to One Connection

The operation of ESP-Now in this kind of configuration can be divided into two parts: master and slave.

3.2.3.1.1 Master

Figure 26 shows the program flow of ESP-Now in master configuration. The first task was configuring the Master in Station (STA) Wi-Fi mode. For this, the *mode()* function of the Wi-Fi library was used. At first, this function obtained the current mode of the device and checks whether the device is configured in STA mode or not. If the device is in STA mode, then nothing was done, and true value was returned. But if the device was not set in STA mode, then it called the *esp_wifi_set_mode()* function which finally set the device in STA mode. Then, the MAC address of the device was obtained. For this *macAddress()* function was used. This function called the *esp_wifi_get_mac()* function which gave the MAC address in STA mode. After the setup was completed in STA mode, the ESP32 was initialized.

Once the initialization of master was done, the master was told to scan for any available access points devices in the surrounding. When a device was found, then the SSID, RSSI, and BSSID of each found devices were saved. SSID stands for service set identifier which is just a technical term for a network name. RSSI is the received signal strength, and BSSID is the MAC address of

the wireless access point. After each of them was saved, the master was instructed to check if the current device name began with slave or not. This name was given to the slave device during AP configuration. If the name began with the slave, then the MAC address of the slave device was saved in a variable otherwise the slave would be discarded.

Before sending or receiving the data to another device, the device needs to be added to the paired device list. For that *esp_now_add_peer()* function was called. Once the pairing was made, the data could be sent or received. The data was sent using the *esp_now_send()* function, and the *esp_now_register_send_cb()* function was used to register for sending the callback function. This callback function in the transmitter indicated whether the data was successfully transmitted or not. Whereas, *esp_now_register_recv_cb()* was called to register the receiving callback function on the receiving side. This callback function was used to receive the data from the transmitter.

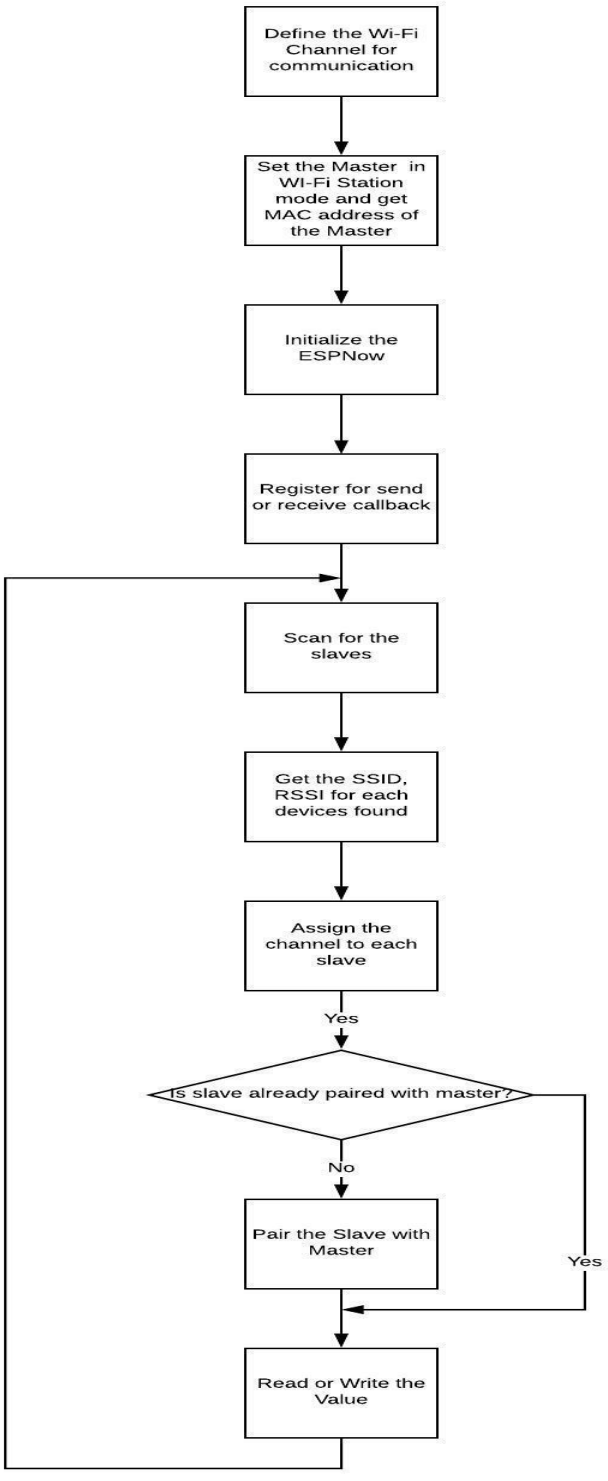

Figure 26: ESP-Now Master

3.2.3.1.2 Slave

Figure 27 shows the program flow in slave side of ESP-Now. The first task that was done was to configure the ESP32 device in soft-AP mode. For this, the *mode()* function of the Wi-Fi library was used. After that, the next task was to generate the access point of the client. For that initially, the client was provided with a MAC Address and prefix in front of the MAC Address. The prefix was provided before the Mac address so that it would be easy for the Master to recognize the slave. For the MAC address, *macAddress()* function was called. This function at first checked whether the device was in null mode or not. In null mode the ESP32 is not initialized as a station or the soft-AP mode. This mode is used to stop both AP and STA mode. If it was in the null mode, then the function was terminated, and nothing happened. But if the device was not in Null mode then it called a function *esp_wifi_get_mac()* to obtain the MAC address. After the generation of the MAC address, the MAC address and prefix were combined to generate the SSID of the client network. The last task was to finalize the AP configuration of the client network. For this the *softAP()* function was called. The parameters provided to this function are combined SSID (prefix + MAC address), password, Channel and hidden network cloaking. Since in this thesis we are configuring the ESP32 device in WPA2 standard, a password of minimum eight characters was provided. The hidden network cloaking is used to broadcast or hide the SSID of the client.

After the successful configuration of ESP32 in soft-AP mode, the initialization of the device was completed. Once the initialization was done, the client successfully broadcasted its SSID and waited for the connection with the server. When the connection was established, register callback functions were called to either send or receive the data from the server.

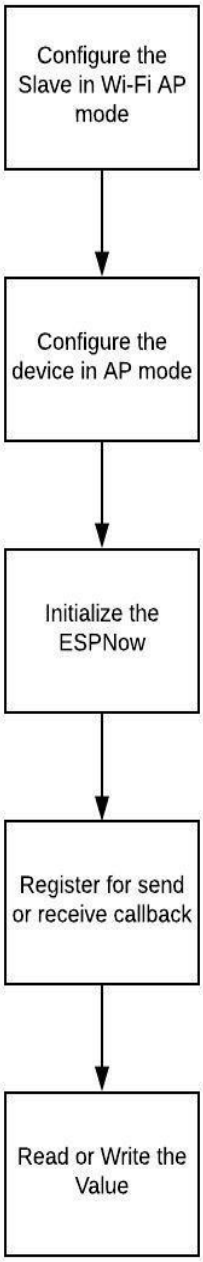

Figure 27: Slave side in ESP-Now

3.2.3.2 Star Connection

Similar to the BLE and nRF24L01+, the star connection for ESP-Now had central hub and peripheral nodes.

3.2.3.2.1 Central Hub

Figure 28 shows the program flow of ESP-Now in the central hub in a star network. In a star network, the central hub was configured as master. There is a slight difference in the configuration process of master in star and one to one connection. In one to one connection, the MAC address of master was generated using Wi-Fi, *macAddress()* function. But in a star connection, the MAC address of the ESP32 was hardcoded in the code. This was done so that the clients can add the master in their paired device list during their initialization. The MAC address was set by using *esp_base_mac_addr_set()* function. Once the MAC address was configured, next the ESP32 was initialized, and the hub began scanning for the available devices. Once the scanning was complete, the devices or nodes were added to the paired list. Initially, the hub was acting as a receiver, so the callback register functions to receive the data from the nodes were called. The end user was determined by using the concept of device ID similar to that of BLE and nRF24L01+.

In this state, the master was expecting three different data from the nodes. The first value was the device ID, and rest was the user data. After all data were received from the node, the master went into a new state. In this state, the hub acted as a transmitter, and for that, the register callback function to send data was called. Then the hub transmitted the user data to the intended end node. The correct node was determined inside the send function. After the data are transmitted, the register callback function gave the status of transmission to the hub.

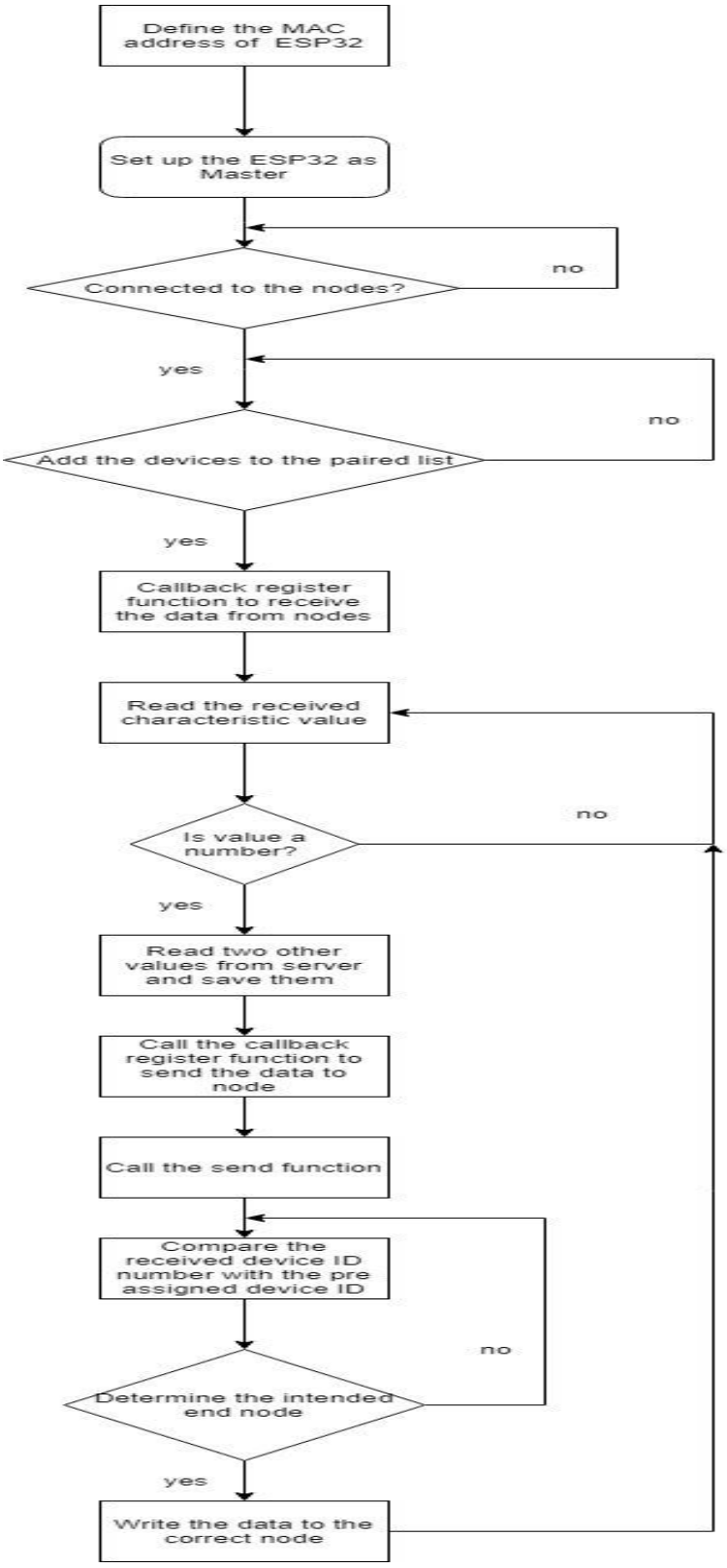

Figure 28: Central Hub in Star Connection of ESP-Now

3.2.3.2.2 Peripheral Node

Figure 29 shows the general flow of the program of ESP-Now in a peripheral node in star connection. The node in the star connection was configured as a client in AP mode. The setup process was same in both star and one to one connection. Once the setup was completed, the ESP32 was initialized. There was a slight difference in the initialization process in star and one to one connection. In star connection, we knew MAC address of the hub. So, during the initialization process, this MAC address of the server was used to add our server in the paired device list.

Now after the connection was made to the hub, the node acted as receiver and called the register callback function to receive the data from the hub. In this state, there was also another condition if the node received a user data from the serial monitor the node was told to act as a transmitter. To act as a transmitter, a register callback function to send data was called by the node. After that, the node transmitted the device ID received from serial monitor to hub using send function. Once this first transmission was completed, the node entered a new state and transmitted the user defined data to the hub as shown. After all data were transmitted to the hub, the client went back to receiving state.

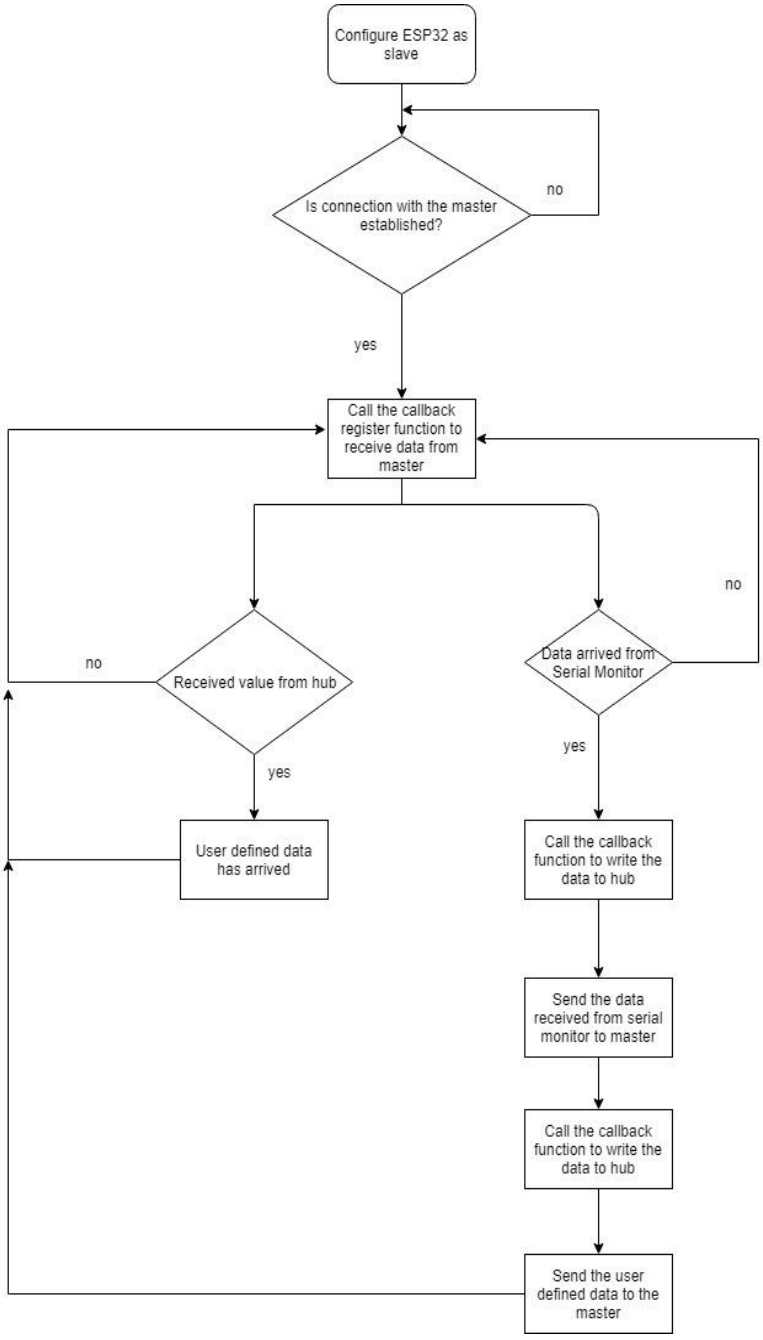

Figure 29: Peripheral node (ESP-Now) in Star Connection

3.2.4 Measurement of Performance Metrics

3.2.4.1 Throughput

Figure 33 shows the methodology used for the throughput calculation. The throughputs were calculated for all three protocols in both one to one and star connections. Different payload sizes were transmitted, and throughput for each case was calculated. In the case of nRF24L01+, throughputs were calculated in different available data rates and power settings. This was done to compare the performance of nRf24L01+ in different available speed and power settings.

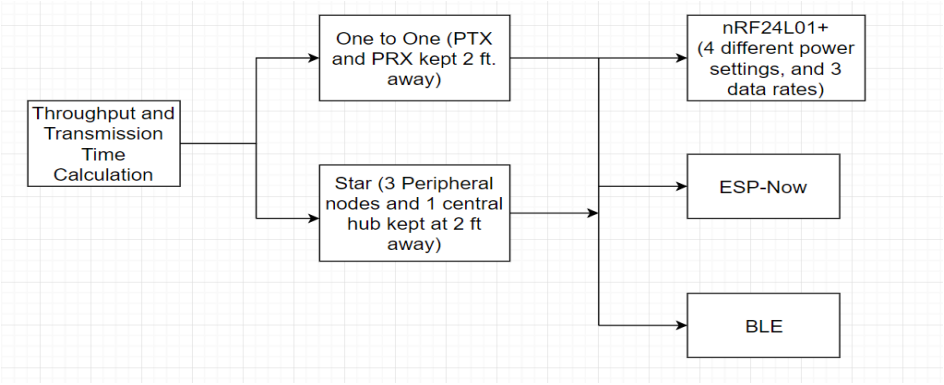

Figure 30: Throughput Calculation Methods

3.2.4.1.1 nRF24L01+

In nRF14L01+, the receiver sends back the acknowledgment to the transmitter when it receives the data. The acknowledgment indicates the successful transmission of data from a transmitter to receiver. In star connection, the transmission of data from one node to another occurs through a central hub. So, the data was transmitted twice, one from transmitting node to the central hub and another from the central hub to end receiving node. Thus, in this thesis for star network throughput was calculated in two phases. The first phase involved calculation of throughput when data was transmitted from transmitting node to a central hub, and another phase involved

transmission from the central hub to end node. The two corresponding transmission time in a single transmission were then added and saved in an array in the program. Now after 50 transmissions those saved values were called, added with each other and averaged out to find the total transmission time of the network. At last Equation 3 was used to calculate the throughput of the system.

$$\text{Throughput} = \frac{(\text{number of bytes sent} \times 8)}{\text{total transmission time}} \quad (3)$$

In case of static payload length, the total length of bytes that is transmitted is 32 bytes, but in dynamic payload length, the packet size varies with the length of the payload. The total transmission time is calculated using the *SysTick* Timer of MSP432. The timer is initialized in such a way that it generates an interrupt in every microsecond. The total number of ticks between the transmission of data and arrival of the acknowledgment signal is measured to calculate the total transmission time. The throughputs for different power and speed configuration of nRF24L01+ are calculated.

The Enhanced ShockBurst packet in nRF24L01+ contains preamble, CRC, address and payload field. Except for payload field, all other fields are used for data verification, reception and error checking. The payload field is the actual user sent data and is the only thing that can be controlled by the user during communication. So, in this thesis, only payload field was considered for throughput calculation and comparison.

3.2.4.1.2 ESP-Now and BLE

The throughput calculation concept was the same for both ESP-Now and BLE. Two different data of certain length were sent one after another. The time difference between the arrival of first and second data was measured at the end node to calculate the transmission time of the packet.

This process was repeated many times to get the several measures of transmission time, and each measured value was saved in an array. At the end of the 50^{th} transmission, those values were extracted from the array and averaged out to obtain the average transmission time. At last using Equation 3**,** the throughputs were calculated for both protocols. The *micros()* function of Arduino IDE was used for time measurement.

ESP-Now packet format contains different data fields like MAC, Category Code, etc. along with payload body. Similar to nRF24L01+, only payload body size was considered for throughput calculation.

3.2.4.2 Current Measurement

3.2.4.2.1 ESP-Now and BLE

To measure the current consumption in the ESP32, a series connection was established using current sensing resistor of 0.005 ohm. A Lithium-Ion battery of rating 3.7 V and 2000mah was used as power supply. The current measurement configuration consisted of two different circuits for transmitter and receiver. If the current was to be measured in the transmitter side, then the transmitter was connected to the Lithium Ion battery through a resistor. There was no USB connection to transmitter in this condition, so it was getting the power only from the battery. Whereas, the receiver was connected to the computer through USB connection, this was done to monitor the values in serial monitor to make sure that the receiver is receiving the correct transmitted values.

Now similarly to measure the current consumption in the receiver side, the circuits were exactly opposite, that is receiver was connected to the battery while the transmitter was connected to the laptop through USB connection. In both conditions continuous transmission was done and the

voltage across the resistor was monitored continuously. Figure 31 shows the circuit diagram of the current measurement circuit.

Different resistors values ranging from 1 ohm to 5 ohm were used initially to calculate the current consumption. But due to the high voltage drop in those resistors, the ESP32 was not able to transmit the data to the receiver. Whereas when a current sensing resistor of 0.005 ohm was used, ESP32 was able to transmit the data to the receiver. Then the voltage across the resistor was measured using a digital multi-meter to obtain the current in the circuit.

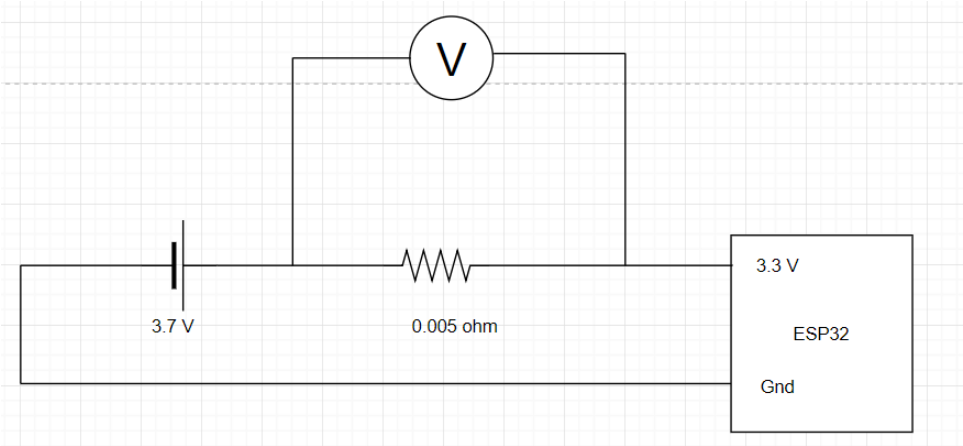

Figure 31: Current Measurement Circuit

3.2.4.2.2 nRF24L01+

The current consumption in nRF24L01+ was done in two methods. In the first method, the EnergyTrace feature of CCS was used to measure the current consumption in TX and RX states. To measure the current consumption of nRF24L01+, the data was transmitted continuously from PTX to PRX with some delay. Then, the EnergyTrace software was run in the Code Composer Studio IDE to measure the current consumption.

In the second method, current measurement circuit of **Error! Reference source not found.** was used to measure the current consumption of nRf24L01+ in TX and RX state. A current sensing

resistor of 0.005 ohm was placed in series between nRF24L01+ and MSP432. The voltage across the resistor was measured using a digital multi-meter to measure the current.

3.2.4.3 Network Recovery Time

The network recovery time for all three protocols was calculated in two different conditions: when power is removed and when device is taken out of range. The brief description of the process involved for the measurement of time is given below:

3.2.4.3.1 Power Removed

nRF24L01+

The nRF24L01+ has four operating modes: power down, standby, TX, and RX mode. The nRF24L01+ consumes high current in TX and RX modes. When nRF24L01+ is powered up, at first, it goes to power down mode and then goes to standby mode followed by either RX or TX mode. In order to calculate the network recovery time, nRF24L01+ was put in power down mode. The standard mode of operation when the module is not transmitting or receiving is to keep module in standby mode. So, the time taken by nRF24L01+ to go from power down mode to TX Mode through standby mode was measured.

ESP-Now and BLE

In ESP-Now, the network recovery time was calculated in only master side since most of the tasks needed for the connection are done by master. The master scans for the slaves available and makes a connection to them. At first, it checks the validity of the slave, and after that, it begins the connection and pairing is established between the master and slave. To calculate the network recovery time in ESP-Now, the input power was removed and again restored, and when the

power was restored the time required between the scanning of devices to the successful pairing of slaves was measured.

In the case of BLE server advertises its services and characteristics, and the client makes a connection to them. The network recovery time was calculated only in client side since most of the work needed to make a connection is done by client. The general pattern for operation is that at first client scans the area for any BLE device. When it finds one, the client is initialized and then it is connected to the server address. After that, it makes a connection to the service and characteristic of the connected device. The input power was removed and again restored. Once the power was restored the time required for the client to perform a scanning operation and establish the connection to the server was measured.

3.2.4.3.2 Out of Range

nRF24L01+

In the nRF24L01+, when it is taken out of the range, it goes to standby mode not the power down mode. When taken back within the range, the nRF24L01+ goes from stand by mode to the TX or RX state and starts transmitting or receiving the data. Thus, the time taken by the nRF24L01+ to reconnect from standby mode to the TX or RX was calculated.

ESP32 and BLE

In ESP-Now, it was observed that the processes involved in reconnecting the device back to the network when taken out of range are same as the ones when the power is removed. This means in ESP-Now when the server is taken out of range and put back within the range, at first master scans for the slave in the surronding and checks the validity of the available slaves. Once the slave is found to be valid, it begins the connection and pairing is established between the master and slave. To calculate the network recovery time, the ESP32 was taken out of range and again

put back in, and the time required between the scanning of devices to the pairing of slaves was measured.

Similarly, in BLE when client is taken out of range and put back in, initially the client scans the surrounding for the advertising BLE device. When a server is found, the client is initialized, and connection to the server is established. Thus, the time required for the client to perform a scanning operation and establish the connection to the server was measured.

3.2.4.4 Range

The experiment for range measurement was conducted in GVSU football field in cloudy weather. There were no obstructions between the transmitter and receiver. The line of sight between the transceivers was kept straight throughout the experiment. Some experiments were conducted by keeping fixed antennas in the module in an angle with each other. The value of the range from those experiments was less than the values obtained by keeping the antennas in a straight line. Those values are not included in this thesis since they do not give the maximum range capability of the device.

3.2.4.4.1 One to One Connection

To measure the range in one to one connection, continuous data transmission from the transmitter to receiver was done. And after that receiver was slowly continuously taken away from the transmitter until the communication ceased. In case of one to one connection, when the communication ceased out the receiver was walked back in the network until it made error free communication and then was again separated. This procedure was repeated several times before a distance was measured.

3.2.4.4.2 Star Connection

To calculate the range in a star connection, two peripheral nodes and one central hub were configured in a star network. It was not possible to use more than two nodes due to the lack of resources (laptops) since the Terminal window was needed for each node to check the successful transmission of the data from one node to another node. In the experiment, two nodes were powered using the same laptop, and two Terminals (Tera Term) were running for each node to monitor the transmission, while the central node was using a separate laptop. The continuous transmission was done from one node to another through the central node, and central node was taken slowly away from the peripheral nodes. Figure 32 shows the experimental procedure used for range calculation. The distance between the peripheral nodes and central node was increased gradually to measure the range. The arrow between the central hub and the peripheral node shows the direction of data transmission. When the communication ceased, the hub was put back in the network until an error free transmission was again re-established. After that, the hub was again walked away from network, and this process was repeated several times before the distance between the central and peripheral node was measured. Since both nodes are in the same position and the communication occurred through the central node, the measured distance between the central and peripheral nodes was multiplied by two to obtain the total range of the network.

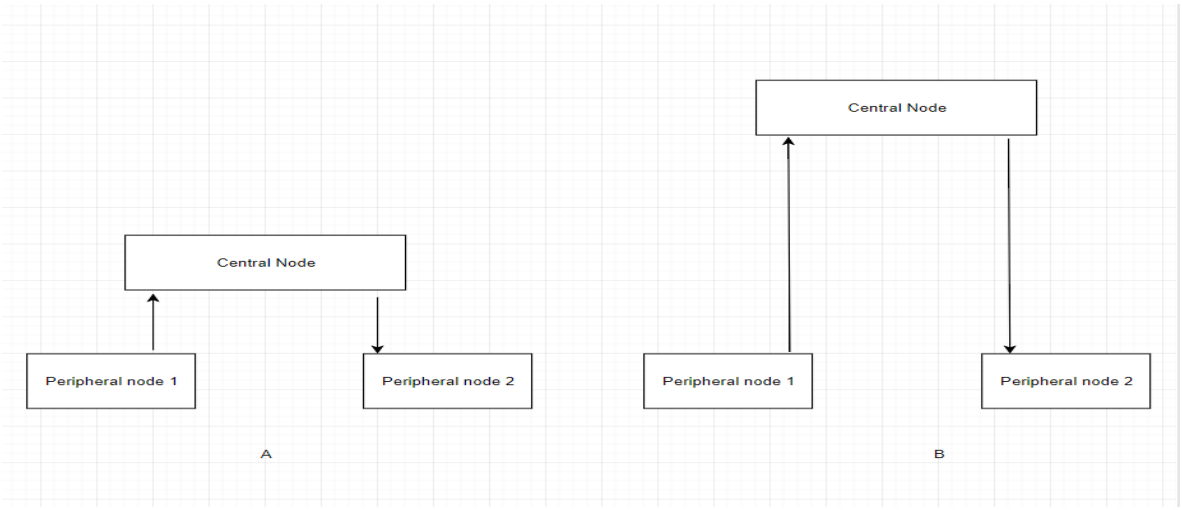

Figure 32: Experiment Procedure of Range Measurement in Star Network

4 Result

4.1 Throughput

4.1.1 nRF24L01+

The throughputs of nRF24L01+ for different payload sizes were measured. The payload length varied from 1 byte to 32 bytes. This radio module is generally used in a sensor network to transmit data. In most applications, the size of the data can be of small size (1 byte or 4 bytes). Due to this reason, throughput for small payload size was also calculated. In addition, throughputs in different power settings and data rates were also calculated for nRF24L01+. This was done to study the performance of nRF24L01+ in different available power and data rate settings.

4.1.1.1 One to One Connection

The throughputs for nRf24L01+ in one to one connection were measured for different payload sizes in different data rate and power setting configurations.

Figure 33 shows the distribution of measured transmission time over fifty transmissions in static payload length (32 bytes) at 250 Kbps data rate. Similarly, the distribution of transmissions times over fifty transmissions at 1 Mbps and 2 Mbps data rates are shown in Figure 34 and Figure 35. The raw data for each transmission is given in Appendix A.

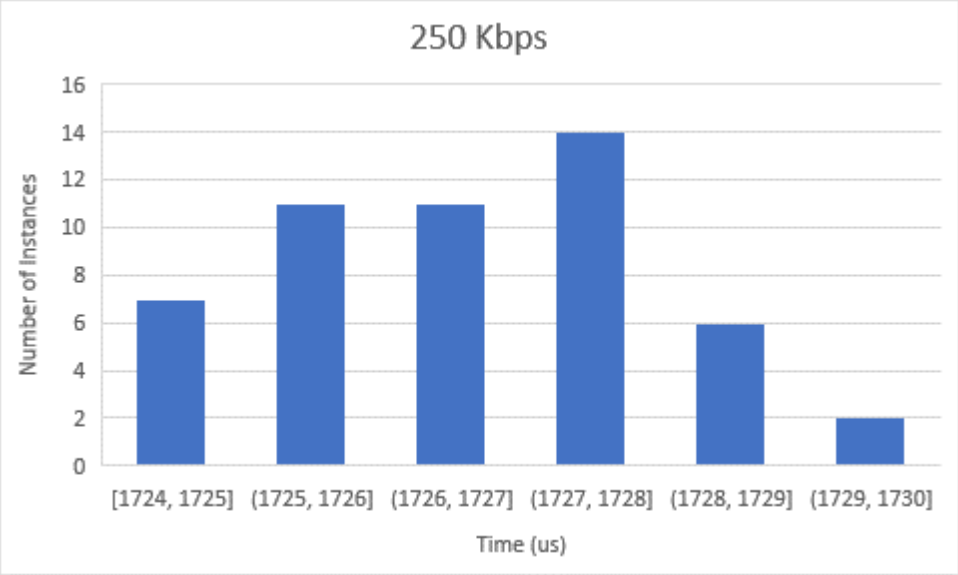

Figure 33: Distribution of Transmission Time at 250 Kbps

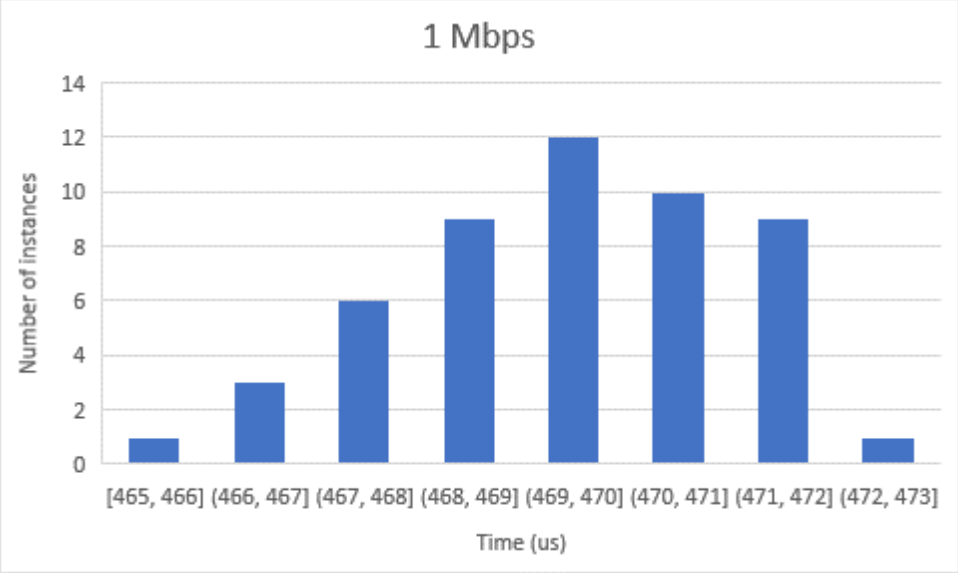

Figure 34: Distribution of Transmission Time at 1 Mbps

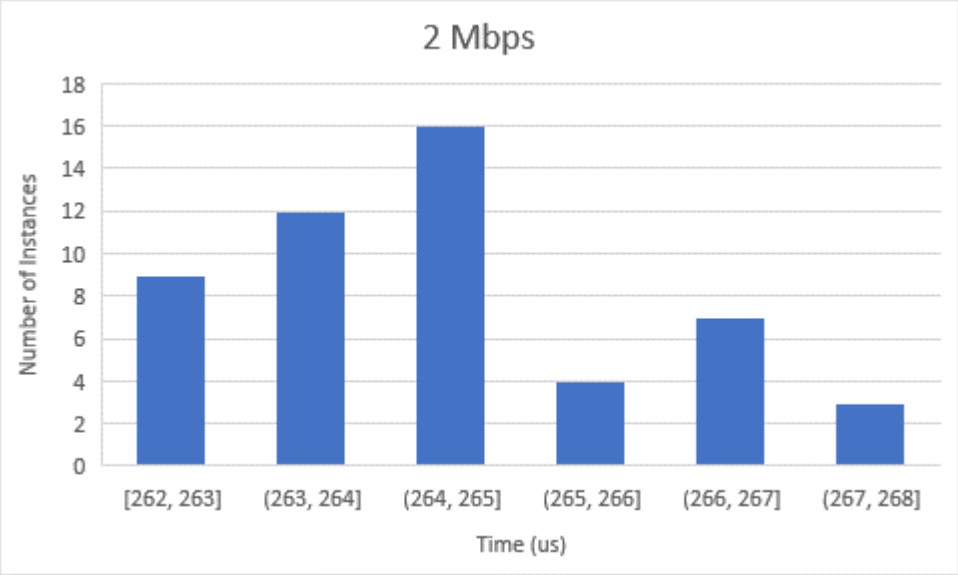

Figure 35: Distribution of Transmission Time at 2 Mbps

Table 1 to Table 4 show the measured transmission time and calculated throughput values for different payload conditions. For each payload, the calculation of throughput was done in different power setting and data rate configurations.

The mean values of measured transmission time over 50 transmissions are given along with their respective standard deviations. Using those measured mean values and Equation 3, the throughputs of the network in different conditions are calculated.

Table 1: Average transmission time and calculated throughput for 32 bytes

SET UP IN nRF24L01+		Mean Values with standard deviations	Calculated Values based on Mean
Data Rate	Power	Time(us)	Throughput(kbps)
250 kbps	0 dBm	1727 (1.43)	148.23
250 kbps	MIN 12dBm	1726 (1.43)	148.31
250 kbps	MIN 18dBm	1728 (1.42)	148.14
1 Mbps	0 dBm	470 (1.65)	544.68
1 Mbps	MIN 12dBm	471 (1.64)	543.52
1 Mbps	MIN 18dBm	471 (1.64)	544.68
2 Mbps	0 dBm	264 (1.51)	969.7
2 Mbps	MIN 12dBm	265 (1.52)	966
2 Mbps	MIN 18dBm	264 (1.51)	969.7

Table 2: Average transmission time and calculated throughputs for 1 byte

SET UP IN nRF24L01+		Mean Values with standard deviations	Calculated Values based on Mean
Data Rate	Power	Time(us)	Throughput(kbps)
250 kbps	0 dBm	733 (0.49)	10.91
250 kbps	MIN 12dBm	733 (0.48)	10.91
250 kbps	MIN 18dBm	733 (0.48)	10.91
1 Mbps	0 dBm	222 (1.25)	36.03
1 Mbps	MIN 12dBm	221 (1.23)	36.19
1 Mbps	MIN 18dBm	222 (1.24)	36.03
2 Mbps	0 dBm	144 (0.54)	55.56
2 Mbps	MIN 12dBm	144 (0.55)	55.56
2 Mbps	MIN 18dBm	144 (0.54)	55.56

Table 3: Average transmission time and calculated throughputs for 4 bytes

SET UP IN nRF24L01+		Mean Values with standard deviations	Calculated Values based on Mean
Data Rate	Power	Time(us)	Throughput(kbps)
250 kbps	0 dBm	850 (1.47)	37.64
250 kbps	MIN 12dBm	852 (1.46)	37.55
250 kbps	MIN 18dBm	849 (1.48)	37.69
1 Mbps	0 dBm	246 (1.48)	130.08
1 Mbps	MIN 12dBm	247 (1.46)	129.55
1 Mbps	MIN 18dBm	246 (1.47)	130.08
2 Mbps	0 dBm	153 (1.28)	209.15
2 Mbps	MIN 12dBm	152 (1.29)	210.52
2 Mbps	MIN 18dBm	154 (1.28)	207.8

Table 4: Average transmission time and calculated throughput for 16 bytes

SET UP IN nRF24L01+		Mean Values with standard deviations	Calculated Values based on Mean
Data Rate	Power	Time(us)	Throughput(kbps)
250 kbps	0 dBm	850 (1.47)	37.64
250 kbps	MIN 12dBm	852 (1.46)	37.55
250 kbps	MIN 18dBm	849 (1.48)	37.69
1 Mbps	0 dBm	246 (1.48)	130.08
1 Mbps	MIN 12dBm	247 (1.46)	129.55
1 Mbps	MIN 18dBm	246 (1.47)	130.08
2 Mbps	0 dBm	153 (1.28)	209.15
2 Mbps	MIN 12dBm	152 (1.29)	210.52
2 Mbps	MIN 18dBm	154 (1.28)	207.8

4.1.1.2 Star Connection

The process involved for throughput calculation in nRF24L01+ in star connection is discussed in section 3.2.4.1.1. From one to one connection it was observed that the throughput did not depend upon the transmitting or receiving powers. So, the transmission time of nRF24L01+ in star connection was measured for only different payload sizes in different data rates.

Figure 36 shows the distribution of measured transmission time in star network over fifty transmissions in static payload length (32 bytes) at 250 Kbps data rate. Similar distributions are also given for 1 Mbps and 2 Mbps at the same payload condition in Figure 37 and Figure 38. The raw data for each transmission in star network is given in Appendix A**.**

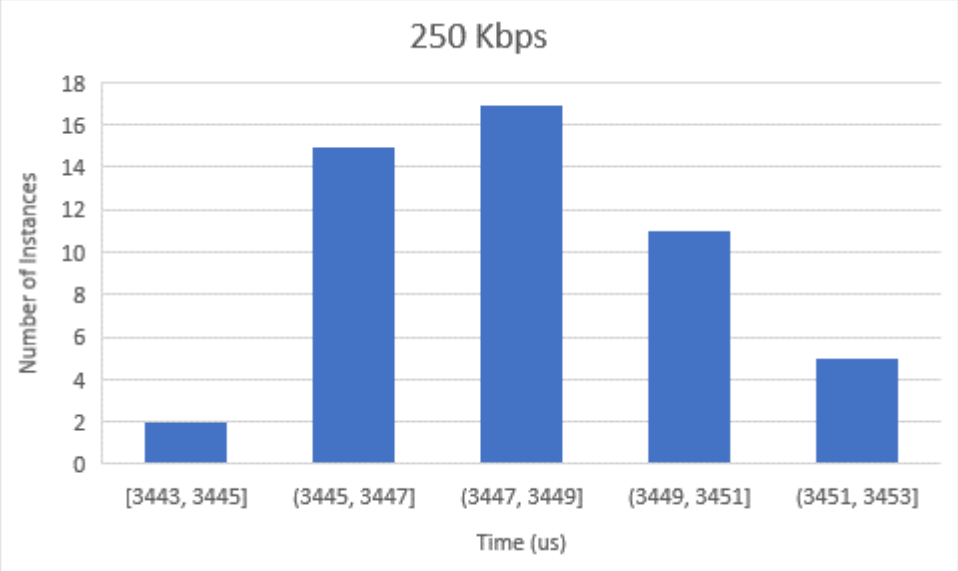

Figure 36: Distribution of Transmission time in Star Connection (32 Bytes)

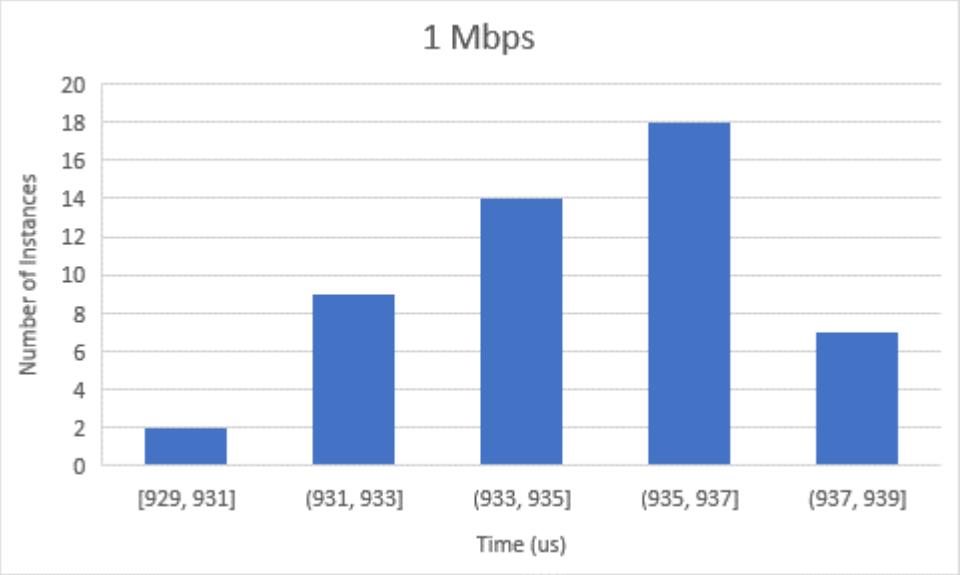

Figure 37: Distribution of Transmission time in Star Connection (32 Bytes)

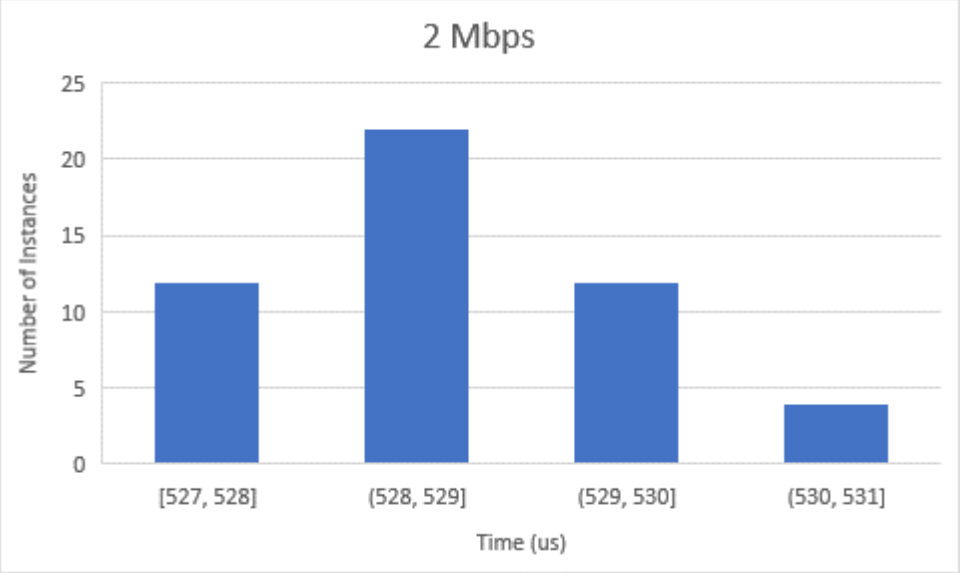

Figure 38: Distribution of Transmission time in Star network (32 Bytes)

Table 5 to Table 8 show the measured transmission time and calculated throughput values for different payload conditions in a star network. For each payload condition, the calculation of throughput was done in different data rate configurations.

The total calculated transmission time (T) of measured mean transmission time (T1 and T2) in each condition is given. Using those total calculated transmission time values and Equation 3, the throughputs of the network in different conditions are calculated.

Table 5: Throughput and Time Taken for 32 bytes (Star Network)

			Transmission Time Mean Values with standard deviations		Calculated Values based on Mean	
Data Rate	Throughput 1 (kbps)	Throughput 2 (kbps)	Time 1(us) (T1)	Time 2 (us) (T2)	Total Throughput(kbps) (PacketSize)/(T1+T2)	Time (us) (T1 + T2)
250 kbps	148.40	148.57	1725 (1.47)	1723 (1.52)	74.24	3448
1 Mbps	545.84	544.68	469 (1.17)	470 (1.22)	272.63	939
2 Mbps	966.03	969.69	265 (0.72)	264 (0.74)	483.93	529

Table 6: Throughput and Transmission Time for 1 byte (Star Network)

			Transmission Time Mean Values with standard deviations		Calculated Values based on Mean	
Data Rate	Throughput 1 (kbps)	Throughput 2 (kbps)	Time 1(us) (T1)	Time 2 (us) (T2)	Total Throughput(kbps) (PacketSize)/(T1+T2)	Time (us) (T1 + T2)
250 kbps	10.91	10.92	733 (0.47)	732 (0.48)	5.46	1465
1 Mbps	36.19	36.19	221 (1.24)	221 (1.25)	18.09	442
2 Mbps	55.94	56.33	143 (0.55)	142 (0.54)	28.07	285

Table 7: Transmission Time and Throughput for 4 bytes (Star Network)

			Transmission Time Mean Values with standard deviations		Calculated Values based on Mean	
Data Rate	Throughput 1 (kbps)	Throughput 2 (kbps)	Time 1(us) (T1)	Time 2 (us) (T2)	Total Throughput(kbps) (PacketSize)/(T1+T2)	Time (us) (T1 + T2)
250 kbps	38.04	38.09	841 (0.83)	840 (0.82)	19.03	1681
1 Mbps	130.61	130.61	245 (0.75)	245 (0.76)	65.30	490
2 Mbps	210.52	210.52	152 (0.73)	152 (0.72)	105.26	304

Table 8: Throughput and Transmission Time for 16 bytes (Star Network)

			Transmission Time Mean Values with standard deviations		Calculated Values based on Mean	
Data Rate	Throughput 1 (kbps)	Throughput 2 (kbps)	Time 1(us) (T1)	Time 2 (us) (T2)	Total Throughput(kbps) (PacketSize)/(T1+T2)	Time (us) (T1 + T2)
250 kbps	105.34	105.43	1215 (0.84)	1214 (0.83)	52.69	2429
1 Mbps	375.36	374.26	341 (0.56)	342 (0.58)	187.40	683
2 Mbps	640	640	200 (0.80)	200 (0.82)	320	400

4.1.2 BLE and ESP-Now

4.1.2.1 One to One

To calculate the throughput for ESP-Now and BLE, the data was transmitted between two ESP32s which were kept at 2.5 ft away from each other. For each transmission, the transmission time was measured and used to calculate the throughput of the system.

Like that in nRF24L01+, throughputs in ESP-Now and BLE were calculated different payload sizes. Unlike nRF24L01+, these protocols do not offer different data rates. Hence the throughput calculation in BLE and ESP-Now were done in the only one configuration. Figure 39 shows the

distribution of transmission time in ESP-Now in One to One Connection when 32 bytes of data are transmitted. The experimental data is given in Appendix A**.**

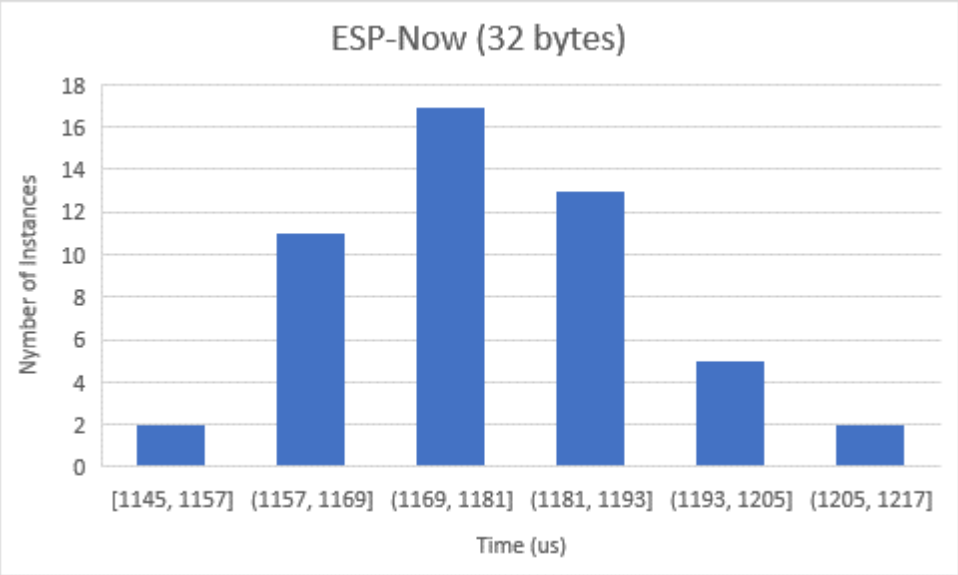

Figure 39: Distribution of Transmission time in ESP-Now

Figure 40 shows the distribution of transmission time in BLE in One to One Connection when 32 bytes of data are transmitted. The experimental data is given in Appendix A**.**

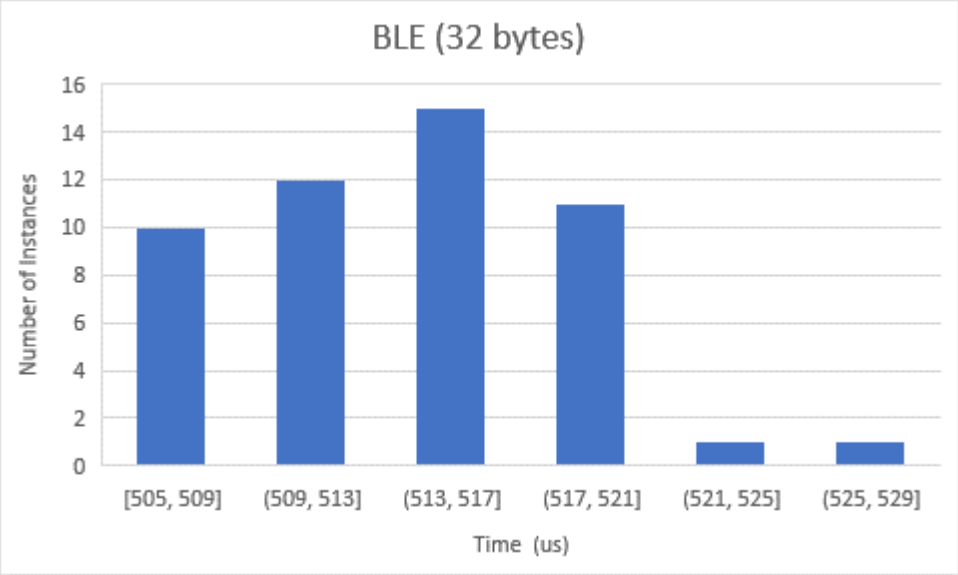

Figure 40: Distribution of Transmission time in BLE

Table 9 contains the measured transmission time and calculated throughput values for different payload conditions in ESP-Now. The mean values of transmission time along with their standard deviation and calculated throughput based on those values are given.

Table 9: Throughput and Transmission time for ESP-Now in different Payload sizes

ESP-Now	Mean Values with standard deviations	Calculated Values based on Mean
Payload Sizes (byte)	Time (us)	Throughput (kbps)
1	929 (10.08)	8.611
4	966 (12.07)	33.126
16	1058 (17.30)	120.98
32	1179 (14.22)	217.13

Table 10 contains the measured transmission time and calculated throughput values for different payload conditions in BLE. The mean values of transmission time along with their standard deviation and calculated throughput based on those values are given.

Table 10: Throughput and Transmission time for BLE in different Payload sizes

BLE	Mean Values with standard deviations	Calculated Values based on Mean
Payload Sizes (byte)	Time (us)	Throughput (kbps)
1	114 (0.68)	69.89
4	142 (0.92)	225.35
16	334 (6.16)	338.23
32	514 (4.85)	498.05

4.1.2.2 Star Connection

The process involved for throughput calculation in ESP-Now and BLE in star connection is discussed in section 3.2.4.1.2.

Figure 41 shows the distribution of transmission time in ESP-Now in a star connection when 32 bytes of data are transmitted. The experimental data is given in Appendix A**.**

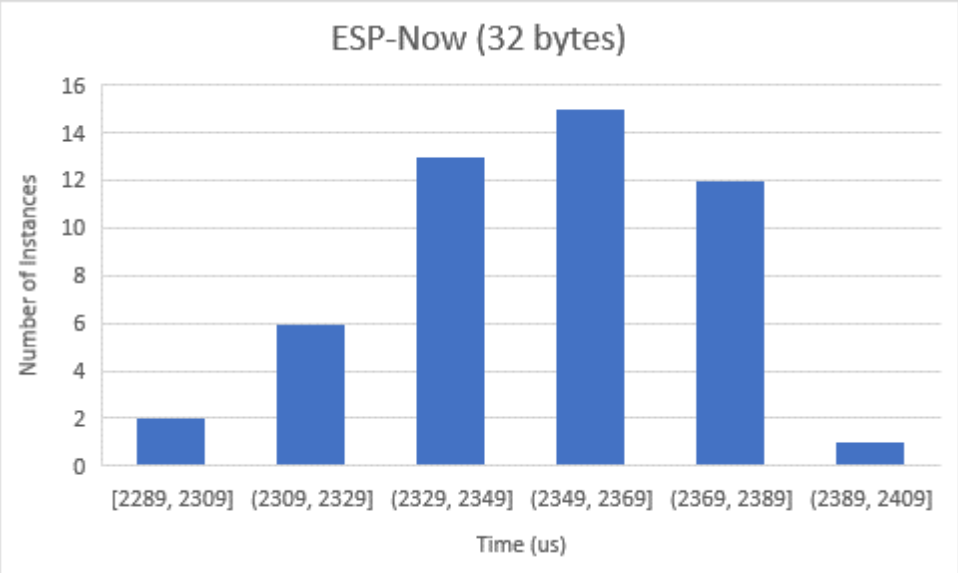

Figure 41: Distribution of Transmission time in Star Network (ESP-Now)

Table 11 shows the measured transmission time and calculated throughput values in a star network for different payload conditions in ESP-Now. The total transmission time (T1)

calculated based on the mean transmission values (T1 and T2), and calculated throughput values are given.

Table 11: Throughput and Transmission Time in ESP-Now in Star Connection

			Transmission Time Mean Values with standard deviations		Calculated Values based on Mean	
Payload Size (Byte)	Throughput 1 (kbps)	Throughput 2 (kbps)	Time 1(us) (T1)	Time 2 (us) (T2)	Total Throughput(kbps) (PacketSize)/(T1+T2)	Time (us) (T1 + T2)
1	8.57	8.60	933	930	4.29	1863
4	33.07`	33.02	967.60	967	16.54	1934.6
16	121.21	120.98	1056	1058	60.54	2114
32	217.74	217.31	1175.7	1178	108.76	2353.7

Table 12 shows the measured transmission time and calculated throughput values in a star network for different payload conditions in BLE. The total transmission time (T1) calculated based on the mean transmission values (T1 and T2), and calculated throughput values are given. Table 12 shows that the measured transmission Time 2 (T2) when data was transmitted from the central hub to the end node is not valid data.

Table 12: Throughput and Transmission Time in BLE in Star Connection

			Transmission Time Mean Values with standard deviations		Calculated Values based on Mean	
Payload Size (Byte)	Throughput 1 (kbps)	Throughput 2 (bps)	Time 1(us) (T1)	Time 2 (s) (T2)	Total Throughput(bps) (PacketSize)/(T1+T2)	Time 2(s) (T1 + T2)
1	59.26	1.6	135	5.025	1.6	5.025
4	205.12	6.25	156	5.12	6.25	5.12
16	336.84	25.34	380	5.05	25.64	5.05
32	467.15	50.50	548	5.069	50.50	5.069

4.2 Range

4.2.1 One to One

The process involved for range measurement in one to one connection is described in section 3.2.4.4.1. The experiment was repeated thirty times to measure the distance .

Figure 42 shows the distribution of the measured range in nRF24L01+in maximum power settings. Similar range measurements were also done for other power settings. The experimental data is data is given in Appendix B**.**

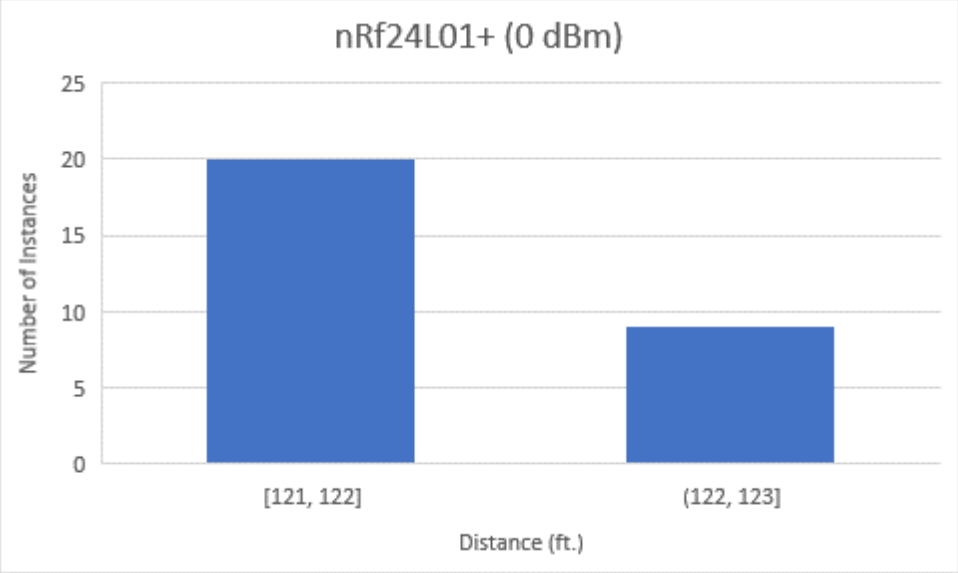

Figure 42: Distribution of Range in nRF24L01+ (0dBm)

Figure 43 shows the distribution of measured range for BLE; the experimental data is data given in Appendix B**.**

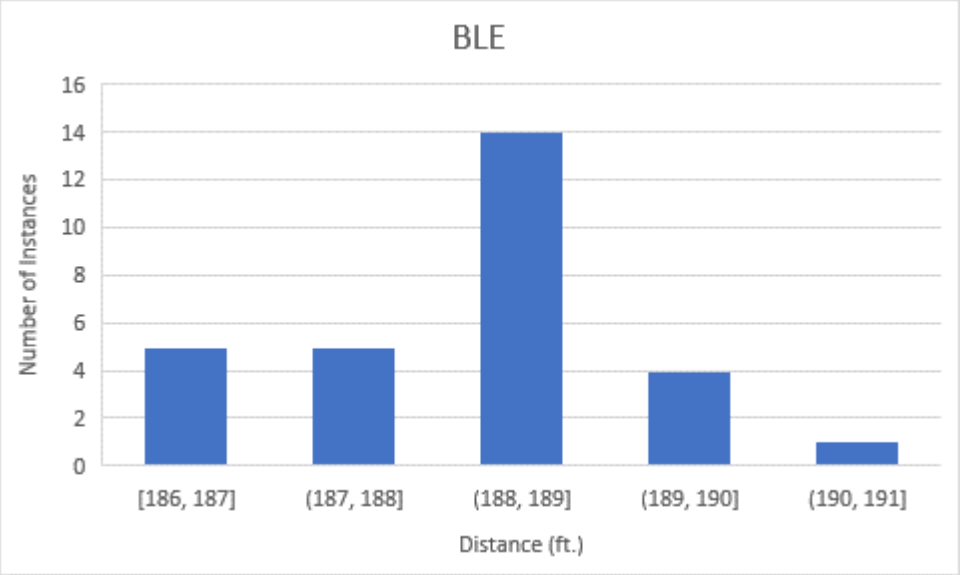

Figure 43: Distribution of Range in BLE

Figure 44 shows the distribution of measured range for BLE; the experimental data is data given in Appendix B**.**

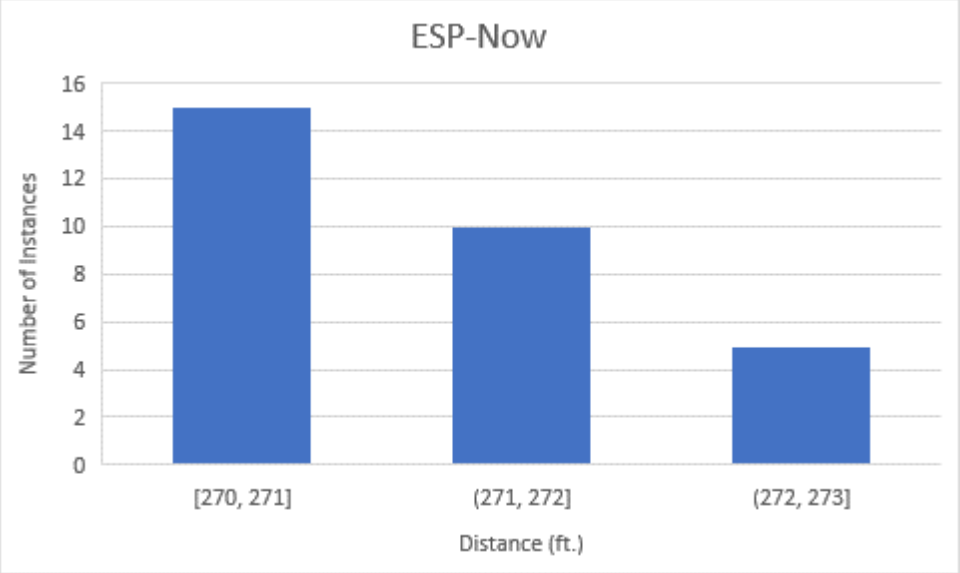

Figure 44: Distribution of Range in ESP-Now

Table 13 shows the maximum range of three protocols in one to one connection. For nRf24L01+, the range calculation has been done in different available power settings.

Table 13: Range of nRF24L01+, ESP-Now and BLE in One to One connection

Wireless Module	Power (in case of nRF24L01+ Only)	Range Mean Values with standard deviations (ft)
nRF24L01+	Min 18dBm	45
	Min 12 dBm	84
	Min 6 dBm	97
	0 dBm	122
BLE		189
ESP-Now		271

4.2.2 Star Connection

The process involved for range measurement in one to one connection is described in section 3.2.4.4.2. The experiment was done for thirty times and distance was measured in each instance. Figure 45 shows the distribution of measured range for nRF24L01+ in star connection at maximum power; the experimental data is data given in Appendix B**.**

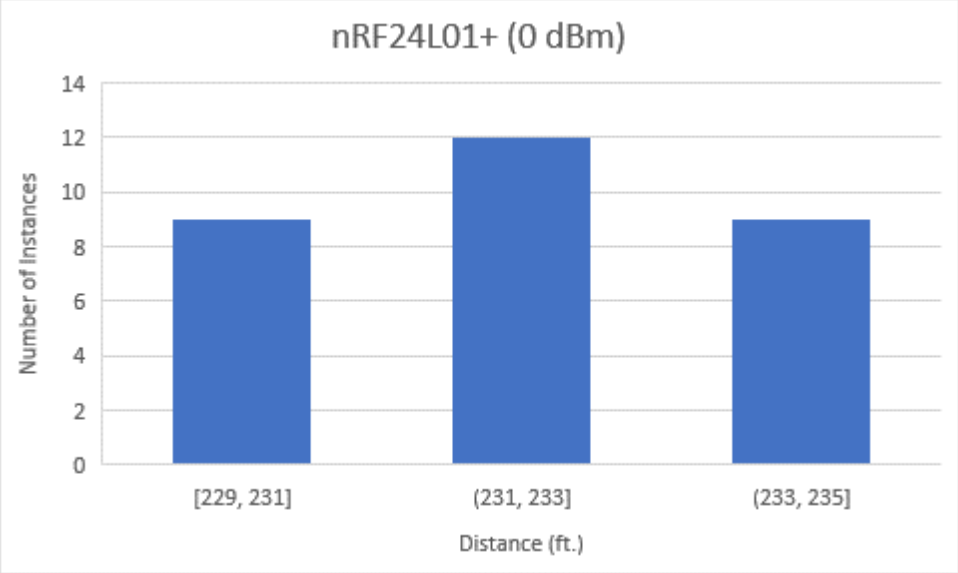

Figure 45: Range of nRF24L01+ in a star network

Figure 46 shows the distribution of measured range for BLE in a star connection; the experimental data is data given in Appendix B**.**

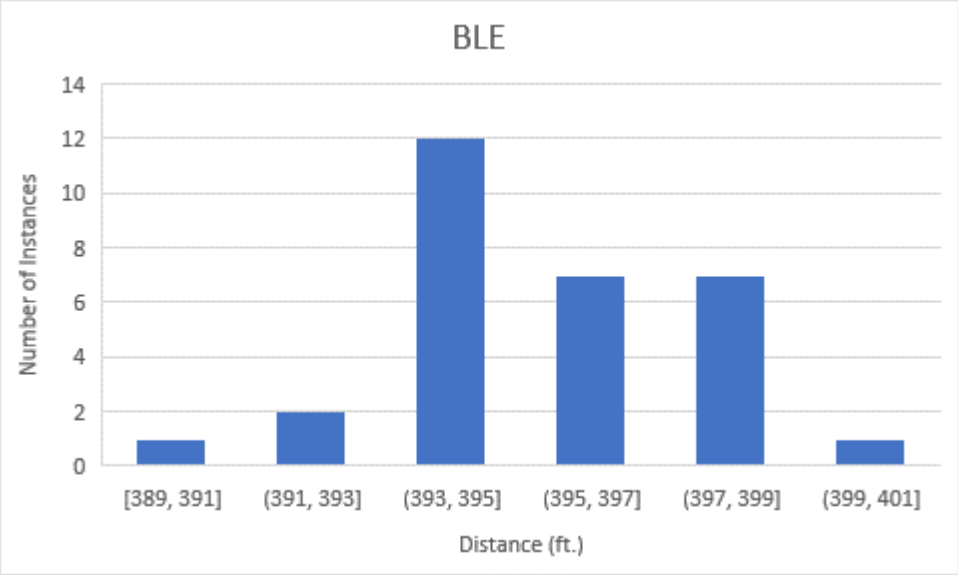

Figure 46: Range of BLE in a star network

Figure 47 shows the distribution of measured range for ESP-Now in a star connection; the experimental data is data given in Appendix B**.**

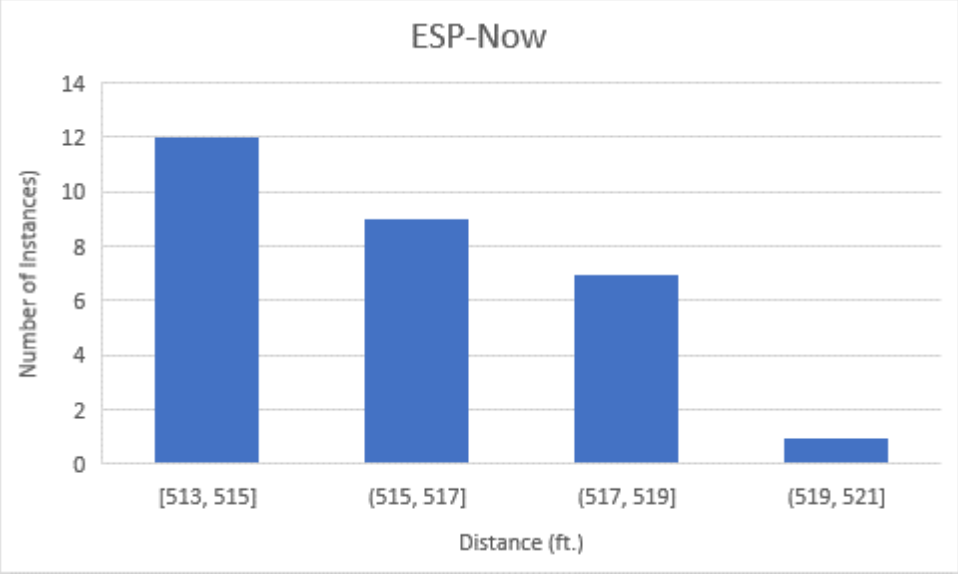

Figure 47: Range of ESP-Now in a star network

Table 14 shows the maximum range of three protocols in star connection. For nRf24L01+, the range calculation has been done in different available power settings.

Table 14: Range of nRf24L01+, ESP-Now and BLE in a star connection

Wireless Module	Power (in case of nRF24L01+ Only)	Range Mean Values with standard deviations (ft.)
nRF24L01+	Min 18dBm	90
	Min 12 dBm	168
	Min 6 dBm	193
	0 dBm	233
BLE		393
ESP-Now		513

4.3 Network Recovery Time

4.3.1 Power Removed

nRF24L01+

The process involved for measuring network recovery time in nRF24L01+ when power was removed is discussed in section 3.2.4.3.1.

Figure 48 shows the distribution of network recovery time of nRF24L01+ when power was removed. The average time taken by nRF24L01+ is 335 us. The experimental data is given in Appendix B**.**

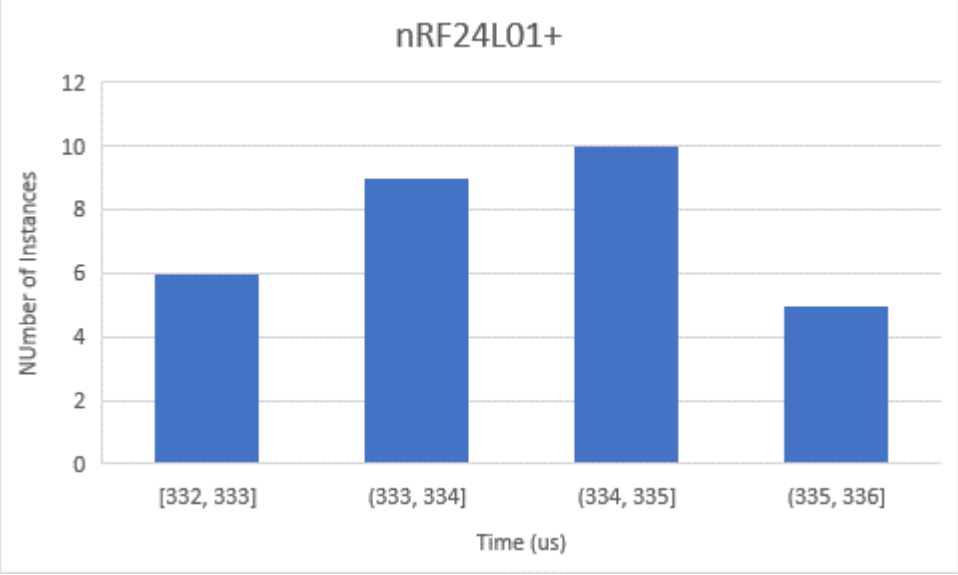

Figure 48: Distribution of Network Recovery Time in nRF24L01+ (power removed)

ESP-Now and BLE

The process involved for measuring network recovery time in ESP-Now and BLE when power was removed is discussed in section 3.2.4.3.1.

Figure 49 shows the distribution of network recovery time in ESP-Now when power was removed. In most instances, the network recovery time for ESP-Now was in between 16ms to 36ms with a mean value of 31ms. The experimental data is given in Appendix B**.**

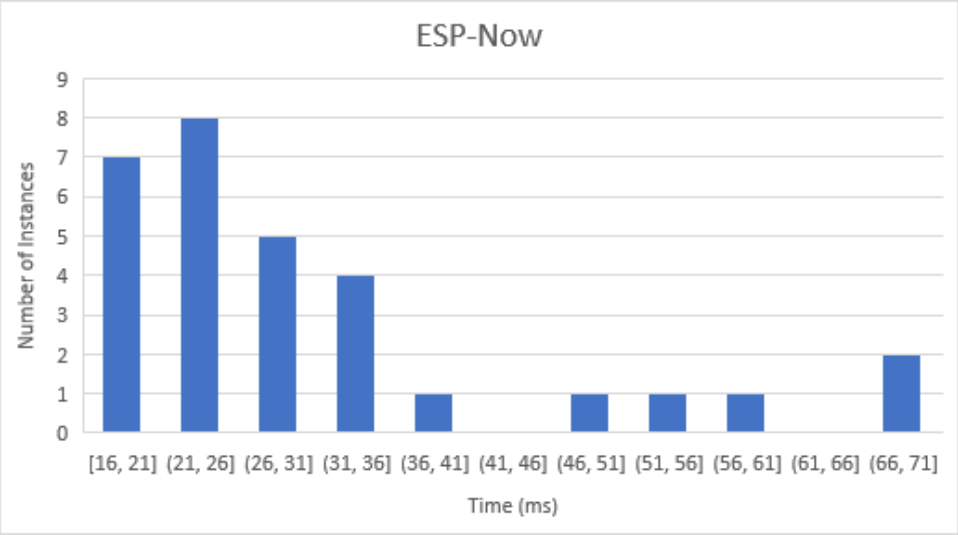

Figure 49: Distribution of Network Recovery Time in ESP-Now (Power Removed)

Figure 50 shows the distribution of network recovery time in BLE when power was removed. The average time taken by BLE to reconnect to the network was 1.3s. The experimental data is given in Appendix B**.**

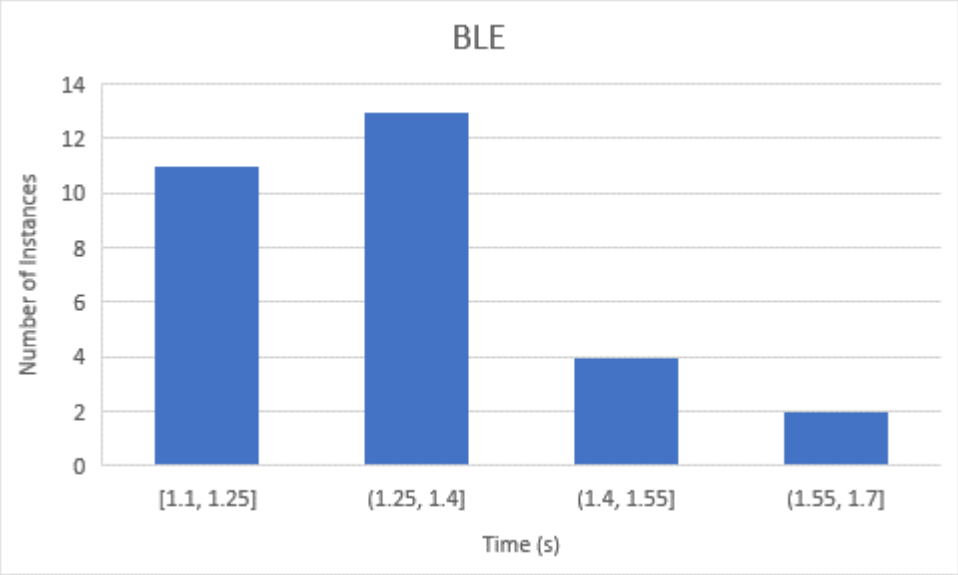

Figure 50: Distribution of Network Recovery Time in BLE (Power Removed)

4.3.2 Out of Range

nRF24L01+

The process involved for measuring network recovery of nRF24L01+ when it is taken out of the range is discussed in section 3.2.4.3.2.

Figure 51 shows the distribution of network recovery time in nRF24L01+ when taken out of the range. The average time required to reconnect back to the network was 224 us.

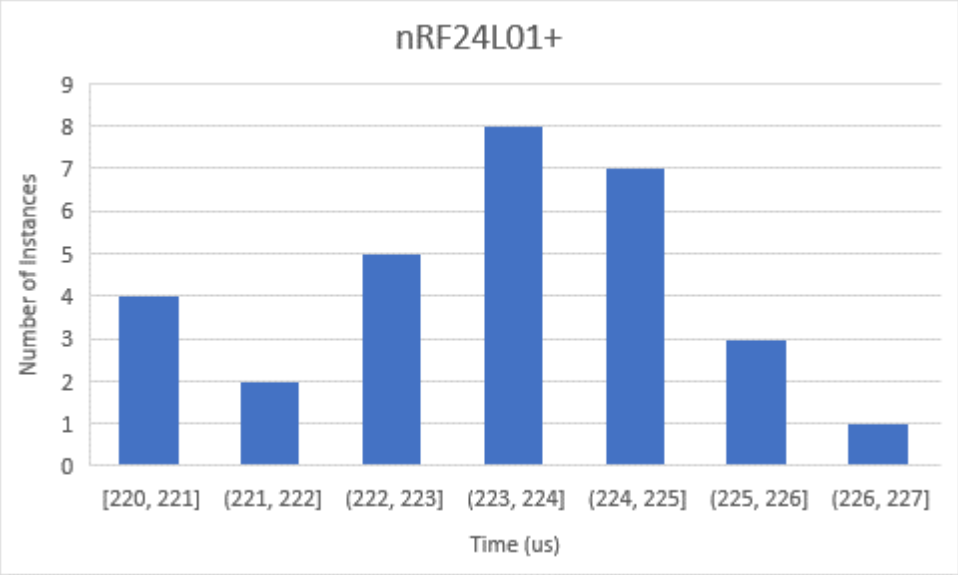

Figure 51: Distribution of Network Recovery Time in nRF24L01+ (out of range)

ESP-Now and BLE

The process involved for measuring the network recovery time in ESP-Now and BLE when the device is taken out of range is discussed in section 3.2.4.3.2.

Figure 52 shows the distribution of network recovery time in ESP-Now when taken out of the range. In most instances, the network recovery time for ESP-Now was in between 15ms to 35ms with a mean value of 33ms. The experimental data is given in Appendix B**.**

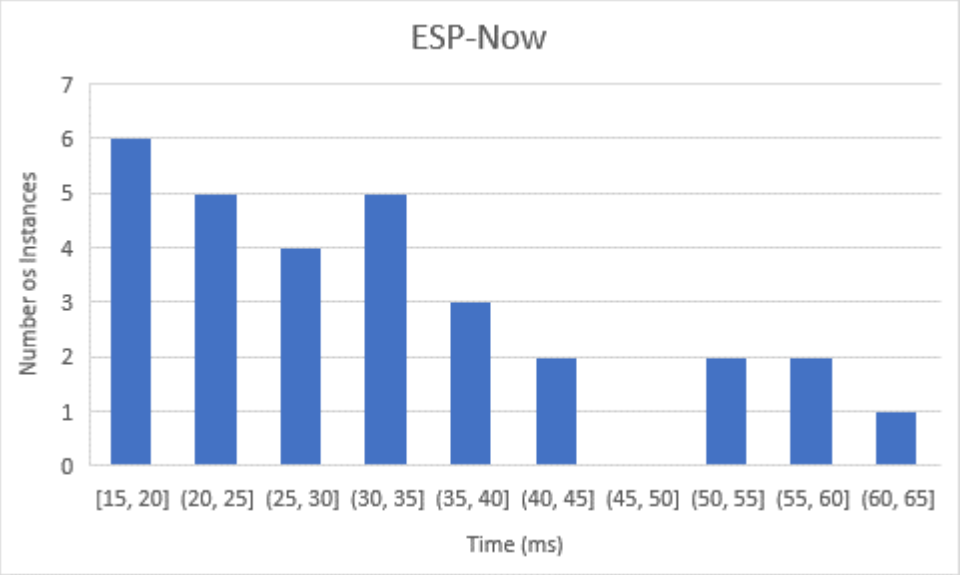

Figure 52: Distribution of Network Recovery Time in ESP-Now (out of range)

Figure 53 shows the distribution of network recovery time in BLE when taken out of the range. The average time taken by BLE to reconnect to the network was 1.34s. The experimental data is given in Appendix B**.**

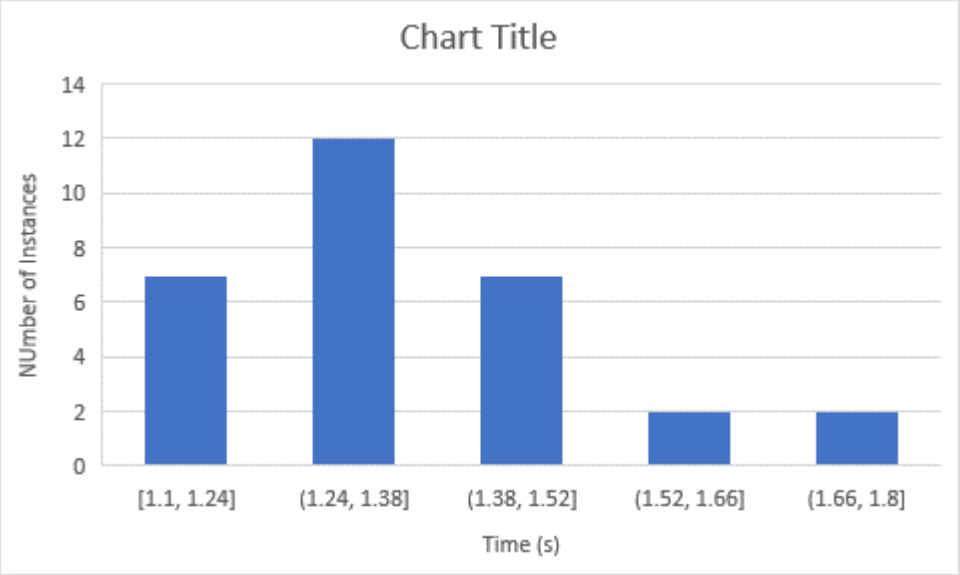

Figure 53: Distribution of Network Recovery Time in BLE (out of range)

4.4 Current Measurement

4.4.1 nRF24L01+

The process for measuring current in nRF24L01+ is discussed in section 3.2.4.2.2. The delays used in the experiment were 2.5us, 250us, and 25000us. The delay was used to provide some time to the receiver to process the received data before it receives next one. When data was transmitted without delay, many packets were lost due to latency in receiving side.

4.4.1.1 Current Measurement using EnergyTrace

Figure 54 shows the current consumption by nRF24L01+ during transmission of data. The current consumption is the minimum when nRF24L01+ is not transmitting and maximum when transmission occurs. Since there is a fixed delay between transmissions, the current consumption is high during transmission and low during the fixed delay period.

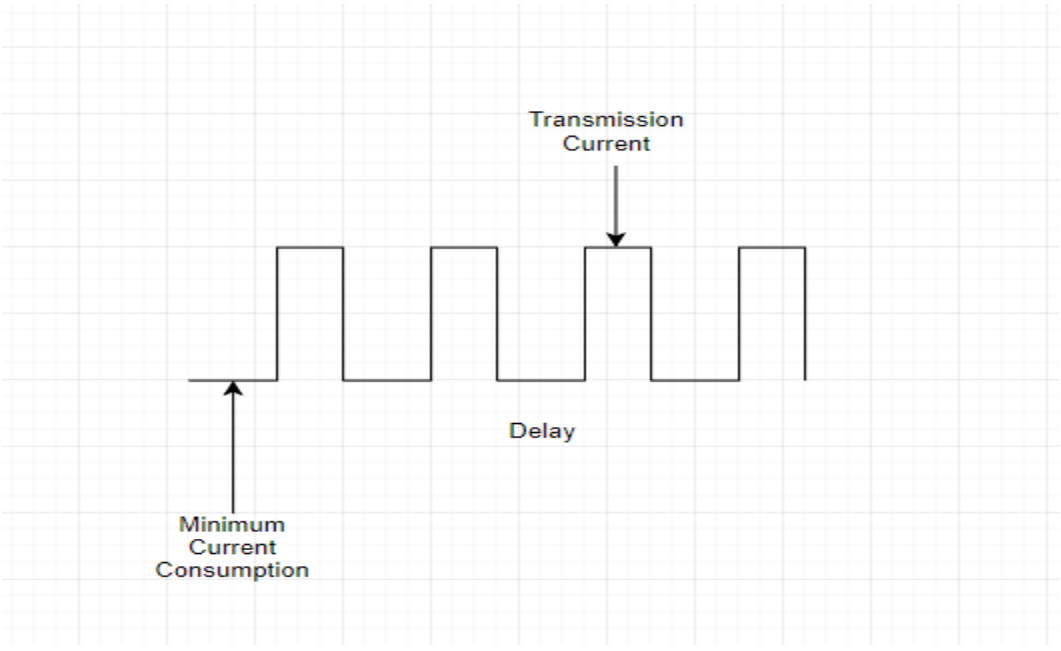

Figure 54: Current Consumption of nRF24L01+

Figure 55 shows the plot when nRF24L01+ powers up and begins transmitting. The power is about 34 mW in the initial condition, but as nRf24L01+ starts transmitting the relative power go up to 70 mW.

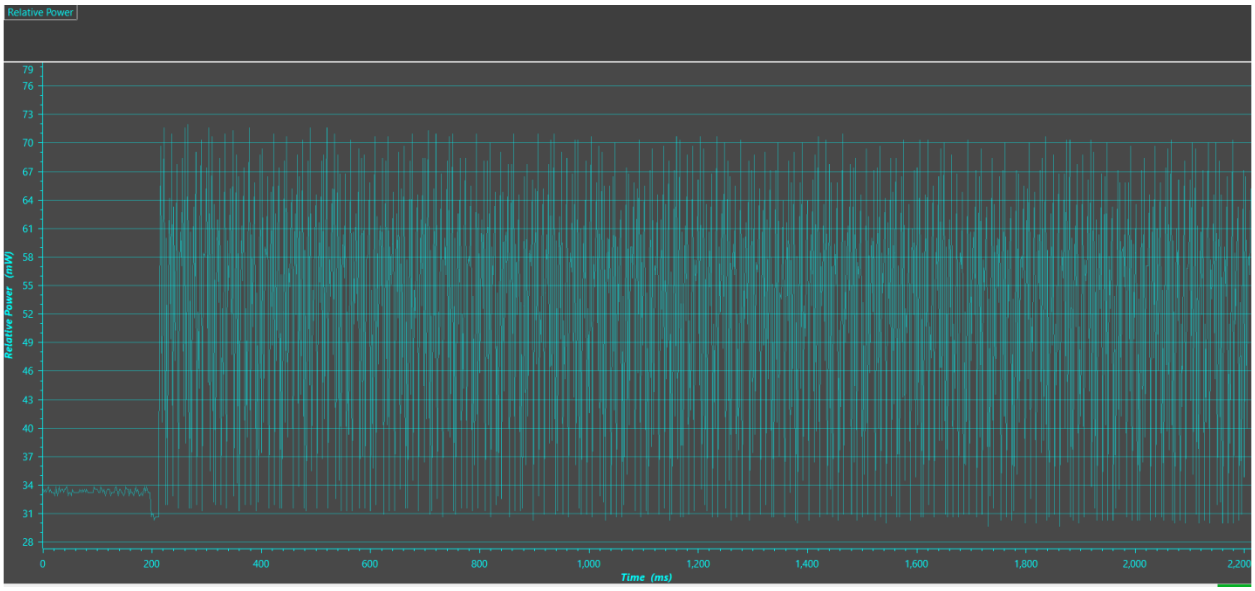

Figure 55: EnergyTrace plot when nRF24L01+ starts transmitting

Figure 56 and Figure 57 show the EnergyTrace plot for power measurement at 2.5 us delay at 1Mbps and 2 Mbps data rates respectively. In both figures, we observe the effect of current draw at periodic intervals due to the delay in the transmission. The minimum power consumption in the circuit is about 25 mW. As the nRf24L01+ starts transmitting, the power in the circuit goes high in the periodic intervals. That periodic interval depends upon the delay in transmission.

EnergyTrace™ Profile (Relative Measurement)

Name	Live
⌄ System	
Time	16 sec
Energy	803.629 mJ
⌄ Power	
Mean	48.5885 mW
Min	22.8063 mW
Max	61.7879 mW
⌄ Voltage	
Mean	3.3000 V
⌄ Current	
Mean	14.7238 mA
Min	6.9110 mA
Max	18.7236 mA
Battery Life	CR2032: 0.5 day (est.)

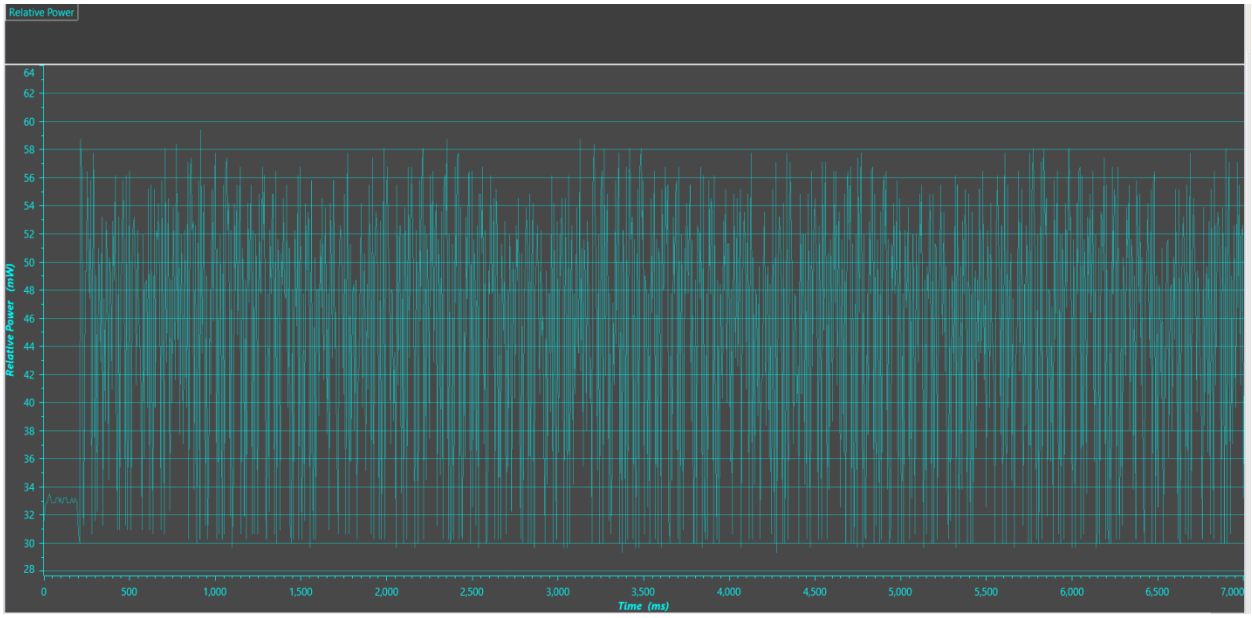

Figure 56: EnergyTrace Measurement at 2.5 us delay at MAX Power and 1Mbps

Name	Live
System	
Time	10 sec
Energy	408.759 mJ
Power	
Mean	41.9307 mW
Min	20.5392 mW
Max	61.2530 mW
Voltage	
Mean	3.3000 V
Current	
Mean	12.7063 mA
Min	6.2240 mA
Max	18.5615 mA
Battery Life	CR2032: 0.7 day (est.)

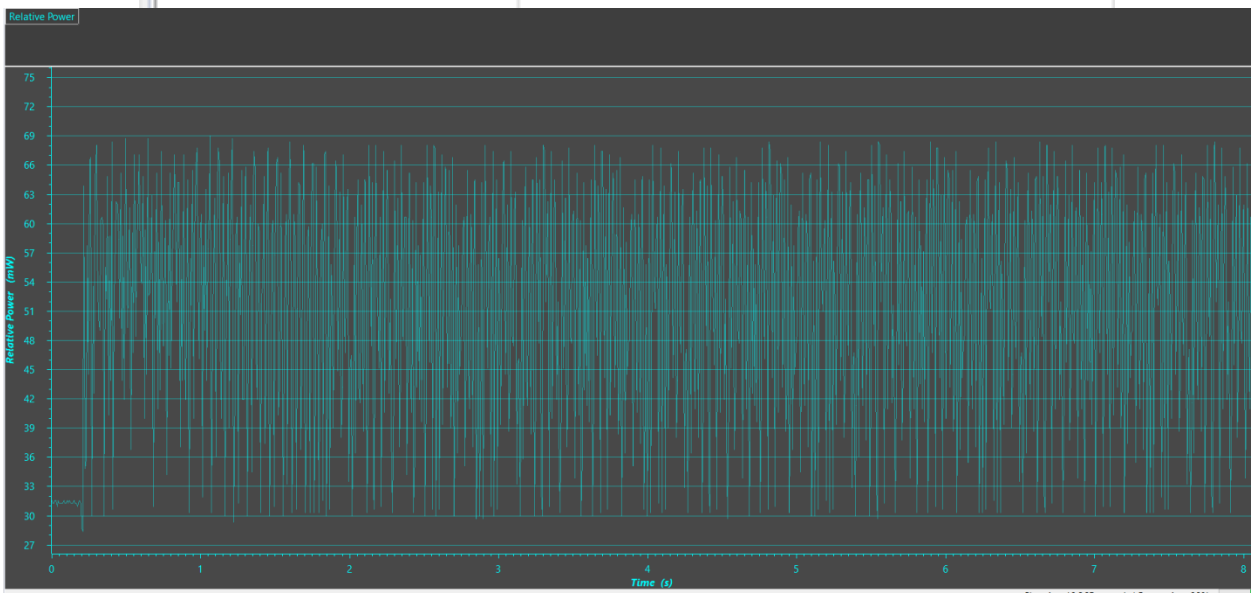

Figure 57: EnergyTrace Measurement at 2.5 us delay at MAX Power and 2Mbps

Transmitter

Table 15 shows the current consumption of nRF24L01+ while transmitting at max power (0 dBm) in different dates and delays. The current measurement was also made for other power settings in different data rates and delays which are given in Appendix C**.**

Table 15: Current Consumption of nRF24L01+ at 0 dBm in different data rates and delays

Delay(us)	Speed	Current(mA)
2.5	250 Kbps	15.97
250	250 Kbps	15.97
25000	250 Kbps	15.88
2.5	1 Mbps	11.41
250	1 Mbps	7.99
25000	1 Mbps	8.46
2.5	2 Mbps	11.25
250	2 Mbps	5.08
25000	2 Mbps	5.573

Receiver

Table 16 shows the current consumption of nRF24L01+ while receiving at max power (0 dBm) in different dates and delays. The current was also measured for other power settings in different data rates and delays; those measured values are given in Appendix C**.**

Table 16: Current Consumption of nRF24L01+ in receiver side at 0 dBm in different data rates and delays

Delay(us)	Speed	Current(mA)
2.5	250 Kbps	12.75
250	250 Kbps	12.89
2.5	1 Mbps	12.49
250	1 Mbps	12.27
2.5	2 Mbps	12.19
250	2 Mbps	12.70

4.4.1.2 Current Measurement using Circuit

This current measurement technique is discussed in section 3.2.4.2.2. The continuous transmission was done, and the current was measured in different data rates and power settings in both the transmitter and receiver side.

Transmitter

Table 17 shows the current consumption of nRF24L01+ while transmitting at max power (0 dBm) in different dates. The current measurement was also made for other power settings in different data rates which are given in Appendix C**.**

Table 17: Measured Current values in 0 dBm power (TX)

Speed	Current(mA)
250 Kbps	9.8
1 Mbps	6.51
2 Mbps	4.85

Receiver

Table 18 shows the current consumption of nRF24L01+ while receiving at max power (0 dBm) in different dates. The current measurement was also made for other power settings in different data rates which are given in Appendix C**.**

Table 18: Measured Current values in 0 dBm power (RX)

Speed	Current(mA)
250 Kbps	13.15
1 Mbps	11.67
2 Mbps	10.42

4.4.2 ESP-Now and BLE

The current measurement technique used for measuring the current in both ESP-Now and BLE is described in section 3.2.4.2.1.Table 19 shows the measured current values in BLE in both transmitter and receiver.

Table 19: Current Measurement of BLE

	Voltage across resistor(mv)	Resistance(ohm)	Current(mA)
TX	0.407	0.005	81.4
RX	0.333	0.005	66.6

Table 20 shows the measured current values in ESP-Now in both transmitter and receiver.

Table 20: Current Measurement of ESP-Now

	Voltage across resistor(mv)	Resistance(ohm)	Current(mA)
TX	0.650	0.005	130
RX	0.740	0.005	148

5 Discussion

5.1 Throughput

5.1.1 Performance of nRF24L01+ in different configurations

The throughput of nRF24L01+ was found to be independent of different power settings. This can be observed from Table 1 to Table 4. As expected, the throughput of nRF24L01+ was changing with data rates in both one to one and star connections. Figure 58 shows the relation of throughput with different data rates in nRF24L01+ in a star network when payloads of different sizes were transmitted.

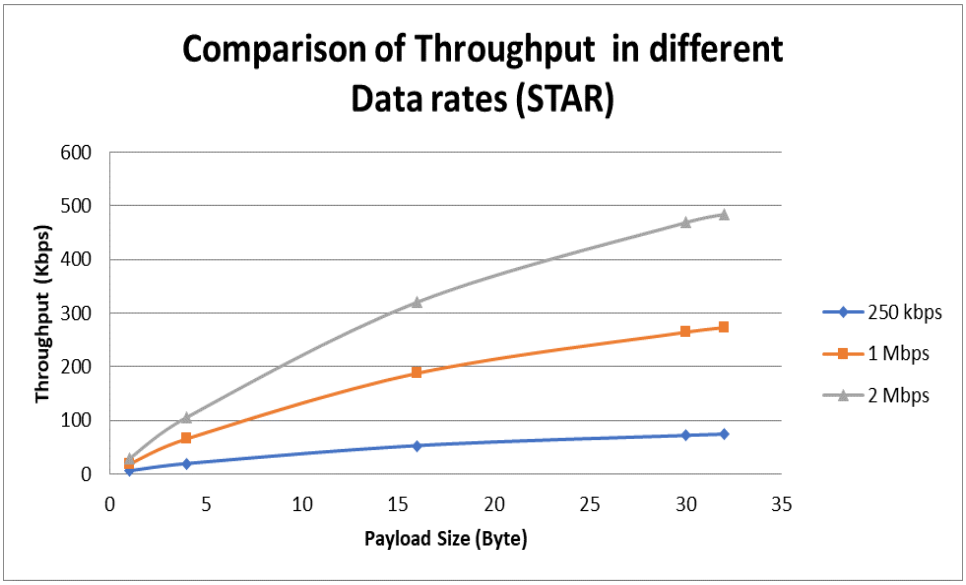

Figure 58: Throughput of nRF24L01+ in different Data rates

Besides this, the throughput of nRF24L01+ was also found to be changing when acknowledgment with payload feature was enabled**.** Figure 59 shows the comparison of throughput of nRF24L01+ in one to one, star and when acknowledgment with payload is enabled

in star connection. As seen from Figure 59, the throughput becomes less when acknowledgment with payload feature is enabled.

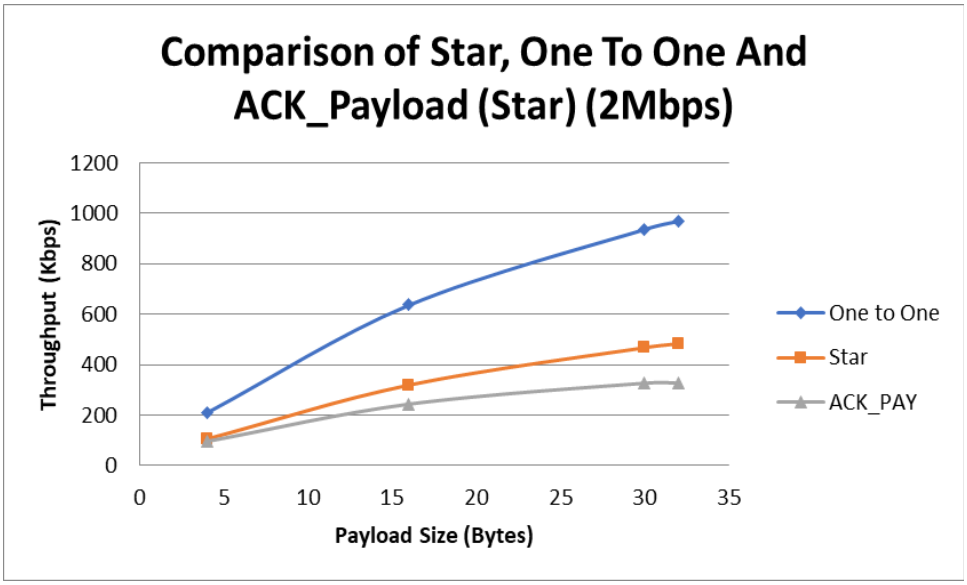

Figure 59: Comparison of Throughput in Star, One to One and ACK_Payload (Star)

5.1.2 Comparison of three protocols

5.1.2.1 One to One Connection

For the comparison of throughput between three protocols, the maximum throughput condition (2Mbps) of nRF24L01+ is considered. Table 21 shows the comparison of throughput between three protocols at different payload sizes in one to one connection.

Table 21: Throughput of three protocols in different Payload Sizes in One to One

Throughput (Kbps)			
Payload Size	nRF24L01+	ESP-Now	BLE
1	55.56	8.611	69.89
4	209.15	33.126	225.35
16	636.81	120.98	338.23
32	969.7	217.13	498.05

Figure 60 shows the comparison of throughput in graphical form. From Figure 60 it can be seen that the throughput of BLE is higher than nRF24L01+ when the payload size is smaller (1byte and 4 bytes). But as the payload size increases, the throughput of nRF24L01+ becomes greater than the BLE's throughput. The throughput of ESP-Now is the lowest one in all payload sizes condition.

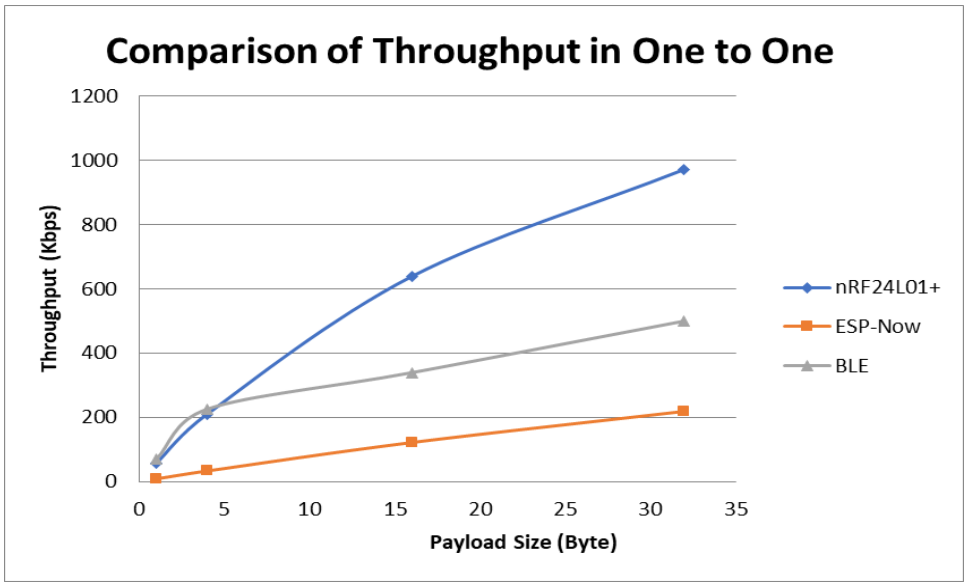

Figure 60: Comparison of Throughput in One to One connection

Thus from Figure 60 and Table 21, it can be concluded that in one to one connection the throughput of nRF24L01+ is highest among these three protocols when the payload size is bigger (more than 4 bytes).

5.1.2.2 Star Connection

Similar to one to one connection, maximum throughput condition of nRF24L01+ (2Mbps) is taken for throughput comparison in a star network. Table 22 shows the throughput value of three protocols in different payload sizes in star connection. The throughput value of BLE in Table 22 is not a valid one.

Table 22: Comparison of Throughput of three protocols in Star connection

Throughput (Kbps)			
Payload Size	nRF24L01+	ESP-Now	BLE (bps)
1	28.07	4.29	1.6
4	105.26	16.54	6.25
16	320	60.54	25.64
32	483.93	108.76	50.50

In case of BLE, during the second transmission, from the central hub to end peripheral node the measured time taken value was longer than expected. There was always some delay from the client (node) side when trying to read two different data in quick succession. After it received the first data, it took some time before making another read request to the server. The library that was used in this thesis was using *getValue()* function to make a read request to the server. The client can only read the server characteristic's value after making a read request. During implementation, it was seen that this function *getValue()* was not able to make read request in quick succession, that is why data was being transmitted from server to client with a delay. Due to this reason, the throughput value of BLE in Table 22 is not the valid one. Many data packets were sent one after another without any delay in server side to check the reading capability of the client, but many packets were lost during the transmission.

To overcome this issue, the library contributors were contacted. They advised another method known as notify() to send data from server to client. In this method, the server notifies the client as soon as the characteristic's value in the server side is changed. Using this method, the server was able to notify the client and client was able to read two different values in quick succession, and thus giving the correct time taken value and throughput. But since the current BLE library used in this thesis does not support the multi-connection feature of BLE, the notify() could not

send data to more than one client even though server was connected to multiple clients. Thus, notify() method could not be used to realize the star connection.

Figure 61 shows the comparison of throughput between in nRF24L01+ and ESP-Now in star connection. The throughput value for BLE is not included since the calculated data is not the valid one. The throughput for nRF24L01+ is greater than ESP-Now in star connection.

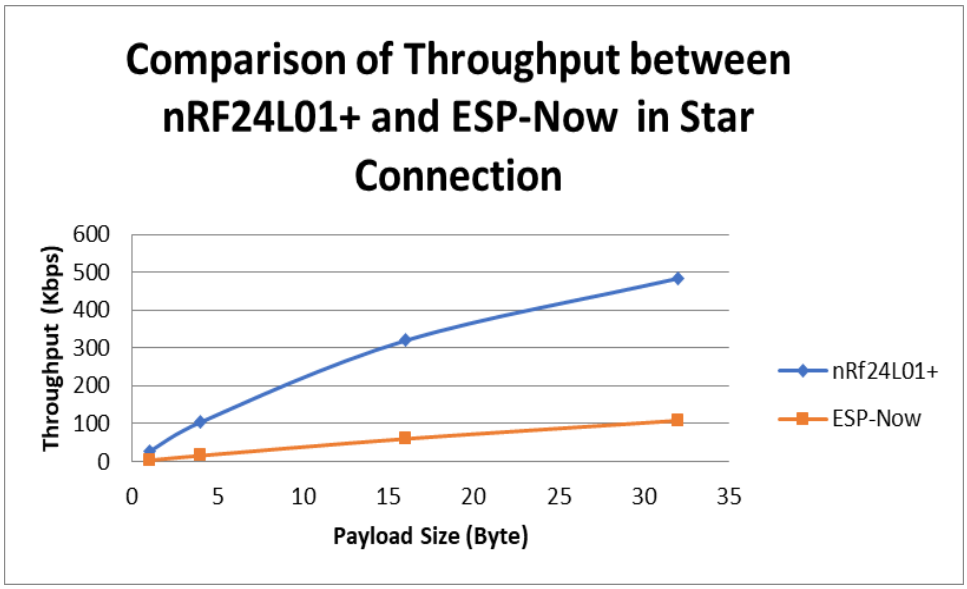

Figure 61: Comparison of Throughput in Star Connection

5.2 Range

Figure 62 shows the comparison of the range between nRF24L01+, ESP-Now and BLE in star connection. In the case of nRF24L01+, the range for different available power settings has also been calculated. The data for Figure 62 is given in Table 14. From Figure 62, it can be observed that the ESP-Now has the highest range among the three protocols. BLE comes in second while the nRF24L01+ has the smallest range. Figure 62 also shows that the range of nRF24L01+ can be increased by increasing the reception and transmission power.

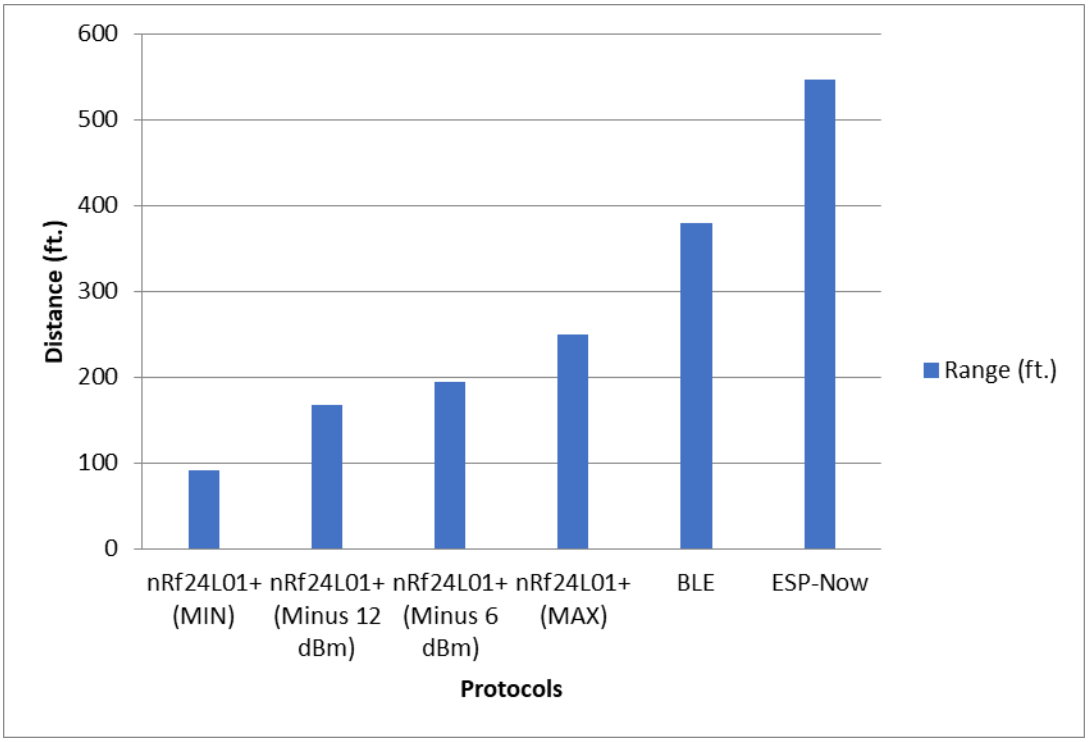

Figure 62: Comparison of range between nRF24L01+, BLE and ESP-Now in Star Connection.

5.3 Network Recovery Time

Table 23 shows the comparison of Network Recovery Time between three protocols in both cases. The network recovery time of ESP-Now and BLE was found to be significantly higher than nRF24L01+. When power was removed from the devices, nRF24L01+ took 335 us, ESP-Now took about 31ms on average, and BLE took 1.3 s to reconnect to the network.

Similarly, when taken out of range, the network recovery time for nRF24L01+, ESP-Now and BLE were 224us, 34ms, and 1.34s respectively. In both cases, the Network Recovery Time for BLE was the highest one. The reason for large Network Recovery Time in BLE might be due to the presence of many procedures involved to make a successful connection. Unlike in ESP-Now, in the BLE during scanning, a callback function is called for every possible advertising device (server) found in the surrounding. Also, when a device is found, initially it is checked whether

the new device has a service ID or not, and if it has one then, the found service ID is compared with the one defined in the program. If both service IDs are same, then it is assumed that the wanted device has been found, and scanning is stopped. Then, the next step in the BLE is to create an instance of client. Once the client is successfully created the connection to the desired device is established. Hence, the Network Recovery Time of the BLE is large due to presence of more procedures involved in connection process than the ESP-Now and nRF24L01+.

Table 23: Comparison of Network Recovery Time between three protocols

Condition		Time
Power Removed	nRF24L01+	335us
	ESP-Now	31ms
	BLE	1.3s
Out of Range	nRF24L01+	224us
	ESP-Now	34ms
	BLE	1.34s

5.4 Current Measurement

5.4.1 Performance of nRF24L01+ in different configurations

The current consumption in nRF24L01+ was found to be dependent upon both the transmission power and data rate. The current consumption increased as the power was increased. Whereas in the case of data rate, it was found that the current consumption in nRF24L01+ is inversely proportional to the data rate. The same result has also been found in another study [31]. It might be due to the fact that in higher data rate the transceiver spends less time in the power consuming mode, due to which the current consumption might be lesser in this condition compared to low data rate condition where the transmitter spends more time in power consuming mode.
Figure 63 shows the relationship between the current at different data rates at 0 dBm power at the 25000us delay.

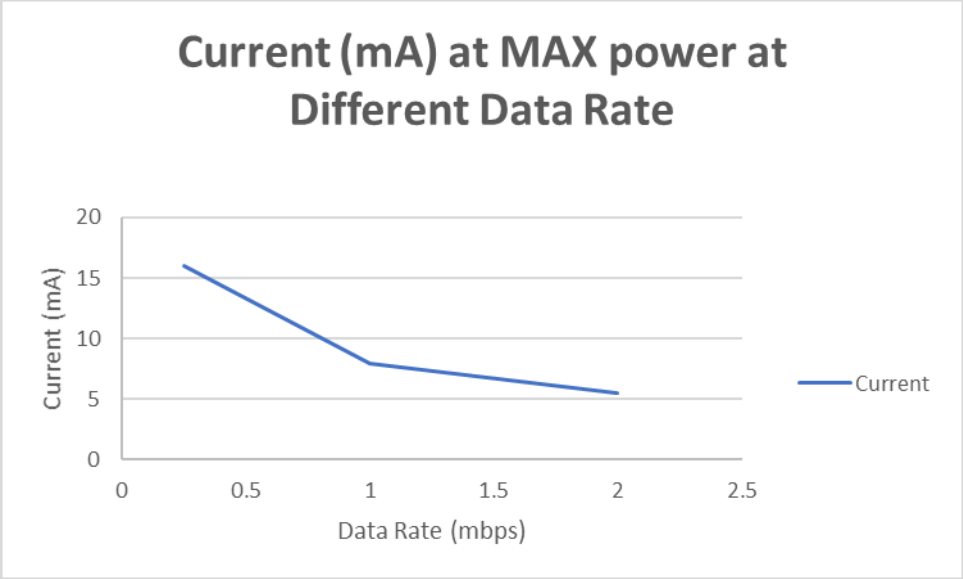

Figure 63: Current Consumption of PTX at Different Data Rate at MAX Power using EnergyTrace

Figure 64 shows the comparison of current at different power settings at 250 Kbps data rate. The current values used in Figure 64 were measured using EnergyTrace.

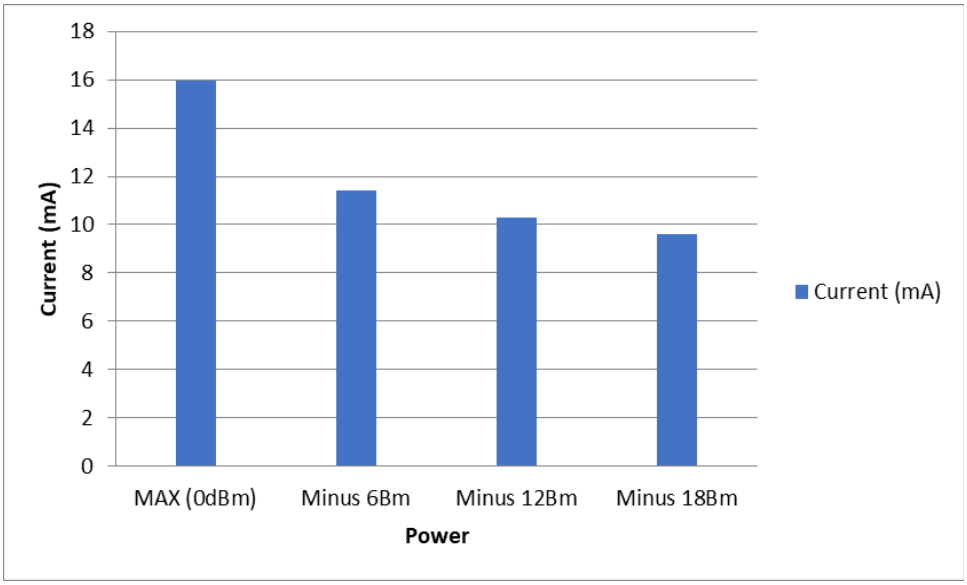

Figure 64: Current Consumption in Transmit Condition at Different Power Ratings

Besides these two things, the current consumption at PTX side when data rates were 1Mbps or 2Mbps was observed to be depending upon the delay between the transmissions. When the transmission was done quickly, the current consumption was higher than the condition when the transmission was done slowly as shown. This might be because when nRf24L01+ is transmitting with less delay (2.5us), many packets are lost due to the latency in the receiver. Due to this reason, the TX tries to resend the lost packet which increases the time spent in TX mode that might cause an increase in current consumption. Figure 65 shows the relationship between current and transmission delay at 0 dBm power and 2 Mbps data rate.

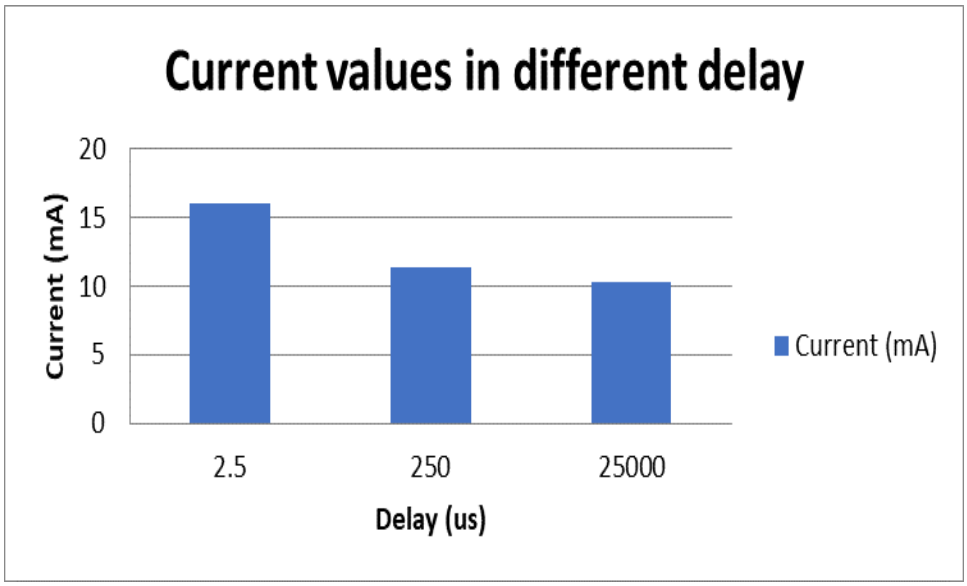

Figure 65: Current Consumption at Different Delays at MAX Power (0 dBm) and MAX Data Rate (2Mbps)

The current consumption values of the nRf24L01+ was measured to be about 15.97mA and 12.89mA in the Transmitter and Receiver sides respectively. These measurements were done using the EnergyTrace software. Past studies [23] [28] have measured that the current

consumption of nRf24L01+ to be around 12mA and 13mA in Transmitter and Receiver sides respectively. Thus, it can be said that the measured current values in this thesis are almost similar to the values obtained from past studies.

5.4.2 Comparison of three protocols

Transmitter

Table 24 shows the current comparison between the three protocols. For nRf24L01+, measured current values at 0 dBm power and 250 Kbps data rate are taken for comparison.

Table 24: Comparison of Current Values in Transmitter

	Current(mA)
nRF24L01+ (EnergyTrace)	15.97
nRF24L01+ (Current Measurement Circuit)	9.8
BLE	81.4
ESP-Now	130

From Table 24, it can be seen that ESP-Now consumes the maximum amount of current while nRF24L01+ consumes the minimum one.

Receiver

Table 25 shows the current comparison between three protocols in the receiver side. For nRf24L01+, measured current values at 0 dBm power and 250 Kbps data rate are taken for comparison.

Table 25: Comparison of Current Values in Receiver

	Current(mA)
nRF24L01+ (EnergyTrace)	12.89
nRF24L01+ (Current Measurement Circuit)	13.15
BLE	66.6
ESP-Now	148

Table 25 clearly shows that in receiving side also, ESP-Now has the maximum current consumption while nRF24L01+ has the lowest one.

5.5 Bandwidth, Spectral Efficiency and Noise Reduction

All three protocols that are compared in this thesis operate in 2.4 GHz spread spectrum. The bandwidth of the nRF24L01+depends upon the data rate of the device. The highest available bandwidth in the nRF24L01+ is 2 MHz. This protocol provides 126 different channels in its frequency spectrum for communication. Whereas, the BLE has 40 channels in its frequency spectrum and each channel has a bandwidth of 2 MHz. Similarly, ESP-Now has 11 channels in its frequency band and each channel has bandwidth of 22 MHz. Among these three protocols, the low bandwidth technologies like BLE and nRF24L01+ can be used to transmit the small packet size in an application, but if large bandwidth is needed in an application, the ESP-Now can be used.

The nRF24L01+ has a spectral efficiency of 1.111bits/s/Hz at 2 Mbps and 1 Mbps data rate. The spectral efficiencies of BLE and ESP-Now are 0.35bits/s/Hz and 1.3 bits/s/Hz respectively. Spectral efficiency is the ability of a modulation scheme to accommodate data within a limited bandwidth. It measures how many bps can be transmitted in 1 Hz of bandwidth. So, it can be assumed that more data can be transmitted in high spectral efficiency scheme. Thus, in terms of spectral efficiency the ESP-Now is superior compared to BLE and nRF24L01+, and can transmit more data within its limited bandwidth.

ESP-Now and BLE provide different level of protection from the noise. BLE uses adaptive frequency hopping in its 37 data channels to avoid interference. BLE's adaptive frequency hopping enables BLE radio to identify the congested frequencies which are then avoided in future transactions. This helps to increase the throughput of the system while minimizing interference for the system. ESP-Now uses the same frequency and channel as Wi-Fi does. Wi-Fi uses direct sequence spread spectrum (DSSS) or orthogonal frequency division multiplexing

(OFDM) modulation over 11 RF channels (in US). Wi-Fi uses IEEE 802.11 DCF protocol to maximize the throughput of system by preventing the packet collisions. The DCF protocol is discussed in section 1.3.1.3.2. This protocol allows the Wi-Fi device to know about the availability of a channel before transmission is made. Beside this, DSSS or OFDM modulation technique provides significant resistance to transmission interference and jamming in Wi-Fi. Unlike ESP-Now and BLE, the nRF2401+ Enhanced ShockBurst Protocol does not provide effective noise mitigating techniques. It is recommended to use auto acknowledgment feature in this protocol to monitor the status of transmission. Also, while selecting the channel in nRF24L01+, it is recommended to select high frequency channels as other RF radios like Wi-Fi mostly work in low frequency.

In the latest version of ESP32 BLE, it is possible to connect 7 slaves at a time. While, for ESP-Now up to 20 peer devices can be connected at a time and for nRF24L01+ 6 devices can be connected to a PRX in a Multiceiver network.

5.6 Maximum Payload Size

Table 26 shows the maximum payload size available in three protocols. The BLE has the maximum payload size while nRF24L01+ has the minimum one. This indicates that unlike nRF24L01+, BLE and ESP-Now can be used to transmit the large payload.

Table 26: Maximum Payload Sizes in Three Protocols

Protocol	Payload Size (byte)
nRF24L01+ (ShockBurst)	32
ESP-Now	250
BLE	500

5.7 Security

ESP32 has its own native hardware encryption for AES.The security in ESP32 BLE is defined by Bluetooth Specification version 4.2. It is implemented on the ESP-IDF, specifically on the Security Manager Protocol API. The security in BLE consists of three concepts: pairing, bonding and encryption. Pairing means the exchange of security features and types of keys needed. The generation and exchange of shared keys is handled by pairing procedure. ESP32 supports both leagacy and Secure Connections pairing. Once the shared keys are exchanged, next task is to store the exchanged keys. This process of storing is known as bonding. At last, the encrprtion of the palin texts ca be done using AES-128 engine and the shared keys. ESP32 offers different types of security modes for ESP-Now such as WPA, WPA2,WEP,etc. The different modes of security in ESP32 Wi-Fi or ESP-Now can be done by choosing proper auhentication mode in *wifi_auth_mode_t* parameter. However, nRF24L01+ does not provide any way to encrypt the signals in Enhanced ShockBurst Protocol. The encryption in nRF24L01+ can be done using the hardware accelerated encryption available in adajcent microcontroller. For example, the encryption of data would be possible by using AES256 accelerator module present in MSP432 which performs the encryption and decryption according to the AES256 encryption standard.

5.8 Design cost

The nRF24L01+ requires an external microcontroller to make communication with each other. The microcontroller and nRF24L01+ are connected with each other by using wires. Due to this fact, the design of the circuit is not that compact, and there are always some risks for loose connections. Also, nRF24L01+ is more prone to damage than ESP32. In this thesis, the

nRF24L01+ is getting its power from MSP432 Launchpad. In some cases, the nRF24L01+ drew high current from MSP432 may be due to loose connections and got damaged.

Unlike nRF24L01+, ESP32 does not need any external microcontrollers. Due to this reason, the design was more rigid and reliable. Thus, it can be concluded that ESP32 is much more rigid than nRF24L01+ and can handle more harsh conditions than RF chip. Table shows the cost of nRF24L01+ and ESP32.

Table 27: Cost of radio modules

ESP-Now	$14.95
nRF24L01+	$2

6 Conclusion

6.1 Summary

Three wireless protocols the proprietary Enhanced ShockBurst, Bluetooth Low Energy and the proprietary ESP-Now, were implemented in both one to one and star connections. The Nordic manufactured nRF24L01+ radio module was used for Enhanced ShockBurst while the Espressif ESP32 was used to implement BLE and ESP-Now. Once the communication was established different network performance metrics such as throughput, transmission time, network recovery time, range and current consumption were measured, and compared to each other to find the advantage and disadvantage of each protocol over one another.

The result tells that the nRF24L01+ has the highest throughput among three protocols when the payload size is large. But when the payload size is less than 4 bytes, the throughput of BLE is greater than the nRF24L01's throughput value. Similarly, in the case of node to node transmission time, the BLE has the lowest transmission time value in small payload size but as payload size increases the transmission time for BLE becomes greater than nRF24L01+. The ESP-Now has maximum transmission time and lowest throughput for all payload sizes. The experimental result shows that the current consumption of nRF24L01+ depends upon the transmission power, data rate, and delay. The nRF24L01+ has maximum current consumption at max power (0dBm), minimum transmission delay and minimum data rate (250 Kbps). Among three protocols, the nRF24L01+ consumes least amount of current than BLE and ESP-Now in both transmitting and receiving states. ESP-Now consumes more current than BLE. Also, the result shows that ESP-Now has maximum range followed by BLE and nRF24L01+. The nRF24L01+ has the least range and the range of nRF24L01+ changes with the transmission and

reception power. Now in the case of network recovery time, the result shows that nRF24L01+ takes the least amount of time to reconnect to the network compared to the ESP-Now and BLE. BLE has the longest network recovery time in both conditions, when power is removed and when the device is taken out of range.

Thus, in this thesis, three different wireless protocols were implemented in star and one to one connection, and performance metrics were measured and compared with each other. From the results, it was seen that the nRF24L01+ has maximum throughput when transmitting large payloads (344% higher than ESP-Now in 32 bytes payload size, BLE throughput was 50.50 bps in 32 bytes payload size), least current consumption (ESP-Now consumes 714% more current and BLE consumes 409.7% more current), and shortest network recovery time (nRF24L01+ took 335us, ESP-Now took 31ms and BLE took 1.3s on average), while the Wi-Fi based ESP-Now has a maximum range(30.53% better than BLE and 120% better than nRF24L01+).

6.2 Future Work

The star connection implementation of BLE network done in this thesis is not that reliable. The existing library does not support the multi-client connection. The new library is expected to be out in couple of months. After the new library is introduced, the star connection can be done to establish more reliable and efficient network. In the future implementation using new library the *notify()* method can be used to read the value form server. Beside this, during the implementation of star network it was found out that the client is not able to make read requests to server in quick succession. In the future, further research can be done in this issue. In this thesis, in ESP-Now the data transmission is being done using the call back functions. In future, the concept of MQTT can be applied to implement the ESP-Now and the results from the current implementation can be compared with the MQTT's implementation to find the advantage and disadvantage of one

over another. MQTT stands for Message Queuing Telemetry Transport, is an ISO standard machine to machine, Internet of Things connectivity protocol. This messaging protocol is for IoT devices and low bandwidth, high latency or unreliable networks. In addition, security concerns of intercepting wireless communications were not addressed in detail by this thesis. This security concern can also be compared in the future work. This work can be taken into the next level by comparing more network metrics and may by adding more wireless protocol or module.

Appendix A

A.1 Transmission Times

The following tables show the transmission times measured during 50 experiments for nRF24L01+, BLE and ESP-Now.

A.1.1 nRF24L01+

One to One connection

Table 28 shows the measured transmission time values for the nRF24L01+ in one to one connection when 32 bytes of data was transmitted.

Table 28: Transmission Time values for static payload length (32 bytes)

Transmission Time(us)		
250Kbps	1Mbps	2Mbps
1728	472	265
1726	470	263
1728	465	264
1727	467	266
1727	468	267
1728	467	264
1726	472	265
1725	470	263
1726	471	267
1727	472	264
1728	471	267
1727	472	265
1726	468	264
1729	467	265
1728	470	266
1725	470	267
1729	471	265
1730	471	263
1728	469	262
1725	468	268
1724	470	265
1727	471	266
1727	472	264
1728	470	267
1727	469	265
1726	468	268
1728	469	265
1728	470	263
1728	471	262
1727	472	265
1726	470	264
1726	469	267

1728	468	265
1726	469	263
1727	469	264
1728	470	265
1729	471	268
1726	472	265
1728	469	264
1729	470	264
1725	470	263
1727	471	267
1726	469	264
1728	468	263
1726	470	264
1727	471	265
1729	472	265
1725	473	266
1724	472	265
1730	471	265
1729	469	264

Table 29 shows the measure transmission time values for the nRF24L01+ in one to one connection when 1 byte of data was transmitted.

Table 29: Transmission Time values for 1 byte

Transmission Time(us)		
250Kbps	1Mbps	2Mbps
733	222	144
733	221	144
732	222	144
733	221	144
733	222	144
732	221	144
733	220	144
733	222	144
731	221	144
733	221	142
733	219	144
733	223	144
732	222	144
733	221	144
733	221	144
731	222	144
733	223	144
732	224	144
733	221	144
733	223	143
733	221	144
733	220	144
733	221	144
733	224	144
733	222	144
733	223	144
732	222	144
733	221	144
733	222	144
733	223	141
733	220	144
733	221	144
733	224	144
733	220	143

733	221	144
733	219	144
733	222	144
733	223	144
733	224	143
732	219	144
733	220	144
733	221	144
733	223	144
733	222	143
733	221	144
733	222	144
733	223	144
733	222	144
733	221	144
733	221	144
733	222	144

Table 30 shows the measured transmission values for the nRF24L01+ in one to one connection when 4 bytes of data was transmitted.

Table 30: Transmission Time values for 4 bytes

Transmission Time(us)		
250Kbps	1Mbps	2Mbps
852	247	154
851	246	153
851	244	154
848	246	152
849	244	153
850	243	154
850	243	156
851	244	153
849	243	154
848	244	154
849	243	153
850	245	156
852	246	152
850	245	157
849	244	154
849	243	155
850	244	154
852	242	156
851	244	154
852	246	156
848	247	153
849	243	156
850	244	154
847	244	154
852	241	153
852	246	156
851	244	154
849	245	155
850	246	154
851	243	155
849	247	156
850	241	153
850	243	156
851	246	153
852	244	154
850	248	156

848	245	154
849	243	153
850	246	153
850	244	154
848	244	156
850	244	154
852	246	153
853	244	154
851	245	154
846	243	156
850	245	157
850	244	156
851	243	153
852	244	154
852	244	155

Table 31 shows the measured transmission time values for the nRF24L01+ in one to one connection when 16 bytes of data was transmitted.

Table 31: Transmission Time values for 16 bytes

Transmission Time(us)		
250Kbps	1Mbps	2Mbps
1213	341	257
1213	342	256
1212	340	257
1214	341	257
1215	342	257
1216	339	255
1215	340	258
1214	340	254
1215	342	257
1216	341	256
1217	340	254
1213	342	255
1214	343	256
1215	342	253
1213	342	254
1214	342	256
1216	341	253
1214	340	257
1215	343	253
1216	342	255
1214	341	254
1214	342	253
1215	343	254
1216	342	254
1217	341	255
1214	342	254
1217	340	254
1214	342	256
1213	344	253
1212	340	256
1214	339	253
1214	344	254
1214	342	256
1215	340	253
1216	341	254
1217	141	253
1218	342	256
1214	341	253

1216	343	254
1214	342	253
1217	341	254
1218	340	256
1214	342	254
1216	341	253
1214	342	254
1213	341	253
1214	340	254
1215	340	255
1214	341	254
1214	342	255
1215	341	254

Star Connection

Table 32 shows the measured transmission time values for the nRF24L01+ in star connection when 32 bytes of data was transmitted.

Table 32: Transmission Time values for 32 bytes in star connection

Transmission Time (us)					
250 Kbps		1 Mbps		2 Mbps	
T1	T2	T1	T2	T1	T2
1724	1723	469	469	265	264
1723	1725	469	468	265	265
1725	1725	467	468	264	263
1724	1723	468	469	265	264
1726	1722	467	469	264	265
1723	1724	468	469	265	264
1723	1725	468	467	265	266
1724	1726	466	467	265	264
1722	1724	468	465	264	265
1723	1723	467	466	265	265
1724	1724	466	467	263	264
1725	1727	465	464	265	264
1723	1728	466	469	264	265
1724	1722	468	467	266	265
1722	1724	469	469	264	266
1724	1725	469	468	265	264
1725	1726	469	469	264	265
1726	1723	468	468	264	263
1725	1724	467	466	265	264
1722	1727	469	468	264	265
1723	1725	468	467	265	265
1723	1726	469	466	263	265
1724	1725	468	465	264	264
1723	1723	466	466	265	265
1722	1724	468	468	265	263
1721	1722	467	469	265	265
1724	1724	466	469	264	264
1725	1725	466	469	265	266
1727	1726	465	469	263	264
1728	1725	466	468	264	266
1722	1722	468	467	266	264
1724	1723	469	469	264	265
1725	1723	469	468	266	264

1726	1724	469	469	264	264
1723	1723	468	467	265	265
1724	1724	467	469	264	264
1727	1725	469	468	264	265
1725	1726	468	468	265	263
1726	1725	469	470	266	264
1725	1727	468	468	265	264
1723	1728	466	465	264	265
1724	1723	469	469	265	264
1723	1725	469	468	265	265
1725	1725	467	468	264	263
1724	1723	468	469	265	264
1726	1722	467	469	264	265
1723	1724	468	469	265	264
1723	1725	468	467	265	266
1724	1726	466	467	265	264
1722	1724	468	465	264	265

Table 33 shows the measured transmission times for the nRF24L01+ in star connection when 1 byte of data was transmitted.

Table 33: Transmission Time values for 1 byte in star connection

Transmission Time (us)					
250 Kbps		1 Mbps		2 Mbps	
T1	T2	T1	T2	T1	T2
733	733	222	222	144	144
733	733	221	221	144	144
733	733	222	222	144	144
733	733	223	221	141	145
733	733	224	220	144	144
732	732	219	222	144	144
733	733	220	221	144	144
733	733	221	221	144	144
733	733	223	224	142	144
732	732	222	222	144	144
733	733	221	223	144	144
733	733	222	222	144	143
733	732	223	221	144	144
733	733	222	222	144	144
733	733	221	223	144	144
733	732	221	220	144	144
733	733	222	221	144	142
733	733	224	224	144	144
731	731	221	220	144	144
733	731	223	223	144	144
733	733	221	221	144	144
733	732	220	220	144	144
733	733	221	221	144	143
732	733	224	224	143	144
733	733	222	222	144	144
733	733	223	223	144	144
733	733	222	222	144	144
731	733	221	221	144	144
733	733	222	222	144	144
733	733	223	223	145	144

733	732	220	220	144	144
731	733	221	221	144	144
733	733	224	224	144	144
733	732	220	220	144	144
732	733	221	221	144	144
733	733	222	222	145	145
733	733	223	221	144	144
733	731	224	221	144	144
733	733	221	219	144	144
731	733	223	223	144	144
733	733	221	222	144	144
733	731	220	221	144	144
733	733	221	221	144	144
733	733	224	222	144	144
733	733	222	223	144	144
733	733	223	224	144	144
732	733	222	221	144	143
733	733	221	223	143	144
733	732	221	221	144	144
733	733	221	220	144	144
733	733	222	222	144	144

Table 34 shows the measured transmission times for the nRF24L01+ in star connection when 4 byte of data was transmitted.

Table 34: Transmission Time values for 4 bytes in star connection

Transmission Time (us)					
250 Kbps		1 Mbps		2 Mbps	
T1	T2	T1	T2	T1	T2
840	841	246	245	152	153
840	840	245	245	152	151
839	841	246	246	152	152
840	839	247	245	151	153
840	841	246	246	152	154
840	841	245	245	153	151
841	840	244	245	152	153
839	840	245	246	152	152
840	839	245	244	151	151
840	841	246	245	151	152
839	840	246	245	150	153
841	839	245	246	151	152
841	840	246	245	152	153
840	841	246	245	151	152
839	841	246	246	152	152
839	840	245	246	153	151
841	840	246	245	151	152
841	840	246	245	152	152
842	840	246	245	152	152
841	841	246	244	153	153
840	842	245	245	150	152
840	841	245	245	151	153
841	839	245	246	152	152
842	840	244	246	153	152
841	840	245	245	152	153
839	839	245	246	152	152

840	841	246	246	152	152
840	841	246	246	152	152
839	842	247	247	153	151
839	841	245	245	152	151
841	840	244	245	152	150
841	840	245	246	152	151
840	841	246	245	151	152
840	839	245	245	153	151
841	840	246	245	152	152
842	840	246	246	153	153
840	842	245	246	152	151
839	841	246	245	152	152
840	840	246	246	151	152
841	840	247	246	152	153
841	841	245	245	152	152
840	842	245	246	153	151
841	841	246	246	152	152
841	842	245	245	151	151
840	841	244	246	152	152
841	841	246	245	152	152
840	840	245	244	153	152
841	842	246	245	152	152
840	841	245	245	152	153
841	841	246	245	152	151
840	842	244	245	152	152

Table 35 shows the measured transmission time values for the nRF24L01+ in star connection when 4 byte of data was transmitted.

Table 35: Transmission Time values for 16 bytes in star connection

Transmission Time (us)					
250 Kbps		1 Mbps		2 Mbps	
T1	T2	T1	T2	T1	T2
1215	1215	342	342	200	200
1215	1213	341	341	200	201
1214	1214	342	341	201	200
1213	1215	342	342	200	200
1215	1215	342	341	199	201
1214	1214	341	341	200	200
1214	1214	342	342	201	199
1213	1216	342	341	200	202
1214	1214	342	343	201	201
1215	1215	342	341	200	199
1213	1214	341	342	200	200
1215	1215	342	342	199	201
1214	1216	342	342	200	200
1213	1214	342	341	201	201
1214	1215	341	342	201	200
1215	1214	342	342	202	201
1214	1215	342	342	200	200
1214	1214	342	341	200	200
1215	1215	342	342	201	201
1214	1215	342	343	200	202
1215	1216	341	342	199	201
1213	1215	342	341	198	200

1215	1214	342	341	200	200
1215	1215	342	342	200	201
1214	1215	342	342	200	200
1214	1216	341	342	200	198
1213	1214	343	342	201	200
1215	1215	342	342	199	201
1214	1216	341	341	198	200
1215	1215	342	341	200	200
1216	1215	342	342	200	200
1215	1215	342	341	200	199
1215	1215	342	342	200	200
1214	1213	342	342	201	201
1215	1214	341	343	200	201
1216	1215	343	341	202	202
1214	1216	342	341	199	200
1215	1214	341	342	200	200
1215	1213	341	342	200	201
1216	1214	342	341	201	200
1215	1215	342	342	200	199
1214	1216	343	342	200	198
1215	1214	342	342	199	200
1215	1215	342	342	201	200
1216	1214	342	342	200	200
1214	1215	341	341	200	200
1215	1216	341	342	201	201
1216	1214	342	342	200	200
1215	1213	343	343	200	201
1215	1214	343	342	201	200
1216	1215	342	342	200	199

A.1.2 ESP-Now

Table 36 shows the measured transmission time values for the ESP-Now in both one to one and star connections when 1 byte of data was transmitted.

Table 36: Transmission Time of ESP-Now for 1 byte

Transmission Time(us) (1byte)		
One to One	Star	
	T1	T2
929	932	935
930	931	935
929	932	934
931	931	932
929	931	932
929	934	934
928	933	931
927	932	935
930	932	931
929	932	932
930	932	933
929	933	934
930	932	931
929	934	931
931	935	932
930	936	933
929	933	932
930	934	934
929	932	931
930	931	932
929	935	933
932	932	932
929	932	931
930	931	932
929	935	933
930	936	932
929	931	934
933	933	932
929	932	932
931	933	932
929	931	932
930	932	932
929	931	935
929	931	932
930	931	932
928	931	932
930	932	933
929	932	932
930	933	934
928	934	935
930	931	936
928	931	933
930	932	934
928	933	932

930	932	931
930	934	935
928	931	932
931	932	934
928	933	934
930	932	935

Table 37 shows the measured transmission time values for the ESP-Now in both one to one and star connections when 4 byte of data was transmitted.

Table 37: Transmission Time of ESP-Now for 4 bytes

Transmission Time(us) (4bytes)		
One to One	Star	
	T1	T2
955	957	953
955	957	953
955	957	953
964	966	962
963	944	961
959	978	957
982	952	980
990	955	988
991	955	989
955	993	959
973	983	977
945	984	949
942	962	946
976	951	980
950	958	954
953	959	957
953	955	957
991	993	995
981	983	985
982	984	986
960	962	964
955	957	953
962	964	960
963	965	961
957	959	955
958	976	956
959	983	957
980	975	978
987	971	985
979	970	977
975	958	973
974	959	972
962	960	960
963	989	961
964	981	962
974	976	991
977	979	983
969	971	979
970	972	978
950	952	966
962	964	967

958	976	970
963	979	981
973	971	973
975	972	974
956	952	954
959	960	962
964	966	968
955	957	953
955	957	953

Table 38 shows the measured transmission time values for the ESP-Now in both one to one and star connections when 16 byte of data was transmitted.

Table 38: Transmission Time of ESP-Now for 16 bytes

Transmission Time(us) (16bytes)		
One to One	Star	
	T1	T2
1021	1019	1023
1031	1029	1033
1060	1066	1062
1035	1073	1019
1064	1051	1029
1071	1122	1058
1049	1059	1033
1120	1078	1062
1057	1054	1069
1076	1066	1047
1052	1064	1118
1064	1062	1055
1062	1060	1074
1037	1035	1050
1070	1068	1072
1071	1069	1073
1063	1061	1065
1056	1054	1058
1064	1062	1066
1059	1057	1062
1045	1043	1057
1043	1041	1043
1036	1034	1041
1039	1037	1034
1047	1045	1037
1059	1057	1045
1065	1061	1057
1078	1047	1063
1077	1045	1079
1074	1038	1076
1072	1041	1074
1073	1049	1075
1080	1061	1082
1055	1067	1057
1056	1080	1058
1057	1079	1059
1054	1052	1056
1064	1062	1066

1067	1065	1069
1066	1068	1071
1067	1069	1078
1034	1036	1053
1040	1042	1054
1047	1049	1055
1045	1047	1052
1058	1060	1062
1060	1062	1062
1046	1048	1048
1021	1019	1023
1031	1029	1033

Table 39 shows the measured transmission time values for the ESP-Now in both one to one and star connections when 16 byte of data was transmitted.

Table 39: Transmission Time of ESP-Now for 32 bytes

Transmission Time(us) (32bytes)		
One to One	Star	
	T1	T2
1145	1147	1142
1180	1177	1187
1185	1182	1207
1205	1202	1174
1172	1169	1170
1168	1165	1208
1206	1203	1184
1182	1179	1199
1197	1194	1172
1170	1167	1193
1191	1188	1179
1177	1174	1176
1174	1171	1171
1164	1166	1161
1171	1173	1168
1197	1199	1194
1182	1184	1179
1185	1187	1182
1160	1157	1157
1181	1178	1186
1175	1172	1168
1176	1173	1180
1184	1181	1192
1166	1163	1194
1178	1175	1166
1190	1187	1168
1192	1189	1176
1164	1161	1174
1166	1163	1160
1174	1171	1164
1172	1169	1169
1158	1155	1155
1162	1164	1159
1188	1190	1185
1185	1187	1182

1186	1188	1183
1195	1197	1192
1179	1181	1176
1204	1185	1201
1209	1173	1211
1167	1172	1169
1188	1186	1190
1176	1165	1178
1175	1156	1177
1189	1167	1191
1168	1170	1170
1159	1161	1161
1170	1172	1172
1145	1147	1142
1180	1177	1187

A.1.3 BLE

Table 40 shows the measured transmission time values for the ESP-Now in both one to one and star connections when 1 byte of data was transmitted.

Table 40: Transmission Time of BLE for 1 byte

Transmission Time (1 byte)		
One to One (us)	Star	
	T1 (us)	T2 (s)
114	135	5.025
114	134	5.02
113	135	5.03
114	135	5.025
115	135	5.03
114	135	5.02
114	135	5.04
115	134	5.03
113	135	5.03
115	134	5.025
114	134	5.01
115	135	5.015
114	135	5.025
114	135	5.03
115	135	5.02
113	135	5.025
114	134	5.025
114	134	5.03
115	135	5.02
114	134	5.03
114	134	5.025
115	135	5.03
114	134	5.02
114	135	5.03
114	134	5.03
113	135	5.02
114	134	5.03
114	135	5.025
115	134	5.025
114	135	5.03
115	135	5.04

115	134	5.03
115	135	5.03
116	133	5.025
114	135	5.01
115	134	5.015
116	134	5.025
115	135	5.03
115	135	5.02
115	134	5.03
114	135	5.02
114	135	5.03
114	135	5.03
114	134	5.02
114	135	5.03
114	135	5.025
115	133	5.025
114	135	5.025
115	134	5.024
114	134	5.023

Table 41 shows the measured transmission time values for the ESP-Now in both one to one and star connections when 4 byte of data was transmitted.

Table 41: Transmission Time of BLE for 4 bytes

Transmission Time (4 bytes)		
One to One (us)	Star	
	T1 (us)	T2 (s)
141	156	5.025
143	162	5.02
141	155	5.03
142	159	5.025
141	157	5.03
142	160	5.02
141	153	5.04
143	154	5.03
142	159	5.03
141	161	5.025
142	155	5.01
143	157	5.015
142	158	5.025
141	159	5.03
143	160	5.02
144	155	5.025
142	154	5.025
140	158	5.03
142	157	5.02
141	159	5.03
142	153	5.025
143	155	5.03
142	156	5.02
141	154	5.03
142	158	5.03
143	157	5.02
141	161	5.03
140	155	5.025

142	156	5.025
141	155	5.03
140	157	5.04
141	159	5.03
142	158	5.03
141	156	5.025
142	157	5.01
142	158	5.015
141	154	5.025
142	158	5.03
142	157	5.02
141	156	5.03
141	157	5.02
143	158	5.03
140	154	5.03
143	158	5.02
141	157	5.03
142	161	5.025
143	155	5.025
141	156	5.025
142	159	5.024
141	158	5.023

Table 42 shows the measured transmission time values for the ESP-Now in both one to one and star connections when 16 byte of data was transmitted.

Table 42: Transmission Time of BLE for 16 bytes

Transmission Time (16 bytes)		
One to One (us)	Star	
	T1 (us)	T2 (s)
353	380	5.025
334	385	5.02
330	384	5.03
329	378	5.025
335	374	5.03
328	375	5.02
330	376	5.04
334	386	5.03
330	387	5.03
328	383	5.025
335	387	5.01
328	372	5.015
330	376	5.025
334	374	5.03
330	372	5.02
329	379	5.025
336	390	5.025
328	405	5.03
330	365	5.02
325	378	5.03
334	388	5.025
332	395	5.03
321	397	5.02
331	379	5.03
330	376	5.03

350	381	5.02
347	380	5.03
345	383	5.025
335	384	5.025
330	396	5.03
340	397	5.04
339	391	5.03
337	392	5.03
338	400	5.025
329	371	5.01
330	378	5.015
332	395	5.025
333	388	5.03
331	384	5.02
337	405	5.03
336	365	5.02
334	378	5.03
331	388	5.03
330	395	5.02
342	397	5.03
343	379	5.025
341	384	5.025
330	385	5.025
337	376	5.024
332	378	5.023

Table 43 shows the measured transmission time values for the ESP-Now in both one to one and star connections when 32 byte of data was transmitted.

Table 43: Transmission Time of BLE for 32 bytes

Transmission Time (32 bytes)		
One to One (us)	Star	
	T1 (us)	T2 (s)
528	548	5.025
505	520	5.02
515	535	5.03
511	540	5.025
508	536	5.03
512	550	5.02
514	545	5.04
505	537	5.03
511	539	5.03
509	536	5.025
512	537	5.01
514	564	5.015
513	547	5.025
505	543	5.03
510	542	5.02
509	541	5.025
511	540	5.025
513	541	5.03
517	536	5.02
516	537	5.03
514	530	5.025
518	538	5.03

508	539	5.02
520	543	5.03
521	542	5.03
519	556	5.02
514	542	5.03
517	540	5.025
518	547	5.025
511	556	5.03
510	539	5.04
521	536	5.03
520	537	5.03
513	564	5.025
516	547	5.01
517	543	5.015
518	542	5.025
509	541	5.03
518	540	5.02
522	552	5.03
514	546	5.02
515	547	5.03
509	555	5.03
508	559	5.02
514	560	5.03
513	548	5.025
517	537	5.025
520	535	5.025
514	536	5.024
521	545	5.023

Appendix B

B.1 Network Recovery Time

Table 44 shows the measured Network Recovery times for three protocols in 30 experiments.

Table 44: Network Recovery Time for three protocols

nRF24L01+		ESP-Now		BLE	
Power is removed (us)	Out of Range (us)	Power is removed (ms)	Out of Range (ms)	Power is removed (s)	Out of Range (ms)
335	224	18	17	1.15	1.2
332	223	19	19	1.2	1.3
334	220	19	21	1.3	1.18
336	222	20	25	1.25	1.15
335	221	21	56	1.35	1.3
335	225	22	39	1.2	1.4
334	226	23	35	1.3	1.5
336	227	21	15	1.27	1.4
332	224	24	40	1.34	1.25
334	226	67	23	1.3	1.35
335	223	56	31	1.4	1.30
334	225	50	32	1.5	1.7
332	224	70	54	1.2	1.2
333	224	29	42	1.1	1.19
334	225	39	27	1.3	1.55
335	221	35	60	1.5	1.42
336	223	25	22	1.2	1.45
335	226	36	33	1.4	1.50
334	224	60	35	1.5	1.39
333	225	26	18	1.6	1.26
335	223	27	24	1.4	1.27
335	224	33	26	1.3	1.29
336	225	29	34	1.2	1.37
334	224	16	52	1.2	1.22
333	225	35	27	1.3	1.55
335	224	26	29	1.7	1.3
334	225	24	31	1.1	1.4
336	222	28	32	1.2	1.35
334	221	27	31	1.4	1.27
335	223	23	24	1.5	

B.2 Range

One to One Connection

Table 45 shows the measured range values for three protocols in one to one connection. The experiment was conducted for thirty times.

Table 45: Measured Range values for three protocols in one to one connection

Distance (Ft.)					
nRF24L01+				BLE	ESP-Now
Minimum (-18dBM)	Medium (-12dBM)	(-6dBM)	Maximum (0 dBM)		
44	84	97	122	189	271
45	85	97	121	188	270
46	84	96	123	189	272
45	85	97	122	190	271
46	83	98	122	189	270
45	84	95	123	188	272
43	84	97	121	187	271
45	84	98	122	189	271
46	85	96	123	187	271
43	84	97	122	186	270
45	85	96	123	189	272
45	86	97	121	190	273
44	85	98	122	189	271
46	84	97	123	190	270
45	85	97	121	189	271
47	84	96	123	189	272
44	85	97	122	187	273
45	85	97	121	189	272
45	86	96	123	189	273
44	85	97	121	188	271
46	83	97	122	188	272
43	84	98	122	189	273
44	84	97	123	189	272
46	84	98	121	190	271
44	85	97	122	187	272
45	84	96	123	188	273
46	85	97	122	189	272
43	86	98	121	191	271
47	85	97	122	189	271
46	84	96	122	189	272

Appendix C

C.1 Current Measurement

EnergyTrace

Table 46 to Table 48 show the measured current values of nRF24L01+ (TX) in different power settings using EnergyTrace.

Table 46: Current Consumption in Min 6 dBm power

Delay(us)	Speed	Current(mA)
2.5	250 Kbps	11.41
250	250 Kbps	11.798
25000	250 Kbps	11.89
2.5	1 Mbps	9.51
250	1 Mbps	6.99
25000	1 Mbps	6.86
2.5	2 Mbps	8.50
250	2 Mbps	4.326
25000	2 Mbps	4.625

Table 47: Current Consumption in Min 12 dBm power

Delay(us)	Speed	Current(mA)
2.5	250 Kbps	10.27
250	250 Kbps	10.40
25000	250 Kbps	11.1
2.5	1 Mbps	8.62
250	1 Mbps	6.35
25000	1 Mbps	6.10
2.5	2 Mbps	8.32
250	2 Mbps	4.57
25000	2 Mbps	4.69

Table 48: Current Consumption in Min 18 dBm power

Delay(us)	Speed	Current(mA)
2.5	250 Kbps	9.64
250	250 Kbps	9.64
25000	250 Kbps	9.45
2.5	1 Mbps	7.74

250	1 Mbps	5.84
25000	1 Mbps	6.05
2.5	2 Mbps	5.98
250	2 Mbps	5.12
25000	2 Mbps	4.94

Table 46 to Table 48 show the measured current values of nRF24L01+ (RX) in different power settings using EnergyTrace.

Table 49: Current Consumption in Min 6 dBm power (PRX)

Delay(us)	Speed	Current(mA)
2.5	250 Kbps	9.90
250	250 Kbps	10.85
2.5	1 Mbps	11.28
250	1 Mbps	11.06
2.5	2 Mbps	11.80
250	2 Mbps	11.82

Table 50: Current Consumption in Min 12 dBm power (PRX)

Delay(us)	Speed	Current(mA)
2.5	250 Kbps	9.586
250	250 Kbps	9.7
2.5	1 Mbps	10.9
250	1 Mbps	11.17
2.5	2 Mbps	11.49
250	2 Mbps	11.701

Table 51: Current Consumption in Min 18 dBm power (PRX)

Delay(us)	Speed	Current(mA)
2.5	250 Kbps	10.85
250	250 Kbps	11.06
2.5	1 Mbps	11.70
250	1 Mbps	11.49
2.5	2 Mbps	11.701
250	2 Mbps	11.701

Using Current Measurement Circuit

Table 52 and Table 53 show the measured current values of nRF24L01+ in TX side in different power settings.

Table 52: Measured Current values in Min 12 dBm power (TX)

Speed	Current(mA)
250 Kbps	6.51
1 Mbps	5.41
2 Mbps	4.85

Table 53: Measured Current values in Min 18 dBm power (TX)

Speed	Current(mA)
250 Kbps	5.45
1 Mbps	4.5
2 Mbps	3.84

Table 54 and Table 55 show the measured current values of nRF24L01+ in TX side in different power settings.

Table 54: Measured Current values in Min 12 dBm power (RX)

Speed	Current(mA)
250 Kbps	13.15
1 Mbps	11.67
2 Mbps	10.42

Table 55: Measured Current values in Min 18 dBm power (RX)

Speed	Current(mA)
250 Kbps	13.15
1 Mbps	11.67
2 Mbps	10.42

Appendix D

The following source codes are for star connection network for three protocols. The source codes of both central hub and peripheral node for each protocol have been given. The library used for ESP32 BLE is given in Bibliography section [32].

D.1 nRF24L01+ Code (Star)

Central Hub

```
// Central Hub in Star Network
#include "driverlib.h"
#include "clk.h"
#include "ST7735.h"
#include "msprf24.h"
#include "nrf_userconfig.h"
#include "nRF24L01.h"
#include <string.h>
#include <stdio.h>
uint8_t user = 0;
int nrf_state = 0;
volatile uint32_t tick = 0;//Systick Timer variable
volatile uint32_t start_trans = 0 , end_trans = 0;//transmission time when packet is sent to the nodes
volatile uint32_t  diffTrans[10] = {0}, total_diffTrans = 0;
volatile long double  total_throughput, throughput[10] = {0};
int th =0;
uint8_t r_length;
void uart_init(){//initialization of UART
   const eUSCI_UART_Config uartConfig =
   {
         EUSCI_A_UART_CLOCKSOURCE_SMCLK,         // SMCLK Clock Source
         104,                                    // BRDIV = 26
         0,                                      // UCxBRF = 0
         0,                                      // UCxBRS = 0
         EUSCI_A_UART_NO_PARITY,                 // No Parity
         EUSCI_A_UART_LSB_FIRST,                 // MSB First
         EUSCI_A_UART_ONE_STOP_BIT,              // One stop bit
         EUSCI_A_UART_MODE,                      // UART mode
         EUSCI_A_UART_LOW_FREQUENCY_BAUDRATE_GENERATION  // Low Frequency Mode
   };
   /* Selecting P1.2 and P1.3 in UART mode. */
   MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1,
      GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

   /* Configuring UART Module */
   MAP_UART_initModule(EUSCI_A0_BASE, &uartConfig);
```

```
   /* Enable UART module */
   MAP_UART_enableModule(EUSCI_A0_BASE);
//
//   UART_enableInterrupt(EUSCI_A0_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT);
//   Interrupt_enableInterrupt(INT_EUSCIA0);

}
void rf_rx_init()
{
   uint8_t addr[5], addr1[5],addr2[5];
   uint8_t buf[32];
   user = 0xFE;
   /* Initial values for nRF24L01+ library config variables */
      rf_crc = RF24_EN_CRC | RF24_CRCO; // CRC enabled, 16-bit
      rf_addr_width      = 5;
      rf_speed_power     = RF24_SPEED_MAX | RF24_POWER_MAX;
      rf_channel         = 108;

      msprf24_init();//initialization of nRf24L01+
      MAP_Interrupt_enableMaster();
      msprf24_set_pipe_packetsize(0, 0);//enabling dynamic payload length in pipe 0
      msprf24_open_pipe(0, 1);  // Open pipe#0 with Enhanced ShockBurst
      msprf24_set_pipe_packetsize(1, 0);//enabling dynamic payload length in pipe 1
      msprf24_open_pipe(1, 1);  // Open pipe#1 with Enhanced ShockBurst
      msprf24_set_pipe_packetsize(2, 0);//enabling dynamic payload length in pipe 2
      msprf24_open_pipe(2, 1);  // Open pipe#2 with Enhanced ShockBurst

      // Set our RX addresses for each node
      addr[0] = 0xAA; addr[1] = 0xBB; addr[2] = 0xCC; addr[3] = 0xDD; addr[4] = 0xEE;//for node 1
      addr1[0] = 0xB4; addr1[1] = 0xB5; addr1[2] = 0xB6; addr1[3] = 0xB7; addr1[4] = 0xC0;//for node 2
      addr2[0] = 0xB4; addr2[1] = 0xB5; addr2[2] = 0xB6; addr2[3] = 0xB7; addr2[4] = 0xC0;//for node 3

      w_rx_addr(0, addr);//opening pipe 0 for node 1
      w_rx_addr(1, addr1);//opening pipe 1 for node 2
      w_rx_addr(2, addr2);//opening pipe 2 for node 3

      // Receive mode
      if (!(RF24_QUEUE_RXEMPTY & msprf24_queue_state())) {
         flush_rx();
      }
      msprf24_activate_rx();
      nrf_state = 1;
}
uint8_t rx_buff[32];
uint8_t buf[32];

void rf_rx(){
     if (rf_irq & RF24_IRQ_FLAGGED) {
       rf_irq &= ~RF24_IRQ_FLAGGED;
       msprf24_get_irq_reason();
   }
   if (rf_irq & RF24_IRQ_RX || msprf24_rx_pending()) {
      start_rec= tick;
      r_length = r_rx_peek_payload_size();
      memset(buf,0x00,32);
```

```
        memset(rx_buff,0x00,32);

        r_rx_payload(r_length, buf);
        reg1 =  r_reg(0x07);

        msprf24_irq_clear(RF24_IRQ_RX);
        memcpy(rx_buff,buf,32);
        printf("Received from node: %s",rx_buff);
//        printf("%s", rx_buff);
        nrf_state = 2;
    } else {
        user = 0xFF;
    }
}
void rf_init()
{
    uint8_t addr[5];
    uint8_t test = 0x00;

    user = 0xFE;
    /*Initial Values for NRF24L01+ library config variables*/
    rf_crc = RF24_EN_CRC | RF24_CRCO; // CRC enabled, 16-bit
    rf_addr_width      = 5;
    rf_speed_power     = RF24_SPEED_MAX | RF24_POWER_MAX;
    rf_channel         = 108;
    msprf24_init();
   test = r_reg(0x00);
    nrf_state = 3;
}
char txbuff[50];
int k;
void rf_transtx(){
    uint8_t addr[5], addr1[5];
    if (rx_buff[0] == '1')//if received device id is 1 then open pipe 0 for node 1
    {
          msprf24_set_pipe_packetsize(0, 0);
            msprf24_open_pipe(0, 1);  // Open pipe#0 with Enhanced ShockBurst enabled
            auto3 = r_reg(RF24_EN_AA);

            msprf24_standby();

            addr[0] = 0xAA; addr[1] = 0xBB; addr[2] = 0xCC; addr[3] = 0xDD; addr[4] = 0xEE;//node 1 address
            w_tx_addr(addr);
            w_rx_addr(0, addr);  // Pipe 0 receives auto-ack's, autoacks are sent back to the TX addr so the PTX
node
                              // needs to listen to the TX addr on pipe#0 to receive them.
    }
    else if (rx_buff[0] == '2')//if received device id is 1 then open pipe 0 for node 2
    {
          msprf24_set_pipe_packetsize(0, 0);
            msprf24_open_pipe(0, 1);  // Open pipe#0 with Enhanced ShockBurst enabled
            auto4 = r_reg(RF24_EN_AA);

            msprf24_standby();
            //user = msprf24_current_state();
```

```
            addr1[0] = 0xB4; addr1[1] = 0xB5; addr1[2] = 0xB6; addr1[3] = 0xB7; addr1[4] = 0xC0;//node 2
address
            w_tx_addr(addr1);
            w_rx_addr(0, addr1);  // Pipe 0 receives auto-ack's, autoacks are sent back to the TX addr so the PTX
node
                                // needs to listen to the TX addr on pipe#0 to receive them.
   }
   else if (rx_buff[0] == '3')//if received device id is 2 then open pipe 0 for node 2
    {
          msprf24_set_pipe_packetsize(0, 0);
            msprf24_open_pipe(0, 1);  // Open pipe#0 with Enhanced ShockBurst
            msprf24_standby();
            //user = msprf24_current_state();
            addr1[0] = 0xB4; addr1[1] = 0xB5; addr1[2] = 0xB6; addr1[3] = 0xB7; addr1[4] = 0xC1;//node 3
address
            w_tx_addr(addr2);
            w_rx_addr(0, addr2);  // Pipe 0 receives auto-ack's, autoacks are sent back to the TX addr so the
PTX node
                                // needs to listen to the TX addr on pipe#0 to receive them.
    }
   for ( k = 0; k<50 ; k++)
   {
      txbuff[k] = rx_buff[k+1];//remove device ID before sending the packet to the end user
   }
   nrf_state = 4;
}
void rf_tx(){
    w_tx_payload(r_length, txbuff);
   msprf24_activate_tx();//activate tx and send packet
   nrf_state = 5;
   if (rf_irq & RF24_IRQ_FLAGGED) {
      rf_irq &= ~RF24_IRQ_FLAGGED;
//       timer = count;
      msprf24_get_irq_reason();
      if (rf_irq & RF24_IRQ_TX){
         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2,GPIO_PIN1);
         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2,GPIO_PIN0);
      }
      if (rf_irq & RF24_IRQ_TXFAILED){
         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2,GPIO_PIN0);
         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2,GPIO_PIN1);
      }
      msprf24_irq_clear(rf_irq);
   }
}

void main(void)
{
   WDTCTL = WDTPW | WDTHOLD;          // Stop watchdog timer
   clk_init();
   uart_init();
   MAP_SysTick_enableModule();
   MAP_SysTick_setPeriod(48);//Systick Interrupt in every 1 us
   MAP_SysTick_enableInterrupt();//enable interrupt
   MAP_Interrupt_enableMaster();//enable master
```

```
    printf("STAR NETWORK IMPLEMENTATION\r\n");
    printf("Central Hub of Network\r\n");
    while (1)
    {
        if(nrf_state == 0)//state machine to control transceiver
        {
            rf_rx_init();//initialization in receiver mode
        }
        else if (nrf_state ==1)
        {
            rf_rx();//received data from node 0
            _delay_cycles(12000000);
        }
        else if (nrf_state == 2)
        {
            rf_init();//initialization as transmitter
//            _delay_cycles(12000000);
        }
        else if (nrf_state ==3 )
        {
            rf_transtx();//assigning end node in accordance to the device ID
        }
        else if (nrf_state == 4)
        {
            rf_tx();//transmit the packet
            _delay_cycles(12000000);
            printf("Transmitted to node %c\r\n", rx_buff[0]);

        }
        else if (nrf_state == 5)
        {
             diffTrans[th] = end_trans - start_trans - 130; //finding the time difference
             throughput[th] = ((r_length * 8) * 1000000) / diffTrans[th];
             tick = 0;
             nrf_state = 0;//go back to initial state
        }
    }
}
char buff[50];
void EUSCIA0_IRQHandler(void)
{
}
void SysTick_Handler(void)
{
//    MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
    tick++;
}
```

Peripheral Node

This is for only one node. Same code is also used for other nodes also, but the address of pipe will be different.

```
//*****************************************************************************
//nRF24L01+ star network node
//*****************************************************************************
#include "msp.h"
#include "driverlib.h"
#include "clk.h"
#include "msprf24.h"
#include "nrf_userconfig.h"
#include "nRF24L01.h"
#include "msp432_spi.h"
#include <string.h>
#include <stdio.h>
#define CPU_CLK_MHZ     48
#define SYSTICK_TICK_MS       10
uint8_t state_transceiver = 0;
char txbuff[50];
uint32_t packetSize = 0;
uint8_t  r_length;
uint8_t user = 0;
volatile uint32_t tick = 0;
volatile uint32_t start_trans = 0, end_trans = 0;//variables to calculate the time
volatile long double diffTrans[10] = {0}, throughput[10] = {0};
volatile long double total_diffTrans= 0, total_throughput = 0;
int th = 0;
void rf_rx_init()
{
   uint8_t addr[5];
   uint8_t buf[32];

   user = 0xFE;

   /* Initial values for nRF24L01+ library config variables */
      rf_crc = RF24_EN_CRC | RF24_CRCO; // CRC enabled, 16-bit
      rf_addr_width      = 5;
      rf_speed_power     = RF24_SPEED_MAX | RF24_POWER_MAX;
      rf_channel         = 108;

      msprf24_init();//initializing nRF24L01+
      MAP_Interrupt_enableMaster();
      msprf24_set_pipe_packetsize(0, 0);//Enabling Dyanmic Payload Length
      msprf24_open_pipe(0, 1);  // Open pipe#0 with Enhanced ShockBurst
      auto1 = r_reg( RF24_EN_AA );
      // Set our RX address
      addr[0] = 0xAA; addr[1] = 0xBB; addr[2] = 0xCC; addr[3] = 0xDD; addr[4] = 0xEE;//pipe address
      w_rx_addr(0, addr);

      // Receive mode
      if (!(RF24_QUEUE_RXEMPTY & msprf24_queue_state())) {
         flush_rx();
```

```
        }
        msprf24_activate_rx();
        state_transceiver = 1;
//        value_reg = r_reg(RF24_FEATURE);
}
uint8_t rx_buff[32];
char buf[32];
void rf_rx(){
    reg = r_reg(0x00);
    uint8_t addr[5];

    uint8_t temp;

    static int g_tcount= 0;
  //  int i;


    if (rf_irq & RF24_IRQ_FLAGGED) {
        rf_irq &= ~RF24_IRQ_FLAGGED;
        msprf24_get_irq_reason();
    }
    if (rf_irq & RF24_IRQ_RX || msprf24_rx_pending()) {
        r_length = r_rx_peek_payload_size();
       // temp = r_reg(9);
       // printf("RF Power: %d\r\n", temp);
        memset(buf,0x00,32);
        memset(rx_buff,0x00,32);
        r_rx_payload(32, buf);
        msprf24_irq_clear(RF24_IRQ_RX);
       // user = buf[0];
        memcpy(rx_buff,buf,r_length);
        printf("Received data:%s\r\n",rx_buff);

    } else {
        user = 0xFF;
    }

}
void rf_init()
{

    uint8_t addr[5];
    uint8_t test = 0x00;
    user = 0xFE;

    /*Initial Values for NRF24L01+ library config variables*/
    rf_crc = RF24_EN_CRC | RF24_CRCO; // CRC enabled, 16-bit
    rf_addr_width      = 5;
    rf_speed_power     = RF24_SPEED_MAX | RF24_POWER_MAX;
    rf_channel         = 108;

    msprf24_init();
    test = r_reg(0x00);
    w_reg(0x00,0x5);
    test = r_reg(0x00);
```

```
    msprf24_set_pipe_packetsize(0, 0);//Enabling Dyanmic Payload length
    msprf24_open_pipe(0, 1);//opening pipe 0 with Enhanced ShockBurst feature

    msprf24_standby();
    //user = msprf24_current_state();
    addr[0] = 0xAA; addr[1] = 0xBB; addr[2] = 0xCC; addr[3] = 0xDD; addr[4] = 0xEE;
    w_tx_addr(addr);
    w_rx_addr(0, addr);  // Pipe 0 receives auto-ack's, autoacks are sent back to the TX addr so the PTX node
    state_transceiver = 3;
}
uint8_t regadd = 0;
void rf_tx(){
    w_tx_payload(packetSize - 2, txbuff);
    msprf24_activate_tx();//activating transmitter
    state_transceiver = 4;
    if (rf_irq & RF24_IRQ_FLAGGED) {
        rf_irq &= ~RF24_IRQ_FLAGGED;
        msprf24_get_irq_reason();
        if (rf_irq & RF24_IRQ_TX){
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2,GPIO_PIN1);
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2,GPIO_PIN0);
        }
        if (rf_irq & RF24_IRQ_TXFAILED){
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2,GPIO_PIN0);
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2,GPIO_PIN1);
        }
        msprf24_irq_clear(rf_irq);
    }
}

void uart_init(){
    const eUSCI_UART_Config uartConfig =
    {
            EUSCI_A_UART_CLOCKSOURCE_SMCLK,         // SMCLK Clock Source
            104,                                    // BRDIV = 26
            0,                                      // UCxBRF = 0
            0,                                      // UCxBRS = 0
            EUSCI_A_UART_NO_PARITY,                 // No Parity
            EUSCI_A_UART_LSB_FIRST,                 // MSB First
            EUSCI_A_UART_ONE_STOP_BIT,              // One stop bit
            EUSCI_A_UART_MODE,                      // UART mode
            EUSCI_A_UART_LOW_FREQUENCY_BAUDRATE_GENERATION  // Low Frequency Mode
    };
    /* Selecting P1.2 and P1.3 in UART mode. */
    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1,
        GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

    /* Configuring UART Module */
    MAP_UART_initModule(EUSCI_A0_BASE, &uartConfig);

    /* Enable UART module */
    MAP_UART_enableModule(EUSCI_A0_BASE);

    UART_enableInterrupt(EUSCI_A0_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT);
    Interrupt_enableInterrupt(INT_EUSCIA0);
```

```
}
void main(void)
{

   WDTCTL = WDTPW | WDTHOLD;// Stop watchdog timer
   clk_init();//initialization of clock
   uart_init();//initialization of uart
//   rf_init();
   MAP_SysTick_enableModule();
   MAP_SysTick_setPeriod(48);//systick timer gives interrupt in every micro second

   MAP_SysTick_enableInterrupt();//enabling interrupt

   MAP_Interrupt_enableMaster();
//   value_reg = r_reg(RF24_STATUS );
   printf("STAR NETWORK IMPLEMENTATION\r\n");
   printf("This is node 1. Please enter 2 at the beginning of message to transmit to node 2: \r\n");
   while (1)
   {//state machine is used to control the transceivers processed

      if (state_transceiver == 0)
      {
         rf_rx_init();//initialization as receiver
       }
      else if (state_transceiver == 1)
        {
        rf_rx();//ready to receive
        }
     else if (state_transceiver == 2)
     {
       rf_init();//initialization as transmitter
     }
     else if (state_transceiver == 3)
     {
       printf("Packet to send is:\r\n%s\r\n",txbuff);
        rf_tx();//transmitted the packet
//        state_transceiver = 0;
     }
     else if (state_transceiver == 4)
           {
       diffTrans[th] = end_trans - start_trans - 130;// calculating the time taken for transmission
      throughput[th] = ( (packetSize - 2)*8) * 1000000)/ diffTrans[th];//calculating the throughput only packet size is
taken into consideration
      total_throughput =total_throughput + throughput[th];
      total_diffTrans = total_diffTrans + diffTrans[th];
      tick = 0;
      state_transceiver = 0;
           }

     __delay_cycles(12000000);

   }
}
void EUSCIA0_IRQHandler(void)
{
```

```
    static int i =0;
    int receive = UCA0RXBUF;//taking input from keyboard through terminal

    EUSCI_A_UART_transmitData(EUSCI_A0_BASE, receive);//using UART capability of MSP432
    txbuff[i++] = receive;//buffer to store the input date and time
    if(receive == 0x0a){
        txbuff[i] = '\0';
        packetSize = i;
        state_transceiver = 2;
        i = 0;
    }else{

    }
}
void SysTick_Handler(void)
{
    tick++;
}
```

D.2 BLE Code (Star)

Central Hub

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include "time.h"
//volatile uint32_t isrCounter = 0;
//volatile uint32_t islCounter = 0;
//volatile long difCounter = 0;
//volatile long throughput = 0;
int state = 0, states = 0, states1 =0;
char data[20];
char  node[20], rec12[20], rec_data1[20], rec_data2[20];
char  node2[20];
bool result;
///bool result2;
int comp1,comp2;
int num =0;//to check whether device ID is receive or not
int reception_complete = 0;//to make sure data is completely received
int state_num = 0;
int state_num_trans = 0;
int node_num = 0;//to check from which node initially data was received

volatile uint32_t rec_start = 0;
volatile uint32_t rec_end = 0;
volatile long difCounter = 0;


#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"// service UUID
#define CHARACTERISTIC_UUID1 "beb5483e-36e1-4688-b7f5-ea07361b26a8"//characteristic 1 UUID
#define CHARACTERISTIC_UUID2 "beb5483e-36e1-4688-b7f5-ea07361b26a9"//characteristic 2 UUID


bool bleConnected = false;

 BLEAdvertising *pAdvertising;//for advertising
 BLECharacteristic *pCharacteristicclient1;//for client 1
  BLECharacteristic *pCharacteristicclient2;//for client 2


bool IsValidNumber (char * rec12)//function to check whther received user data is device ID (number) or not
{
   if (rec12[0] < 48 || rec12[0] > 57)
     {
       return false;

     }
   return true;
}

class MyServerCallbacks: public BLEServerCallbacks {//callback function to connect or disconnect with client
```

```
  void onConnect(BLEServer *pServer){
    bleConnected = true;
    pAdvertising->start();
  };

  void onDisconnect(BLEServer* pServer)
  {
    bleConnected = false;
  }
};

class MyCallbacks: public BLECharacteristicCallbacks {
    void onRead(BLECharacteristic *pCharacteristic)
    {
        if (pCharacteristic == pCharacteristicclient1 ){
         if ( reception_complete == 1 && num == 1 && node_num == 2)//if data received from client 2 transmit it to
client 1
         {
           if (state_num_trans == 0)
           {
             pCharacteristic->setValue(rec_data1);//transmit first data
            Serial.println("transmitted rec_data1 node1:  ");
            Serial.println(rec_data1);
            state_num_trans = 1;
           }
           else if (state_num_trans == 1)
           {
             pCharacteristic->setValue(rec_data2);
             Serial.println("transmitted rec_data2 node1:  ");//transmit second data
           Serial.println(rec_data2);
            state_num_trans = 0;
            nodes = 0;
            num = 0;
           }
         }
         else
         {
          pCharacteristic->setValue("Hello");//arbitary value Hello is written to make sure that client is in receiving
state
         }

        }
        else  if (pCharacteristic == pCharacteristicclient2){
         if ( reception_complete == 1 && num == 1 && node_num == 1)//if data received from client 1 transmit it to
client 2
         {
             if (state_num_trans == 0)
           {

           pCharacteristic->setValue(rec_data1);
             Serial.println("transmitted rec_data1 node 2:  ");//transmit first data
           Serial.println(rec_data1);
            state_num_trans = 1;
           }
```

```
          else if (state_num_trans == 1)
          {
           pCharacteristic->setValue(rec_data2);
           Serial.println("transmitted rec_data2 node 2:  ");//transmit second data
          Serial.println(rec_data2);
           state_num_trans = 0;
           nodes = 0;
           num = 0;
        }
        }
        else
        {
         pCharacteristic->setValue("Hello");//arbitary value Hello is written to make sure that client is in receiving
state
        }

       }
    }
    void onWrite(BLECharacteristic *pCharacteristic){

  if (pCharacteristic == pCharacteristicclient1){
    node_num = 1;//value received from node 1

          if ( num == 0)//state to check whether the received value is device ID (number or not)
             {

               std::string value1 = pCharacteristic->getValue();//get value from client
               strcpy(rec12,value1.c_str());
               result = IsValidNumber(rec12);//check whether data is number or not
//               Serial.print(result);
               if (result)
               {
                 strcpy(node,rec12);
                 num  = 1;//reecived data is device ID now wait for  user data
                Serial.print("The data will be transmitted to node: ");
                Serial.println(node);

               }
          }
          else if (num == 1)
          {
           if (state_num == 0)//state to receive first user data
           {
             std::string value1 = pCharacteristic->getValue();
              strcpy(rec_data1,value1.c_str());//first user data received
               Serial.print("Value Received: ");
                 Serial.println("rec_data1: ");
                 Serial.println(rec_data1);
                 rec_start = micros();//arrival of first user data
                  Serial.println("rec_start");
                  Serial.println(rec_start);
                 state_num = 1;//receive second user data
           }
           else if (state_num == 1)//state to receive second user data
           {
```

```
                std::string value1 = pCharacteristic->getValue();
                strcpy(rec_data2,value1.c_str());//second user data arrived
                 Serial.print("Value Received: ");
                  Serial.println("rec_data2: ");
                  Serial.println(rec_data2);
                  rec_end = micros();//arrival of second user data
                  Serial.println("rec_end");
                  Serial.println(rec_end);
                  state_num = 0;//change it back to zero for another transmission
                  reception_complete = 1;
          }

         }
    }
    else if (pCharacteristic == pCharacteristicclient2){
  node_num = 2;//value received from node 2

        if ( num == 0)//state to check whether the received value is device ID (number or not)
           {
            std::string value1 = pCharacteristic->getValue();//get value from client
            strcpy(rec12,value1.c_str());
            result = IsValidNumber(rec12);//check whether data is number or not
//              Serial.print(result);
            if (result)
            {
             strcpy(node,rec12);
             num  = 1;//reecived data is device ID now wait for  user data
            Serial.print("The data will be transmitted to node: ");
            Serial.println(node);//data received from node now ready to transmit

           }
        }
        else if (num == 1)
        {
         if (state_num == 0)//state to receive first user data
         {
           std::string value1 = pCharacteristic->getValue();
           strcpy(rec_data1,value1.c_str());//first user data received
            Serial.print("Value Received: ");
             Serial.println("rec_data1: ");
             Serial.println(rec_data1);
             rec_start = micros();//arrival of first user data
              Serial.println("rec_start");
              Serial.println(rec_start);
             state_num = 1;//receive second user data
         }
         else if (state_num == 1)//state to receive second user data
         {

              std::string value1 = pCharacteristic->getValue();
              strcpy(rec_data2,value1.c_str());//second user data arrived
               Serial.print("Value Received: ");
                 Serial.println("rec_data2: ");
                 Serial.println(rec_data2);
```

```
            rec_end = micros();//arrival of second user data
            Serial.println("rec_end");
            Serial.println(rec_end);
            state_num = 0;//change it back to zero for another transmission
            reception_complete = 1;//data received from node now ready to transmit
          }
           }
     }
    }
};
void setup() {
  Serial.begin(115200);

  Serial.println("1- Download and install an BLE scanner app in your phone");
  Serial.println("2- Scan for BLE devices in the app");
  Serial.println("3- Connect to MyESP32");
  Serial.println("4- Go to CUSTOM CHARACTERISTIC in CUSTOM SERVICE and write something");
  Serial.println("5- See the magic =)");

  BLEDevice::init("MyE");//initialization of server
  BLEServer *pServer = BLEDevice::createServer();//create server
  pServer->setCallbacks(new MyServerCallbacks());//set callback function for server to connect or disconnect

  BLEService *pService = pServer->createService(SERVICE_UUID);//create Service of given UUID

  pCharacteristicclient1 = pService->createCharacteristic(//create Characteristic of defined UUID and open it in read
or write mode for client 1
                              CHARACTERISTIC_UUID1,
                              BLECharacteristic::PROPERTY_WRITE |
                              BLECharacteristic::PROPERTY_READ
                             );

  pCharacteristicclient2 = pService->createCharacteristic(//create Characteristic of defined UUID and open it in read
or write mode for client 2
                              CHARACTERISTIC_UUID2,
                              BLECharacteristic::PROPERTY_WRITE |
                              BLECharacteristic::PROPERTY_READ
                             );
  pService->start();//start service
  pAdvertising = pServer->getAdvertising();//get advertising
  pAdvertising->addServiceUUID(pService->getUUID());
  pAdvertising->start();//start advertising
}
void loop() {
  if (state == 0)//state where server is in receing stage, listening to clients for data
  {
    pCharacteristicclient1->setCallbacks(new MyCallbacks());

  pCharacteristicclient2->setCallbacks(new MyCallbacks());
     if (result)//device ID (or whole data received) received
     {
        state = 1;
     }

  }
```

```
 else if ( state == 1)//state where server is ready to transmit the data to client
 {
  comp1 = strcmp(node, "1");//comparing the received number to determine end user
  if (comp1 == 0 && reception_complete == 1 )
  {
   num1_transmission++;
   pCharacteristicclient1->setCallbacks(new MyCallbacks());//call callback function for only client 1
   state = 0;//going back to receiver state
  }
  comp2 = strcmp(node, "2");//comparing the received number to determine end user
  if (comp2 == 0 && reception_complete == 1 )
  {
   num2_transmission++;
   pCharacteristicclient2->setCallbacks(new MyCallbacks());//call callback function for only client 2
   state= 2;//calculating the time difference between arrival of data
  }
 }
 else if (state == 2 )
 {
  difCounter = rec_end - rec_start; //calculating the transmission time when data is received from node to central
hub
  state = 0;//going back to receiver state
 }
 delay(1000);
}
```

Peripheral Node

This is for only one node. Same code is also used for other nodes also, but the characteristic ID will be different.

//A BLE node

```
#include "BLEDevice.h"
#include <string.h>
//#include "BLEScan.h"
// The remote service we wish to connect to.
static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
// The characteristic of the remote service we are interested in.
static BLEUUID    charUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");
int state = 0;
int i =1 , j =1 ;
static BLEAddress *pServerAddress;
static boolean doConnect = false;
static boolean connected = false;
```

```
static BLERemoteCharacteristic* pRemoteCharacteristic;
char incomingchar;
int state_ble = 0;
char rec[10];
int result;
int throughput_state =0;
volatile uint32_t isrCounter = 0;//variable to store arrivable time of first character
volatile uint32_t islCounter = 0;//variable to store arrivable time of second character
volatile long difCounter;
static void notifyCallback(
  BLERemoteCharacteristic* pBLERemoteCharacteristic,
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    Serial.print("Notify callback for characteristic ");
    Serial.print(pBLERemoteCharacteristic->getUUID().toString().c_str());
    Serial.print(" of data length ");
    Serial.println(length);
}
bool connectToServer(BLEAddress pAddress) {
  if (state == 0)
{
    Serial.print("Forming a connection to ");
    Serial.println(pAddress.toString().c_str());

    BLEClient*  pClient  = BLEDevice::createClient();
    Serial.println(" - Created client");
    // Connect to the remove BLE Server.
    pClient->connect(pAddress);
    Serial.println(" - Connected to server");

    // Obtain a reference to the service we are after in the remote BLE server.
    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    if (pRemoteService == nullptr) {
      Serial.print("Failed to find our service UUID: ");
      Serial.println(serviceUUID.toString().c_str());
      return false;
```

```
  }
  Serial.println(" - Found our service");
  // Obtain a reference to the characteristic in the service of the remote BLE server.
  pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
  if (pRemoteCharacteristic == nullptr) {
    Serial.print("Failed to find our characteristic UUID: ");
    Serial.println(charUUID.toString().c_str());
    return false;
  }
  Serial.println(" - Found our characteristic");
}
else if (state == 1)
{
  std::string myValue = pRemoteCharacteristic->readValue();//reading value from server
  strcpy(rec, myValue.c_str());
  result = strcmp(rec,"Hello");//Checking whether the received data is user data or not
  if (result != 0)
  {
    if (throughput_state == 0)
    {
      isrCounter = micros();
      Serial.print("The characteristic value was ");
      Serial.println(rec);
      Serial.println(isrCounter);//arrival of first character
      throughput_state = 1;
    }
    else if (throughput_state ==1)
    {
      islCounter = micros();
      Serial.print("The characteristic value was ");
      Serial.println(rec);
      Serial.println(islCounter);//arrival of second character
      throughput_state = 2;
    }
    else if (throughput_state ==2)
    {
      difCounter = islCounter - isrCOunter;//transmission time from server to node
```

```
      }
    }
  }
    else if (state == 2)
  {
    pRemoteCharacteristic->writeValue(incomingchar);//write the value entered by user to the server
  }
   else if (state == 3)
  {
    pRemoteCharacteristic->writeValue('a');//writing two data one after another to server
    pRemoteCharacteristic->writeValue('b');
  }
    pRemoteCharacteristic->registerForNotify(notifyCallback);
}
/**
 * Scan for BLE servers and find the first one that advertises the service we are looking for.
 */
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
 /**
   * Called for each advertising BLE server.
   */
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.print("BLE Advertised Device found: ");
    Serial.println(advertisedDevice.toString().c_str());
    // We have found a device, let us now see if it contains the service we are looking for.
    if (advertisedDevice.haveServiceUUID() && advertisedDevice.getServiceUUID().equals(serviceUUID)) {
      //
      Serial.print("Found our device!  address: ");
      advertisedDevice.getScan()->stop();

      pServerAddress = new BLEAddress(advertisedDevice.getAddress());
      doConnect = true;
    } // Found our server
  } // onResult
}; // MyAdvertisedDeviceCallbacks
void setup() {
  Serial.begin(115200);
```

```
  Serial.println("Starting Arduino BLE Client application...");
  BLEDevice::init("");
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);//starting the scanning and running it for 30 seconds
  pBLEScan->start(30);
} // End of setup.
void loop() {
    if (state == 0)//making connection to server
    {
        if (doConnect == true) {
       if (connectToServer(*pServerAddress)) {
         Serial.println("We are now connected to the BLE Server.");
         connected = true;
       } else {
         Serial.println("We have failed to connect to the server; there is nothin more we will do.");
       }
       doConnect = false;
       state = 1;
       }
     }
   else if (state == 1)
   {
      connectToServer(*pServerAddress);//reading value from server
     if (Serial.available () > 0 )
       {
         incomingchar = Serial.read();//reading value from serial monitor
     state = 2;//state where we write the user entered value from serial monitor to server
     connectToServer(*pServerAddress);
     state = 3;
       }
   }
    else if (state == 3)//writing two characters 'a' and 'b' to server
    {
     Serial.print("Transmission:");
     Serial.println(i);
     connectToServer(*pServerAddress);
```

```
   state = 1;
   i++;
  }
 delay(5000); // Delay a second between loops.
}
```

D.3 ESP-Now Code (Star)

Central Hub

//central hub for ESPNOW

```
#include <esp_now.h>
#include <WiFi.h>
#include "time.h"

// Global copy of slave
#define NUMSLAVES 6
esp_now_peer_info_t slaves[NUMSLAVES] = {};
int SlaveCnt = 0;
uint8_t new_mac[8] = {0x30, 0xAE, 0xA4, 0x11, 0x11, 0x11};//mac address of hub
 uint8_t data_received ;
 uint8_t  dataID ;
 uint8_t data1;
 uint8_t data2;
#define CHANNEL 1//channel
#define PRINTSCANRESULTS 0
int espnow_hub = 0;
int state_trans = 0;
volatile uint32_t rec_start = 0, rec_end = 0;//transmission time when data received from node
volatile uint32_t trans_start = 0, trans_end = 0;//transmission time when data is sent to node
volatile long difcounter1, difcounter2;
volatile long difCounter[20] = {0};
volatile long throughput[20] = {0};
volatile long total_difCounter = 0, total_throughput = 0;
int th = 0;
int trans_state = 0;//state to keep track in star connection
// Init ESP Now to connect
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  }
  else {
    Serial.println("ESPNow Init Failed");
    //  Restart if connection failed
    ESP.restart();
  }
}
// Scan for slaves in AP mode
void ScanForSlave() {
  int8_t scanResults = WiFi.scanNetworks();
  //reset slaves
  memset(slaves, 0, sizeof(slaves));
  SlaveCnt = 0;
  Serial.println("");
  if (scanResults == 0) {
    Serial.println("No WiFi devices in AP Mode found");
  } else {
```

```
    Serial.print("Found "); Serial.print(scanResults); Serial.println(" devices ");
    for (int i = 0; i < scanResults; ++i) {
      // Print SSID and RSSI for each device found
      String SSID = WiFi.SSID(i);
      int32_t RSSI = WiFi.RSSI(i);
      String BSSIDstr = WiFi.BSSIDstr(i);
      if (PRINTSCANRESULTS) {
        Serial.print(i + 1); Serial.print(": "); Serial.print(SSID); Serial.print(" ["); Serial.print(BSSIDstr);
Serial.print("]"); Serial.print(" ("); Serial.print(RSSI); Serial.print(")"); Serial.println("");
      }
      delay(10);
      // Check if the current device starts with `Slave`
      if (SSID.indexOf("Slave") == 0) {
        // SSID of interest
        Serial.print(i + 1); Serial.print(": "); Serial.print(SSID); Serial.print(" ["); Serial.print(BSSIDstr);
Serial.print("]"); Serial.print(" ("); Serial.print(RSSI); Serial.print(")"); Serial.println("");
        // Get BSSID => Mac Address of the Slave
        int mac[6];
        if ( 6 == sscanf(BSSIDstr.c_str(), "%x:%x:%x:%x:%x:%x%c",  &mac[0], &mac[1], &mac[2], &mac[3],
&mac[4], &mac[5] ) ) {
          for (int ii = 0; ii < 6; ++ii ) {
            slaves[SlaveCnt].peer_addr[ii] = (uint8_t) mac[ii];
          }
        }
        slaves[SlaveCnt].channel = CHANNEL; // pick a channel
        slaves[SlaveCnt].encrypt = 0; // no encryption
        SlaveCnt++;
      }
    }
  }

  if (SlaveCnt > 0) {
    Serial.print(SlaveCnt); Serial.println(" Slave(s) found, processing..");
  } else {
    Serial.println("No Slave Found, trying again.");
  }

  // clean up ram
  WiFi.scanDelete();
}
// Check if the slave is already paired with the master.
// If not, pair the slave with master
void manageSlave() {
  if (SlaveCnt > 0) {
//    Serial.println(SlaveCnt);
    for (int i = 0; i < SlaveCnt; i++) {
      const esp_now_peer_info_t *peer = &slaves[i];
      const uint8_t *peer_addr = slaves[i].peer_addr;
      Serial.print("Processing: ");
      for (int ii = 0; ii < 6; ++ii ) {
        Serial.print((uint8_t) slaves[i].peer_addr[ii], HEX);
        if (ii != 5) Serial.print(":");
      }
      Serial.print(" Status: ");
      // check if the peer exists
```

```
      bool exists = esp_now_is_peer_exist(peer_addr);
      if (exists) {
        // Slave already paired.
        Serial.println("Already Paired");
      } else {
        // Slave not paired, attempt pair
        esp_err_t addStatus = esp_now_add_peer(peer);
        if (addStatus == ESP_OK) {
          // Pair success
          Serial.println("Pair success");
        } else if (addStatus == ESP_ERR_ESPNOW_NOT_INIT) {
          // How did we get so far!!
          Serial.println("ESPNOW Not Init");
        } else if (addStatus == ESP_ERR_ESPNOW_ARG) {
          Serial.println("Add Peer - Invalid Argument");
        } else if (addStatus == ESP_ERR_ESPNOW_FULL) {
          Serial.println("Peer list full");
        } else if (addStatus == ESP_ERR_ESPNOW_NO_MEM) {
          Serial.println("Out of memory");
        } else if (addStatus == ESP_ERR_ESPNOW_EXIST) {
          Serial.println("Peer Exists");
        } else {
          Serial.println("Not sure what happened");
        }
        delay(100);
      }
    }
  } else {
    // No slave found to process
    Serial.println("No Slave found to process");
  }
}
// send data
void sendData() {

  const uint8_t *peer_addr;
   if ( dataID == 1)
   {
    SlaveCnt = 0;
      const esp_now_peer_info_t *peer = &slaves[SlaveCnt];
    const uint8_t *peer_addr = slaves[SlaveCnt].peer_addr;
      esp_err_t result = esp_now_send(peer_addr, &data1, sizeof(data1));//transmitting data1 to node 1 from central
hub
      esp_err_t result1 = esp_now_send(peer_addr, &data2, sizeof(data2));//transmitting data1 to node 1 from central
hub
        Serial.print("Send Status: ");
    if (result == ESP_OK && result1 == ESP_OK) {
      Serial.println("Success");
//      state_trans = 0;
    } else if (result == ESP_ERR_ESPNOW_NOT_INIT && result1 == ESP_ERR_ESPNOW_NOT_INIT ) {
      // How did we get so far!!
      Serial.println("ESPNOW not Init.");
    } else if (result == ESP_ERR_ESPNOW_ARG && result1 == ESP_ERR_ESPNOW_ARG) {
      Serial.println("Invalid Argument");
    } else if (result == ESP_ERR_ESPNOW_INTERNAL && result1 == ESP_ERR_ESPNOW_INTERNAL) {
```

```
      Serial.println("Internal Error");
    } else if (result == ESP_ERR_ESPNOW_NO_MEM && result1 == ESP_ERR_ESPNOW_NO_MEM) {
      Serial.println("ESP_ERR_ESPNOW_NO_MEM");
    } else if (result == ESP_ERR_ESPNOW_NOT_FOUND && result1 == ESP_ERR_ESPNOW_NOT_FOUND) {
      Serial.println("Peer not found.");
    } else {
      Serial.println("Not sure what happened");
    }
    delay(100);
   }
   else if ( dataID == 2)
    {
      SlaveCnt = 1;
        const esp_now_peer_info_t *peer = &slaves[SlaveCnt];
      const uint8_t *peer_addr = slaves[SlaveCnt].peer_addr;
        esp_err_t result = esp_now_send(peer_addr, &data1, sizeof(data1));//transmitting data1 to node 2 from central
hub
      esp_err_t result1 = esp_now_send(peer_addr, &data2, sizeof(data2));//transmitting data1 to node 2 from central
hub
         Serial.print("Send Status: ");
   if (result == ESP_OK && result1 == ESP_OK) {
      Serial.println("Success");
//      state_trans = 0;
    } else if (result == ESP_ERR_ESPNOW_NOT_INIT && result1 == ESP_ERR_ESPNOW_NOT_INIT ) {
      // How did we get so far!!
      Serial.println("ESPNOW not Init.");
    } else if (result == ESP_ERR_ESPNOW_ARG && result1 == ESP_ERR_ESPNOW_ARG) {
      Serial.println("Invalid Argument");
    } else if (result == ESP_ERR_ESPNOW_INTERNAL && result1 == ESP_ERR_ESPNOW_INTERNAL) {
      Serial.println("Internal Error");
    } else if (result == ESP_ERR_ESPNOW_NO_MEM && result1 == ESP_ERR_ESPNOW_NO_MEM) {
      Serial.println("ESP_ERR_ESPNOW_NO_MEM");
    } else if (result == ESP_ERR_ESPNOW_NOT_FOUND && result1 == ESP_ERR_ESPNOW_NOT_FOUND) {
      Serial.println("Peer not found.");
    } else {
      Serial.println("Not sure what happened");
    }
    delay(100);
    }
  }
// callback when data is sent from Master to Slave
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
   if (trans_state == 0)
   {
    trans_start = micros();//first data is sent to end node from hub
    trans_state = 1;
   }
   else if (trans_state == 1)
   {
    trans_end = micros();//second data is sent to end node from hub
    trans_state = 0;
   }
```

```
}
void setup() {
  Serial.begin(115200);
  //Set device in STA mode to begin with
  esp_base_mac_addr_set(new_mac);//set MAC address of hub or server.
  WiFi.mode(WIFI_STA);
  Serial.println("ESPNow/Multi-Slave/Master Example");
  // This is the mac address of the Master in Station Mode
  Serial.print("STA MAC: "); Serial.println(WiFi.macAddress());
  // Init ESPNow with a fallback logic
  InitESPNow();
   ScanForSlave();
   manageSlave();
  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
   esp_now_register_recv_cb(OnDataRecv);
//  esp_now_register_send_cb(OnDataSent);
}
void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  char macStr[18], macStr1[18];

       snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
              mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);


         data_received = *data;
       state_trans++;

       if (state_trans == 1)
       {
        dataID = data_received ;//received data ID

       }
       else if (state_trans == 2)
       {
        rec_start = micros();//arrival time of first user data
        data1 = data_received ;//received first user data
        Serial.print("data1: ");
        Serial.println(data1);
        Serial.print("rec_start:  ");
        Serial.println(rec_start);
       }
        else if (state_trans == 3)
       {
        rec_end = micros();//arrival time of second user data
        data2 = data_received ;//received second user data
        Serial.print("data2: ");
        Serial.println(data2);
        Serial.print("rec_end:  ");
        Serial.println(rec_end);
       }

}
void loop() {
  if (espnow_hub == 0)//central hub is in receiving state
```

```
 {
  esp_now_register_recv_cb(OnDataRecv);//register receiving callback function to receive the data
  if (state_trans == 3)//all data are successfully received
  {
   espnow_hub = 1;//go in transmitting state
  }
 }
 else if (espnow_hub == 1)
 {
   esp_now_register_send_cb(OnDataSent);//register sending callback function to transmit the data
   sendData();//send data to node
   espnow_hub = 2;
   state_trans = 0;
 }
 else if (espnow_hub == 2)
 {
   difcounter1 = rec_end - rec_start;
   difcounter2 = trans_end - trans_start;
   Serial.print("difcounter1: ");
   Serial.println(difcounter1);
   Serial.print("difcounter2: ");
   Serial.println(difcounter2);
   difCounter[th] = difcounter1 + difcounter2;
   throughput[th] = 48000000/difCounter[th];
     th++;
   espnow_hub = 0;//go back to receiving state
 }

  delay(1000);
}
```

Peripheral Node

Code for only one node is given; same code can be used for other nodes.

```
//ESPNOW node (slave)
#include <esp_now.h>
#include <WiFi.h>
#define CHANNEL 1//defining channel
uint8_t data1, data2;
uint8_t rec_data1, rec_data2;
//char macStrr[18];
uint8_t mac_master[8] = {0x30, 0xAE, 0xA4, 0x11, 0x11, 0x11};//mac address of server
int st ;
esp_now_peer_info_t slave;
int state_node = 0;//state to track star network
int state_rec = 0;//state to track receiving of data from hub
// Init ESP Now with fallback
```

```
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  }
  else {
    Serial.println("ESPNow Init Failed");
     ESP.restart();
  }
  memset(&slave, 0, sizeof(slave));
  for (int i = 0; i < 6; ++i)
    slave.peer_addr[i] = (uint8_t)mac_master[i];
  slave.channel = CHANNEL; // pick a channel
  slave.encrypt = 0; // no encryption
  const esp_now_peer_info_t *peer = &slave;
  const uint8_t *peer_addr = slave.peer_addr;
  esp_err_t addStatus = esp_now_add_peer(peer);
}
// config AP SSID
void configDeviceAP() {
  String Prefix = "Slave:";
  String Mac = WiFi.macAddress();
  String SSID = Prefix + Mac;
  String Password = "123456789";
  bool result = WiFi.softAP(SSID.c_str(), Password.c_str(), CHANNEL, 0);
  if (!result) {
    Serial.println("AP Config failed.");
  } else {
    Serial.println("AP Config Success. Broadcasting with AP: " + String(SSID));
  }
}
void sendData()
{
  if (state_node  == 0)//send device ID
  {
    state_node  = 1;
    Serial.println("state_node");
```

```
  Serial.println(state_node);
    const uint8_t *peer_addr = slave.peer_addr;
esp_err_t result = esp_now_send(peer_addr, &data1, sizeof(data1));
Serial.print("Send Status: ");
if (result == ESP_OK)
{
  Serial.println("Success:");
 } else if (result == ESP_ERR_ESPNOW_NOT_INIT) {
    // How did we get so far!!
    Serial.println("ESPNOW not Init.");
  } else if (result == ESP_ERR_ESPNOW_ARG) {
    Serial.println("Invalid Argument");
  } else if (result == ESP_ERR_ESPNOW_INTERNAL) {
    Serial.println("Internal Error");
  } else if (result == ESP_ERR_ESPNOW_NO_MEM) {
    Serial.println("ESP_ERR_ESPNOW_NO_MEM");
  } else if (result == ESP_ERR_ESPNOW_NOT_FOUND) {
    Serial.println("Peer not found.");
  } else {
    Serial.println("Not sure what happened");
  }
  delay(100);
}
else if (state_node  = 1)//send the user data
{
  state_node  = 0;
  Serial.println("state_node");
  Serial.println(state_node);
  data1 = 0;//data1
  data2 = 1;//data2
   const uint8_t *peer_addr = slave.peer_addr;
esp_err_t result = esp_now_send(peer_addr, &data1, sizeof(data1));
esp_err_t result1 =esp_now_send(peer_addr, &data2, sizeof(data2));
Serial.print("Send Status: ");
if (result == ESP_OK && result1 == ESP_OK)
{
  Serial.println("Success:");
```

```
   } else if (result == ESP_ERR_ESPNOW_NOT_INIT && result1 == ESP_ERR_ESPNOW_NOT_INIT) {
     // How did we get so far!!
     Serial.println("ESPNOW not Init.");
   } else if (result == ESP_ERR_ESPNOW_ARG && result1 == ESP_ERR_ESPNOW_ARG) {
     Serial.println("Invalid Argument");
   } else if (result == ESP_ERR_ESPNOW_INTERNAL && result1 == ESP_ERR_ESPNOW_INTERNAL) {
     Serial.println("Internal Error");
   } else if (result == ESP_ERR_ESPNOW_NO_MEM && result1 == ESP_ERR_ESPNOW_NO_MEM) {
     Serial.println("ESP_ERR_ESPNOW_NO_MEM");
   } else if (result == ESP_ERR_ESPNOW_NOT_FOUND && result1 == ESP_ERR_ESPNOW_NOT_FOUND  )
{
     Serial.println("Peer not found.");
   } else {
     Serial.println("Not sure what happened");
   }
   delay(100);
  }

}
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.print("Last Packet Sent to: "); Serial.println(macStr);
  Serial.print("Last Packet Send Status: "); Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery
Success" : "Delivery Fail");
}

void setup() {
  Serial.begin(115200);
  Serial.println("ESPNow Star Connection Node 1");
  //Set device in AP mode to begin with
  WiFi.mode(WIFI_AP_STA);
  // configure device AP mode
  configDeviceAP();
  // This is the mac address of the Slave in AP Mode
  Serial.print("AP MAC: "); Serial.println(WiFi.softAPmacAddress());
```

```
  // Init ESPNow
  InitESPNow();
}

// callback when data is recv from Master
void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {

  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
if (state_rec == 0)
   {
    rec_data1 = *data;//received data 1
    Serial.print("Last Packet Recv Data: "); Serial.println(rec_data1);
    state_rec = 1;
   }
   else if (state_rec == 1)
   {
    rec_data2 = *data;//received data 2
    Serial.print("Last Packet Recv Data: "); Serial.println(rec_data2);
    state_rec = 0;
   }
  Serial.print("Last Packet Recv from: "); Serial.println(macStr);
  Serial.println("");
}

void loop() {
  if (state_node == 0)
  {
   esp_now_register_recv_cb(OnDataRecv); //register receiving call back function to receive data
    if (Serial.available () > 0)//data arrival from serial monitor
    {
      data1 =Serial.read() - '0';
      esp_now_register_send_cb(OnDataSent);//register send  call back function to send data (device ID)
        sendData();
     }
  }
```

```
  else if (state_node == 1)
  {
   esp_now_register_send_cb(OnDataSent);//register send  call back function to send data (user data)
     sendData();
     state_node = 0;//go back to initial state
  }
 delay(5000);
}
```

REFERENCES

[1] T. Simplelink and M. S. P. E. Launchpad, “Development Kit (MSP - EXP432P401R),” no. March 2015, 2018.

[2] Texas Instruments, “MSP432P401R, MSP432P401M MSP432P401R, MSP432P401M SimpleLink™ Mixed-Signal Microcontrollers 1 Device Overview,” *MSP432P401R, MSP432P401M datasheet*, 2017.

[3] Nordic Semiconductor, “nRF24L01+ Datasheet,” *Nord. Semicond.*, no. March, p. 75, 2008.

[4] “How nRF24L01+ wireless module works & Interface with Arduino,” *Arduino Projects*, 2016. [Online]. Available: https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/.

[5] E. Systems, “ESP32-WROOM-32,” 2018.

[6] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, and D. Formica, “Performance evaluation of Bluetooth low energy: a systematic review,” *Sensors*, vol. 17, no. 12, p. 2898, 2017.

[7] Belbin, “"Frequently Asked Questions,” pp. 1–2, 2012.

[8] Bluetooth, “Specification of the Bluetooth System Wireless connections made easy-v4.0,” vol. 0, no. June, 2010.

[9] M. O. Al Kalaa and H. H. Refai, “Selection probability of data channels in Bluetooth Low Energy,” in *Wireless Communications and Mobile Computing Conference (IWCMC), 2015 International*, 2015, pp. 148–152.

[10] P. Di Marco, P. Skillermark, A. Larmo, P. Arvidson, and R. Chirikov, “Performance

Evaluation of the Data Transfer Modes in Bluetooth 5," *IEEE Commun. Stand. Mag.*, vol. 1, no. 2, pp. 92–97, 2017.

[11] "ESP32 Bluetooth Architecture," 2018.

[12] Rohde & Schwarz, "Bluetooth Adaptive Frequency Hopping on a R&S CMW," *Appl. Note 11.2016_1C108_0e*, 2016.

[13] MikroElektronika, "Bluetooth Low Energy - Part 1: Introduction To BLE," 2016. [Online]. Available: www.mikroe.com/blog/bluetooth-low-energy-part-1-introduction-ble. [Accessed: 01-Dec-2018].

[14] Cypress, "PSoC ® Creator™ Component Datasheet Generic Attribute Profile (GATT) Features □ GATT Client and Server General Description SIG adopted Profiles and Services Comprehensive APIs," pp. 408–943, 2015.

[15] "Your Go-To-Guide for Channel & Transmit Powe on Wi-Fi Networks," *engeniustech*, 2015. [Online]. Available: https://www.engeniustech.com/go-guide-channel-transmit-power-wi-fi-networks-2/. [Accessed: 01-Dec-2018].

[16] "ESP-NOW User Guide," 2016.

[17] E. G. Villegas, E. Lopez-Aguilera, R. Vidal, and J. Paradells, "Effect of adjacent-channel interference in IEEE 802.11 WLANs," in *Cognitive Radio Oriented Wireless Networks and Communications, 2007. CrownCom 2007. 2nd International Conference on*, 2007, pp. 118–125.

[18] I. S. Association, "802.11-2012-IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specific," *Retrived from http//standards. ieee.*

org/about/get/802/802.11. html, 2012.

[19] B. Mitchell, “How many Devices Can Connect to One Wireless Router?” [Online]. Available: https://www.lifewire.com/how-many-devices-can-share-a-wifi-network-818298.

[20] C. C. Studio, “Code Composer - Getting Started Guide,” *October*, no. October, 2006.

[21] B. Finch and W. Goh MSP, “Application Report MSP430™ Advanced Power Optimizations: ULP Advisor™ Software and EnergyTrace™ Technology,” no. June, pp. 1–29, 2014.

[22] S. Chakkor, E. A. Cheikh, M. Baghouri, and A. Hajraoui, “Comparative performance analysis of wireless communication protocols for intelligent sensors and their applications,” *arXiv Prepr. arXiv1409.6884*, 2014.

[23] H. Saha, S. Mandal, S. Mitra, S. Banerjee, and U. Saha, “Comparative Performance Analysis between nRF24L01+ and XBEE ZB Module Based Wireless Ad-hoc Networks,” *Int. J. Comput. Netw. Inf. Secur.*, vol. 9, no. 7, p. 36, 2017.

[24] J.-S. Lee, Y.-W. Su, and C.-C. Shen, “A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi,” in *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, 2007, pp. 46–51.

[25] M. T. Arefin, M. H. Ali, and A. K. M. F. Haque, “A Comparative Analysis of Short Range Wireless Protocols For Wireless Sensor Network.”

[26] Z. Chen, C. Hu, J. Liao, and S. Liu, “Protocol architecture for wireless body area network based on nRF24L01,” *Proc. IEEE Int. Conf. Autom. Logist. ICAL 2008*, no. September, pp. 3050–3054, 2008.

[27] S. S. Sonavane and B. P. Patil, “Experimentation for Packet Loss on MSP430 and

nRF24L01 Based Wireless Sensor Network," vol. 01, no. 01, pp. 25–29, 2009.

[28] Y. Wang, C. Hu, Z. Feng, and Y. Ren, "Wireless transmission module comparison," *2014 IEEE Int. Conf. Inf. Autom. ICIA 2014*, no. July, pp. 902–907, 2014.

[29] A. Maier, A. Sharp, and Y. Vagapov, "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things," *2017 Internet Technol. Appl. ITA 2017 - Proc. 7th Int. Conf.*, pp. 143–148, 2017.

[30] Espressif Systems, "Read the Docs Template Documentation," 2017.

[31] N. Zhu and I. O'Connor, "Energy Performance of High Data Rate and Low Power Transceiver based Wireless Body Area Networks.," in *SENSORNETS*, 2013, pp. 141–144.

[32] N. Kolban, "ESP32 BLE Arduino Library," 2017. [Online]. Available: https://github.com/nkolban/ESP32_BLE_Arduino.