DATA SHARING USING MQTT AND ZIGBEE-BASED DDS ON RESOURCE-CONSTRAINED CONTIKI-BASED DEVICES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TUNAHAN YILDIRIM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2020

Approval of the thesis:

DATA SHARING USING MQTT AND ZIGBEE-BASED DDS ON RESOURCE-CONSTRAINED CONTIKI-BASED DEVICES

submitted by **TUNAHAN YILDIRIM** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Mehmet Halit S. Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Mehmet Halit S. Oğuztüzün
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Ahmet Coşar
Computer Engineering, THK University _____

Prof. Dr. Mehmet Halit S. Oğuztüzün
Computer Engineering, METU _____

Assoc. Prof. Dr. Ertan Onur
Computer Engineering, METU _____

Date:10.01.2020

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Tunahan Yıldırım

Signature :

ABSTRACT

DATA SHARING USING MQTT AND ZIGBEE-BASED DDS ON RESOURCE-CONSTRAINED CONTIKI-BASED DEVICES

Yıldırım, Tunahan
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Mehmet Halit S. Oğuztüzün

January 2020, 107 pages

This thesis describes the implementation of data sharing among resource-constrained IoT devices using two different middleware: MQTT(Message Queuing Telemetry Transport) and DDS (Data Distribution Services) for Real-Time Systems. In our setting, all IoT devices run under the Contiki operating system. In the configuration for DDS, a pair of Texas Instruments' MSP430 processor-based ZigBee powered Advanticsys XM1000 device is used to realize data sharing between wireless sensor network devices without a server node. In order to install and run the proposed application, Ubuntu 64-bit operating system is used since it provides all the dependencies for Contiki and wireless sensor nodes. In the DDS middleware, an example interface definition is given in the XML file which contains the temperature, humidity, light values and the data to be shared is determined by the interface definition. For sharing data, DDS components (DataWriter, DataReader, Publisher, Subscriber, Topics) are used. Two XM1000 wireless sensor nodes are used for sharing sensor data with each other every 10 seconds. These experi-

ments are performed between 2 and more devices using the simulator and in different periods. In the second part of the work, the data structure that contains temperature, humidity and light values is shared via both MQTT and MQTT-SN (MQTT for Sensor Networks) middleware. The configuration of the MQTT middleware application works with the TCP protocol, the following components are used: The Ubuntu-based host computer, the mosquitto (Local MQTT server) application running on the host computer, MSP430 processor-based ZigBee powered wireless sensor node (CM5000 produced by Advanticsys company). In addition, the Python client applications for both publisher and subscriber are run on the host computer. These components above are used for sharing the sensor data structure between wireless sensor nodes which are connected to the host computer. Publisher and subscriber applications share the sensor data structure in every 10 seconds over the mosquitto server and received data is sent to the wireless sensor node by using python subscriber application. The received sensor data structure is sent to wireless sensor node by using the serial interface. In this implementation, ZigBee radio is not used because the ZigBee module could not be activated due to the large memory footprint of the MQTT application. Since the Zigbee module cannot be used in MQTT application, it is not included in the comparison. In order to run and share the data structure that contains temperature, humidity and light values by using the MQTT-SN middleware, RSMB (Really Small Mosquitto Broker) is installed and run on a Linux based host computer. The devices must obtain the IP address from the host computer to communicate with the Mosquitto server running on the host computer. Therefore, the border-router application used to obtain an IP address on the computer is running on wireless sensor node named CM5000 . Another CM5000 wireless sensor node gets the IP address by using border-router application installed CM5000 wireless sensor node and can write sensor data on RSMB by using MQTT-SN middleware installed on the CM5000 wireless sensor node. The sensor data structure is shared in different period of times(5,10,20,30,60 seconds) and 2 and more sensor devices. In addition, sensor data sent from CM5000 wireless sensor node is shared with the RSMB server running on the host computer. The shared sensor data can

be viewed on the host computer by using the Mosquitto client which reads data from RSMB. These three configurations are compared in terms of memory usage, power consumption, and bandwidth utilization. With respect to memory usage, DDS middleware software is the least memory consuming software. In addition, the effects of the changes made to fit the image of the operating system on the memory usage are calculated. Similarly, it was seen that the DDS application consumed the least power among all the applications. For MQTT-SN, power consumption could not be calculated, due to power estimation module which could not be activated because of the lack of memory.With the application developed using calculated raw values, the calculation is made for MQTT-SN and the energest module of Contiki is used for power estimation.

Keywords: Dds, Mqtt, Contiki,Interoperable,Memory Footprint

ÖZ

KAYNAK KISITLARI BULUNAN CONTİKİ TABANLI CİHAZLARDA MQTT VE ZİGBEE TABANLI DDS KULLANARAK VERİ PAYLAŞIMI

Yıldırım, Tunahan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Mehmet Halit S. Oğuztüzün

Ocak 2020 , 107 sayfa

Bu tez, iki farklı katman yazılımı kullanan kaynak kısıtlı IoT aygıtları arasında veri paylaşımının uygulanmasını açıklamaktadır: Gerçek Zamanlı Sistemler için MQTT (Message Queuing Telemetry Transport) ve DDS (Veri Dağıtım Hizmetleri). Konfigürasyonumuzda, tüm IoT cihazları Contiki işletim sistemi altında çalışmaktadır. DDS yapılandırmasında, bir sunucu düğümü olmadan kablosuz sensör ağ aygıtları arasında veri paylaşımını gerçekleştirmek için bir çift Texas Instruments'ın MSP430 işlemci tabanlı ZigBee destekli Advanticsys XM1000 cihazı kullanılmıştır. Uygulamanın kurulumu ve çalıştırılması için, Contiki ve kablosuz sensör düğümleri için tüm bağımlılıkları sağladığı için Ubuntu 64-bit işletim sistemi kullanılır. DDS arakatman yazılımında, sıcaklık, nem, ışık değerleri içeren XML dosyasında örnek bir arayüz tanımı verilmiştir ve paylaşılacak veriler bu arayüz tanımı tarafından belirlenir. Veri paylaşımı için, DDS bileşenleri (DataWriter, DataReader, Publisher, Subscriber, Topic) kullanılır. İki XM1000 kablosuz sensör düğümü, sensör verilerini her 10 saniyede bir birbirleriyle paylaşmak için kullanılır.

Bu testler, simülatör kullanarak 2 ve daha fazla cihaz arasında ve farklı periyotlarda gerçekleştirilir. Çalışmanın ikinci bölümünde sıcaklık, nem ve ışık değerleri içeren veri yapısı hem MQTT hem de MQTT-SN (Sensör Ağları için MQTT) ara katman yazılımı ile paylaşılıyor. MQTT ara yazılımı uygulamasının yapılandırması TCP protokolüyle çalışır ve aşağıdaki bileşenler kullanılır: Ubuntu tabanlı ana bilgisayar, ana bilgisayarda çalışan mosquitto (Yerel MQTT sunucusu) uygulaması, MSP430 işlemci tabanlı ZigBee destekli kablosuz sensör düğümü (MSP430 işlemci tabanlı) Advanticsys firması tarafından üretilen CM5000). Ayrıca, hem yayıncı hem de dinleyici için Python istemci uygulamaları ana bilgisayarda çalıştırılır. Yukarıdaki bu bileşenler, sensör veri yapısını ana bilgisayara bağlı olan kablosuz sensör düğümleri arasında paylaşmak için kullanılır. Yayıncı ve abone uygulamaları, sensör veri yapısını mosquitto sunucusu üzerinden her 10 saniyede bir paylaşır ve alınan veriler python abone uygulaması kullanılarak kablosuz sensör düğümüne gönderilir. Alınan sensör veri yapısı da seri arayüz kullanılarak kablosuz sensör düğümüne gönderilir. Bu uygulamada, ZigBee modülü, MQTT uygulamasının yüksek bellek ihtiyacınedeniyle ZigBee modülü etkinleştirilemediğinden kullanılamaz.MQTT uygulamasında Zigbee modulü kullanılamadığından dolayı karşılaştırılmaya dahil edilmemiştir. MQTT-SN ara katman yazılımını kullanarak sıcaklık, nem ve ışık değerleri içeren veri yapısını çalıştırmak ve paylaşmak için RSMB (Really Small Mosquitto Broker) Linux tabanlı bir ana bilgisayara kurulup çalıştırılır. Cihazlar, ana bilgisayarda çalışan Mosquitto sunucusuyla iletişim kurmak için ana bilgisayardan IP adresini almalıdır. Bu nedenle, bilgisayarda IP adresi almak için kullanılan yönlendirici uygulaması CM5000 adlı kablosuz algılayıcı düğümde çalıştırılır. Başka bir CM5000 kablosuz sensör düğümü, CM5000 kablosuz sensör düğümünü yükleyen yönlendirici uygulamasını kullanarak IP adresini alır ve CM5000 kablosuz sensör düğümüne yüklenen MQTT-SN ara yazılımı kullanarak RSMB üzerindeki sensör verilerini yazabilir. Sensör veri yapısı farkı sürelerde 2 veya daha fazla sensör düğümü arasında paylaşılır. Ayrıca, CM5000 kablosuz sensör düğümünden gönderilen sensör verileri, ana bilgisayarda çalışan RSMB sunucusuyla paylaşılır. Paylaşılan sensör verileri, ana bilgisayarda RSMB'den veri okuyan Mosquitto istemcisi kullanılarak görün-

tülenebilir. Bu üç yapılandırma hafıza kullanımı, güç tüketimi ve bant genişliği kullanımı açısından karşılaştırılmıştır. Bellek kullanımıyla ilgili olarak, DDS arakatman yazılımı en az bellek tüketen yazılımdır. Ek olarak, işletim sisteminin görüntüsünün cihazlar üzerine yüklebilmesi için yapılan değişikliklerin bellek kullanımı üzerindeki etkileri hesaplanmaktadır. Benzer şekilde, DDS uygulamasının tüm uygulamalar arasında en az gücü tükettiği görülmüştür. MQTT-SN için bellek yetersizliği nedeniyle etkinleştirilemeyen güç tahmin modülü nedeniyle güç tüketimi hesaplanamaz. Bu nedenle bu modül tarafından hesaplanan ham değerler kullanılarak geliştirilen uygulama ile MQTT-SN için de hesaplamalar yapılır. Contikinin "Energest modülü" güç tüketimi tahmini için kullanılır.

Anahtar Kelimeler: Dds, Mqtt, Contiki, Birlikte Çalışabilirlik,Hafıza Kullanımı

To my family..

ACKNOWLEDGMENTS

I would like to express my inmost gratitude to my supervisor Prof. Dr.Halit Oğuztüzün for his patience, vision and understanding throughout this thesis. I am also indebted to Assoc.Prof Ertan Onur for his knowledge, ideas and motivation. I also would like to thank my parents, sister and my aunt Sevgi Yıldırım for their love and support.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
DDS-XRCE	DDS for Extremely Resource Constrained Environments
DDS	Data Distribution Service
IOT	Internet of Things
IP	Internet Protocol Address
LPM	Low Power Mode
M2M	Machine to Machine
MQTT-SN	MQTT For Sensor Network
MQTT	Message Queuing Telemetry Transport
OMG	Object Management Group
QOS	Quality of Service
RAM	Random Access Memory
RIME	Radio Interface Module
ROM	Read Only Memory
RSMB	Really Small Mosquitto Broker
RTI	Real Time Infrastructure
RTPS	Real Time Publish Subscribe
RX	Receive
SDDS	Sensor Data Distribition Service
SNPS	Sensor Network Publish
TCP	Transmission Control Protocol

TX	Transmit
UDP	User Datagram Protocol
USB	Universal Serial Bus
XML	Extensible Markup Language Subscribe

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Today applications are developed by using various software languages and can be run on many different systems. In many usage scenarios, communication between devices and systems is required. The disintegration of systems led to the development of the concept of discrete systems. The main purpose of this thesis is to run the DDS(Data Distribution Service) protocol on systems that use less memory and fixed memory space. DDS can be defined as a middleware standard for interoperable, data-centric publish/subscribe architectures with real-time capabilities [7]. DDS normally provides secure data transfer over ethernet. In this thesis, it is aimed to update the DDS protocol for tailoring data and convert the DDS middleware to a gateway that can be run on ZigBee by using a sensor node that uses Contiki operating system. Studies are developing data distribution service as middleware according to the Object Management Group (OMG) DDS standard. The OMG Data-Distribution Service for Real-Time Systems (DDS) is the first open international middleware standard directly addressing publish-subscribe communications for real-time and embedded systems [7]. The DDS differs from client-server applications because of several features. The most significant difference between DDS and client-server applications is the requirement of the usage of the centralized server which receiving data from clients and the server replies every request with a response message. Client-server architecture is generally used for web applications. On the contrary, a previously defined data structure, Interface Definition Language (IDL), is used for the

client-server architecture. Common data structures are created according to the definitions in the IDL file format. Data exchange can be successfully performed when these structures are compatible with each other. The data is written via a publisher and the data is read via a subscriber. By making adjustments about parameters of Quality of Service (QoS), DDS events can be customized to identify problems about DDS clients and data sharing between clients. The general concept is that each application has a data writer to send data that is specified in IDL. Also, the subscriber component uses the DDS component, DataReader, to read data from other nodes according to the DDS topic. When data is shared between nodes by using DDS middleware, data transfer takes place only through logical structures such as topics and QoS parameters. Incompatible logical structures are ignored by DDS. Also, each device communicates over a certain domain number which helps to isolate the domain from other domains to avoid miscommunication. In this thesis, devices have ZigBee modules to convert and modify Real Time Publish Subscribe (RTPS) packages to smaller packages which are called Sensor Network Publish-Subscribe (SNPS) packet [8]. In this thesis, the SNPS packet format is modified to be able to use DDS middleware on ZigBee devices. The aim of the usage of the modified SNPS packet format is data communication with the radio interface of Contiki devices, packages, and network layer which is changed to send as a ZigBee packet. The DDS data packets are exchanged with nodes through a radio interface to achieve a successful data transfer. To find and define the differences (memory footprint, advantages, use cases), the performance results of this thesis are compared with Message Queuing Telemetry Transport (MQTT) which is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. MQTT is designed as an extremely lightweight publish/subscribe messaging transport. The MQTT is useful for connections with remote locations where a small code footprint is required and network bandwidth is effectively used [9].

When resource-constrained operating systems are compared, it appears that Tiny-OS is the least resource-consuming operating system. However, it is predicted that the Contiki operating system would be suitable for applica-

tions developed throughout the thesis. Tiny OS cannot be multitasked, insufficient in terms of flexibility and preemptive multi-threading and doesn't support the pure C programming language [10].

Firstly, it is aimed to install tailored DDS middleware on sensor node models which have very low RAM and ROM resources by using the ZigBee model on sensor node instead of using ethernet. To achieve this goal, efforts are given to minimize memory footprints. To optimize the memory footprint of the device, it is possible to load on the device by changing the configuration data of the related platform (Contiki), adding optimization flags of the compiler and optimizing the memory on the Contiki operating system. The procedures will be discussed in Section 3.

Secondly, it is aimed to evaluate various data sharing schemes in resource-constrained environments. The steps are required to install and use this software will be explained in Section 3. A sample code compatible with the CC2538 model has been produced by Texas Instrument to use the MQTT via Contiki. This example could not be run on the existing devices because the CC2538 model has approximately four times more RAM and ROM resources. As a solution, the data is sent to python software by using the serial communication port. The middleware receives the incoming data from the MQTT server and sends the data constructed from the related topic data structure to the MQTT server. Meanwhile, a second client application which is implemented in python listens to the data and periodically share data. The data is transferred to the device via the serial communication port. Although the data reading event on Sky sensor node is available, the data is not received. To fix this problem, changes are made in the data reading mechanism.

Then, it is aimed to reduce the memory footprints by using the User Datagram Protocol (UDP) instead of the Transmission Control Protocol (TCP) for the MQTT packets. To achieve this, it is decided to use the MQTT For Sensor Network (MQTT-SN protocol) which is a derivative of MQTT. For the MQTT-SN experiments, the memory overflow problem is observed in the

memory footprint. To solve this problem, the operating system and compiler settings are changed and the memory problem is fixed. To ensure IP connectivity between devices, the aforementioned CM5000 device controller must work with webserver support by using the rpl-border-router which is one of the examples of IPv6 projects under Contiki applications. Otherwise, the IP connection can not be established. The XM1000 (sensor node) device had problems with getting the IP address, therefore XM1000 [2], is simulated with the Cooja [11] simulator by using the sky sensor node platform. Then, the sensor node sends to the other nodes by using the topic structure that has the same structure with the data which is sent over DDS. Details will be explained in Section 3.

1.2 Approach

Compatible low-power Telosb [12] nodes that have radio interface are examined to run the DDS and MQTT middleware on the Contiki operating system. In this thesis, the devices produced by Advanticsys are used. These devices are Advanticsys XM1000 [2] and CM5000 [3] nodes. The sample DDS application is installed on devices that have ZigBee radio chips. Both XM1000 and CM5000 are kind of Telosb sensor node that has temperature, humidity, light and battery sensors. These values which are received from sensors are collected and stored into a data structure that is sent by using data distribution service. These devices (XM1000 and CM5000) have very low RAM and ROM resource usage. Both devices have MSP430 based processors. The XM1000 [2] device has 114KB ROM and 8KB RAM, while the CM5000 [3] has 48KB ROM and contains 10KB of RAM. XM1000 uses the Zigbee adapter to send moisture, temperature and light status to the other devices. Because of the lack of available memory on RAM and ROM, the binary file for a device could not be loaded via bootstrap in the default setting. Also, overflow error messages are written on the bootstrap loader console.

To resolve the memory overflow obstacle, the changes on platform-conf under the platform folder related to the device. Platform files for the Con-

tiki operation system are downloaded via Texas Instruments(TI)'s website and downloaded files are moved to the Contiki operation system's platform folder. The changes related to macro is done by using this platform-conf file. When changes are done SNPS packet for ZigBee, firstly macros related to IPV6 set as 0 for disabling IPV6 support. The memory overflow problem still exists while creating binary images for tailored DDS, the latest version of MSP430 is not available on the Linux repositories. The latest version(4.6.3) on Ubuntu repositories couldn't support 20-bit registers. Because of this, a newer version is downloaded for github [13] and this memory-efficient version reduces the RAM and ROM size. Compiler flags that provide optimization to reduce RAM and ROM usage are added to the corresponding makefile. However, uploading the binary file to the device is not successful. Besides, a project-related configuration file shall be set by disabling or changing unnecessary components. For example, TSCH log level, TCP and UDP settings, values for routing tables and neighboring tables, process strings can be activated by using a platform configuration file, therefore memory space can be gained by optimizing or disabling Contiki macros. Optimization is applied for the byte values occupied by members held in the source code, initial values are removed in source code to reduce the RAM size. Both the MQTT part and the DDS part are tuned according to the methods above and the reference [14]. In the first part of the thesis, the DDS application is modified for ZigBee after adapting DDS for the radio interface. The second problem is found while trying to upload DDS software to the sky sensor node. Makefile is updated for building the DDS project for MSP430 CPU architecture. All dependencies for(sensor source file and DDS source files) are added to the makefile by overriding makefile templates, as a result of this the exceptions related to memory messages seen in the terminal window. These problems are fixed by applying the methodologies defined above. Application is uploaded to devices and data exchange operations successfully done. The second implementation is related to MQTT. Similar to the DDS sensor data application, the makefile was created to install the MQTT software on the sensor nodes, and the configuration settings were updated similar to methods mentioned in the part related to DDS software. However, despite

these changes, the binary file could not be loaded to the device because TCP and UDP could not be turned off in the MQTT software. In the first attempt of MQTT, the data obtained from the serial port was transmitted to the mosquitto server which is running on the Linux computer via the client software running on the same computer and the data exchange between the two sensor devices is made. In the third stage of the thesis, a data structure was created using MQTT-SN. infrastructure produced by IBM for sensor networks that can work with MQTT. MQTT-SN uses UDP and not TCP for its transport [15]. UDP is a connectionless protocol whereas TCP is connection orientated. Also, the arrangements on memory usage applied in other stages were made in this software. Updated memory sharing of the corresponding processor structure under MSP430 because of overflow on the RAM. These changes will be discussed later in the thesis. Thus, a certain portion of the empty ROM was assigned as ram and the corresponding software could be loaded onto the device. For the server, a python-based MQTT-SN compatible version of the server was installed and connected with MQTT was achieved. Contiki operating system uses a java-based Cooja simulator which has features for adding nodes by using binary file and these nodes will be added to the cooja's panel. Ip configuration for the sensor nodes is provided by the border-router application. The simulation is firstly run on the cooja simulator then run in real devices for testing.

1.3 Improvements

There are several improvements mentioned in the items defined above.
-In current implementation modified DDS software only supports on data available option, on liveliness lost, on sample lost and other necessary QoS features will be added if memory footprint is enough to enable these features
-The complete version of MQTT will be run on real devices using Contiki operating system instead of the client python program running on the computer.

-RTI and opensplice are announced the resource constraint version of DDS software. Data read from sensors will be sent M2M by using this lightweight DDS software versions and memory footprint will be compared if the binary file fits the sky sensor nodes.

CHAPTER 2

BACKGROUND INFORMATION AND RELATED WORK

2.1 Contiki

2.1.1 Operating System

Contiki is an open-source operating system for the Internet of Things [11]. Contiki, a lightweight operating system with support for dynamic loading and replacement of individual programs and services. Contiki is built around an event-driven kernel but provides optional preemptive multithreading that can be applied to individual processes [16]. The Contiki operating system requires very low system resources, therefore it has worked in harmony with most resource-constraint devices. Also, the operating system is compatible with most of the wireless sensor network devices. Even if the platform doesn't exist in the platform folder under the Contiki-OS source folder, the platform will be easily ported for a new device because of the flexibility of Contiki OS. The ported source code for the platform is moved under the platform folder. The code can be built for the newly added platform by changing the target device. Contiki is designed for low-cost, low-power microcontrollers also serve the cooja simulator to test an application without installing the real device. This simulator is configurable for devices with specific processor architecture, the developed code can be tested with the simulator without being installed on the device. On the instant Contiki virtual computer provided by Contiki, all programs required to install the operating system to the device are installed by default. Since it is open-source, it is possible to make flexible changes to the files in the operating sys-

tem. The Contiki operating system supports Ipv4 and Ipv6 and RIME(radio interface support). Devices using the Contiki operating system can easily get an IP address with the border-router project under the example folder of the operating system. Although the operating system has a very small memory footprint, it can run multiple operations at the same time. Libraries that allow data to be easily received from the sensors on the device are available in the operating system's source code. It is possible to optimize the memory footprint by setting values that will not be used or diminished through the configuration file of the device.

2.1.2 Energest Power Module

Another topic to be compared in the thesis is power consumption. For power consumption, Contiki offers the energest module. In the power consumption calculation, the power consumption of the devices is calculated with the values presented in the datasheet of the device (voltage, current), other values CPU tick number, transmit, receive values. Using this feature, the hourly power consumption of the applications which are implemented in the thesis will be compared in the following chapters.

2.2 MSP430 Based Telosb Mote

AS-XM1000 and MTM-CM5000-MSP models produced by Advanticsys are used to install the software in this thesis. Since the used devices have a USB interface, they can be easily installed via the serial communication interface of the Linux operating system. These devices support Contiki and TinyOS operating systems, these platform files can be reachable by the website of the products [2] [3]. The devices use the first and second-generation versions of the MSP430 processor. Both devices have a ZigBee radio. Devices can be used wireless as there is a battery compartment on them. There are also temperature, humidity and light sensors on the device. Besides, the battery status of the Contiki operating system can also be queried. It can get IPv6

addresses by using a router through USB.

Figure 2.1: Telosb Motes XM1000 [2], CM5000 [3]

2.3 Zigbee

Zigbee is commonly used in wireless sensor networks. When sensor networks are wired, data sharing is more secure and faster, but wireless sensor networks come to the forefront because of the easy installation. The IEEE 802.15.4 wireless data standard developed by the Zigbee alliance is used. The Zigbee module uses energy better than other wireless data standards and operates affordably and securely. Besides, the high density of the node network is provided by Zigbee. Compared to the estimated battery life, Zigbee can work for years, while Bluetooth can work for days and wireless can work for hours. In addition, Zigbee requires 4-32 KB, Bluetooth requires 250 KB and wireless requires around 1MB memory space [17]. Although Zigbee has the lowest data transmission speed, the Zigbee is sufficient in the applications used in the thesis. Range varies between 10-100 meters according to environmental factors. The data transmission rate is 250kb / second for 2.4 GHz, at a slower rate than other wireless devices; however, this speed is sufficient for transmitting sensor data. Also, the interference detection feature is used for reliability. It calculates Packet Error Range for interference detection [18].

2.4 Data Distribution Service

The DDS (Data Distribution Service) is a kind of middleware protocol and API(Application Programming Interface) standard for data-centric connectivity from the OMG(Object Management Group) [19]. Today, there are many DDS implementations developed by various communities. OMG aims to produce a solution that can be used on all versions by partnering with the methods used on DDS QoS. Another important feature of DDS is that it is a middleware that can be used by many software languages. The interoperability feature of the DDS middleware stands out when compared to similar protocols. The DDS middleware is a software layer that abstracts the application from the details of the operating system, network transport, and low-level data formats. The same concepts and APIs are provided in different programming languages allowing applications to exchange information across operating systems, languages, and processor architectures [7]. The DDS is more secure in comparison to socket APIs because the data type is defined. When DDS compares with web services, web services do not provide data type reliability because they use HTTP and XML protocols. In addition, web services must be configured before a connection can be established. In addition, QoS settings can be done more flexibly on DDS. [20] All entities are shown in Figure 2.2 must be created for realizing the secure data sharing scenario. The main element of the DDS model is the participant, every component which is used in DDS software is created by using the participant component. While the user can create many participants on the same computer, DDS allows the creation of participants on different computers. In order for the communication to take place, users must be on the same network, their quality of service settings should be the same and data connection between participants shall be successful, all data writer and readers shall be matched and all participants have to be in the same domain. The communication of the devices or computers with each other is possible with the use of a common data structure. The software developing communities create data structures that can be used in the application layer in the software language of the user by taking the file containing the data structures in IDL

format as input. Each participant should have the same structure. Also, the datawriters of each data structure can be adjusted to QoS. For example, if it is data that is continuously updated and does not pose a problem in case of data hijacking, it might be reasonable to use the best-effort method instead of reliable in the QoS properties of data losses. If the data is of critical importance, the relevant option is selected as reliable, and in case there is no data in the Qos events, it subscribes to an event and informs the participant about this situation and makes it operate accordingly. The inform event is also configurable by specifying the timeout period. Data is read over structures named DataReader and data is sent from the DataWriter structure to the relevant domain. A topic structure is created for any data that is sent and written. There are cases where different topics with the same data structure can be produced. For example, suppose that two separate temperature data are received from the telosb models used in the Contiki operating system, the first device measures the CPU temperature and the other device measures the atmospheric temperature. Both data structures listen to the data structure related to temperature information on different topics and data readers, as a result of this prevents from creating an extra separate data structure for the same data. The data published in the DDS domain is sent by DataWriters. Similar to data readers, the quality of service settings is configurable for DataWriters. DDS middleware can inform on_sample_lost, on_subscription_lost and other similar events that are helpful in different use cases. Data readers can read data in two different ways. These are handled in two ways: Listener-based data access and Wait-based data access [21]. If there is a change in all subscribed QoS events in the Listener-Based version, the component that listens to either DataWriter or DataReader will notify the every event changes separately about the data transfer. In the version with WaitBase, one or more event information comes with a single message. The component notifies the user when the desired event information is acquired. Guard conditions are predefined when creating the component which read data and quality of the service events, therefore the user is only notified when the desired event information changed or new data is available. The overall architecture can be seen in Figure 2.2

.

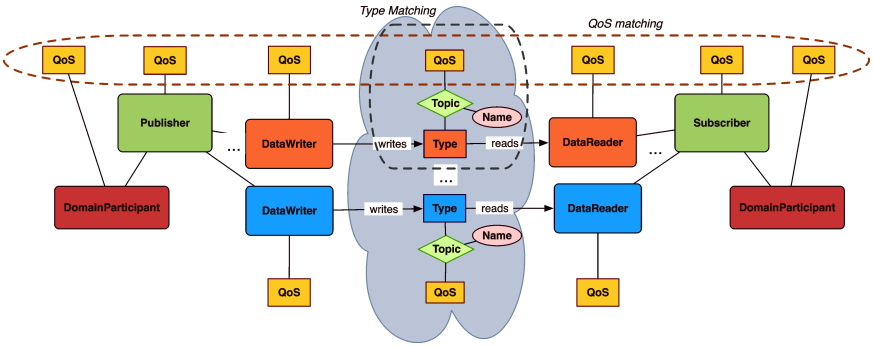

Figure 2.2: General structure for DDS [4]

2.5 MQTT

MQTT is a low-resource protocol that runs on devices with resource problems using publish-subscribe logic. This protocol includes a central server service. The service structure used on MQTT can be either a mosquitto server running on a local network or a Watson platform developed by IBM can be used. It is the duty of sending the messages from the server to the subscribers who listen to the messages published on it. This protocol, which is running in the application layer, performs message transfers using TCP/IP. The devices communicating in this structure do not have to recognize each other directly. Internet of Things is useful for the Publish-Subscriber structure because the sent data can be transmitted to all devices on the same network. Besides, any small delays while sending sensor data will not cause any problems. Data can be received from sensor nodes by using topic names. For reading the data sent on MQTT, the predefined topic name must be created similar to DDS. Applications that listen to the MQTT server can communicate with each other if they have the same topic name as the sender. The topic is used to make it more specific separating names with "/" character. If we try to give an example of the sensors we use, if we assume that the microcontroller measures both atmospheric temperature and CPU temperature, only CPU data can be read by using the "temperature/CPU" topic.

If the subscriber wants to read all the temperature data under this topic, it can access all the temperature data using the temperature/# topic. Similar wildcard expressions can also be used as a topic structure. If multiple data is sent, REST structures such as JSON and XML can be sent over MQTT.

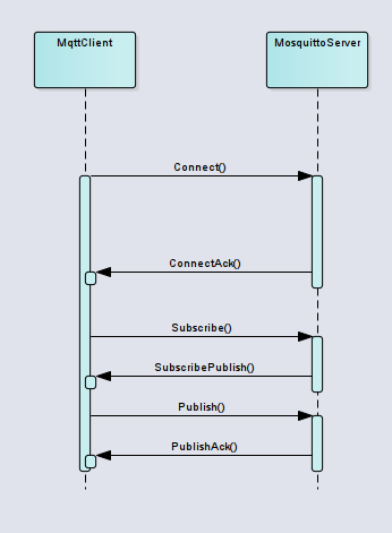

Figure 2.3: MQTT general connection architecture

2.6 Mosquitto

Mosquitto is a server application that performs a publisher-subscriber structure for MQTT protocol [22]. The mosquitto software developed by Eclipse has an interface that allows you to start the mosquitto server automatically when the computer is opened. The log messages from the server can be seen from the terminal application. The mosquitto server can be run on a local network with the desired port(default 1883) and address, the mosquitto publisher writes data with specific topic name data to mosquitto server via the terminal interface and mosquitto_sub component of mosquitto listens to the data on the specified topic from the mosquitto server. Mosquitto_sub can be configurable for desired data by giving the topic name if the user wants to read all data subscribed from the server, the mosquitto_sub reads the topic name with # character for reading all data.

2.7 MQTT-SN

Although the MQTT protocol is used for resource-constraint devices, data sharing on some devices are still not possible due to the size of the memory footprint. For this reason, the MQTT-SN platform has been developed specifically for sensor networks working on UDP instead of TCP [15]. Besides, the connection between MQTT-SN and MQTT protocol is provided by establishing a connection with RSMB on the Mosquitto server. Both server applications run interoperable. In this thesis MQTT-SN application realizing the data transfer with MQTT-SN through the mosquitto server. Unlike MQTT, resources are used to consume less energy at lower bandwidth. First of all, the link message is translated into three messages. Connection messages are sent via a topic named "Will". Topic names are implemented with 2 bytes instead of string notation. For the devices with power problems, all messages are stored on the server and used to send messages to the devices that are reopened [5].

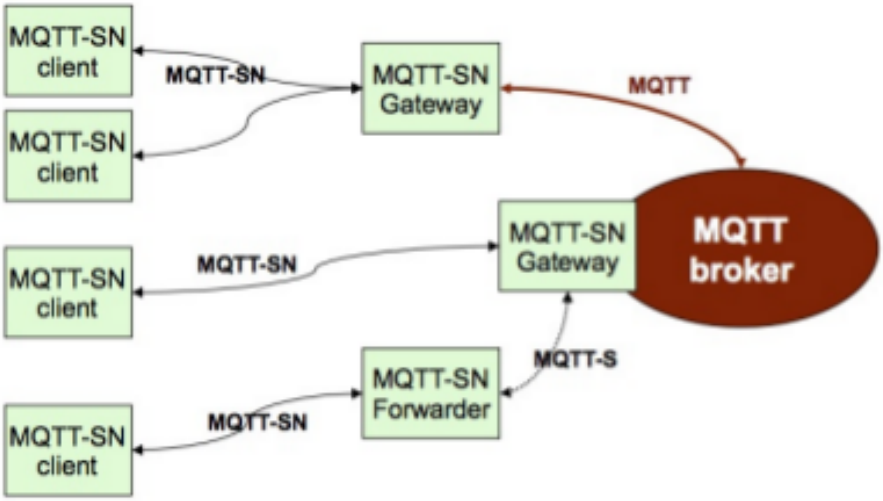

Figure 2.4: MQTT-SN Architecture [5]

2.8 Literature Survey on DDS Gateway for Resource-Constrained Enviroments

The DDS version developed by Opensplice and RTI works on powerful devices. However, the Contiki operating system used in the writing of the thesis could not be ported in two versions due to its size. Similarly, these two DDS distributors are working to reduce the memory footprint. RTI microprocessor-level DDS micro version for the operation of DDS has been released. [23]. This version only aims to use less memory space by implementing the basic features of DDS. In the current implementation, the settings related to the operating system can be defined generically, thus providing the possibility to port to other operating systems in the future. Currently, FreeRTOS supports Windows, VxWorks operating systems, but part of how to port to other operating systems is available on the document [24]. It is not possible to work on the device used for the thesis with its current size. A similar operation can be created when the package is sent via ZigBee af-

ter the port operation for Contiki is completed. Similarly, the Lite version of OpenSplice aims to reduce the memory footprint of the DDS. Although it can go down to 200kb in size, it is not possible to install it again on the Contiki operating system due to the files needed for the sensor. Another solution is similar to the client-server software described in the beginning. In this solution produced by EProsima micro, low-source devices can join the DDS topology by exchanging data with the xrce-server as a client. Currently, Linux, Windows and Nuttx operating systems are used. It can be handled in future jobs by porting in accordance with Contiki [25]. The DDS-XRCE standard used by eprosima and RTI was also published by OMG [6]. Similarly, SDDS software has modified beyond the standards specified by OMG DDS to create a separate packet structure for sensor networks. This software provides limited support for RIOT, Linux, Contiki, and Tiny-OS. The existing software supports devices with Avr processor architecture in the Contiki operating system. At this stage of the thesis, the code generated for the Avr processor will be updated to run for the MSP430 architecture. Also, Contiki will be added to the operating system and the makefile where the required files for DDS are compiled again for the MSP430 processor architecture. Finally, the packet structure sent over ethernet will be sent over ZigBee and data communication will be provided.

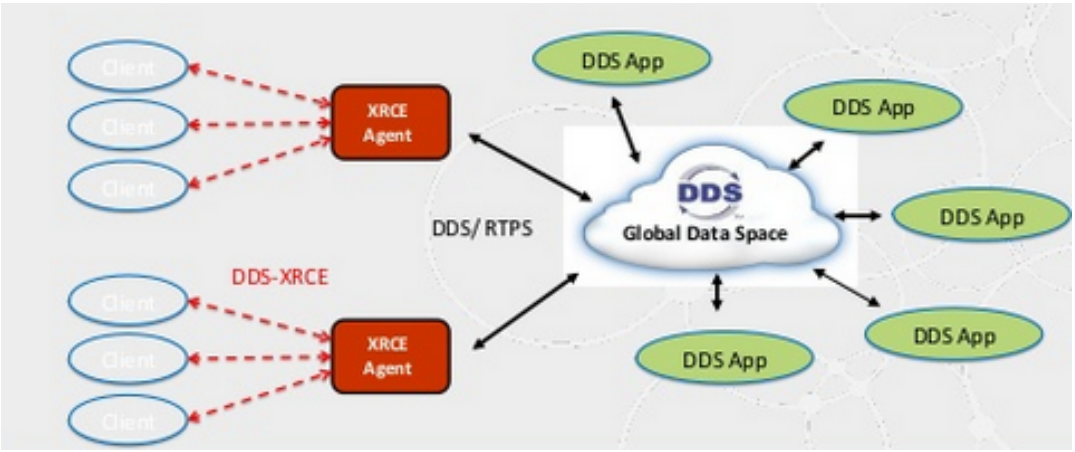

Figure 2.5: DDS-XRCE architecture [6]

2.9 Literature Survey for MQTT

MQTT is an open-source standard developed by IBM. As it is known, when using the MQTT protocol, TCP is used for providing and realizing the reliable transmission of packets. The most effective implementation for embedded systems is discussed in the referenced document. It is observed that the complexity and memory usage of the protocol increased when Qos levels increased. Also, the increase in Qos levels led to a slowdown in the time taken for data transfer. [26]. In this article, the differences with CoAP (Constrained Application Protocol) are examined in terms of memory usage. These two protocols differ in terms of TCP usage. If data transfer is performed in an environment where message loss is high, it is understood that MQTT comes to the fore. In another article, the MQTT-SN version of UDP used in the network layer was examined by updating the MQTT protocol to be used more effectively on devices with resource shortages of [15] memory space. HTTP and MQTT in the article comparing the similarities of MQTT with QoS levels were examined. QoS1 (Assured transmission) has been mentioned to have one-to-one similarities in terms of communication with HTTP and reliability. Since MQTT is an asymmetric protocol, it is mentioned that it consumes fewer resources and has more performance. CoAP and MQTT [27] for smart home systems are examined and the similarities of these two protocols to HTTP are mentioned. It has been mentioned that MQTT comes to the fore by consuming 5 times less power than HTTP. According to CoAP, it is ahead in both power and security issues.

CHAPTER 3

IMPLEMENTATION AND METHODS

3.1 Dataset

The XM1000 telosb sensor node which is produced by Advanticsys [12] company is decided to run the tailored DDS version for the sensor sharing on the Contiki operating system via Zigbee protocol instead of the UDP protocol. A similar device called CM5000 sensor node which is produced by the same company has the same sensors(temperature, humidity, light) on the board. The MQTT version of the sensor data transfer is run on the CM5000 telosb sensor nodes. Both sensor nodes have a reset button on the board, the application restarts at the time when the button is triggered. The devices also have an input button which receives input from user and triggers a button press event, the SHT11 humidity, temperature and light sensors are available on the board. Data received from sensors are shared with either serial communication interface or radio interface on the sensor node. In the thesis, data obtained from the sensors using the modified version of the DDS software and MQTT will be shared on the devices. It is planned that the LEDs on the device are lit each time data is received. Since the device has Sht11 sensors, meaningful data are obtained according to the datasheet of these sensors and the web site of the official Contiki repository [1].

Table 3.1: The calculation of the values by using sensor data. [1]

Value	Formula
Temperature	(Temperature Sensor Value* / 10) - 396) / 10
Light	10 * LightSensor Value / 7
Humidity	Same Humidity Value

Before the data is received, the sensors to be received via the Contiki operating system are activated and then received and processed in a loop.

3.2 Proposed Methods and Models

3.2.1 DDS

3.2.1.1 Installing Contiki Development Environment for Sensor DDS

The Contiki operating system is characterized by dynamic loading and unloading of code compared to other operating systems, allowing multiple operations, and thus distinguished from other similar operating systems. Also, being open-source provides accessibility advantage [28]. Operations can interact with each other and decide when to proceed. It is also important that the code developed with the cooja simulator can be used independently of the actual hardware for the selection of the operating system. The Contiki operating system can be used and applications could be developed via the instant Contiki virtual machine, where all platform data of the operating system is pre-loaded. However, there is a need for more space to try sample applications related to the thesis. Besides, a more optimized loading of the code requires version 4.7.3 of the compiler for MSP430. Ubuntu 16.04 64 bit version was used during the development of the sample applications since this version only supports 64-bit architecture and could be downloaded via Github. 4.7.3 version of the MSP430 software was downloaded from [13] at

the given reference. Since the latest version of this application was downloaded from the ubuntu repository, 4.6 was used when compiling the code. Ubuntu uses a default version of the MSP430, so the memory footprint problem persists while developing a sample application.

```
1 sudo visudo
2 edit secure path value similar to first
```

3.2.1.2 Tailoring DDS Software and Enabling Zigbee Gateway for Contiki OS

Sensor DDS software includes DataReader, DataSink, DataWriter, History, Locator, Topic, and SNPS package structure, which is converted RTPS into a smaller package. By sending the UDP packet over the network structure, data can be shared over DDS via Linux and other operating systems. The part used over UDP has been removed from the Contiki operating system by using configuration file or makefile. All components mentioned above are initialized in the first part of the software produced. To talk about the operation briefly, the topic that is produced by reading the contents of the XML file is initialized. To reduce memory footprint and provide much space on RAM and ROM, the variables are used at the core of the Contiki operating system and SDDS software-related source files are updated. The first change is done in loops uint8_t containing 1 byte is used instead of the integer value of 4 bytes. The components are used in data sharing with DDS via ZigBee, these components are briefly mentioned below.

- **DataSink:** First checks whether the incoming data in the SNPS protocol has the correct structure. In a normal implementation, this data is received via UDP, while in the updated version it is received via the Zigbee package. For Zigbee devices to communicate, both the same domain and the same channel must communicate. DataSink first checks the software version. During this check, convert the incoming byte value using the Marshaling class and check the version value. If this

value is different from the expected value, the message is ignored and the next message is checked. The second byte of the protocol specifies the length of the other bytes in the message packet. The third byte contains the domain address. Only devices within the same domain can communicate in the DDS protocol. Therefore, it compares the domain value in the incoming message with the domain value in the listening application; if the value is the same as the expected value, the message is accepted and the remaining byte values are converted to message structure. The message structure is identified by topic id which is decoded in the SNPS package. Since the devices that broadcast and listen to the data use the same data structure, the order of the data in the message structure must be same order. The text and numeric values are contained in the byte array. This value is decoded in the same way and converted into a meaningful string and numeric values.

- **DataReader:** This class created by Datasink is adding the incoming data to the history class. History class data is stored in a list and the oldest element of this list is read by the data reader with the data-available event. The incoming data can be viewed via the ubuntu terminal via the serial interface.
- **DataWriter:** A new locator is created for each data to be sent via Publisher. The DataWriter class also uses the SNPS package to create the data. The deadline event can also be set via the project macros. For example, if the deadline event is active, then the data has not been received for a certain period. However, in the exemplary embodiment developed, this structure is not active. The compatibility of the generated SNPS package with the UDP is checked in normal implementation. This part has been modified to check its suitability with the ZigBee package. The size of the data is calculated and sent according to the protocol.
- **Zigbee Network Component:** The component reads the Zigbee packet

from a predefined channel, the component is bind to DataSink. Incoming packages are forwarded to DataSink and locators are created in this component. These components also used as a network layer. The datawriter directly sends the data through this component. The current position of the data, buffer length and other values are retrieved from data by using the netbuff component. Significant data is retrieved from this class, these data are copied into the byte array to be sent, and created data is sent using the ZigBee radio interface.

- **SNPS:** The SNPS component is used for writing domain id, parsing and reading sub-messages from SNPS data. It also contains self methods to discard data, read data and addresses. For the Zigbee gateway reading channel, the feature is added to this component. All read-/write operations on data are realized in this component.
- **Marshalling:** The marshaling component is used for encoding and decoding the integer and text values.

3.2.1.3 Creating Source Code for MSP430 based Telosb Motes for Sensor DDS

As described in section 2 of the DDS, a code structure is created according to the specified data structures. During the generation of this code, the application specified on the reference was used. SDDS [8] software is configured over XML and converted into meaningful code fragments by GSL(Generator Scripting Language) [29]. The history depth, number of locators, buffer size, publisher and subscribers will be configurable through the XML file in Appendix D. Generated source codes are created the root directory of the script. The GSL script is updated for Contiki and MSP430 architecture. The reference paths are also updated and Makefile's first and second level versions are generated. In this sensor sharing example the following files are auto-generated:

- sensor_data-ds
- sensordata_test_publisher
- sensordata_test_publisher_imp

The topic details are defined in another XML file. Topic names are referenced in the sdds.xml file by using include tag and the path of the file. This file contains the variables in the structure. The telosb sensor nodes have a temperature, humidity and light sensors. To share these pieces of information, the topic structure created and defined in the sensordata.xml file. The content of the XML is seen as below. The device id is defined as a primary key for every device. The data will be identified by the device id and these ids are predefined in source code. Also, the domain id and topic id are defined in the same XML file in Appendix B.

To explain briefly, the type of temperature value, light, humidity, and device id are DDS_Short.

3.2.1.4 Building and Creating a binary file from source code

SDDS currently runs on only avr-based microprocessors. The data fields, the method of creating a binary file, a compiler for gcc and some extra changes are required to build and create a binary file. Besides Zigbee related files are added and SDDS types are added for msp430 architecture. The default makefile could be accessible to the platform folder of the device. In the sample application, the rule defined in default makefile is overridden in some cases. The makefile is seen in the code block below. The compiler cc flag is set as gcc-msp430 which version is 4.7.3. In the makefile in Appendix A, both SDDS and Contiki operating system makefiles are included. The Contiki operating system can use the makefile under the platform according to the specified Target value. Binary file creation is done by adding a

dependency to files related to SDDS and Zigbee binary file in makefiles and the related files could be compiled in this architecture. In Msp430 version 4.7.3, SMALL = 1 is assigned to ensure that the produced binary file is optimized for memory space. IPV6 is also set as inactive to reduce memory footprint. The platform target can be configured through the TARGET variable to support both devices (XM1000 and sky sensor nodes). Because the application name of a binary file to be created is defined as a variable, when this name is to be changed, the code is generated in a well-formed structure automatically. In XM1000 and sky platform files, the production of binary files consists of the same stages. Since the main difference between the two is caused by the target name, this value is set to depend on the target variable on the makefile. Both devices can compile the code with the same infrastructure. An installer script file is prepared to upload the generated file to the devices. This script file decides which device to load by giving the number of devices with the MOTES variable. The motes value starts at 1 in contrast to the array index numbering which starts from 0. The sequence diagram of the build process is given in figure 3.1 below. The final state of the sequence diagram is the creation of the binary file. Detailed information about the meaning of special characters used in the makefile is found in the referenced link [30].

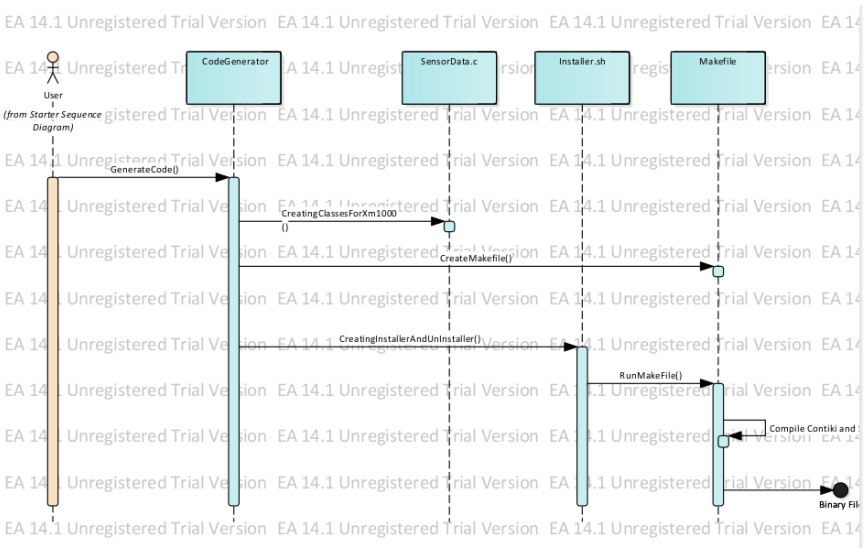

Figure 3.1: Build process for sensor DDS

3.2.1.5 Sequence Diagram for Sending Sensor Data

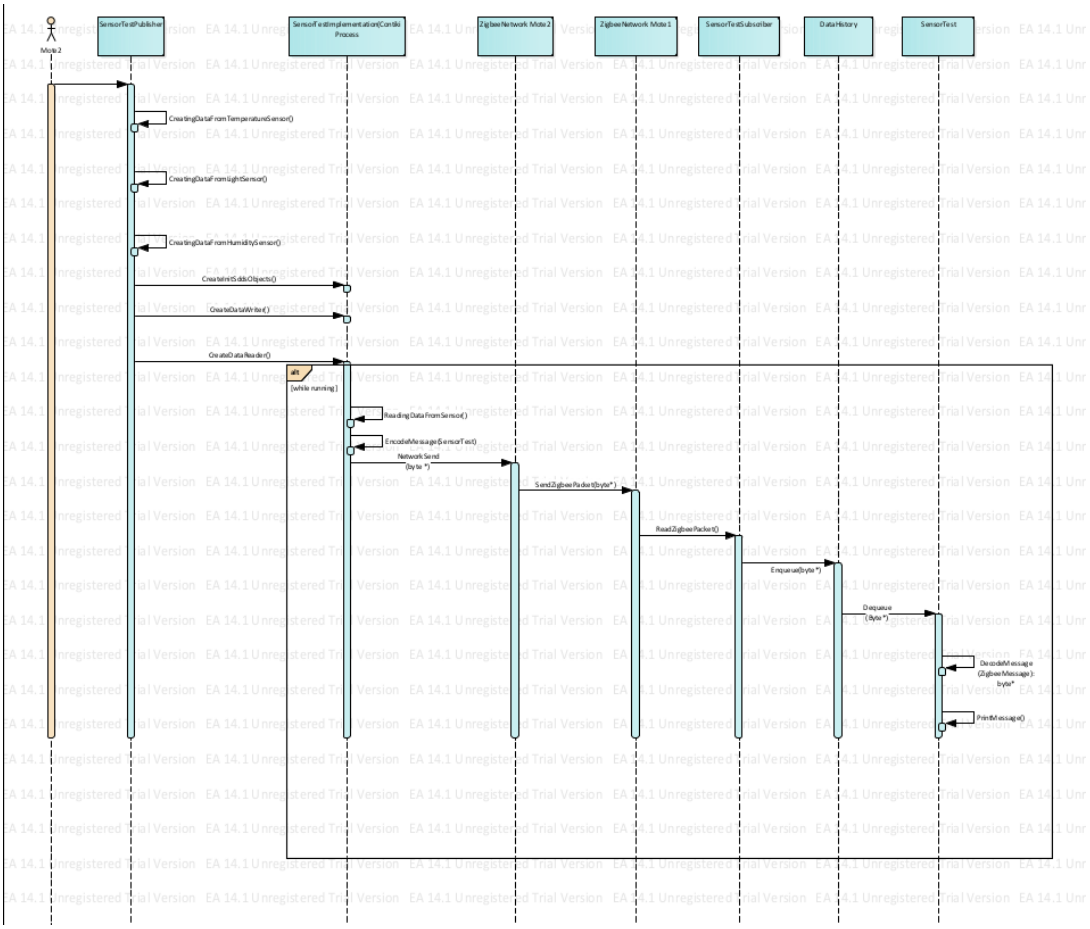

Figure 3.2: Sequence diagram for sending data

In the previous sequence diagram, it is mentioned how to create a binary file. This section shows how the sensor data is received and sent by the telosb sensor node. Roughly, the first sensor reads over the sensors, processes the data and converts them into meaningful values. This data is filled per the data structure by using the topic and sent via the ZigBee packet by adding the device id, domain id and topic ids and topic which are specified as hard-coded. Another telosb sensor node that listens to the same domain captures this incoming data and inserts that data into the DataHistory. The data reader reads the incoming data and prints it to the terminal, thus completing the data transfer.

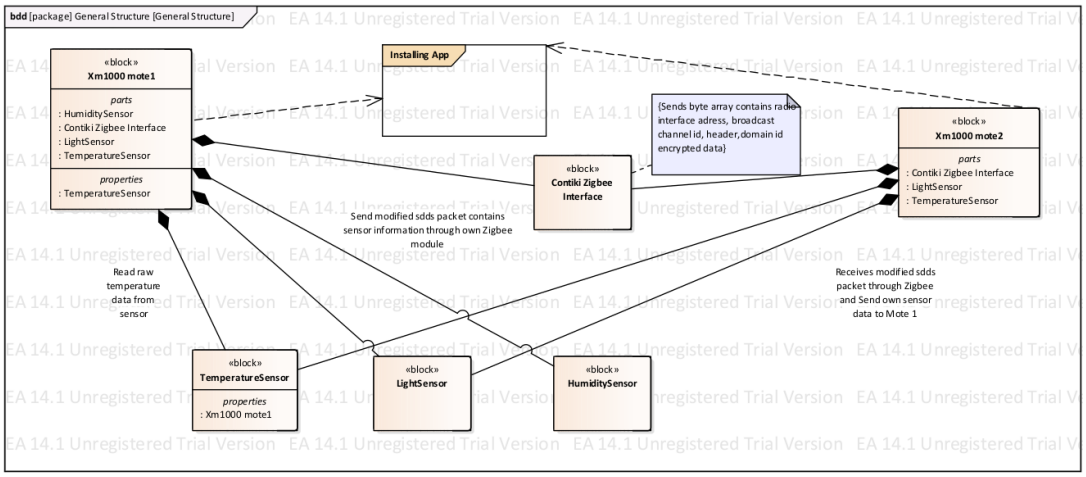

Figure 3.3: Overall block diagram

The overall block diagram and dependencies of the components could be seen in Figure 3.3.

3.2.1.6 Output of the Demo application

In the exemplary application, a separate configuration is prepared for both devices on the makefile. The two devices broadcast data to each other, the device IDs 1 and 2 are assigned to both devices. Humidity, temperature, and light values are shared in the data sent. The data extracted from the serial interface is formatted and printed on the screen via the terminal. Data sharing can be seen from the following figures.

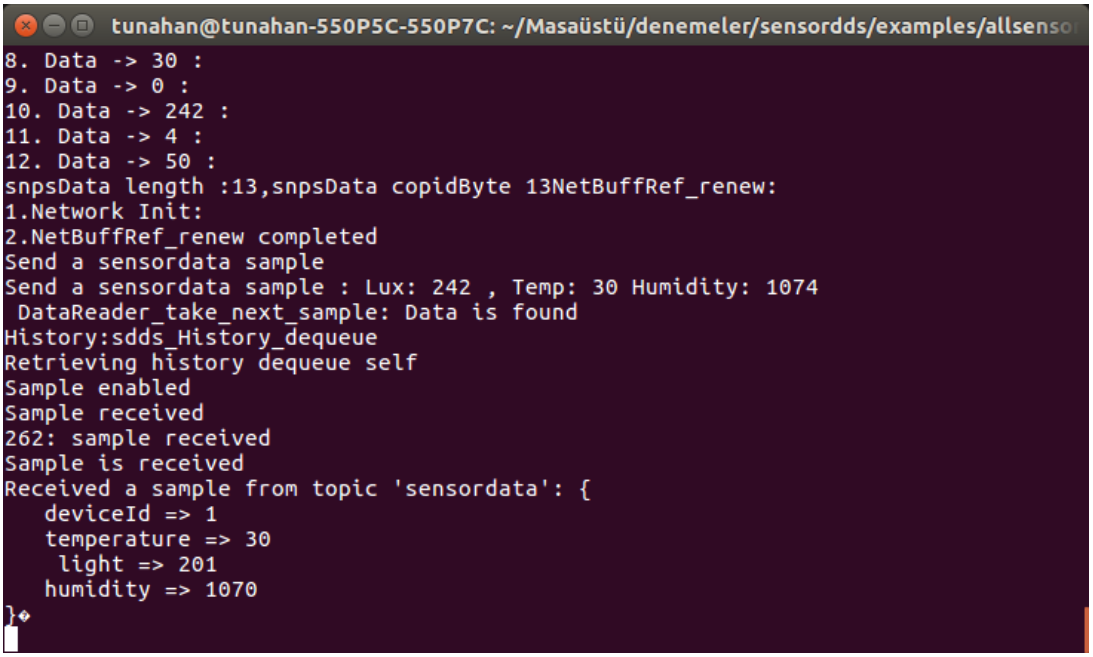

Figure 3.4: Receiving data from XM1000 Mote1

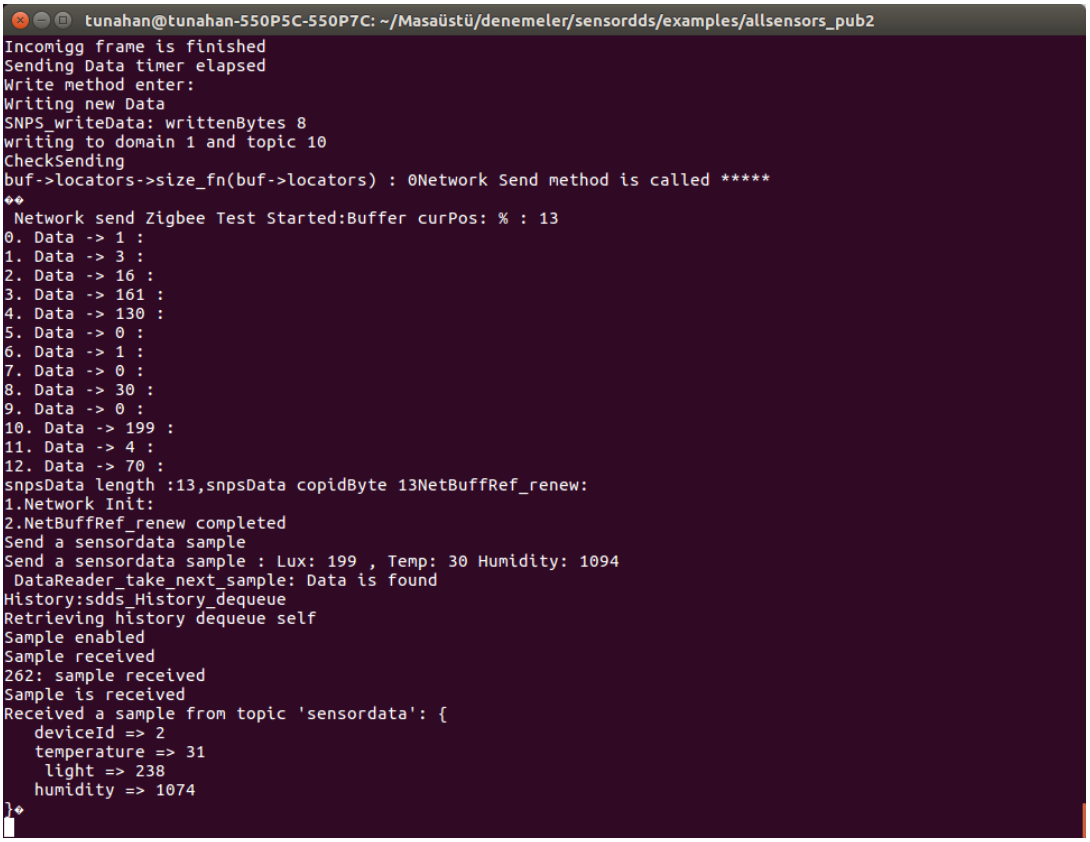

Figure 3.5: Receiving data from XM1000 Mote2

3.2.2 MQTT-SN

3.2.2.1 Installing Contiki Development Environment for MQTT-SN

The TCP network stack version of MQTT could not be installed on the device (XM1000 and CM5000) even though the configuration file related to Contiki and the macros defined on the project were updated. To solve this problem, it was decided to use MQTT-SN [15] software for sensor networks that use UDP instead of TCP. The other change is related to the topic name. It doesn't use the full name of the topic name, the number equivalent two-byte values are used as the topic id for both client and server sides. Also, the connect message of MQTT is divided into three parts and the QoS level is defined as 0. The mosquitto server is not compatible with MQTT-SN. The Really Small Message Broker [31] and Mosquitto are using together in a new project called Eclipse mosquitto. It is compatible with the mosquitto server, this feature provides interoperability with mosquitto, the data sent by MQTT-SN will be received by an MQTT client.

3.2.2.2 Using MQTT-SN for Sharing Sensor Data

In the previous section, data exchange over MQTT was attempted by communicating with the mosquitto server over the devices. Due to the use of MQTT TCP and QoS (Quality of Service) features, efforts were made to reduce the memory footprint, but not enough reduction was achieved. In this step, it is aimed to provide communication over the device by using special-purpose MQTT-SN for sensor networks. In the same way, data exchange and listening of data can be done via the mosquitto server. However, in this stage, the Really Small Message Broker [31] application can communicate with mosquitto and work properly with MQTT. The RSMB is low power consuming and works better for sensor devices. It has a memory space of approximately 200 kb. RSMB supports the MQTT-SN protocol and can interpret UDP messages. However, it can also connect to mosquitto by establishing a bridge function with the MQTT protocol. The commands necessary

to establish the connection will be explained in detail under the following headings. In addition to this, the telosb mode has to be an IP address to achieve communication with RSMB and Mosquitto. The sample under the sky platform will be modified and the border-router application shall be run before the run MQTT-SN protocol. The other sensor node gets the IP address by using the border-router. Also, a cooja simulator is used for using multiple publishers and subscribers. The details of the border-router and other settings are also mentioned in the following headings [32].

3.2.2.3 General Structure

Before going into detail, the following illustration shows the communication of all components used in the topology. Sky Mote 1 provides a connection to the operating system using the border router application, while the other sky sensor node shares the sensor data by connecting with the computer via this device. To successfully transmit this sensor data, it must be transferred over the serial interface via RSMB.

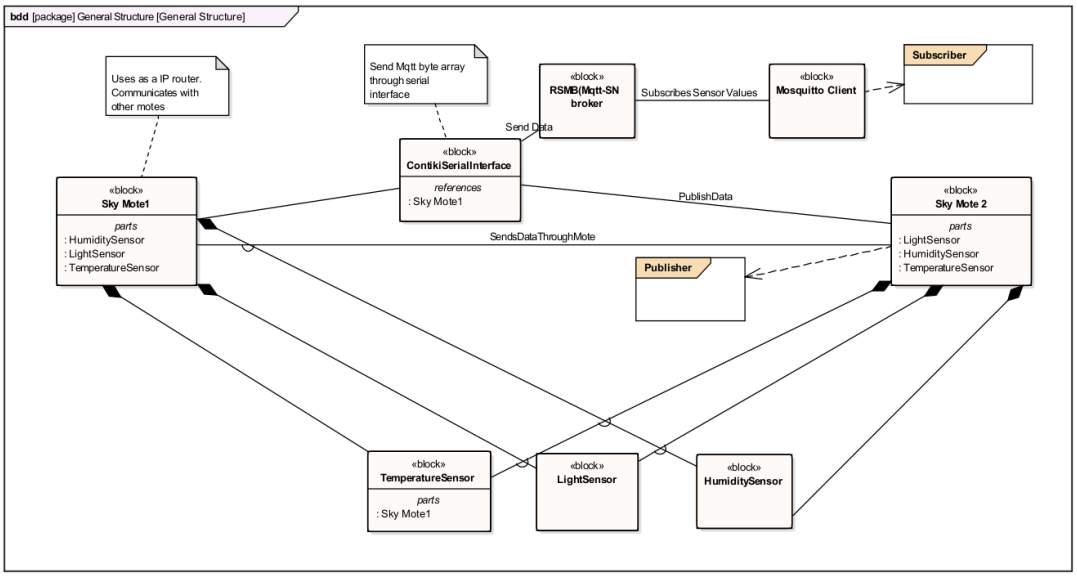

Figure 3.6: The General Structure For MQTT-SN

3.2.2.4 Installing RSMB on Ubuntu OS

The mosquitto.rsmb software is downloaded the following link given [33] the reference. The downloaded files are extracted in a directory. The root directory is opened and the working path is changed as source code directory and the following code snippets are executed. Assume that the command prompt working in the root of RSMB download from the link in the reference. The most important point to note here is that in the version used for MQTT, 1883 works by default. Since this port is used each time the computer is turned on, mosquitto's program id is taken from the programs running using the following commands and the program running with the corresponding id value is terminated.

Listing 1 Running RSMB server

```
1        cd /rsmb/src
2        sudo ./broker_mqtts config.mqtt
```

If the output of the RSMB contains the port is already using error message the mosquitto shall be closed before running this server application. The commands below are used for receiving program id and the program contains the program id is killed for running RSMB properly.

Listing 2 Killing Mosquitto server before running RSMB

```
1        ps -ef | grep mosquitto
2        sudo kill xxx \\ xx defines the PID value.
```

3.2.2.5 Burning Node ID for Border-Router

For the border router to work properly, the node id value must be assigned. To assign the node value, the #include "sys/node-id.h" header file must be

added to the Contiki operating system's example border-router code. Immediately after starting the process, this value can be determined by calling the node_id_burn (id) method. The node value will also be assigned during the compilation and uploading image process described in the next stage.

3.2.2.6 Installing Border-Router on Telosb Mote

To get IP over the network of telosb sensor node devices, the border-router application should be used. This application allows the device to receive IP over the serial interface. After the IP reception process is completed, other devices are added to the network via this device. The sample has been updated for makefile serial ports on the border-router application. Once this application is installed on one of the devices, it should be started as a server with tunslip6. The sample application provides a web interface for viewing other nearby devices. Also, routing addresses can be accessed via the web interface of the border router. The following code snippets are used for active serial interfaces. The tunslip use the serial interface as a parameter then start the device as a server. The memory footprint problem reappears when the web interface is activated. If the CM5000 model is taken as an example, the memory.x file under opt/compilers/ msp430/lib/ldscripts/ msp430f1611 needs to be updated. The last folder shows the architecture of the CPU. Since this field requires admin privileges, changes should be made by opening memory.x file with admin privileges. The updated memory file is seen in Appendix E. The memory area ram (wx) and rom (rx) are rearranged by calculating the required memory area. The problem is related to overflow in the ROM memory area. This file is updated by reducing the RAM and increase the ROM memory area. The end of the memory address could be calculated by adding origin and length. The connect-router interface is updated by using a border-router makefile. The serial address of the device is queried with the python script below.

Listing 3 Listing serial ports using python

```
1         python -m serial.tools.list_ports
```

All active serial ports are seen the output of the command. The program uses the following command to install on telosb sensor node. The result of the Python command is the current active serial interfaces. The current serial interface is given as a parameter to the command used to load the border-router. The usage is seen in the code snippet below. The communication port of the serial interface is assigned to the MOTES variable.

Listing 4 Uploading border router image

```
1         sudo make border-router.upload TARGET=sky MOTES=/dev/ttyUSB2
2         sudo make sky-reset && sudo make connect-router_ttyUSB0 #_devId
```

When there is no error message occured by uploading image stage, the xxx bytes is uploaded message seen at the end of the process.

3.2.2.7 Running Tunslip for SLIP Server

The source code of the existing Contiki operating system contains tools folder and the tunslip6 is accesible in tools folder. With make tunslip6 command, this application is compiled and made executable. This executable application is used to establish a connection with the border-router. In the example above, the following line of code is executed in order to run the device with border-router as a server.

Listing 5 Running server bu using tunslip6 tool

```
1         make tunslip6
2         ./tunslip6 -s /dev/ttyUSB2 aaaa::1/64
```

When the application is successfully started, the output shall be similar to Figure below. The webpage of the server is accesible through the first ipv6 address on Figure 3.9

Figure 3.7: Running server on telosb sensor node

In Figure 3.10, nearby devices and route information can be accessed using the server IP address that is run on the tunslip.

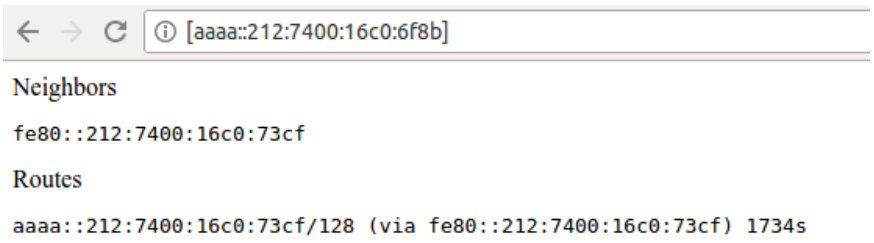

Figure 3.8: The web interface of Tunslip6

In the sequence diagram indicated in the figure below, the compilation of the code, loading it on the device after compiling, running the code loaded on the device with tunslip application and transferring the incoming messages to the network using serial interface are discussed.

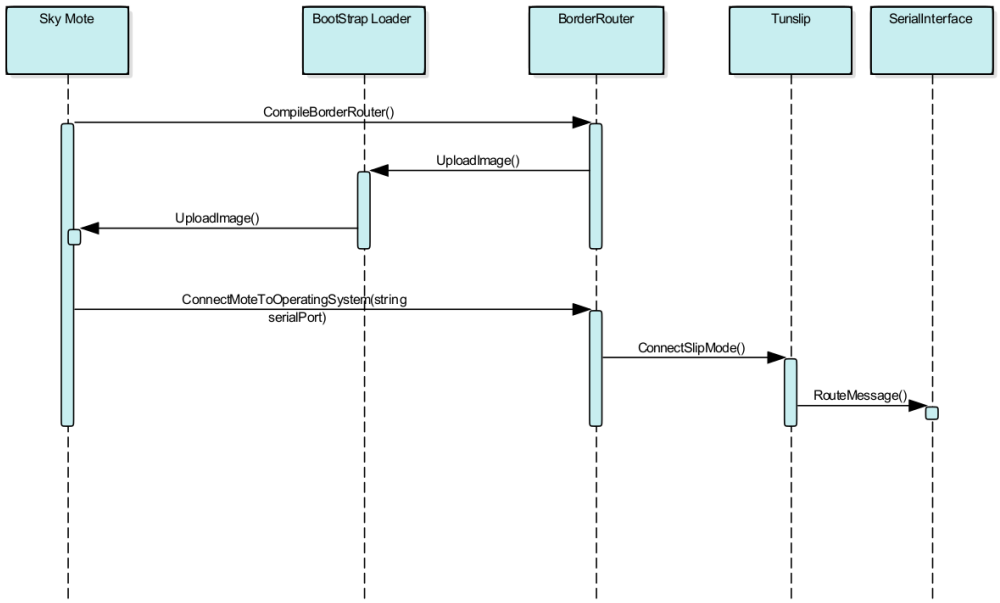

Figure 3.9: Sequence diagram fomgr border-router

3.2.2.8 Creating Binary File and Sending Data

In contrast to the full version MQTT application, the application which uses the MQTT-SN protocol has been successfully installed on, CM5000 telosb sensor node. MQTT-SN protocol uses UDP and little changes have been made to the package structure to reduce memory-footprint. Similar to the DDS application, an infrastructure has been prepared to send sensor data using the MQTT-SN library in the reference [34]. Every 10 seconds, sensor data is sent to the RSMB via the border router. To listen to the sent data, the mosquitto client libraries that were previously established are used. When the border-router application runs on the XM1000, it tries repeatedly to get IP over the server, but it fails. Instead of this device a new virtual sky sensor node, cooja simulator load application as an MQTT client. For listening to all data transfer the client library application, mosquitto_sub, is used to trace data changes. The command is given below is used for listening to all topics

using the following command.

Listing 6 Running mosquitto subscriber for all topics

```
1         mosquitto_sub -t "#" -v
```

The simulator cooja could be built and run the directory below. The tools folder under the root directory of Contiki source code. The simulator is run by using the following commands.

Note: Assume that command prompt shows the root directory

```
1 cd tools
2 ant run #wait until build finish
```

3.2.2.9 Running Simulation on Cooja

In the previous chapters, two binary files were created to send data via border-router and MQTT-SN. In the first part, a binary file of the border-router application is loaded into one of the virtual telosb sensor nodes to verify this structure. After this application, the serial server menu is entered under tools and the server starts to listen with port number 60001. The tun-slip6 application installed in the previous sections. The server is run when the following command is executed. In the command, the IP equivalent address of the localhost (127.0.0.1) is used.

Listing 7 Running tunslip6 on localhost

```
1          sudo ./tunslip6 -a 127.0.0.1 aaaa :: 1/64
```

Thus, the web interface can be accessed from the IP address of the server. Adding a new device updates neighboring devices and route information via the web interface. After the RSMB has been checked successfully as described in the previous sections, a binary file is uploaded to the devices that send sensor data using the MQTT-SN protocol. The binary file creates topics and listens to MQTT-SN events. The sensor values are sent by using the temperature topic. The sequence of creating a simulation for sending temperature data through Cooja is mentioned below. The RSMB shall be run before the simulation is started. In the first stage, the border-router image is added to the Contiki network.

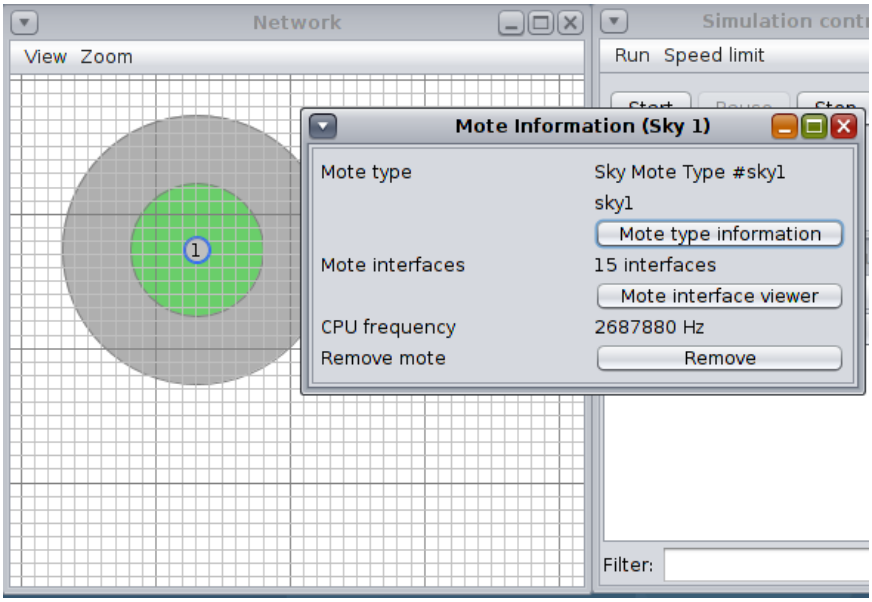

Figure 3.10: Adding a sky sensor node to border router application on Cooja simulator

Secondly, applications that send sensor data to the medium using MQTT-SN should be added. In the same way, the sensor type is selected as the sky and the devices are added next to the server device with border-router installed.

To run the border router application, the listen port feature of the server device must be started. The server starts broadcasting with local tunslip commands under the "Creating Binary File and Sending Data" section. The simulation starts with the addition of newly added virtual devices. Sensor data can be accessed via the outputs of the devices.

The newly added sensor node is selected from the serial settings(SERVER) option under the settings menu of the cooja. The listening process starts from port 60001, which is set as default.

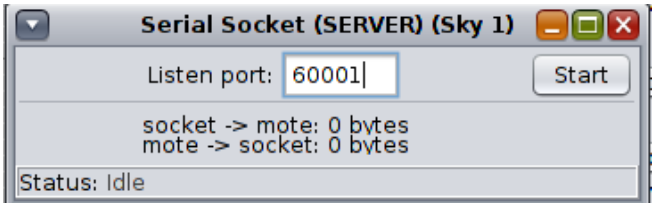

Figure 3.11: Running sensor node as a server

Secondly, applications that send sensor data to the environment using the MQTT-SN protocol must be added. In the same way, the sensor type is selected as the sky and the devices are added next to the server device which border-router installed. To run the border router application, the listen port feature of the server device must be started. The server starts broadcasting with using local tunslip commands under the "Creating Binary File and Sending Data" section. The simulation starts with the addition of newly added virtual devices. Sensor data can be accessed via the outputs of the devices. All data traffic on the network can be listened to by using the mosquitto_sub command.

3.2.2.10 Running Application on Real Hardware

Sky sensor nodes [3] that communicate successfully on simulation are communicating on real hardware in this section. In this case, a border-router

application is installed on one of the devices and then the IP address acquisition and server installation process is completed by using a tunslip application under the tools folder in Contiki OS. First, the MQTT-SN connection is used with the second device where the temperature data is installed. As the default value MQTT-SN application runs through 1884, the configuration on the program is set to Qos level 1 and the port value is set to 1884. After activating the sensors on the device programmatically, the obtained data is converted to meaningful values and sent to the application where the MQTT-SN server is running via border-router. This server also listens port 1883 in MQTT data. The incoming data can be read by both a third device and mosquitto_client. For the border router application to work in harmony with the Linux operating system, the serial interface information to which it is connected must be given as a parameter. It can automatically run the tunslip6 interface with the related command by adding recipe information for the serial port on the makefile.

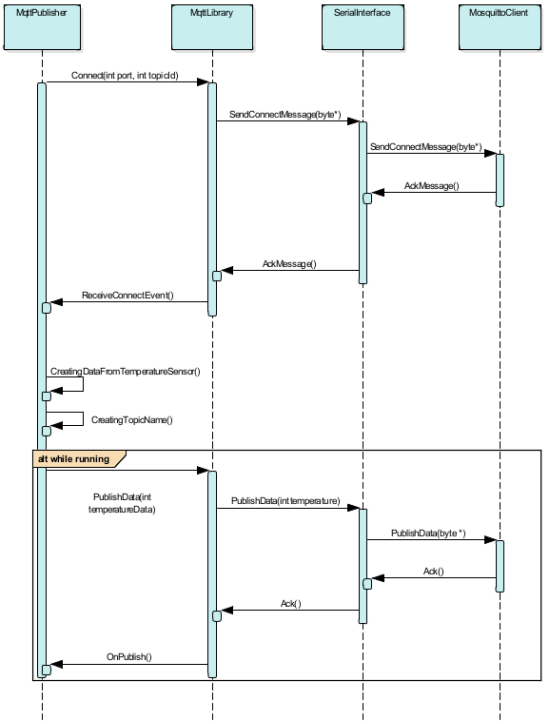

Figure 3.12: The sequence of publishing data

The figure above shows the sequence diagram illustrating the messages sent and received by the sky sensor node and the data received by the RSMB server. In addition to this when all flow diagrams are successfully performed, the following output is shown by the mosquitto client.

```
tunahan@tunahan-550P5C-550P7C:~$ mosquitto_sub -t "#" -v
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
/topic_1 Temp:32
```

Figure 3.13: The output of the mosquitto client application

3.3 Encountered Difficulties and Their Resolution

3.3.1 Sensor DDS

The platform files of the XM1000 developed by advanticsys company, which are used for the thesis, are not installed by default on the Contiki operating system. To resolve this problem, the platform files on the device's website were downloaded via the link in the reference [2]. In the example application using the SHT11 [1] sensor doesn't work properly in default, because the sensors could not be found, the missing parts of the SHT11 sensors were copied from the files under the sky sensor node and made available to the relevant telosb sensor node via Contiki. The bootstrap loader for XM1000 is not compatible with a newer version of the python library. The older version of platform files is used for providing compatibility.Besides, DDS middleware could not be installed due to the memory problem. To resolve the problem, the newer version of the MSP430 compiler is installed. However, the Ubuntu operating system uses the 4.6.3 version that is available on the Ubuntu repository. The newer version couldn't be installed on the 32-bit operating system. Therefore the customized OS for Contiki OS(InstantContiki) couldn't be used. The newer version(MSP430 4.7.3) is only available for 64-bit architecture and installed on the 64-bit Ubuntu operating system. Makefile uses the older version of the because the path values show the binary files for the older version. For using the newer version of MSP430, the sudovise and /.bashrc files are edited with new path values for MSP430 4.7.3 to run applications with the newer version. The command below shall be run before the application start for a normal user. The basic package components will be seen in the Figure below.

```
1 gedit HOME/.bashrc
2 export PATH=PATH:/opt/compilers/mspgcc4.7.3/binary
/opt/compilers/mspgcc4.7.3/binary is an absolute path.
It will be changed according to the path on computer
```

Figure 3.14: Changing the path values

In some cases, the applications shall be run as sudo user. When the application starts with a superuser, the path values couldn't be visible and the program couldn't be run as expected. To resolve the path problem for a superuser the code below is applied for sudo user. By applying these changes, the program can be installed on the XM1000 wireless sensor node. With version 4.7.3 of MSP430, the application cannot be installed on the CM5000, and an error occurs due to register support problems. To solve this problem, only the temperature sensor is activated and compiled with version 4.6.3 without 20-bit register support. In the MSP430 folder under the cpu folder of the Contiki operating system, the compilation process is completed by replacing CC(Compiler binary path) LD(Linker path) variables with binary files of version 4.6.3.

3.3.2 MQTT

The bootstrap loader uses the python-based bootloader. This bootloader version is compatible with an older version of the serial library, the python serial library which is part of the bootloader is updated with a new interface.

3.3.3 MQTT-SN

The server used for the MQTT-SN application shares the same port as the server used for the MQTT. Therefore, unused server software must be closed.

3.4 Power Consumption Calculation for Sensor Sharing Applications

The Contiki operating system is used to receive sensor data via the both sensor DDS software, MQTT and MQTT-SN. Energest module is used on Contiki operating system to calculate energy consumption on devices with resource constraints. The operations of this module can be accessed via reference [35]. The power consumption can be calculated by using the macros given in the table below and the current and voltage values specified on the data documents of the devices.

$$P_{CPU} = \frac{(Current * Voltage * EnergestValue_{CPU})}{RTIMER * IntervalTime} \tag{31}$$

$$P_{LPM} = \frac{(Current * Voltage * EnergestValue_{LPM})}{RTIMER * IntervalTime} \tag{32}$$

$$P_{TRANSMITTER} = \frac{(Current * Voltage * EnergestValue_{TX})}{RTIMER * IntervalTime} \tag{33}$$

$$P_{RECEIVER} = \frac{(Current * Voltage * EnergestValue_{RX})}{RTIMER * IntervalTime} \tag{34}$$

$$P_{TOTAL} = P_{TRANSMITTER} + P_{RECEIVER} + P_{CPU} + P_{LPM} \tag{35}$$

$$P_{BATTERY} = Current * Voltage \tag{36}$$

The total power and current values of the two batteries used in the test are 7.5W. The equations above are used in calculating hourly and total power consumption and these formulas shall be defined and implemented for all three implementations to compare power consumption results. The calculation is done for CPU, LPM, TX, and RX. For testing the relation of message sending period and power consumption, the tests are done for a period of 5,10,20,30,60 seconds. The output of the sensor nodes is redirected to the text file, the developed java application processes this data and creates charts for all units(CPU(Central Processing Unit),LPM(Low Power Mode),TX(Transmit),RX(Receive)).

Table 3.2: The macro definitions for energest module

ENERGEST_TYPE_CPU	The CPU is active.
ENERGEST_TYPE_LPM	The CPU is in low power mode.
ENERGEST_TYPE_DEEP_LPM	The CPU is in deep low power mode.
ENERGEST_TYPE_TRANSMIT	The radio is transmitting.
ENERGEST_TYPE_LISTEN	The radio is listening.

In order to use the required module for energy calculation, energest module must be activated on contiki-conf.h file. To use the energest module, the file defined in the below shall be included the project under tools directory of the Contiki OS.

Listing 8 Including makefile for Power Module

```
1         include $ (CONTIKI) /tools/powertrace/Makefile.powertrace
2
```

Listing 9 Activating the Energest module via configaration file

```
1         #ifndef ENERGEST_CONF_ON
2         #define ENERGEST_CONF_ON 1
3         #endif /* ENERGEST_CONF_ON */
4
```

CHAPTER 4

RESULTS AND DISCUSSION

The first objective of the thesis is installing the applications successfully using DDS and MQTT protocols by optimizing memory usage. Several methods have been used to reduce memory usage [14]. Some of these are examined as follows.

- Instead of assigning the initial values of the variables defined in the operating system, the assignment operator could not be performed and the automatic value assignment is realized as 0. Value assignment operations in the operating system are updated all the source codes in this way.
- The components in the Contiki operating system such as IPV6 and TCP and UDP features can be activated and deactivated by the predefined macros.
- The third solution is updating the MSP430 version. When the DDS application is run lower versions(4.6.3 and below), the binary file for an application cannot be compiled because of overflow on ROM and RAM.
- The fourth solution is made on the "ENERGEST" library, which is used to measure the energy use of the devices. When this library is disabled, resulting in a significant memory reduction in ROM and RAM usage. This library is activated or deactivated by defining a macro in the project config file.
- As a fifth solution, less RAM and ROM space have been achieved by

using the following commands on the compiler. For these values to work, the commands specified in the MSP430 file using the SMALL value in the makefile are implemented. By using the "#define TSCH LOG CONF LEVEL 0" macro, the log operations are closed and memory gain is achieved.

The changes mentioned in the examples have been used in applications required to transmit sensor data.

4.1 Result of Memory Optimization for DDS application

Since the Zigbee package will be used, the following commands must be updated on platform-conf.h or makefile. The size of the binary file is calculated using the SMALL = 0 command on the makefile, This means no optimization commands are used in the first step. In this case, ROM overflowed by 8362 bytes error causes the binary file not to be produced. Since the binary file could not be generated, SMALL = 1 value is set on the makefile to run optimization settings. With the execution of this command, the created binary file is optimized and memory-footprint is reduced. Changing parameter SMALL triggers the execution of the following commands for the MSP430 makefile under the Contiki CPU folder.

```
1 CFLAGS += -ffunction-sections
2 LDFLAGS += -Wl,--gc-sections,--undefined=_reset_vector__,
3 --undefined=InterruptVectors,--undefined=_copy_data_init__
4 ,--undefined=_clear_bss_init__,--undefined=_end_of_init__
5 \caption{Optimization flags for MSP430}
```

Other commands are used to reset the fields used on memory areas on the processor.

Table 4.1: The meaning of the MSP430 flags

Command	Effect
ffunctionsections	switch to tell the compiler not to mix functions that could be referenced externally
–gc-sections	switch telling the linker to do garbage collection and discard unreferenced routines

4.1.1 Memory Optimization for Changing Network Macros

When UIP_CONF_TCP and UIP_CONF_UDP are active since the modules specified in the DDS application are included, there is still an overflow issue on the memory area and no binary file is generated. When the values given above are set as 0 on changing Project-conf.h file, the problem of overflow in the memory space is resolved.
In this case, a total of 41617 bytes can be loaded onto the device using a binary file.

4.1.2 Changing Energest Module on XM1000 Mote

The Energest module is used to access the power information of the device. This module is modified via project-conf.h. To make this change, the command ENERGEST_CONF_ON 0 must be assigned. In the tests performed after the change, it was observed that a 40491-byte size binary file was generated.

4.1.3 Prevent From Initializing Variables and Printing String

Assigning the first value on the Contiki caused data to accumulate on the data area on the memory. If the initial value is not assigned, the data is marked as 0 and thus did not cause data accumulation. As a result of the modification of the entire operating system and the sample application, an

area of approximately 100 bytes was achieved. When the string values are written via the serial interface, there was an area loss of approximately 18 bytes for each value. During the development of the code, a large area acquisition was achieved by removing the codes intentionally for debugging the source code.

4.1.4 Disabling Process Name

Another memory space is obtained by not using the process names during the program start. The process name is only used to show the user and does not cause any problem in the execution of the code.
In the exemplary application, #define PROCESS_CONF_NO_PROCESS_NAMES has been set to 1, decreasing to software size 41345 bytes.

4.1.5 Configuration for Power Consumption

The size of the binary file used for calculating power consumption is 42699. In order to use the power calculation module, the Energest module shall be activated. The power calculation method and a string print function are called for this binary file. The compiler optimization flag is used by definition of SMALL=1 and process name is disabled by changing configuration file similar to 4.1.4.

4.1.6 Result Table

The changes and effect on the tailored DDS version are obviously seen in Table 4.2

Table 4.2: The Memory Usage of The Sample DDS Application

Change	Text	Data	Bss	Dec
Define SMALL= 0	Overflow	Overflow	Overflow	Overflow
Define SMALL= 1	38149	126	3342	41617
Define ENERGEST= 0	37109	126	3256	40491
DEFINE TSCH LOG CONF LEVEL 0	38149	126	3342	41617
Disable Print(4.1.3)	38097	126	3342	41565
Disable TCP and IPV4	38105	126	3340	41571
Disable Process c Name	37889	114	3342	41345
Configuration for Power Calculation	39217	125	3356	42699

4.2 Result of Memory Optimization for MQTT-SN application

Even though the above changes are made for the MQTT application, it could not be installed on Sky sensor node because of the size of the TCP packet. Similar to the DDS application, the device is loaded using the above meth-

ods to reduce the memory footprint. Also, the border-router application could not be installed on the device with default values. The installation was achieved by changing the memory areas in memory.x file(Appendix E) under the compiler folder. The changes for DDS are applied for MQTT-SN. Two situations are considered in this software when calculating the results. Firstly, it is calculated only for the case where the temperature sensor is activated, and secondly, it is calculated for the case where the temperature and humidity sensor are activated together. The memory size is queried by 'size' command.

4.2.1 Memory Optimization For Changing Network Macros

Since the MQTT-SN application provides communication over UDP, there is no need to use TCP protocol. The TCP macro value is initialized with value 0 for disabling this module. With the deactivation of TCP, a binary file of 48175 bytes was created. However, in the case of the temperature sensor is active, a binary file could not be generated due to overflow error on ROM.

4.2.2 Disabling Process Name

In the exemplary application, #define PROCESS_CONF_NO_PROCESS_NAMES has been set to 1, decreasing to software size 47933 bytes. However, in the case of the temperature sensor is active, a binary file could not be generated due to overflow error on ROM. But the amount of 250 bytes is saved by disabling process name.

4.2.3 Prevent From Initializing Variables and Printing String

When the change in title is made, 74 bytes of memory space gain is achieved on both applications. However, change doesn't allow the production of binary files when the state in which both temperature sensors are used.

4.2.4 Changing Energest Module on Sky Mote

In addition to the other changes, deactivation of the Energest module resulted in a large amount of memory gain. Thus, when temperature sensor and humidity sensors are used, a binary file containing 55903 bytes can be generated and uploaded to the device.

4.2.5 Enable Energest Module Without Self Calculation

Changes are done in sections 4.2.1, 4.2.2 and 4.2.3 are applied in this section. In addition to changes, self power consumption function is removed from the source code. The raw values are printed for CPU, LPM, TX, and RX and these values are given as input to the developed java application.

Table 4.3: The Memory Usage of The MQTT-SN Application for Only Temperature Sensor

Change	Text	Data	Bss	Dec
Define SMALL= 0	Overflow	Overflow	Overflow	Overflow
Define SMALL= 1	48175	264	7782	56221
Define PROCESS-NAME= 0	47933	248	7762	55943
Input defined in 4.2.5	47631	248	7778	55657

4.3 Result for Power Consumption

The power consumption of the MQTT and DDS applications is calculated in this section. The application and source code used in the DDS power consumption for the XM1000 sensor node is also used on the CM5000 sensor node by using the COOJA simulator. The power consumption is calculated for both XM1000 and CM5000 sensor nodes at 5, 10, 20, 30 and 60 seconds intervals. As a result of this calculation, the effect of the number of sensor

nodes on the energy consumption is seen. When the DDS application is run on the CM5000 via the simulator, the data sent are shared without being measured by the sensor, embedded in the code. Therefore, the battery life seems to be longer than the XM1000 sensor node.

4.3.1 Result for Tailored DDS on XM1000(Two Nodes)

The following table is created according to the data sent at 5 seconds intervals. The total column indicates the hourly power consumption in miniwatt.

Table 4.4: Average power consumption in microwatt for interval 5 seconds

TIME	CPU	LPM	TX	RX	Total
0	317127	0	0	0	317127
5	112776	3196	733106	2102263	2951341
10	160572	3046	314461	2532216	3010295
15	136825	3136	0	3594880	3734841
20	164258	3044	523465	2352524	3043291
25	187812	2966	688502	2563886	3443166
30	160246	3057	748081	2422060	3333444
35	162844	3049	553732	2485744	3205369
40	159708	3059	164399	1995205	2322371
45	161080	3055	688820	4428625	5281580
50	160729	3056	464205	2407258	3035248
55	160204	3057	583681	2589704	3336646
60	161159	3055	329436	2090214	2583864
65	118541	3196	389652	2225500	2736889
70	159461	3050	658872	2853735	3675118
75	159521	3060	419282	4104697	4686560
80	164367	3044	688502	3327063	4182976
85	137726	3132	0	4466835	4607693
90	164210	3045	478861	4786633	5432749
95	188670	2963	883168	5714355	6789156

Table 4.4 continued from previous page

TIME	CPU	LPM	TX	RX	Total
100	159364	3060	344091	6275119	6781634
105	204785	3043	284194	3112602	3604624
110	159515	3096	479179	2305363	2947153
115	159600	3060	284194	2058545	2505399
120	159243	3060	284194	2248220	2694717
125	159781	3060	268582	2254760	2686183
130	159581	3060	44604	2137031	2344276
135	159013	3061	269219	2410700	2841993
140	160198	3058	254564	2712597	3130417
145	136064	3137	0	2200371	2339572
150	189196	2962	1212604	2312248	3717010
155	159877	3058	449230	2493317	3105482
160	160294	3058	703157	2406569	3273078
165	160186	3057	718450	2482646	3364339
170	160319	3057	448912	2180405	2792693
175	161841	3052	119476	2320854	2605223
180	160143	3058	718132	2339787	3221120
185	160355	3058	643897	2413798	3221108
190	136433	3136	0	2167668	2307237
195	189111	2962	1107465	2504677	3804215
200	137484	3133	0	2457861	2598478
205	187866	3023	747762	2537724	3476375
210	161370	3053	718450	2516037	3398910
215	137901	3132	0	2487810	2628843
220	161467	3053	284194	2534970	2983684
225	164440	3044	418645	2366982	2953111
230	163672	3046	568707	2456828	3192253
235	188458	2963	1002645	2402094	3596160
240	160566	3057	658872	2381440	3203935
245	160337	3057	404307	2443059	3010760

Table 4.4 continued from previous page

TIME	CPU	LPM	TX	RX	Total
250	136330	3138	0	2426191	2565659
255	136439	3137	0	2627915	2767491
260	191667	2953	583363	2788329	3566312
265	135593	3139	0	2186601	2325333
270	137345	3135	0	2716384	2856864
275	139925	3126	0	2673354	2816405
280	137629	3133	0	2683337	2824099
285	140312	3125	0	2491252	2634689
290	220725	2856	1031319	2708811	3963711
295	164331	3044	568707	3901940	4638022
300	165557	3039	553095	2706057	3427748
305	141200	3121	0	2511906	2656227
310	168089	3031	643579	2565952	3380651
315	164210	3044	388696	2402094	2958044
320	194090	2945	897506	2661650	3756191
325	164162	3044	433938	2576623	3177767
330	216937	2869	1959730	2606916	4786452
335	137049	3134	0	2634111	2774294
340	164053	3046	702839	2330493	3200431
345	216755	2869	1151751	2790051	4161426
350	137013	3135	0	2838933	2979081
355	138245	3131	0	2549772	2691148
360	165515	3040	373721	2554936	3097212
365	163606	3047	568707	2422749	3158109
370	164645	3042	329117	2463024	2959828
375	214870	2876	1331125	2546330	4095201
380	160300	3057	703476	2434797	3301630
385	161872	3052	479179	2498825	3142928
390	161116	3054	613948	2399685	3177803
395	160609	3056	523784	2308806	2996255

Table 4.4 continued from previous page

TIME	CPU	LPM	TX	RX	Total
400	136505	3136	0	2445124	2584765
405	164693	3043	747762	2377998	3293496
410	189002	2962	1137095	2294692	3623751
415	160355	3057	763692	2475073	3402177
420	160276	3058	509128	2359753	3032215
425	163098	3048	657916	2540134	3364196
430	160524	3057	747762	2081953	2993296
435	163835	3045	299487	2276792	2743159
440	162373	3071	718450	2064052	2947946
445	160911	3056	718450	2432731	3315148
450	161291	3054	194666	2137719	2496730
Average(nanowatt)	163289,5	3021,8	473405,9	2620432,1	3260149,4

The following table is created according to the data sent at 10 seconds intervals.

Table 4.5: Average power consumption in microwatt for interval 10 seconds

TIME	CPU	LPM	TX	RX	Total
0	157744	0	0	0	157744
10	67002	3210	0	2551494	2621706
20	68367	3213	0	2750119	2821699
30	61010	3239	127122	2465606	2656977
40	92655	3129	246599	2704163	3046546
50	79942	3176	0	2560272	2643390
60	79105	3180	247077	2713458	3042820
70	79080	3179	97333	2565091	2744683
80	82869	3167	119794	2458721	2664551

Table 4.5 continued from previous page

TIME	CPU	LPM	TX	RX	Total
90	79165	3179	127282	2501579	2711205
100	68869	3214	0	2917247	2989330
110	68832	3213	0	2868365	2940410
120	69890	3210	0	2858382	2931482
130	70727	3207	0	3133256	3207190
140	70594	3208	0	2716039	2789841
150	70539	3208	0	2848743	2922490
160	83201	3166	22302	2852186	2960855
170	71422	3205	0	2785748	2860375
180	71826	3204	0	2858038	2933068
190	70388	3208	0	2773527	2847123
200	71298	3206	0	2883167	2957671
210	73083	3199	0	2790223	2866505
220	74839	3194	0	2816557	2894590
230	128457	3015	403511	2598998	3133981
240	84806	3160	82199	3207956	3378121
250	71068	3206	0	2807779	2882053
260	113764	3064	732947	2627570	3477345
270	83495	3164	172364	2594523	2853546
280	72080	3203	0	2828433	2903716
290	71213	3205	0	2566640	2641058
300	137717	2984	792685	2548912	3482298
310	81138	3172	321630	2527741	2933681
320	68709	3214	0	2828261	2900184
330	68905	3213	0	2779379	2851497
340	70530	3208	0	2758553	2832291
350	71905	3203	0	2557862	2632970
360	70757	3207	0	2884544	2958508
370	83570	3164	149584	2850809	3087127
380	84827	3160	74553	2802960	2965500

Table 4.5 continued from previous page

TIME	CPU	LPM	TX	RX	Total
390	71056	3206	0	2775249	2849511
400	71316	3206	0	2795386	2869908
410	71926	3203	0	2740308	2815437
420	72530	3202	0	2820860	2896592
430	73331	3198	0	2781789	2858318
440	85120	3160	127122	2826540	3041942
450	73455	3198	0	2860447	2937100
Average(nanowatt)	79654,8	3113,7	83567,5	2690075,7	2856411,6

The following table is created according to the data sent at 20 seconds intervals.

Table 4.6: Average power consumption in microwatt for interval 20 seconds

TIME	CPU	LPM	TX	RX	Total
0	79283	0	0	0	79283
20	39270	3237	183356	1344594	1570457
40	40647	3237	179612	1340722	1564218
60	40187	3239	179612	1314559	1537597
80	40082	3239	100997	1583926	1728244
100	40395	3238	175789	1179532	1398954
120	40212	3238	14974	1153198	1211622
140	40230	3238	190923	1408536	1642927
160	40194	3239	119794	1345025	1508252
180	40194	3238	145920	1177897	1367249
200	40113	3239	179692	1263268	1486312
220	40383	3238	127202	1638402	1809225
240	29244	3275	149664	1138740	1320923
260	40182	3236	93510	1209911	1346839
280	40208	3239	123458	1217484	1384389

Table 4.6 continued from previous page

TIME	CPU	LPM	TX	RX	Total
300	40061	3239	138353	1290893	1472546
320	40176	3239	142176	1352770	1538361
340	40401	3238	142176	1478675	1664490
360	40149	3239	93510	1344680	1481578
380	40245	3239	3743	1214214	1261441
400	40082	3239	187099	1232459	1462879
420	40250	3239	179692	1436420	1659601
440	40674	3237	116130	1314559	1474600
Average(nanowatt)	41428,8	3099,1	129016,6	1260020,2	1433564,7

The following table is created according to the data sent at 30 seconds intervals.

Table 4.7: Average power consumption in microwatt for interval 30 seconds

TIME	CPU	LPM	TX	RX	Total
0	52852	0	0	0	52852
30	26175	3258	119794	1260198	1409425
60	27533	3256	99829	1170410	1301028
90	26847	3259	122237	1344078	1496421
120	26753	3259	119794	1063466	1213272
150	26809	3259	74871	1173680	1278619
180	26745	3259	79863	1067368	1177235
210	27519	3256	77314	1318145	1426234
240	26827	3259	94678	1123077	1247841
270	26862	3259	84695	1069720	1184536
300	27316	3257	99829	1147231	1277633
330	26754	3259	119688	1084006	1233707
360	26867	3259	39931	1084407	1154464
390	27332	3257	87350	1022502	1140441

Table 4.7 continued from previous page

TIME	CPU	LPM	TX	RX	Total
420	26836	3259	102324	1205694	1338113
450	27368	3257	77314	1140805	1248744
Average(nanowatt)	28587,2	3054,5	87469,4	1079674,2	1198785,3

These values below are calculated for period of 60 seconds.

Table 4.8: Average power consumption in microwatt for interval 60 seconds

TIME	CPU	LPM	TX	RX	Total
0	26427	0	0	0	26427
60	13052	3279	31196	1013035	1060562
120	13495	3279	57348	976259	1050381
180	13498	3279	57401	1252855	1327033
240	13517	3279	49914	1007843	1074553
300	14059	3277	59844	925083	1002263
360	13462	3279	59897	870693	947331
420	13500	3279	52383	928668	997830
Average(nanowatt)	15126,2	2868,9	45997,9	871804,5	935797,5

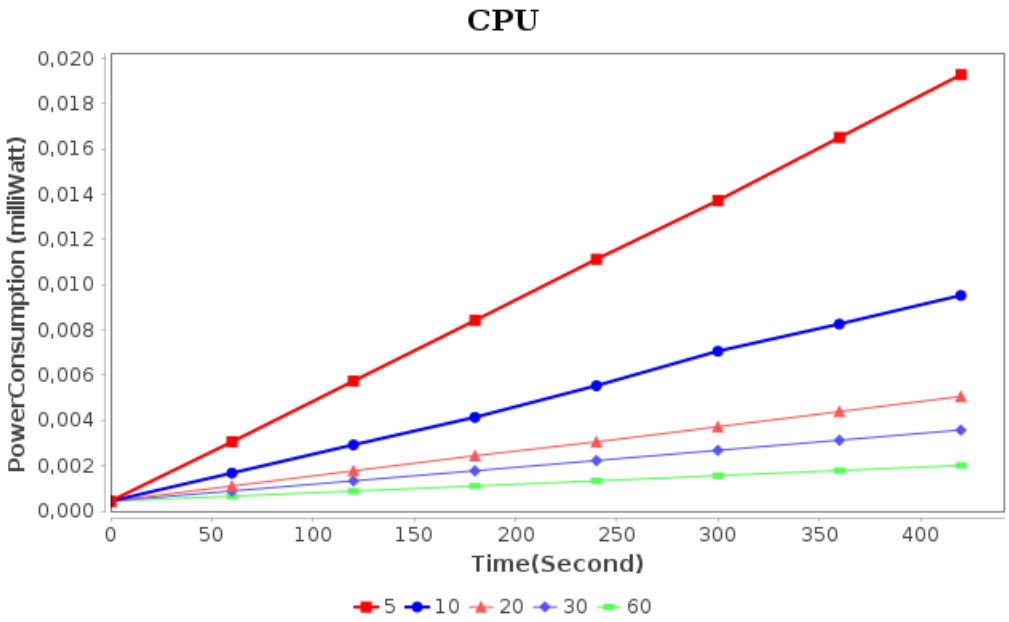

Figure 4.1: The total power consumption in milliwatt in first 400 seconds for CPU

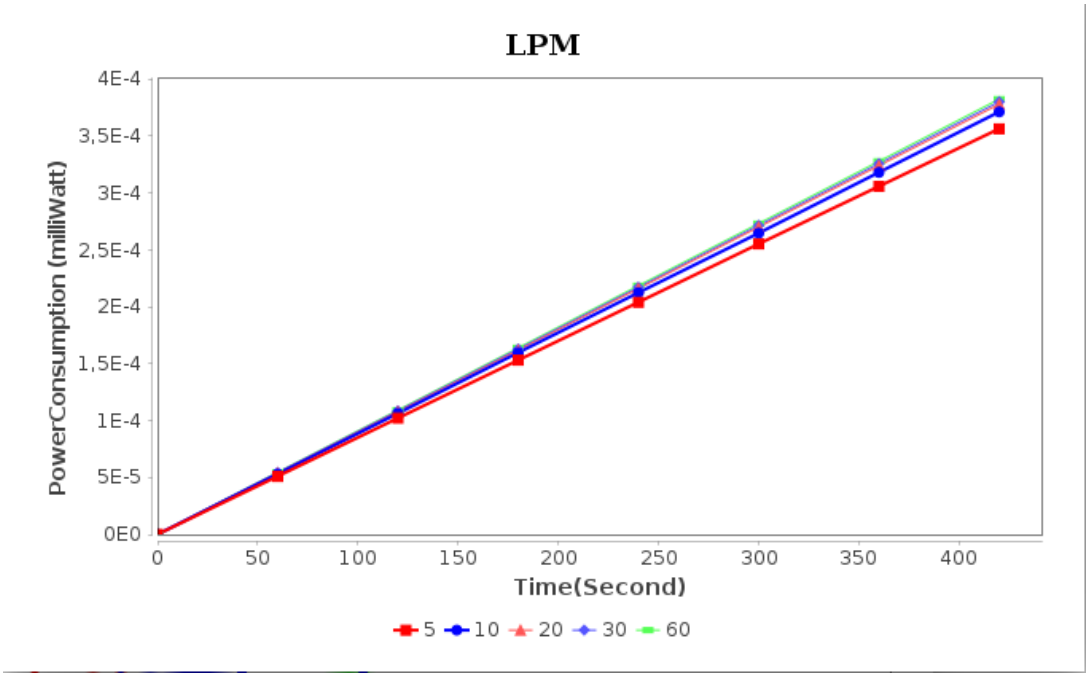

Figure 4.2: The total power consumption in milliwatt in first 400 seconds for LPM

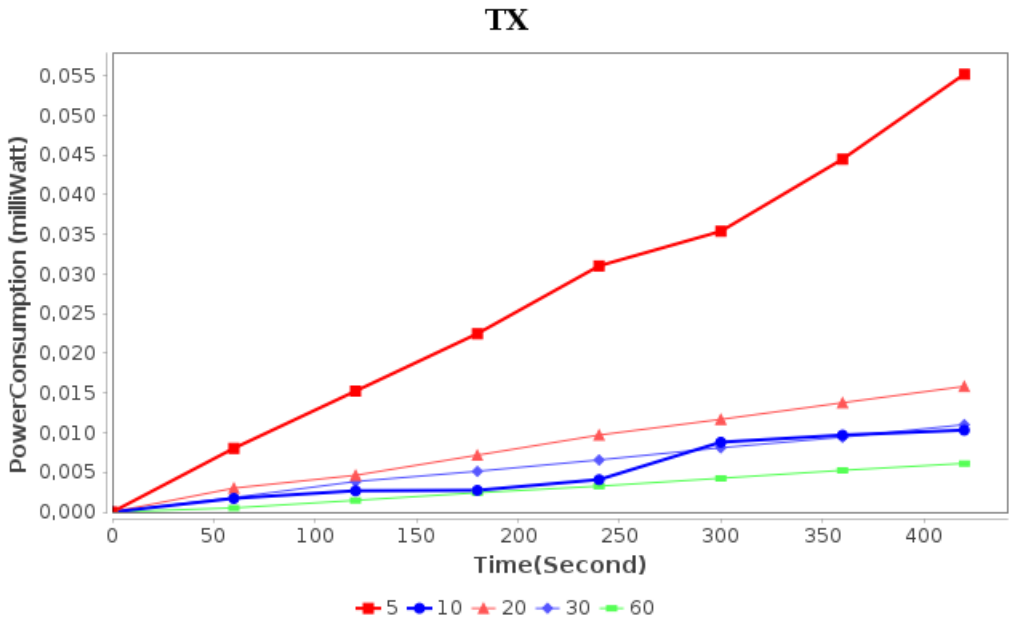

Figure 4.3: The total power consumption in milliwatt in first 400 seconds for TX

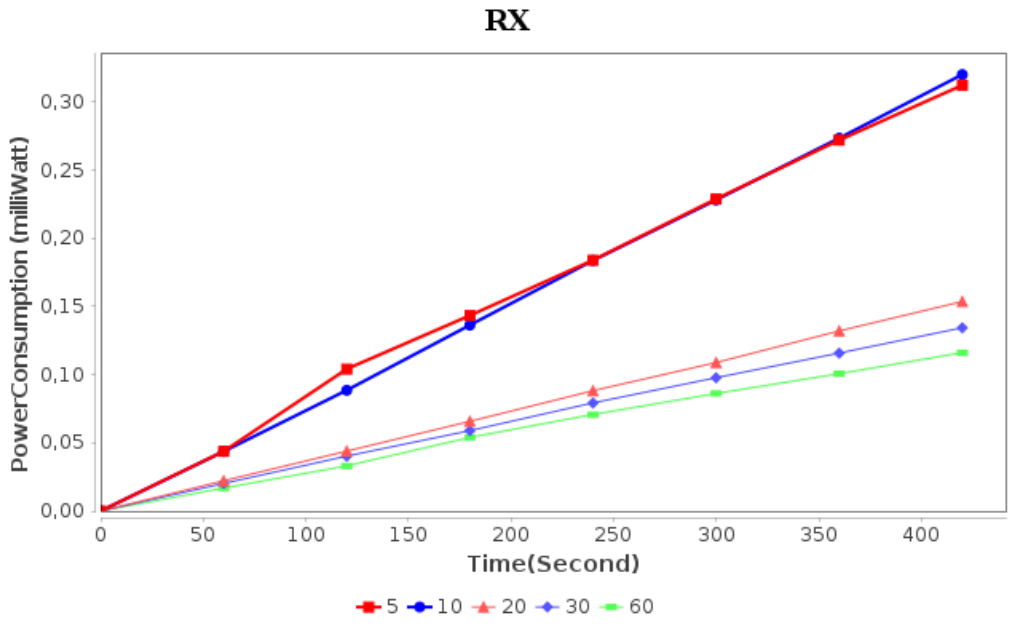

Figure 4.4: The total power consumption in milliwatt in first 400 seconds for TX

Table 4.9: Expected life time for DDS application(XM1000 Two Nodes)

Time(Interval)	Estimated Life Time(year)
5	0.26491
10	0.307197
20	0.631267
30	0.773935
60	1.08155

4.3.2 Result for Tailored DDS on CM5000(Two Nodes)

The power consumption for CM5000 could be seen in the figures below.

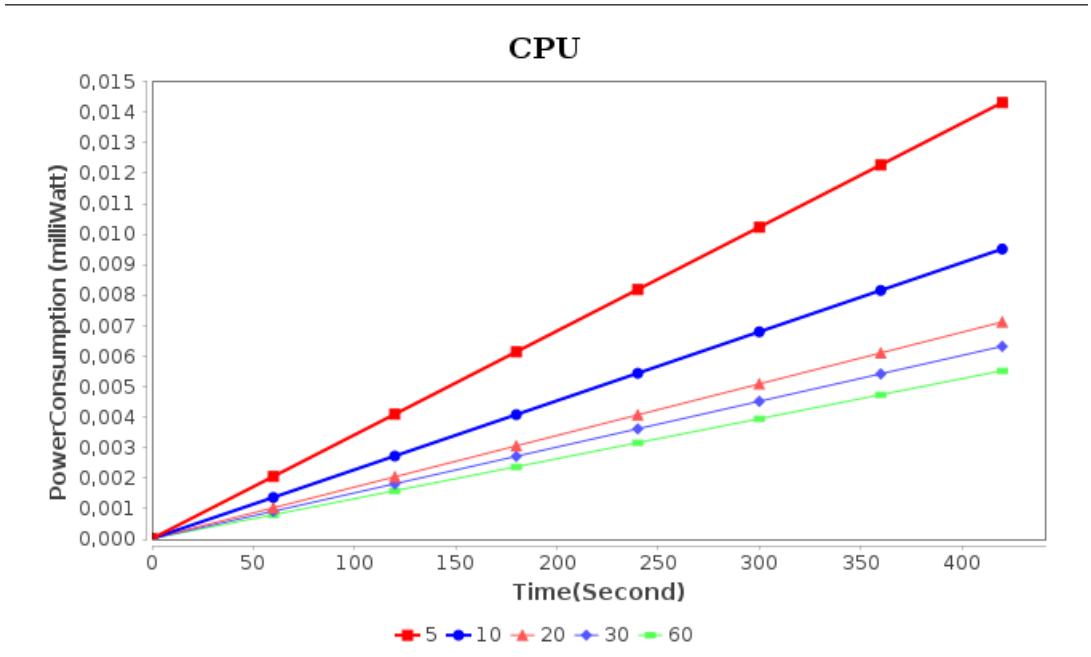

Figure 4.5: The total power consumption in milliwatt in first 450 seconds for CPU

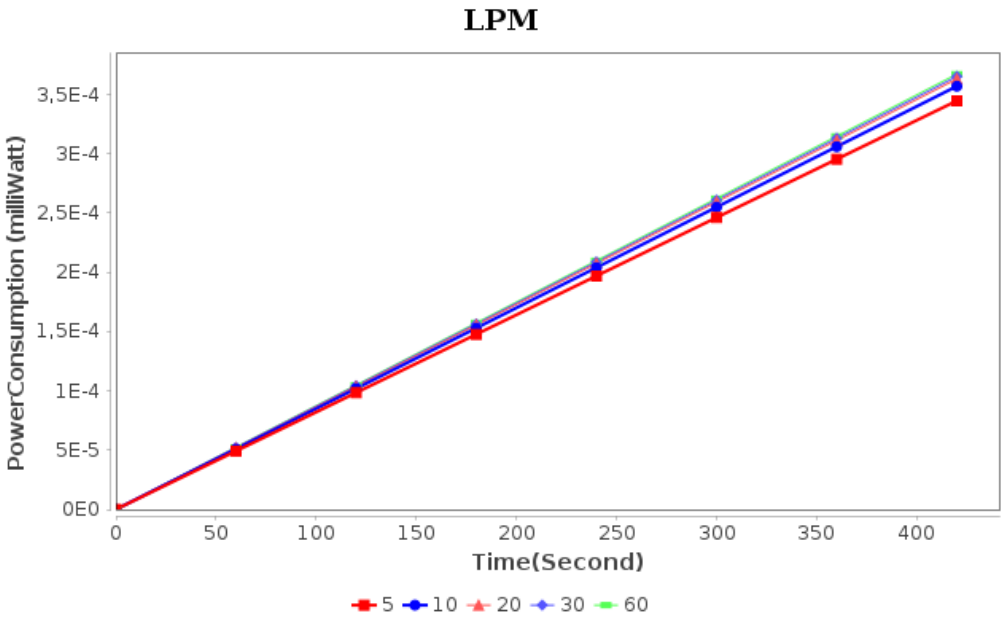

Figure 4.6: The total power consumption in milliwatt in first 450 seconds for LPM

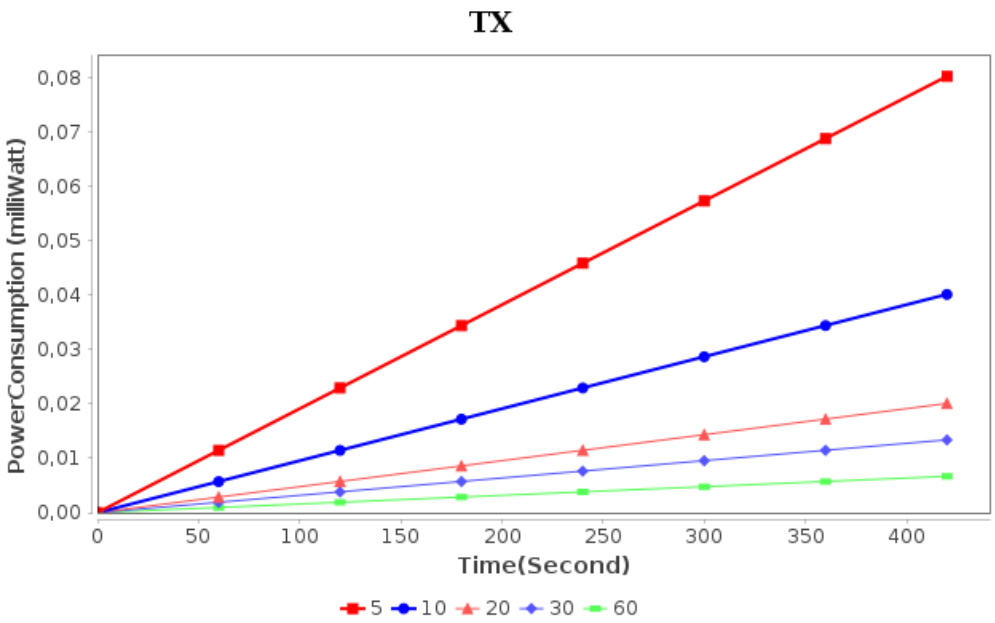

Figure 4.7: The total power consumption in milliwatt in first 450 seconds for TX

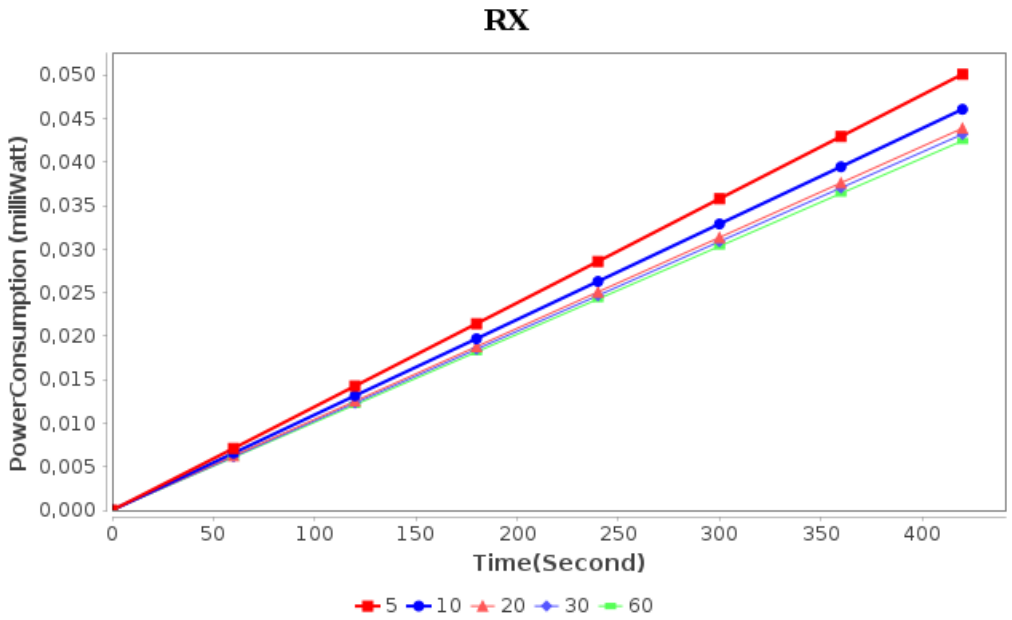

Figure 4.8: The total power consumption in milliwatt in first 450 seconds for RX

Table 4.10: Expected life time for CM5000(Two Nodes)

Time(Interval)	**Estimated Life Time(year)**
5	0.704014
10	1.087074
20	1.537222
30	1.820367
60	2.539766

4.3.3 Result for Tailored DDS on CM5000(Three nodes)

The power consumption for CM5000 (three nodes) could be seen in the figures below.

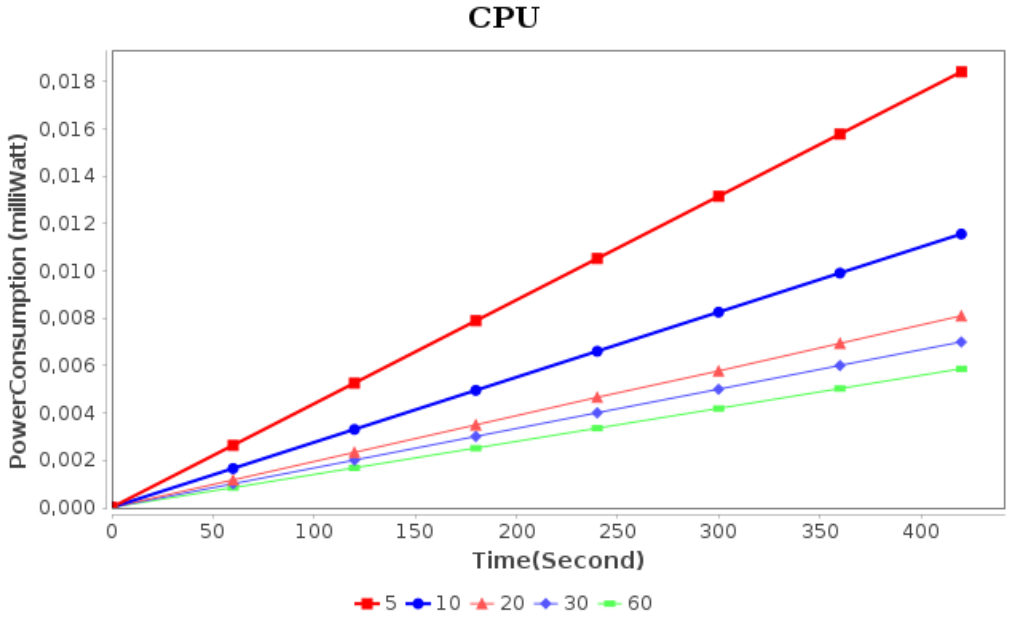

Figure 4.9: The total power consumption in milliwatt in first 450 seconds for CPU

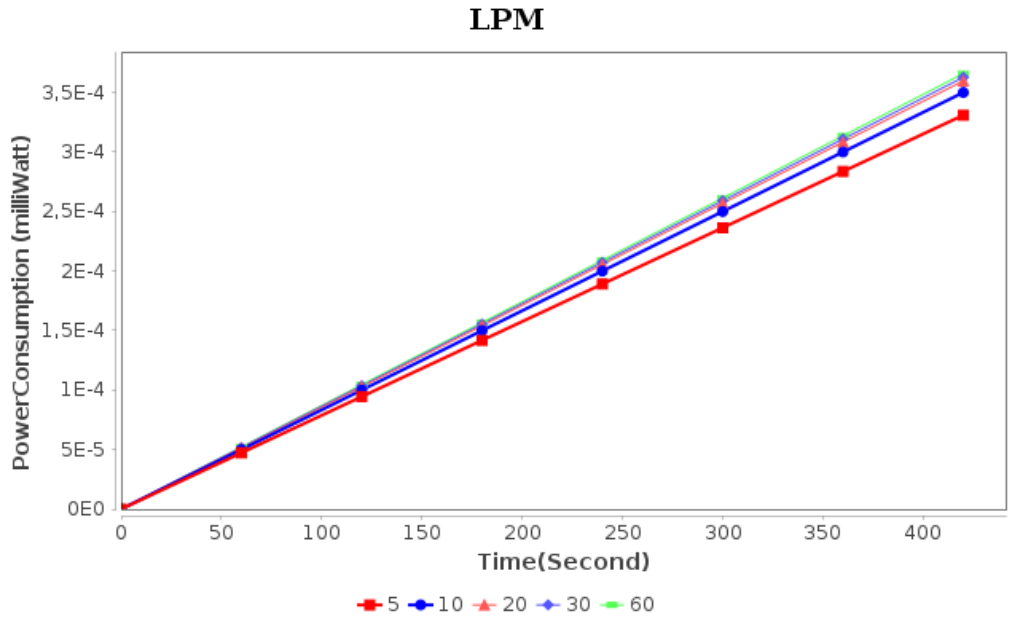

Figure 4.10: The total power consumption in milliwatt in first 450 seconds for LPM

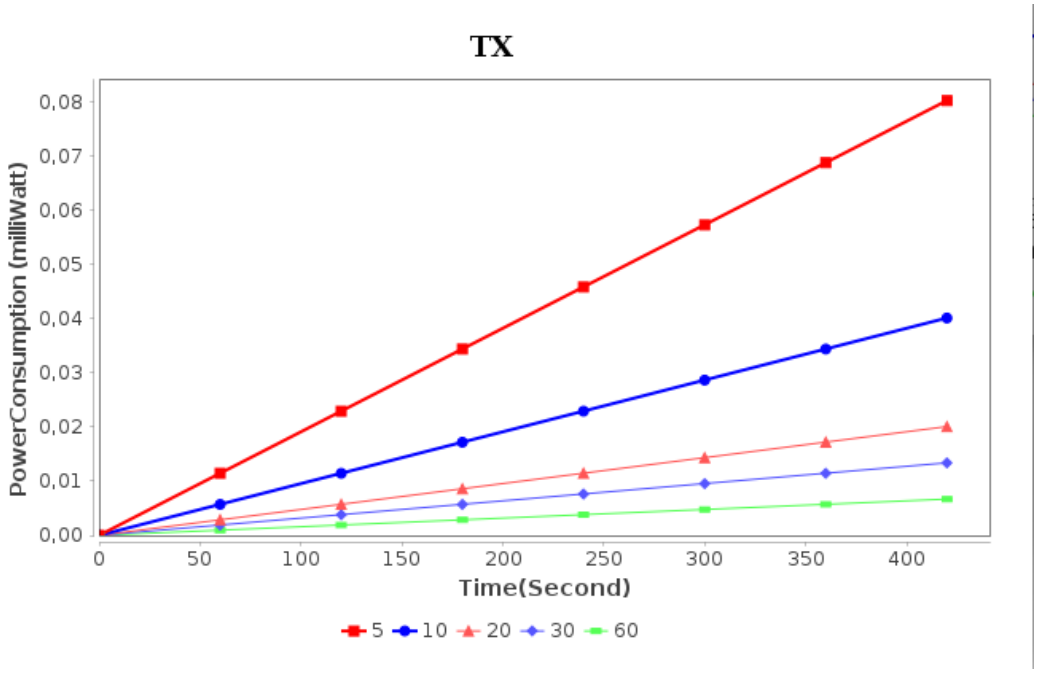

Figure 4.11: The total power consumption in milliwatt in first 450 seconds for TX

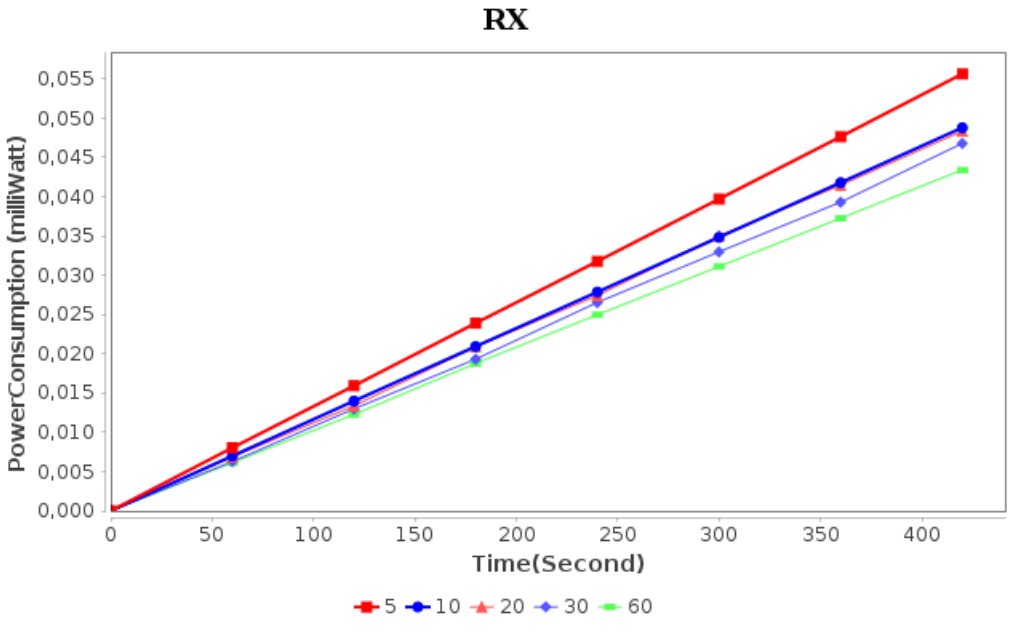

Figure 4.12: The total power consumption in milliwatt in first 450 seconds for RX

Table 4.11: Expected life time for CM5000(Three Nodes)

Time(Interval)	**Estimated Life Time(year)**
5	0.660331
10	1.036143
20	1.425846
30	1.707206
60	2.475717

Since the chemical life of the battery is at most 5 years, the maximum duration can be assumed to be 5 years. With the increase in the number of sensors, energy consumption increases by 10 percent. When the same test is applied with a topology that contains thirteen CM5000 sensor nodes, energy consumption increases by nearly 200 percent.

4.3.4 Result for MQTT

Table 4.12: Average power consumption for MQTT in milliwatt

Time(Second)	**CPU**	**LPM**	**TX**	**RX**	**Total(Nanowatt)**
0	58729	3190	0	1660261	1722180
10	58729	3190	0	749578	811497
20	58666	3190	0	1881434	1943290
30	58971	3189	0	2872668	2934828
40	59044	3191	0	1173852	1236087
50	58738	3190	0	1423941	1485869
60	58787	3190	0	1552514	1614491
70	58811	3190	0	1804841	1866842
Average(nw)	58809.375	3190	0	1639886.125	1701885.5
Average(mw)	0.058809375	0.00319	0	1.639886125	1.7018855

According to formula 41, the estimated battery life is recalculated for MQTT. The battery life is calculated as approximately 0.5 years. The data sent to the python application via the serial interface requires less power than the data sent via the Zigbee in the same period. Because of the Zigbee module is not used in this test, therefore the effect of the node count and data send a period on power consumption is not calculated

4.3.5 Result for MQTT-SN

In order to activate the energest module and calculate the power consumption, the methods used in other applications were added to MQTT-SN. Although the memory reduction methods described in the first part of the conclusion are tried, all the methods are failed. Therefore, no measurement is performed for MQTT-SN, because of the lack of memory. If the current and voltage calculations are removed from the binary file and these calculations are performed via the java application. According to the results below, as the data transmission period increases, energy consumption decreases slightly. In addition, when the number of sensor nodes increases, energy consumption also increases in direct proportion.

4.3.6 Result for Two Nodes MQTT-SN Communication

The communication in this example is provided by two sensor nodes and the RSMB server.

The following table is created according to the data sent at 5 seconds intervals.

Table 4.13: Average power consumption in microwatt for interval 5 seconds

TIME	CPU	LPM	TX	RX	Total
0	25396	3261	0	57015840	57044497
5	24925	3216	119794	56268153	56416088

Table 4.13 continued from previous page

TIME	CPU	LPM	TX	RX	Total
10	30864	3197	147832	56237516	56419409
15	27656	3207	123618	56263334	56417815
20	25456	3215	91757	56298099	56418527
25	28369	3210	113104	56363507	56508190
30	22919	3223	93032	56296729	56415903
35	21064	3229	60853	56332870	56418016
40	21203	3229	60534	56331833	56416799
45	21172	3229	60534	56332185	56417120
50	21070	3229	60853	56332870	56418022
55	25946	3213	93032	56296729	56418920
60	23728	3220	86341	56304294	56417583
65	21064	3229	60853	56332870	56418016
70	21209	3229	60534	56331833	56416805
75	21172	3229	60534	56332185	56417120
80	21064	3229	60853	56332870	56418016
85	21203	3229	60534	56331833	56416799
90	21209	3229	60534	56332185	56417157
95	21064	3229	60853	56332870	56418016
100	21203	3229	60534	56331833	56416799
105	21172	3229	60534	56332185	56417120
110	21064	3229	60853	56332870	56418016
115	21203	3229	60534	56331833	56416799
120	25795	3213	86022	56304985	56420015
125	22465	3225	93032	56297068	56415790
130	21493	3228	60853	56332185	56417759
135	21172	3229	60534	56332185	56417120
140	21064	3229	60853	56332870	56418016
145	21203	3229	60534	56331833	56416799
150	21233	3229	60534	56332185	56417181
155	23124	3222	94625	56295347	56416318

Table 4.13 continued from previous page

TIME	CPU	LPM	TX	RX	Total
160	21203	3229	60534	56332185	56417151
165	21172	3229	60534	56332185	56417120
170	21064	3229	60853	56332870	56418016
175	21209	3229	60534	56331833	56416805
180	21245	3229	60534	56332185	56417193
185	21070	3229	60853	56332870	56418022
190	21203	3229	60534	56331833	56416799
195	21178	3229	60534	56332185	56417126
200	24327	3218	93032	56297414	56417991
205	23468	3221	86022	56304646	56417357
210	21263	3229	60534	56332185	56417211
215	21064	3229	60853	56332870	56418016
220	21203	3229	60534	56331833	56416799
225	21178	3229	60534	56332185	56417126
230	23662	3221	94625	56295347	56416855
235	21209	3229	60534	56332185	56417157
240	21269	3229	60534	56332185	56417217
245	21064	3229	60853	56332870	56418016
250	21203	3229	60534	56331833	56416799
255	21172	3229	60534	56332185	56417120
260	21354	3228	60534	56332524	56417640
265	21209	3229	60534	56332185	56417157
270	21281	3228	60534	56332185	56417228
275	21064	3229	60853	56332870	56418016
280	21203	3229	60534	56331833	56416799
285	21172	3229	60534	56332185	56417120
290	21064	3229	60853	56333216	56418362
295	21203	3229	60534	56331833	56416799
300	21293	3228	60534	56332185	56417240
305	23535	3221	94943	56295347	56417046

Table 4.13 continued from previous page

TIME	CPU	LPM	TX	RX	Total
310	21209	3229	60534	56332185	56417157
315	21178	3229	60534	56332185	56417126
320	21064	3229	60853	56332870	56418016
325	21203	3229	60534	56331833	56416799
330	21299	3228	60534	56332185	56417246
335	21064	3229	60853	56332870	56418016
340	21203	3229	60534	56331833	56416799
345	21172	3229	60534	56332185	56417120
350	21064	3229	60853	56332870	56418016
355	21203	3229	60534	56331833	56416799
360	21311	3228	60534	56332185	56417258
365	21064	3229	60853	56332870	56418016
370	21209	3229	60534	56331833	56416805
375	21172	3229	60534	56332185	56417120
380	21064	3229	60853	56332870	56418016
385	21203	3229	60534	56331833	56416799
390	21335	3228	60534	56332185	56417282
395	21064	3229	60853	56332870	56418016
400	21209	3229	60534	56331833	56416805
405	21172	3229	60534	56332185	56417120
410	21064	3229	60853	56332870	56418016
415	21203	3229	60534	56331833	56416799
420	21342	3228	60534	56332185	56417289
425	21064	3229	60853	56332870	56418016
430	21203	3229	60534	56331833	56416799
435	22635	3224	93350	56297068	56416277
440	21070	3229	60853	56332870	56418022
445	21203	3229	60534	56331833	56416799
450	21348	3228	60534	56332185	56417295
Average(nanowatt)	21882,9	3227,1	66934,2	56333203,0	56425247,2

The following table is created according to the data sent at 10 seconds intervals.

Table 4.14: Average power consumption in microwatt for interval 10 seconds

TIME	CPU	LPM	TX	RX	Total
0	21749	3250	59897	56642860	56727756
10	26106	3212	135884	56251289	56416491
20	23269	3224	102271	56331667	56460431
30	17505	3241	61649	56331667	56414062
40	16710	3244	45560	56349395	56414909
50	18725	3237	61968	56331667	56415597
60	17909	3240	58463	56335795	56415407
70	16704	3244	45560	56349395	56414903
80	16647	3244	45560	56349395	56414846
90	16656	3244	45719	56349734	56415353
100	17012	3243	45560	56349222	56415037
110	16350	3245	45560	56349395	56414550
120	19650	3234	74712	56318246	56415842
130	16849	3243	45719	56349222	56415033
140	16650	3244	45560	56349395	56414849
150	17695	3240	62764	56331155	56414854
160	16707	3244	45560	56349395	56414906
170	16650	3244	45560	56349395	56414849
180	16671	3244	45719	56349734	56415368
190	16704	3244	45560	56349222	56414730
200	19408	3235	74871	56317894	56415408
210	16677	3244	45719	56349734	56415374
220	16704	3244	45560	56349222	56414730
230	17955	3240	62764	56330630	56414589
240	16686	3244	45719	56349734	56415383
250	16707	3244	45560	56349222	56414733
260	16795	3243	45719	56349222	56414979

Table 4.14 continued from previous page

TIME	CPU	LPM	TX	RX	Total
270	16692	3244	45719	56349734	56415389
280	16704	3244	45560	56349222	56414730
290	16650	3244	45560	56349395	56414849
300	17931	3240	62924	56331155	56415250
310	16704	3244	45560	56349395	56414903
320	16650	3244	45560	56349395	56414849
330	16704	3244	45719	56349734	56415401
340	16704	3244	45560	56349222	56414730
350	16650	3244	45560	56349395	56414849
360	17006	3243	45719	56349734	56415702
370	16408	3245	45560	56349222	56414435
380	16650	3244	45560	56349395	56414849
390	16716	3244	45719	56349734	56415413
400	16704	3244	45560	56349222	56414730
410	16653	3244	45560	56349395	56414852
420	16719	3244	45719	56349734	56415416
430	17432	3241	62127	56331833	56414633
440	16647	3244	45560	56349395	56414846
450	16725	3244	45719	56349734	56415422
Average(nanowatt)	17454,3	3241,9	52853,1	56349281,5	56422830,8

The following table is created according to the data sent at 20 seconds intervals.

Table 4.15: Average power consumption in microwatt for interval 20 seconds

TIME	CPU	LPM	TX	RX	Total
0	22068	3238	97890	56447673	56570869
20	18169	3240	74632	56340358	56436399
40	15497	3248	46277	56348960	56413982

Table 4.15 continued from previous page

TIME	CPU	LPM	TX	RX	Total(Hourly)
60	15092	3249	44604	56350944	56413889
80	14412	3251	38152	56358003	56413818
100	14449	3251	38073	56357747	56413520
120	16017	3246	52808	56342252	56414323
140	14943	3250	46675	56348704	56413572
160	14452	3251	38073	56357830	56413606
180	14458	3251	38073	56357830	56413612
200	15801	3247	52649	56342163	56413860
220	15113	3249	46755	56348620	56413737
240	14480	3251	38073	56357830	56413634
260	14500	3251	38073	56357830	56413654
280	14450	3251	38073	56357830	56413604
300	15091	3249	46834	56348620	56413794
320	14433	3251	38152	56358003	56413839
340	14449	3251	38073	56357747	56413520
360	14479	3251	38073	56357830	56413633
380	14440	3251	38152	56358003	56413846
400	14452	3251	38073	56357747	56413523
420	14844	3250	46356	56349139	56413589
440	14446	3251	38152	56358003	56413852
Average(nanowatt)	15240,7	3249,1	45684,6	56357376,8	56421551,1

The following table is created according to the data sent at 30 seconds intervals.

Table 4.16: Average power consumption in microwatt for interval 30 seconds

TIME	CPU	LPM	TX	RX	Total
0	21878	3234	103439	56389845	56518396
30	16543	3244	59472	56334993	56414252

Table 4.16 continued from previous page

TIME	CPU	LPM	TX	RX	Total
60	16715	3244	59950	56334190	56414099
90	15063	3250	50923	56358805	56428041
120	14974	3250	45666	56349853	56413743
150	13711	3254	35577	56360640	56413182
180	14627	3251	45400	56350254	56413532
210	14135	3252	41365	56354500	56413252
240	13765	3254	35683	56360695	56413397
270	13719	3254	35577	56360640	56413190
300	13720	3254	35577	56360640	56413191
330	13724	3254	35577	56360640	56413195
360	14158	3252	41418	56354500	56413328
390	13727	3254	35577	56360640	56413198
420	14151	3252	41365	56354500	56413268
450	13731	3254	35577	56360640	56413202
Average(nanowatt)	14896,3	3250,4	46133,9	56356623,4	56420904,1

The following table is created according to the data sent at 60 seconds intervals.

Table 4.17: Average power consumption in microwatt for interval 60 seconds

TIME	CPU	LPM	TX	RX	Total
0	17227	3246	70517	56381896	56472886
60	13252	3255	38152	56358203	56412862
120	13608	3254	38285	56358058	56413205
180	13508	3254	38152	56358233	56413147
240	13179	3256	36188	56360409	56413032
300	12994	3256	33294	56363451	56412995
360	13119	3256	36028	56360554	56412957
420	13169	3256	36161	56360379	56412965

Table 4.17 continued from previous page

TIME	CPU	LPM	TX	RX	Total
Average(nanowatt)	13757	3254,1	40847,1	56362647,9	56420506,1

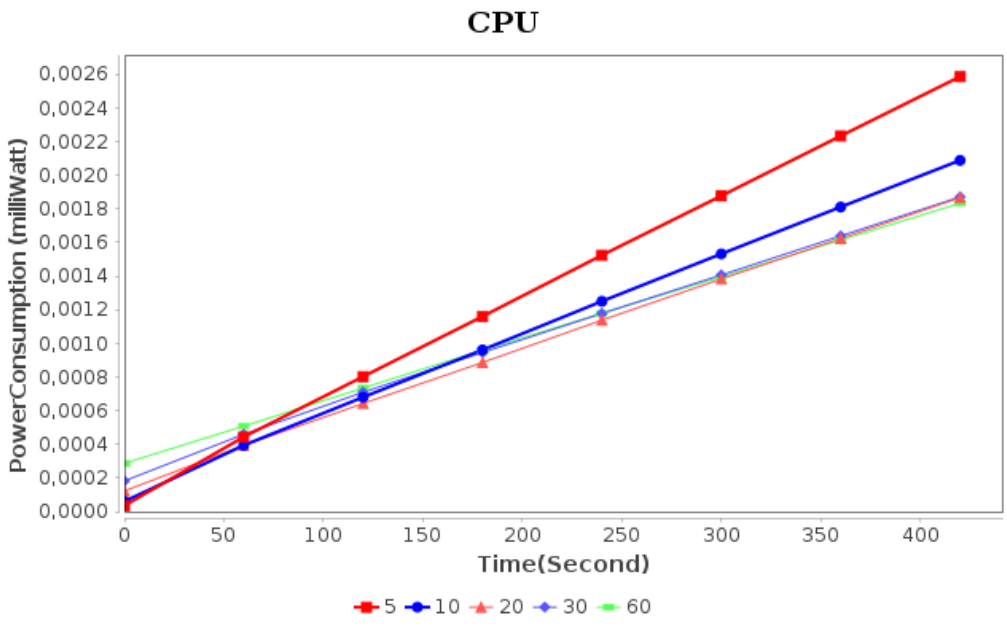

Figure 4.13: The total power consumption in milliwatt in first 450 seconds for CPU

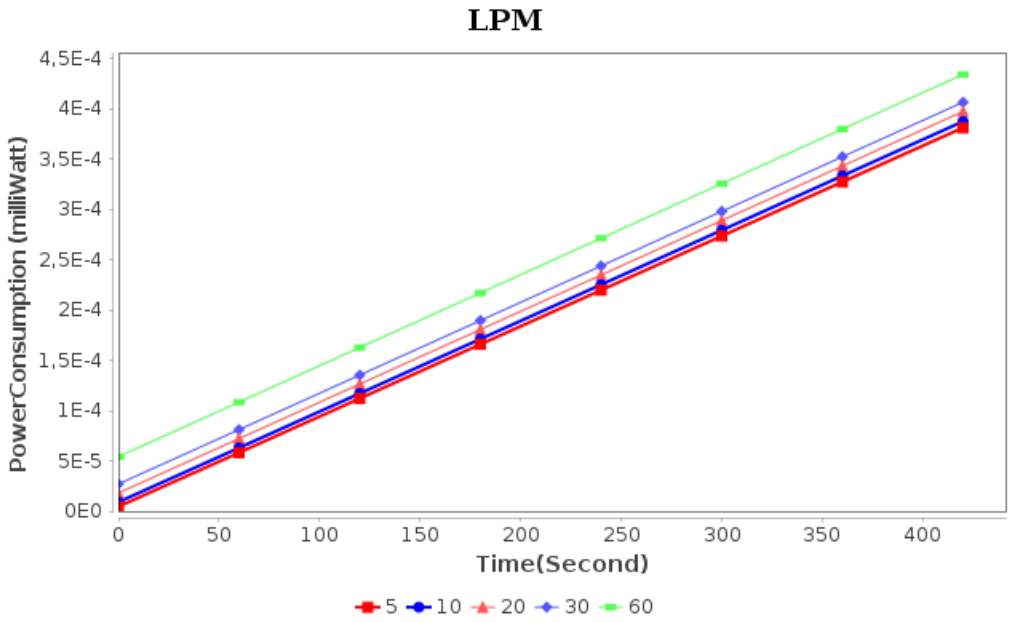

Figure 4.14: The total power consumption in milliwatt in first 450 seconds for LPM

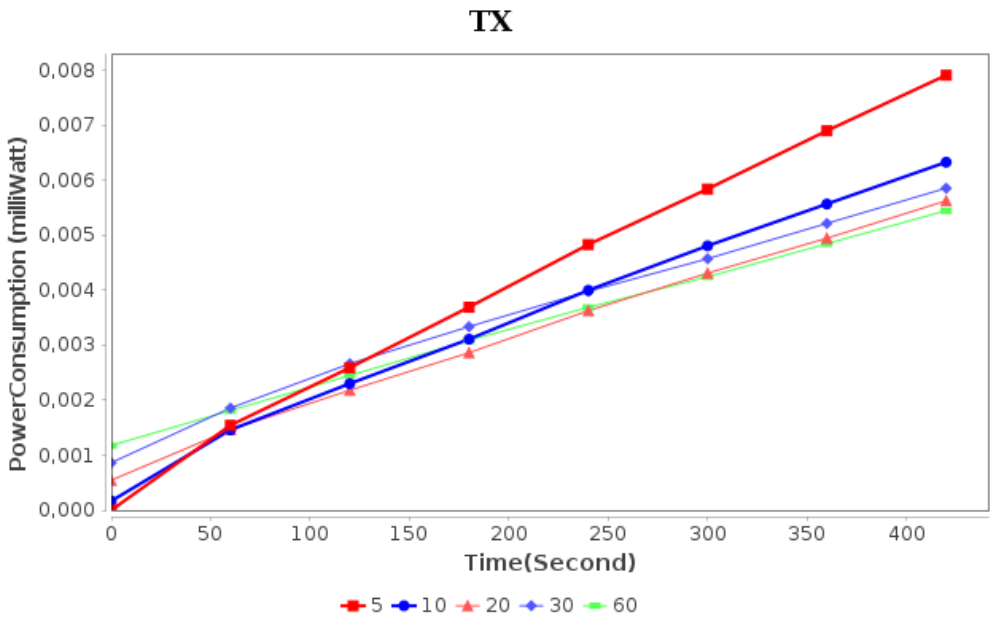

Figure 4.15: The total power consumption in milliwatt in first 450 seconds for TX

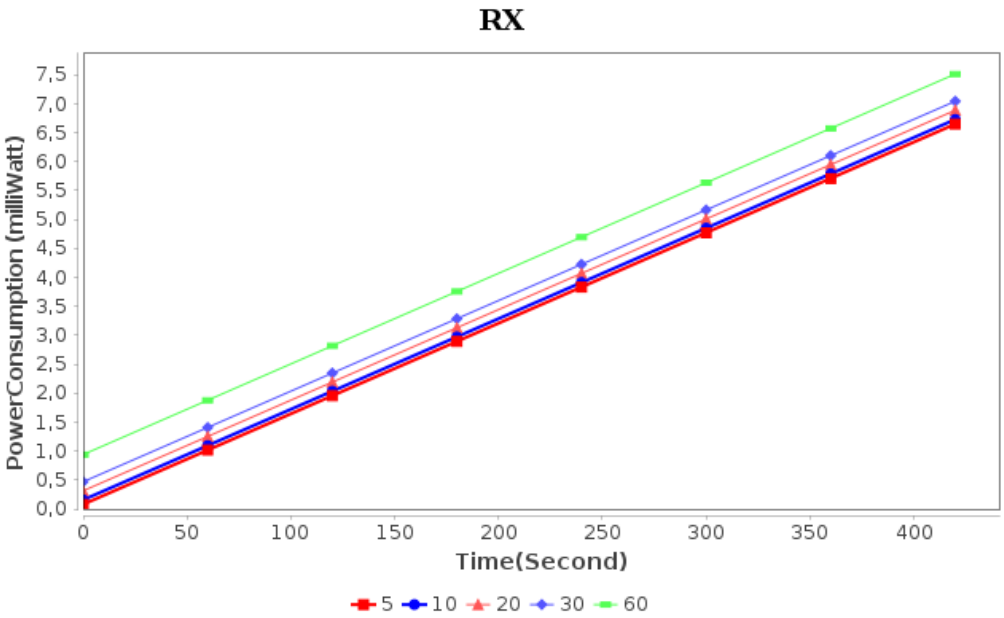

Figure 4.16: The total power consumption in milliwatt in first 450 seconds for RX

Table 4.18: Expected life time for MQTT-SN(Two Nodes)

Time(Interval)	Estimated Life Time(year)
5	0.0153438
10	0.0155188
20	0.0158968
30	0.016258
60	0.017703

4.3.7 Result for Three Nodes MQTT-SN Communication

The communication in this example is provided by three sensor nodes and the RSMB server.

The following table is created according to the data sent at 5 seconds intervals.

Table 4.19: Average power consumption in nanowatt for interval 5 seconds

TIME	CPU	LPM	TX	RX	Total
0	26133	3259	21346	56992774	57043512
5	25904	3260	0	57015840	57045004
10	28127	3206	89846	56300512	56421691
15	30852	3197	124892	56263334	56422275
20	28889	3203	66269	56325990	56424351
25	35294	3182	121069	56266777	56426322
30	27553	3208	121706	56266777	56419244
35	36998	3176	210596	56169702	56420472
40	28738	3204	90164	56301203	56423309
45	26194	3218	87615	56398278	56515305
50	42073	3159	183197	56197932	56426361
55	31553	3194	118520	56268499	56421766
60	26719	3216	113422	56363507	56506864
65	21891	3226	60534	56331833	56417484
70	22151	3226	60853	56333216	56419446
75	21468	3228	60534	56332524	56417754
80	22326	3225	60534	56332185	56418270
85	21626	3227	60534	56332185	56417572
90	22296	3225	60534	56332524	56418579
95	21583	3227	60534	56332524	56417868
100	25348	3215	60853	56333216	56422632
105	22937	3223	93032	56296729	56415921
110	27444	3208	93032	56296729	56420413
115	28496	3204	60534	56332524	56424758
120	25106	3216	86022	56304646	56418990
125	24580	3217	85704	56304294	56417795
130	22405	3225	60534	56332524	56418688
135	21758	3227	60216	56332185	56417386
140	22326	3225	60534	56332524	56418609

Table 4.19 continued from previous page

TIME	CPU	LPM	TX	RX	Total
145	21613	3227	60534	56332524	56417898
150	22296	3225	60534	56332524	56418579
155	21577	3227	60534	56332524	56417862
160	25481	3214	94625	56295692	56419012
165	24520	3218	60853	56332870	56421461
170	22333	3225	60534	56332524	56418616
175	21619	3227	60534	56332185	56417565
180	25523	3214	60534	56332524	56421795
185	24762	3217	94625	56294662	56417266
190	22133	3226	60853	56333216	56419428
195	21481	3228	60853	56332870	56418432
200	25553	3214	60534	56332185	56421486
205	23082	3222	93350	56296729	56416383
210	22296	3225	60534	56332524	56418579
215	21577	3227	60216	56332185	56417205
220	22133	3226	60853	56333216	56419428
225	21468	3228	60853	56332870	56418419
230	23783	3220	93350	56297068	56417421
235	24701	3217	60534	56332185	56420637
240	25946	3213	60534	56332524	56422217
245	28079	3206	86022	56304646	56421953
250	24750	3217	86341	56305337	56419645
255	21481	3228	60853	56332870	56418432
260	22617	3224	60853	56331833	56418527
265	21916	3226	60853	56331833	56417828
270	22296	3225	60534	56332524	56418579
275	21577	3227	60216	56332185	56417205
280	22133	3226	60853	56333216	56419428
285	21474	3228	60853	56332870	56418425
290	22326	3225	60534	56332185	56418270

Table 4.19 continued from previous page

TIME	CPU	LPM	TX	RX	Total
295	21626	3227	60534	56332185	56417572
300	22351	3225	60534	56332524	56418634
305	21638	3227	60534	56332524	56417923
310	22133	3226	60853	56333216	56419428
315	21468	3228	60853	56332870	56418419
320	22326	3225	60534	56332185	56418270
325	21619	3227	60534	56332185	56417565
330	26079	3212	60534	56332524	56422349
335	24792	3217	95262	56295008	56418279
340	21529	3228	60853	56333216	56418826
345	21481	3228	60534	56332524	56417767
350	22326	3225	60534	56332185	56418270
355	21619	3227	60534	56332185	56417565
360	22375	3225	60534	56332524	56418658
365	21650	3227	60534	56332524	56417935
370	22133	3226	60853	56333216	56419428
375	21474	3228	60853	56332870	56418425
380	22326	3225	60534	56332185	56418270
385	21626	3227	60534	56332185	56417572
390	22296	3225	60534	56332524	56418579
395	21577	3227	60534	56332524	56417862
400	22133	3226	60853	56333216	56419428
405	21468	3228	60853	56332870	56418419
410	27299	3208	86022	56304294	56420823
415	24532	3218	60216	56331833	56419799
420	22381	3225	60534	56332524	56418664
425	23384	3221	86022	56304646	56417273
430	22133	3226	60853	56333216	56419428
435	21474	3228	60853	56332870	56418425
440	22326	3225	60534	56332185	56418270

Table 4.19 continued from previous page

TIME	CPU	LPM	TX	RX	Total
445	21619	3227	60534	56332185	56417565
450	22296	3225	60534	56332524	56418579
Average(nanowatt)	23942,7	3220,7	71013,1	56336778,4	56434954,8

The following table is created according to the data sent at 10 seconds intervals.

Table 4.20: Average power consumption in microwatt for interval 10 seconds

TIME	CPU	LPM	TX	RX	Total
0	24943	3240	70570	56631321	56730074
10	24994	3239	62127	56640108	56730468
20	30212	3199	150699	56235110	56419220
30	27816	3207	120591	56268153	56419767
40	27892	3209	119157	56314112	56464370
50	32061	3193	148628	56238380	56422262
60	17363	3242	45560	56349395	56415560
70	20118	3232	57985	56336313	56417648
80	17559	3241	45560	56349395	56415755
90	19088	3236	44285	56350598	56417207
100	21683	3227	58623	56335628	56419161
110	21444	3228	60693	56333216	56418581
120	18390	3238	61968	56332012	56415608
130	21910	3226	57348	56337177	56419661
140	17559	3241	45560	56349395	56415755
150	18212	3239	44445	56350771	56416667
160	17550	3241	45560	56349395	56415746
170	20529	3231	61171	56333043	56417974
180	19127	3236	45719	56349734	56417816
190	26771	3210	121865	56266777	56418623

Table 4.20 continued from previous page

TIME	CPU	LPM	TX	RX	Total
200	19900	3233	62286	56331667	56417086
210	21393	3231	77898	56361958	56464480
220	17553	3241	45560	56349395	56415749
230	17311	3242	45560	56349568	56415681
240	20598	3231	58463	56335628	56417920
250	20514	3231	58623	56335628	56417996
260	17698	3240	45719	56349049	56415706
270	17456	3241	45560	56348876	56415133
280	19160	3236	63083	56330976	56416455
290	18837	3237	45560	56349568	56417202
300	17538	3241	45719	56349734	56416232
310	17293	3242	45719	56349568	56415822
320	19151	3236	45560	56349222	56417169
330	18922	3236	62924	56330464	56415546
340	17556	3241	45560	56349395	56415752
350	17314	3242	45400	56349395	56415351
360	17846	3240	45719	56349734	56416539
370	17299	3242	45719	56349568	56415828
380	17260	3242	45560	56349222	56415284
390	17314	3242	45560	56349222	56415338
400	20568	3231	58463	56335628	56417890
410	18761	3237	45719	56349568	56417285
420	17556	3241	45719	56349734	56416250
430	18163	3239	58463	56335628	56415493
440	17556	3241	45560	56349222	56415579
450	17314	3242	45560	56349222	56415338
Average(nanowatt)	20066,3	3233,8	60638,0	56348627,7	56432565,8

The following table is created according to the data sent at 20 seconds intervals.

Table 4.21: Average power consumption in microwatt for interval 20 seconds

TIME	CPU	LPM	TX	RX	Total
0	21890	3238	67543	56480716	56573387
20	21568	3239	62127	56486316	56573250
40	23021	3223	89686	56301376	56417306
60	24393	3220	104024	56310323	56441960
80	17937	3240	52569	56342508	56416254
100	17304	3242	46516	56349139	56416201
120	16711	3245	51295	56365664	56436915
140	16289	3245	44604	56350860	56414998
160	16826	3243	46516	56348620	56415205
180	16595	3244	46277	56349049	56415165
200	15751	3247	46356	56349139	56414493
220	16270	3245	38073	56357830	56415418
240	17012	3243	44604	56350860	56415719
260	17243	3242	44524	56350860	56415869
280	15979	3246	38152	56358003	56415380
300	16281	3245	46755	56348620	56414901
320	15069	3249	38073	56357651	56414042
340	15344	3248	38073	56357830	56414495
360	15385	3248	38073	56357830	56414536
380	15508	3248	38073	56357830	56414659
400	17486	3241	44604	56351033	56416364
420	17678	3241	53127	56341824	56415870
440	15764	3247	38232	56357747	56414990
Average(nanowatt)	17535,0	3242,1	50342,4	56360070,8	56431190,3

The following table is created according to the data sent at 30 seconds intervals.

Table 4.22: Average power consumption in microwatt for interval 30 seconds

TIME	CPU	LPM	TX	RX	Total
0	24424	3227	108484	56399193	56535328
30	25085	3225	105510	56403613	56537433
60	15848	3247	39931	56355993	56415019
90	15437	3248	41099	56354790	56414574
120	14628	3251	35736	56360755	56414370
150	15039	3249	39878	56355938	56414104
180	15203	3249	41259	56354901	56414612
210	15655	3247	41152	56354845	56414899
240	16030	3246	45613	56349853	56414742
270	16127	3246	39984	56356049	56415406
300	14962	3250	35683	56360755	56414650
330	15034	3249	41418	56354444	56414145
360	15425	3248	39878	56356164	56414715
390	14985	3250	35630	56360695	56414560
420	14681	3251	41152	56354901	56413985
450	15303	3248	39825	56355993	56414369
Average(nanowatt)	16491,6	3245,7	48264,5	56361805,1	56429806,9

The following table is created according to the data sent at 60 seconds intervals.

Table 4.23: Average power consumption in microwatt for interval 60 seconds

TIME	CPU	LPM	TX	RX	Total
0	19689	3238	75296	56376678	56474901
60	20517	3235	70305	56382754	56476811
120	14838	3250	40860	56355217	56414165
180	14555	3251	38763	56357772	56414341
240	14455	3251	35258	56361130	56414094

Table 4.23 continued from previous page

TIME	CPU	LPM	TX	RX	Total
300	14628	3251	38285	56358289	56414453
360	14692	3251	38152	56358058	56414153
420	14537	3251	38179	56358459	56414426
Average(nanowatt)	15988,9	3247,2	46887,2	56363544,6	56429668

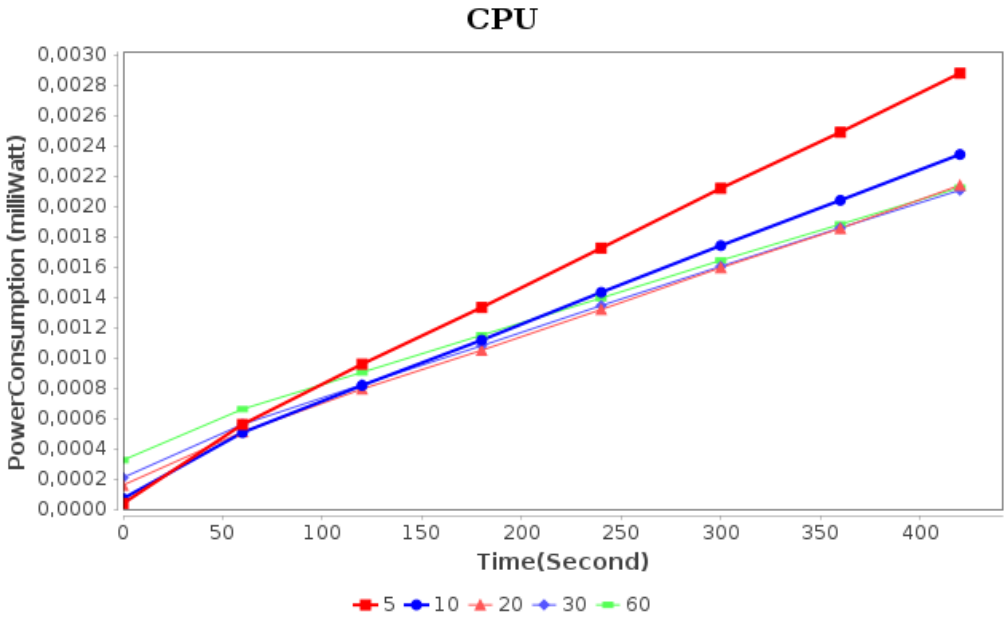

Figure 4.17: The total power consumption in millwatt in first 450 seconds for CPU

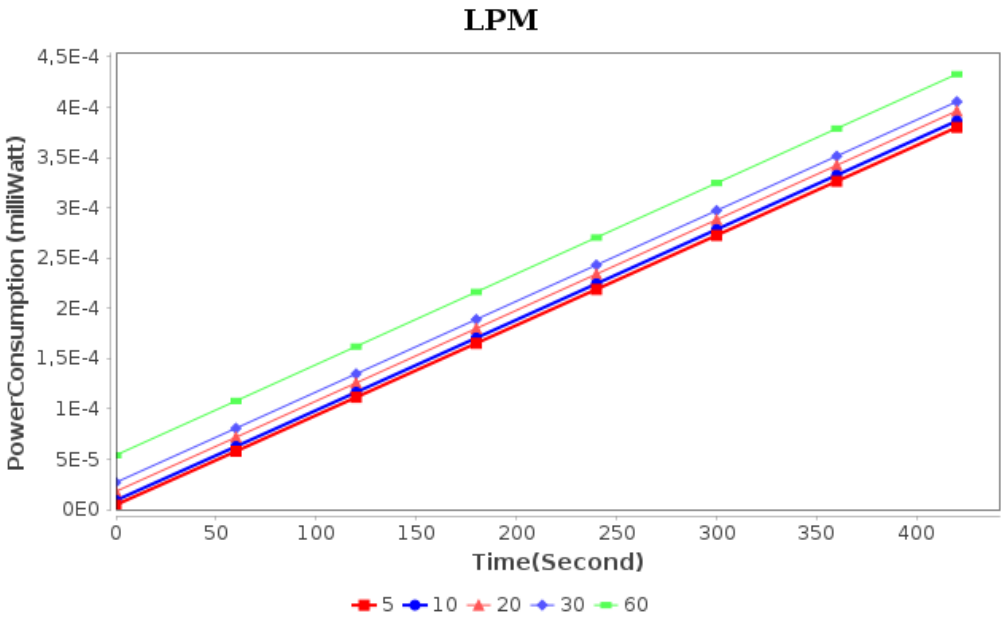

Figure 4.18: The total power consumption in milliwatt in first 450 seconds for LPM

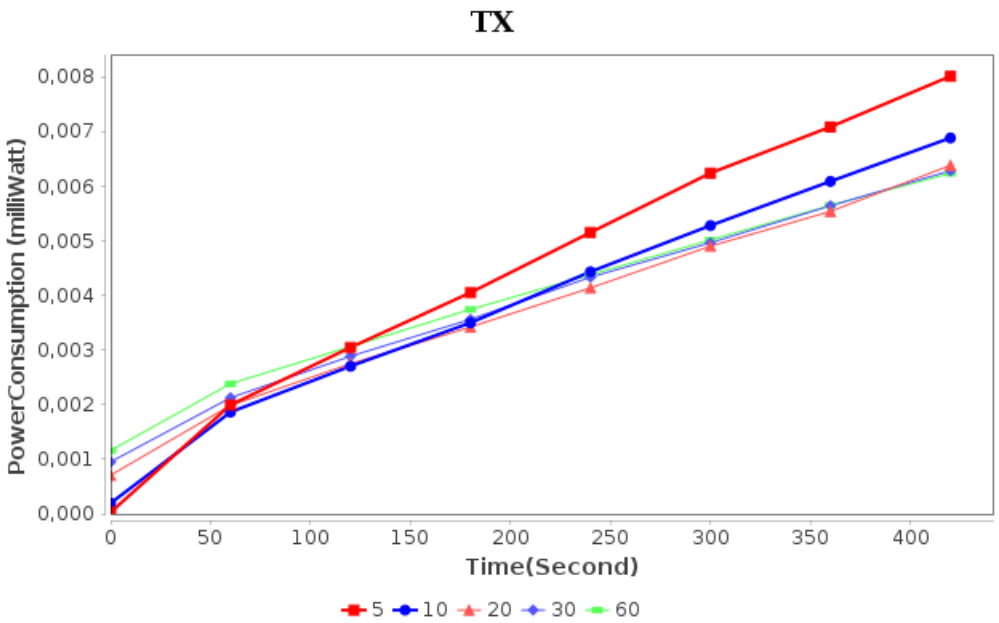

Figure 4.19: The total power consumption in milliwatt in first 450 seconds for TX

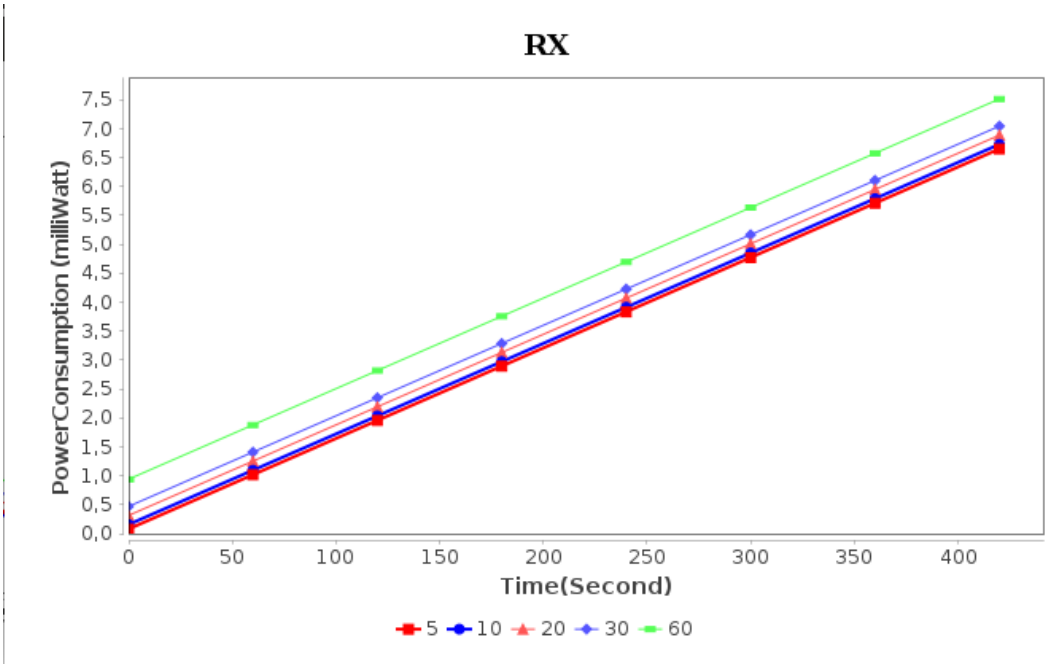

Figure 4.20: The total power consumption in milliwatt in first 450 seconds for RX

Table 4.24: Expected life time for tested periods

Time(Interval)	**Estimated Life Time(year)**
5	0.0153409
10	0.0155161
20	0.0158939
30	0.0162552
60	0.0176993

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, it is seen that the data distribution system, which is widely used today, has been successfully installed on resource-constrained devices that have low power consumption and low memory usage by tailoring the DDS. Since these devices have approximately 8 KB RAM and 128KB ROM values, their power consumption is low and this feature enables them to work for days. The DDS interface provides an abstraction on the network layer. As another point of view, it is aimed to use MQTT middleware using a publisher-subscriber pattern on the same operating system. However, since the memory problems of the device are not overcome in this example, it has been found that the connection to the server can be realized with the python application defined on the computer. However, Python application runs on a computer that provides a USB connection for running application, so it requires more components than other methods. In the MQTT-SN software, unlike MQTT, the studies mentioned in the previous sections are performed and the device is installed successfully as a result of updates. For now, the Zigbee based DDS software only listens to the data received event and the data delay event. In future work, more Qos event information can be added to the Zigbee based DDS. MQTT-SN software can be created automatically source code by reading from an XML file, similar to DDS. Besides differences between bandwidth utilization will be measured and added to the result section.

REFERENCES

[1] Contiki-Os, "contiki-os/contiki." https://github.com/contiki-os/contiki/blob/master/dev/sht11/sht11.c, 2015. [Online; accessed 4-August-2019].

[2] Z. C. Team, "As-xm1000." https://www.advanticsys.com/shop/asxm1000-p-24.html, 2019. [Online; accessed 21-November-2019].

[3] "Cm5000." https://www.advanticsys.com/wiki/index.php?title=CM5000, 2019. [Online; accessed 21-November-2019].

[4] "Quality of service." http://download.prismtech.com/docs/Vortex/html/ospl/DDSTutorial/qos.html. Online; accessed 19-November-2019].

[5] A. Stanford-Clark and H. L. Truong, "Mqtt for sensor networks (mqtt-sn) protocol specification," *International business machines (IBM) Corporation version*, vol. 1, 2013.

[6] "About the dds for extremely resource constrained environments specification version 1.0 beta 2." https://www.omg.org/spec/DDS-XRCE/About-DDS-XRCE/, 2018. [Online; accessed 21-November-2019].

[7] G. Pardo-Castellote, "Omg data-distribution service: Architectural overview," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCSW '03, (Washington, DC, USA), pp. 200–, IEEE Computer Society, 2003.

[8] K. Beckmann and M. Thoss, "A model-driven software development approach using omg dds for wireless sensor networks," Oct 2010.

[9] "Mqtt." http://mqtt.org/, 2019. [Online; accessed 21-November-2019].

[10] T. Reusing, "Comparison of operating systems tinyos and contiki," *Sens. Nodes-Oper. Netw. Appl.(SN)*, vol. 7, pp. 7–13, 2012.

[11] "The open source os for the internet of things." http://www.contiki-os.org/, 2018. [Online; accessed 21-November-2019].

[12] "Wireless sensor networks :: Telosb." http://www.cmt-gmbh.de/Produkte/WirelessSensorNetworks/TelosB.html, 2016. [Online; accessed 21-November-2019].

[13] Pksec, "pksec/msp430-gcc-4.7.3." https://github.com/pksec/msp430-gcc-4.7.3, Feb 2019. Online; accessed 19-November-2019].

[14] Contiki-Os, "contiki-os/contiki." https://github.com/contiki-os/contiki/wiki/Reducing-Contiki-OS-firmware-size, 2019. [Online; accessed 21-November-2019].

[15] "Introduction to mqtt-sn (mqtt for sensor networks)." http://www.steves-internet-guide.com/mqtt-sn/. Online; accessed 19-November-2019].

[16] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, pp. 455–462, Nov 2004.

[17] M. R. H. Khan, R. Passerone, and D. Macii, "Fzepel: Rf-level power consumption measurement (rf-pm) for zigbee wireless sensor network-towards cross layer optimization," in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 959–966, Sep. 2008.

[18] m. a. Sarijari, M. Sharil Abdullah, A. Lo, and R. A Rashid, "Experimental studies of the zigbee frequency agility mechanism in home area networks," in *Experimental studies of the ZigBee frequency agility mechanism in home area networks*, vol. 2014, pp. 711–717, 09 2014.

[19] "What is dds?." https://www.dds-foundation.org/what-is-dds-3/, 2005. Online; accessed 19-November-2019].

[20] "Twin oaks computing (2011) inc twin oaks computing. 2011. what can dds do for you?." https://www.omg.org/hot-topics/documents/dds/CoreDX_DDS_Why_Use_DDS.pdf/, 2019. [Online; accessed 21-November-2019].

[21] G. Pardo-Castellote, "Omg data-distribution service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pp. 200–206, May 2003.

[22] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *The Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[23] "How to build and run rti connext dds micro on a microcontroller." https://www.rti.com/blog/how-to-build-and-run-rti-connext-dds-micro-on-a-micro, journal=Connectivity Software Framework for the Industrial IoT, author=Hidalgo, Jesus Checa, year=2017, note = [Online; accessed 21-November-2019].

[24] https://community.rti.com/static/documentation/connext-micro/2.4.6/doc/html/group__OSAPIUserManuals__PortingModule.html, 2017. [Online; accessed 21-November-2019].

[25] "eprosima micro xrce-dds¶." https://micro-xrce-dds.readthedocs.io/en/latest/index.html, 2018. [Online; accessed 21-November-2019].

[26] O. Deschambault, A. Gherbi, and C. Légaré, "Efficient implementation of the mqtt protocol for embedded systems," *JIPS (Journal of Information Processing Systems)*, vol. 13, no. 1, pp. 26–39, 2017.

[27] Y. Upadhyay, A. Borole, and D. Dileepan, "Mqtt based secured home automation system," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pp. 1–4, March 2016.

[28] T. Reusing, "Comparison of operating systems tinyos and contiki," *Sens. Nodes-Oper. Netw. Appl.(SN)*, vol. 7, pp. 7–13, 2012.

[29] Imatix, "imatix/gsl." https://github.com/imatix/gsl, Sep 2017. [Online; accessed 19-November-2019].

[30] GNU, "Gnu make." https://www.gnu.org/software/make/manual/make.html#Wildcard-Examples, May 2016. [Online; accessed 21-November-2019].

[31] "Rsmb: Mqtt." http://mqtt.org/tag/rsmb, 2016. [Online; accessed 21-November-2019].

[32] IBM, "Really small message broker." https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=d5bedadd-e46f-4c97-af89-22d65ffee070, Jan 2013. [Online; accessed 23-September-2019].

[33] Eclipse, "eclipse/mosquitto.rsmb." https://github.com/eclipse/mosquitto.rsmb/tree/master/rsmb/src, 2016. [Online; accessed 23-September-2019].

[34] Aignacio, "aignacio/mqtt-sn-contiki_example." https://github.com/aignacio/mqtt-sn-contiki_example, Aug 2017. [Online; accessed 23-September-2019].

[35] Contiki-Ng, "contiki-ng/contiki-ng." https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-Energest, 2016. [Online; accessed 21-November-2019].

APPENDIX A

MAKEFILE FOR SENSORDDS

```
SDDS_TOPDIR := ../../
CONTIKI := ../../contiki
DRIVER := $(SDDS_TOPDIR)/include/
ifeq ($(BUILD),sensor_board)
SDDS_OBJDIR := objs-xm1000
TARGET := xm1000
else
SDDS_OBJDIR := objs-sky
TARGET := sky
endif
SDDS_PLATFORM := contiki
SDDS_ARCH := atmega
SMALL = 1
//MODULES += core/net/ipv6/multicast
CONTIKI_WITH_IPV4 = 1
CONTIKI_WITH_RIME = 1
LOCAL_CONSTANTS := local_constants.h
# Object files of the generateted dds data types
DATA_DEPEND_OBJS += $(SDDS_OBJDIR)/sensordata-ds.o
#DATA_DEPEND_OBJS += $(SDDS_OBJDIR)/alpha-ds.o
DATA_DEPEND_OBJS += $(SDDS_OBJDIR)/sensordata_test_publisher_sdds_impl.o
#OBJS = $($(shell ls *-ds.c):.o=.c)
DATA_DEPEND_OBJS += $(addprefix $(SDDS_OBJDIR)/, $(OBJS))
# object files depending on platform
PLATFORM_DEPEND_OBJS += $(SDDS_OBJDIR)/*.o
# object files depending on driver for sensors
#DRIVER_DEPEND_OBJS += $(SDDS_OBJDIR)/sdds-driver-$(SDDS_ARCH)-LED.o
#DRIVER_DEPEND_OBJS +=
↪  $(SDDS_OBJDIR)/sdds-driver-$(SDDS_ARCH)-GammaCorrection.o
# object files of the generates implementation code file of sdds
IMPL_DEPEND_OBJS = $(SDDS_OBJDIR)/sensordata_test_publisher_sdds_impl.o
#TARGET_STARTFILES = sensordata_test_publisher.c
# file for the preprocessor constants of sdds

SDDS_CONSTANTS_FILE :=   sdds_features_config.h   sdds_features.h
↪  sdds_network.h   sdds_profile.h
```

```
#TARGET_LIBFILES += -L. -lhel
ALL_OBJS += $(PLATFORM_DEPEND_OBJS)
ALL_OBJS += $(DRIVER_DEPEND_OBJS)
ALL_OBJS += $(IMPL_DEPEND_OBJS)
ALL_OBJS += $(SDDS_OBJDIR)/$(APPLICATION_NAME).o
ALL_OBJS += $(DATA_DEPEND_OBJS)

include $(SDDS_TOPDIR)/sdds.mk
include $(CONTIKI)/Makefile.include
DATA_DEPEND_SRCS+=$(patsubst $(SDDS_OBJDIR)/%.o,%.c,$(DATA_DEPEND_OBJS))
DATA_DEPEND_SRCS+=$(patsubst $(SDDS_OBJDIR)/%.o,%.h,$(DATA_DEPEND_OBJS))
CLEAN += $(DATA_DEPEND_SRCS)

IMPL_DEPEND_SRCS += $(patsubst $(SDDS_OBJDIR)/%.o,%.c,$(IMPL_DEPEND_OBJS))
IMPL_DEPEND_SRCS += $(patsubst$(SDDS_OBJDIR)/%.o,%.h,$(IMPL_DEPEND_OBJS))
$(info $(IMPL_DEPEND_SRCS))

CLEAN += $(IMPL_DEPEND_SRCS)
CLEAN += $(ALL_OBJS)
CLEAN += $(patsubst %.o,%.d,$(ALL_OBJS))
CLEAN += $(SDDS_CONSTANTS_FILE)
CLEAN += local_constants.h
all:$(APPLICATION_NAME)
$(SDDS_OBJDIR):
mkdir $(SDDS_OBJDIR)
$(LOCAL_CONSTANTS):
touch $(LOCAL_CONSTANTS)
CFLAGS += -I.
CFLAGS += -I $(DRIVER)
CFLAGS += -Os
$(SDDS_OBJDIR)/%.o: %.c
$(COMPILE.c) -MMD $(OUTPUT_OPTION) $<
@echo "compiled"
$(APPLICATION_NAME).c: $(LOCAL_CONSTANTS) $(SDDS_OBJDIR)
$(IMPL_DEPEND_SRCS) $(DATA_DEPEND_SRCS)
$(CC) $(CFLAGS) -MM -MF
$(SDDS_OBJDIR)/$(APPLICATION_NAME).d -MT$@ $^
@echo "********************compile c  $(IMPL_DEPEND_SRCS)"
@echo "$(CC) $(CFLAGS) -c    $(IMPL_DEPEND_SRCS)
$(DATA_DEPEND_SRCS) -MM -MF$(SDDS_OBJDIR)/$(APPLICATION_NAME).d -MT$@ $^"
@echo "compile executed"
$(APPLICATION_NAME).$(TARGET): $(APPLICATION_NAME).co
$(SDDS_OBJS) $(DATA_DEPEND_OBJS)
$(IMPL_DEPEND_OBJS) $(PROJECT_OBJECTFILES) $(PROJECT_LIBRARIES)
contiki-$(TARGET).a
@echo " $(Q)$(LD) $(LDFLAGS) $(SDDS_OBJS) $(DATA_DEPEND_OBJS)
$(APPLICATION_NAME).co obj_sky/contiki-xm1000-main.o contiki-$(TARGET).a
-o $(APPLICATION_NAME).$(TARGET)"

$(Q)$(LD) $(LDFLAGS)     $(APPLICATION_NAME).co
```

```
85  $(SDDS_OBJS) $(DATA_DEPEND_OBJS) obj_$(TARGET)/contiki-$(TARGET)-main.o
86  contiki-$(TARGET).a -o $(APPLICATION_NAME).$(TARGET)
87
88  @echo "StartFiles will be added "
89  @echo "Target generator is done for xm1000 "
90  $(APPLICATION_NAME).ihex: $(APPLICATION_NAME).$(TARGET)
91  $(OBJCOPY) $^ -O ihex $@
92  @echo 'ihex run2'
93  CLEAN += $(APPLICATION_NAME).elf $(APPLICATION_NAME).hex
94  $(APPLICATION_NAME).ihex $(APPLICATION_NAME).out
95  CLEAN += symbols.c symbols.h
96  CLEAN += $(APPLICATION_NAME).d
97  CLEAN += -rf $(SDDS_OBJDIR)
98
99  %-ds.c %-ds.h %_sdds_impl.c %_sdds_impl.h:
100 #$(shell ./generate.sh)
101
102 -include $(patsubst %.o,%.d,$(ALL_OBJS))
103
104 code:
105 #$(shell ./generate.sh)
```

APPENDIX B

INTERFACE DEFINITION FOR DDS

Listing 10 XML file for Interface Definition

```
<profile id = "1">
<topic name = "temperature" domain = "1" id = "A">
Temperature Test
<attribute name = "deviceId" type = "DDS_short" key = "primary">Device ID
</attribute>
<attribute name = "temperature" type = "DDS_short">Temperature value
</attribute>
<attribute name = "light" type = "DDS_short">Light value
</attribute>
<attribute name = "humidity" type = "DDS_short">Humidity value
</attribute>
</topic>
</profile>
```

APPENDIX C

READING SERIAL DATA

Listing 11 Reading serial data from USB

```
#include "contiki.h"
#include "dev/serial-line.h"
#include <stdio.h>
#include "dev/leds.h"
#include "dev/uart1.h"
PROCESS(test_serial, "Serial line test process");
AUTOSTART_PROCESSES(&test_serial);
static struct etimer timer;
static int uart_rx_callback(unsigned char c) {
uint8_t u;
u = (uint8_t)c;
leds_on(LEDS_ALL);
printf("\nReceived temp: %u",u);
//etimer_set(&timer, 1 * CLOCK_SECOND);
//PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer));
leds_off(LEDS_ALL);
}
PROCESS_THREAD(test_serial, ev, data)
{
PROCESS_BEGIN();
uart1_init(BAUD2UBR(115200)); //set the baud rate as necessary
printf("%s\n","Waiting data:" );
uart1_set_input(uart_rx_callback); //set the callback function
leds_on(LEDS_ALL);
for(;;) {
PROCESS_YIELD();
if(ev == serial_line_event_message) {
leds_off(LEDS_ALL);
printf("received line: %s\n", (char *)data);
}
}
PROCESS_END();
}
```

APPENDIX D

CREATING DDS OBJECTS

Listing 12 XML file for creating DDS components

```
<\ project
name = "sensordata_test_publisher"
script = "sdds.gsl"
endian = "little"
os = "contiki"
ip = "fe80::12:13ff:fe14:1516"
port = "23234">
SensorData

<!--
Includes are processed first, so XML in included files will be
part of the XML tree
-->
<define name = "SDDS_NET_MAX_BUF_SIZE" value = "128"/>
<define name = "SDDS_QOS_HISTORY_DEPTH" value = "3"/>
<define name = "SDDS_NET_MAX_LOCATOR_COUNT" value = "10"/>
<include filename = "../topics/sensordata.xml" />

<role topic = "sensordata" type = "publisher"/>
<role topic = "sensordata" type = "subscriber"/>
</project>
```

APPENDIX E

OPTIMIZING MEMORY FOR MSP430

```
1  MEMORY {
2  sfr              : ORIGIN = 0x0000, LENGTH = 0x0010 /* END=0x0010,
   ↪  size 16 */
3  peripheral_8bit  : ORIGIN = 0x0010, LENGTH = 0x00f0 /* END=0x0100,
   ↪  size 240 */
4  peripheral_16bit : ORIGIN = 0x0100, LENGTH = 0x0100 /* END=0x0200,
   ↪  size 256 */
5  ram_mirror (wx)  : ORIGIN = 0x0200, LENGTH = 0x0800 /* END=0x0a00,
   ↪  size 2K */
6  infomem          : ORIGIN = 0x1000, LENGTH = 0x0100 /* END=0x1100,
   ↪  size 256 as 2 128-byte segments */
7  infob            : ORIGIN = 0x1000, LENGTH = 0x0080 /* END=0x1080,
   ↪  size 128 */
8  infoa            : ORIGIN = 0x1080, LENGTH = 0x0080 /* END=0x1100,
   ↪  size 128 */
9  ram (wx)         : ORIGIN = 0x1100, LENGTH = 0x2800 /* END=0x3900,
   ↪  size 10K */
10 rom (rx)         : ORIGIN = 0x4000, LENGTH = 0xbfe0 /* END=0xffe0,
   ↪  size 49120 */
11 vectors          : ORIGIN = 0xffe0, LENGTH = 0x0020 /*
   ↪  END=0x10000, size 32 as 16 2-byte segments */
12 /* Remaining banks are absent */
13 bsl              : ORIGIN = 0x0000, LENGTH = 0x0000
14 infoc            : ORIGIN = 0x0000, LENGTH = 0x0000
15 infod            : ORIGIN = 0x0000, LENGTH = 0x0000
16 ram2 (wx)        : ORIGIN = 0x0000, LENGTH = 0x0000
17 usbram (wx)      : ORIGIN = 0x0000, LENGTH = 0x0000
18 far_rom          : ORIGIN = 0x00000000, LENGTH = 0x00000000
19 }
20 REGION_ALIAS("REGION_TEXT", rom);
21 REGION_ALIAS("REGION_DATA", ram);
22 REGION_ALIAS("REGION_FAR_ROM", far_rom); /* Legacy name, no longer
   ↪  used */
23 REGION_ALIAS("REGION_FAR_TEXT", far_rom);
24 REGION_ALIAS("REGION_FAR_DATA", ram2);
25 PROVIDE (__info_segment_size = 0x80);
26 PROVIDE (__infob = 0x1000);
27 PROVIDE (__infoa = 0x1080);
```