MASTER

Scalability of EADS and Energy-Aware Scheduling for energy-efficient system

Nanadi, Shipha

Award date:
2021

Department of Mathematics and Computer Science

Scalability of EADS and Energy-Aware Scheduling for energy-efficient system

Master Thesis Report

Shipha Nanadi (1422669)

University Supervisor:
dr. Dip Goswami (Electronics Systems, TU/e)
Company Supervisors:
ir. Leslie Aerts (Team Lead, Capgemini Engineering)
ir. Herman Roebbers (Advance Expert, Capgemini Engineering)
External Committee:
dr. ir. Geoffrey Nelissen
Prof. dr. ir. Sonia Heemstra

Eindhoven, November 2021

Abstract

Internet of Things (IoT) provides intelligent services to the users by connecting various smart devices over the internet and allowing these devices to exchange information. Although IoT is proliferating, factors that can limit its growth are energy-constrained devices and the ability of these devices to adapt to the growing network.

Capgemini Engineering aims to create a secure and scalable display system, known as Energy Autonomous Display System (EADS), that can run without batteries. There exists a proof of concept for EADS at Capgemini Engineering that uses Bluetooth Low Energy (BLE) for communication. One of the problems associated with BLE is that only eight nodes can be supported per network simultaneously with a single central hub, thus limiting scalability. Hence, BLE cannot be the best choice when the node density increases (for example, in a larger building). Additionally, BLE only supports point-to-point communication between a BLE device and a central hub. Hence, this thesis aims to investigate and create a framework using wireless mesh protocol that allows low-power scalability of EADS devices. Wirepas Massive protocol was used to establish low-power scalability of these battery-less devices and the performance of EADS was compared with other popular protocols such as BLE and BLE Mesh. The results show that Wirepas Massive consumes significantly less energy when compared to BLE and BLE Mesh. Furthermore, the effect on energy consumption of the device under various scenarios was analyzed, for instance, varying the number of nodes, node configurations, the distance between the nodes. It was observed that varying distance as well the number of nodes do not affect the energy consumption of Wirepas node and Wirepas signal strength. Hence, Wirepas Massive is suitable for the low-power scalability of battery-less devices such as EADS.

Furthermore, as EADS is a battery-less device, it may shut down during the execution of a specific function due to the dynamic behavior of energy storage. Due to this, the task is lost and the energy consumed to execute the interrupted task also depletes the scarcely available energy. An intelligent algorithm is essential to ensure the EADS device does not shut down during the execution of a task. This thesis aims at creating an intelligent algorithm, known as the Energy-Aware Scheduling (EAS) algorithm that keeps track of the available energy before and after a specific task execution. And only executes a task depending on the available energy, the energy cost per task, and the priority of the tasks. The improvement in the EADS functionality with the EAS algorithm was examined and it was observed that the EADS device alleviates the risk of system shut down during the execution of a task.

Preface

The thesis titled - *"Scalability of EADS and Energy-Aware Scheduling for energy-efficient system"*, has been conducted to fulfill the graduation requirements of Master's degree in Embedded Systems at Eindhoven University of Technology (TU/e). The research work presented in this thesis was done in collaboration with TU/e and Capgemini Engineering.

I would like to thank my supervisor, dr. Dip Goswami for his guidance and valuable feedback during the course of the thesis. I would also like to thank my company supervisor, ir. Leslie Aerts for his constant support and positive mindset towards addressing multiple issues I encountered during my thesis work. I would like express my sincere gratitude to my product owner, ir. Herman Roebbers for his willingness to discuss all the issues at any time and providing constructive feedback related to my work.

Finally, I would like to thank my mom, and sister for believing in me and being my pillars of strength. I would also like to thank my fellow friends for all the valuable discussions that have helped me over the past two years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Internet of Things (IoT) consists of smart devices connected via the internet that communicate and exchange data. Over the past few years, the number of connected devices in IoT has exponentially increased as they are cheap and easy to use. It is predicted that by the end of 2022, there will be around 30 billion connected devices globally [11]. IoT has multiple applications as this technology can be adjustable to almost any technology capable of providing relevant information about its operation or environmental conditions needed to be controlled at a distance. Some of the applications where IoT is widely used are smart homes, smart cities, transportation, or even healthcare. IoT devices communicate with each other and a gateway/server through wireless communication protocols. IoT offers a wide range of wireless technologies, as shown in Figure 1.1 depending on the range of communication, data rate, frequency, and energy consumption. For instance, short to medium range communication protocols as Bluetooth Low Energy (BLE), Wi-Fi and ZigBee are suitable for IoT applications with moderate range and data rate. Similarly, long-range communication protocols such as Sigfox, LoRa, and Cellular networks are used for IoT networks that require long-range with minimal data rate.

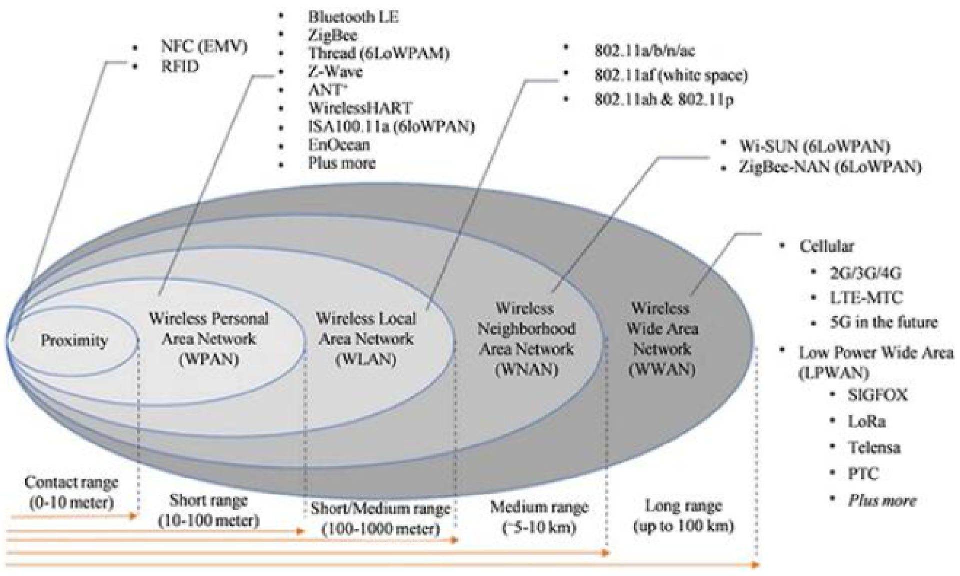


Figure 1.1: Wireless communication protocols for IoT applications [9]

Although IoT has evolved over the past few years, one of the factors preventing this technology from achieving its full potential is the power consumption of IoT devices [10]. IoT devices have reasonably low power consumption and are increasingly battery powered to enable operation in mobile or distributed applications [6]. However, these battery-powered devices in IoT have numerous disadvantages. Batteries contain toxic and flammable materials and production of these batteries releases high greenhouse gas emissions. Furthermore, when the batteries are disposed of carelessly, the toxic chemicals can leach into the surrounding environment and contaminate soil and water, thus damaging the ecosystem. Moreover, batteries also suffer from current peaks that make them degrade faster. IoT devices suffer from such peaks as they spend most of the time in sleep or a low-power state. The power consumption of these devices may jump up orders of magnitude when the device is awake and transmits or receives messages [7]. Another major issue with battery-powered devices is that they involve huge effort in maintenance, for instance, replacing batteries quite often for a high-density network can be challenging. Although rechargeable batteries can offset these issues to some extent, they still suffer performance degradation due to frequent charge and discharge cycles.

Therefore, batteries should be replaced to reduce the negative impact on the environment. To do so, currently, various alternatives such as energy harvesting techniques are used. Energy harvesting requires a harvesting device with an IoT sensor node to provide the energy required or to recharge the onboard battery or capacitor [27]. Some of the harvesting techniques currently used are solar (Photo Voltaic), thermoelectric generators (Seebeck effect), kinetic energy (Piezo-Electric effect), Vibration energy (using a magnet moving inside a coil) and RF energy [27]. These battery-less devices are becoming popular as they are environmentally friendly, require minimal maintenance, and are easy to recycle.

Nowadays, the meeting rooms at workplaces are equipped with meeting room displays that display meeting information, for instance, scheduled time and meeting name. However, these display systems operate on batteries and often need to be recharged. Recharging these batteries can be challenging if the display systems are spread over an entire office building. One such example of a commercially available meeting room display is Joan 6 [17], which requires recharging every three months. To reduce the usage of batteries and eliminate the burden of recharging the batteries, Capgemini Engineering is creating a vision for sustainable IoT by developing an *Energy Autonomous Display System (EADS)*, a secure display system that can run without batteries. EADS will have functionalities similar to the commercially available display systems, however with one main difference i.e., without any batteries. This project serves to demonstrate how scalable and secure wireless IoT systems can operate without batteries.

Energy Autonomous Display System is a battery-less device and uses energy from the ambiance for its operation. It consists of various components such as - an energy harvester unit (solar cells), an energy storage unit (supercapacitor), a power management IC and voltage regulator, processing unit and display (see Figure 1.2). The voltage regulator converts the stored energy into the desired voltage required by the processing unit, and the display. Furthermore, the processing unit communicates with the display through Serial Peripheral Interface (SPI) communication. EADS also consists of a server that is responsible for sending the up-to-date calendar information to the processing unit.

One of the problems related to EADS is the low-power scalability of these devices. This is because, as EADS is a battery-less device, the energy stored in the energy storage (supercapacitor) must be utilized as efficiently as possible. Another factor that makes the scalability of these devices challenging is the sporadic voltage supplied to run the processing unit. The system can shut down in-between operations if the energy stored is insufficient. This thesis aims at investigating and employing a framework to address the EADS low-power scalability issue and make EADS an energy-efficient system. Section 1.2 gives a detailed explanation of the problem statement.

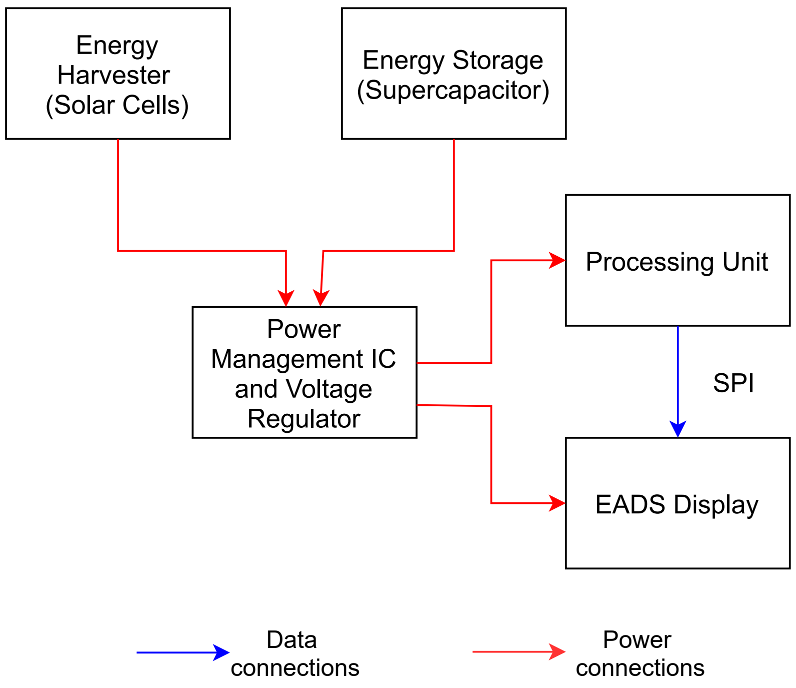

Energy Harvester (Solar Cells)
Energy Storage (Supercapacitor)
Processing Unit
Power Management IC and Voltage Regulator
SPI
EADS Display
Data connections
Power connections

Figure 1.2: EADS hardware modules

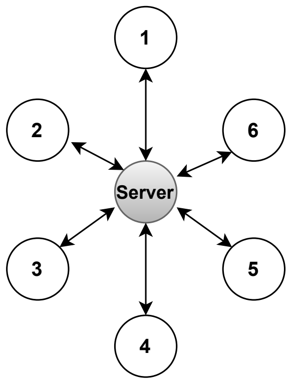
1.2 Problem Statement and Goal

Network scalability is one of the factors limiting the extensive growth of IoT as adding IoT devices in an existing network should not affect the network performance. Scalability refers to ”the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth” [21]. Capgemini Engineering aims at deploying EADS devices in the Eindhoven office building in the near future. The existing EADS prototype is implemented using a star network such as Bluetooth Low Energy (BLE) protocol which gives rise to the following issues:

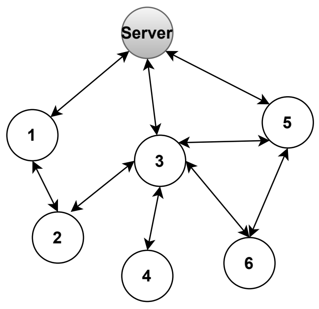
1. In the BLE network, the nodes in the network have a single path to the server as shown in Figure 1.3a. However, if this path is damaged, the node can no longer communicate with the server and can no longer be a part of the network.

2. BLE network can only support up to 8 nodes per server at any moment. As Capgemini Engineering aims to have a scalable network with nodes distributed over the entire building, BLE cannot satisfy this requirement.

These issues can be mitigated by using a wireless mesh protocol. As seen in Figure 1.3b, each node in the network has multiple paths to reach the server. Moreover, wireless mesh protocol also allows multiple nodes (based on IoT protocol) to be connected to the same server, thus allowing the scalability of EADS devices. However, achieving *low-power scalability* of these battery-less devices is challenging as they should consume as little energy as possible. This thesis aims at investigating and employing a suitable wireless mesh protocol to support the low-power scalability of EADS devices. In this work, low-power scalability of battery-less device such as EADS is evaluated which has not been addressed in the literature yet.



(a) Star Network



(b) Mesh Network

Another issue related with battery-less devices such as Energy Autonomous Display System (EADS) is that the energy stored in the supercapacitor is dynamic and scarce and may lead to intermittent execution of the device functions. For instance, as seen in Figure 1.4, the device may turn on and off frequently as it depletes and replenishes the stored energy in the capacitor. This behavior may risk the EADS system shut down when there is insufficient energy to operate the EADS device. The EADS device may shut down during an execution of a specific function or task, thus losing the task. Additionally, this interrupted task execution also leads to wasting of scarce available energy on the task chains that cannot be completed on time anyway. To eliminate this issue, an intelligent algorithm is required that can keep track of available energy. An Energy-Aware Scheduling (EAS) framework that measures the amount of energy consumed before and after every task, and schedules the tasks based on priority and energy available may contribute to efficiently utilizing the stored energy.

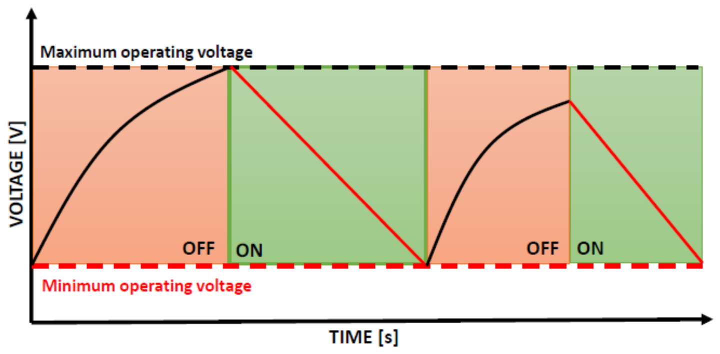


Figure 1.4: Battery-less device intermittent behavior [7]

Hence, considering the problems addressed above, this thesis aims to investigate a low-power wireless mesh network that allows low-power scalability of EADS, in combination with an intelligent algorithm that ensures efficient energy usage. Additionally, this algorithm will also prevent system shut down during task execution.

1.3 Research Questions

This sub-section provides a list of research questions that will act as a guideline to achieve the main goal of this thesis. These questions will be revisited in the later chapters where they are answered based on the findings from the research.

1. **How can a battery-less device such as Energy-Autonomous Display System (EADS) be made scalable with minimal energy consumption?**

   (a) ***What are the currently available wireless mesh protocols for IoT applications and how can a mesh protocol be used to achieve scalability in EADS?***

   (b) ***What are the available Energy-Aware Scheduling frameworks for battery-less energy harvesting IoT devices and how can this functionality be achieved with EADS to efficiently utilize the available energy?***

   (c) ***What is the impact of growing Wirepas Massive network on the energy of the nodes?***

1.4 Requirements

Any system should adhere to a set of requirements, which when met, describe the system's behavior. Similarly, following are the set of functional requirements considered to achieve the main aim of this thesis.

RQ 1. EADS shall have a supercapacitor that harvests the energy from indoor solar cells.

RQ 2. EADS shall configure the Power management IC to charge the supercapacitor to a voltage that is substantially below its rated voltage.

RQ 3. EADS shall provide itself with enough energy harvesting capacity to be able to function energy autonomously from an illumination level of 300 lux and more.

RQ 4. EADS (Node) requests calendar information from the server through Wirepas Massive communication.

RQ 5. EADS will act as a router-node to forward the calendar information to other neighboring nodes.

RQ 6. EADS server should communicate with more than one node at the same time.

RQ 7. EADS working shall be monitored with a software tool for dense network.

RQ 8. EADS shall be equipped with Energy-Aware Scheduling algorithm to efficiently utilize the energy.

RQ 9. EAS (energy-aware scheduling) shall only execute a certain task if the available energy is greater than energy required for that particular task and set threshold value.

RQ 10. EAS shall turn the display off when the available energy falls below minimum voltage.

RQ 11. EAS shall update the energy after execution of every task.

1.5 Outline of the report

This thesis report's outline is as follows: Chapter 1 describes the EADS background, problem statement and research questions. Chapter 2 gives a comparison of different mesh networks and energy-aware scheduling frameworks. Chapter 3 gives a detailed working of the chosen mesh network. Chapter 4 describes the methodology that will be carried out to achieve the main goal of the thesis. Furthermore, Chapter 5 presents the implementation details used to create frameworks for scalability and energy-aware scheduling algorithm. Chapter 6 presents the experimentation set up, evaluation metrics and results obtained. Chapter 7 concludes the work and guides the direction for future work.

Chapter 2

Literature Survey

This chapter will present a comparative analysis of available Wireless Mesh Networks (WMNs) for IoT applications. Based on the analyzis of these WMN, the most suitable one for EADS device will be chosen. Additionally, as this thesis deals with a battery-less device, energy consumption is of paramount importance. Therefore, it is essential to efficiently utilize the available energy of EADS device. A literature study is carried out on various Energy-Aware Scheduling algorithms for battery-less devices with energy harvesting.

2.1 Wireless Mesh Networks (WMN) for IoT

WMNs in IoT networks are easy to configure and manage, provide self-healing algorithms, and are flexible and easy to scale. However, mesh networks incur issues such as channel access problems, load balancing and route management [1]. Moreover, selecting an appropriate mesh network depends on various factors such as application, routing protocol, energy consumption, cost, and the number of devices to be connected. The following describes a comparative analysis of the available IoT mesh networks.

- **WiFi**
  WiFi operates in 2.4 GHz frequency where the band is divided into multiple channels, and only one channel can be used for transmission [18]. The devices running WiFi can transmit up to a range of 95 meters [18]. WiFi allows node mobility, low maintenance, and low-cost infrastructure. It also provides easy node addition to an existing network. WiFi devices may however observe interference from other wireless technologies as WiFi operates in 2.4 GHz [19]. WiFi devices consume almost 10 times more energy compared to other widely used protocols such as Zigbee and Z-Wave [18]. Furthermore, WiFi uses star network and if the path between the central hub and node is cut off, the node can no longer communicate with the hub [13]. As scalability of EADS is one of the main issue addressed in this thesis, WiFi cannot be used due to its inability to support mesh architecture and high power consumption of the nodes.

- **Cellular (5G, 4G/LTE, 3G)**
  Cellular networks allow reliable broadband communication supporting various voice calls and video streaming applications. One of the main disadvantages of cellular networks is that they impose very high operational costs and power requirements [29]. Most of the IoT applications involve battery-powered intelligent devices, thus making cellular networks unsuitable for use. Furthermore, as EADS is a battery-less device, energy consumption should be reduced as much as possible. Hence, it can be concluded that the power requirements of Cellular networks are not suitable for EADS.

- **BLE Mesh**
  BLE Mesh standard released in 2017 by SIG, operates in 2.4 GHz frequency. Unlike BLE that provides two different communication ways among the devices: advertising and connection-oriented mode, BLE Mesh was implemented by a later approach [5]. BLE Mesh networking is based on the flooding technique, ensuring that each intermediate node M can relay incoming messages to the destination node. Moreover, mesh nodes should have a 100% duty cycle for scanning incoming packets on the advertisement channels. The devices advertise their beacons to find each other and repeat the same message three times in three different channels to ensure that at least one device hears the message. This results in collisions when the number of devices in the network increase thus lacking synchronized communication [36]. Hence, due to the flooding mechanism and continuous scanning of devices which requires a 100% duty cycle, may lead BLE Mesh devices to consume more energy which is not a feasible option for battery-less device EADS.

- **Thread**
  Thread is a mesh networking standard based on IEEE 802.15.4, operating in 2.4 GHz frequency, designed for a low rate and low power transmission. Thread protocol is capable of supporting up to 16352 devices [35]. Unlike Zigbee and Z-Wave, Thread does not require a hub device as Thread devices can communicate with each other and directly to the cloud. This reduces the overall cost and infrastructure needs as a gateway or hub is not needed for the interconnection between the network and cloud [5]. In [33], authors examined performance analysis of Thread at Philips Lighting on multicast (flooding) messages in a two-hop network with 100-256 nodes. A proportional increase in the end-to-end latency was observed with increasing node density and their distance from the source. Thread-enabled devices connect without a single point of failure. If a device becomes unavailable, then the Thread network can automatically re-configure [8]. However, the network should be equipped with redundant routers to powered devices to prevent failure, thus increasing the overall cost. The power consumption of Thread devices is $\sim 150\mu$W [5] which is greater than Zigbee ($\sim 125\mu$W). Hence it can be concluded that Thread would not make a good candidate for achieving scalability of EADS devices.

- **Zigbee Mesh**
  Zigbee is based on IEEE 802.15.4 standard and operates in 2.4GHz, 900 MHz and 868 MHz. In the areas of power consumption, Zigbee is nearly similar to BLE i.e., $\sim 125\mu$W and $\sim 150\mu$W respectively [5]. However, one advantage of Zigbee over BLE is that Zigbee also operates in a sub-GHz frequency which means it will have less interference from Wi-Fi signals. As the line-of-transmission of Zigbee is 100 meters, which is greater than BLE (10-90 meters), which makes easy remote access management of Zigbee nodes. Authors in [25] design an indoor experiment with Panasonic's PAN802154 module to test Zigbee functionality. It was observed that as Zigbee uses a layered approach, the test results had considerable overhead for message processing across layers, thus resulting in increased latency and reduced bandwidth utilization. Furthermore, the results also show that reception is slower than transmission. Additionally, it was pointed out that although the theoretical maximum size of the Zigbee network is over 65000 nodes, the bandwidth will limit the number of devices in a mesh network, and added delays would lead to significant overhead. Furthermore, the experiments in performed [25] show that with maximum message size, the data rates of 8.3 kbps can be achieved whereas, the theoretical value is 250 kbps, thus reducing the data rate. Due to the limitations of Zigbee addressed above, it cannot be used as a mesh network for EADS as the effect of growing network would significantly impact the functionality of Zigbee.

- **Z-Wave**
  Z-Wave is a wireless mesh network protocol, that uses low-energy radio waves to communicate between devices that operate in 908 MHz frequency. Z-Wave provides low-latency transmission of small packets at data rates up to 100 kbps over a distance of about 30 meters. Due to the small data size, Z-Wave can only be used in communication such as monitoring and control. Furthermore, unlike Zigbee, Z-Wave can only support up to 232 nodes. As Z-wave offers less coverage, the maximum number of hops is limited to four. Therefore, as a result, more Z-Wave devices should be added to the network that can act as routers, thus increasing the overall cost. Furthermore, Z-Wave devices are battery intensive and consume more power ($\sim 175\mu W$) compared to BLE ($\sim 150\mu W$) and Zigbee ($\sim 125\mu W$) [2].

- **Wirepas Massive**
  Wirepas is a wireless, fully meshed network architecture for low-power IoT applications. It is a decentralized protocol - every node in the network selects its role, maintains, and optimizes the connections autonomously based on its environment [37]. This feature enables a reliable, optimized, and scalable network. Wirepas operates in unlicensed 2.4 GHz frequency with multiple frequency channels (40 channels) [37]. Each router can choose a channel dynamically and independently, so one network can use all 40 channels in parallel. In a Wirepas Massive network, every node can behave as a router, thus forwarding the packets from one node to another. This feature of Wirepas Massive enables the connectivity of numerous nodes in a network, thus achieving scalability. In [37], the authors describe that over 700,000 smart meters were connected in a single mesh network which is believed to be the largest wide-area mesh network deployed on the planet. The authors also mention that more than 1000 radios per cubic meter were proven without collisions. Wirepas Massive network was deployed in a forest with 150 battery-operated nodes in low energy mode with many trees and obstacles. The entire network was supported with a single gateway. It was observed that the routers in the network consumed 45 μA power which is significantly less [24].

Parameters	**BLE Mesh**	**Thread**	**Z-Wave**	**Zigbee Mesh**	**Wirepas Massive**
No. of channels	3	16	16	16	40
Operating frequency	2.4 GHz	868 - 915 MHz	800 - 900 MHz	868 - 915 MHz	2.4 GHz
Maximum hops	+++	+++	- -	+++	+++
Range	+	+	+++	+	++
Network Capacity	- -	+	+	++	+++
Data Rate (bps)	+++	+	- -	+	+++
Routing	NA	+++	+++	+++	+++
Flooding	+++	+++	- - -	+++	+++
Power Consumption	$\sim 150\mu$W	$\sim 100\mu$W	$\sim 175\mu$W	$\sim 125\mu$W	$\sim 75\mu$W

Table 2.1: Comparison of different Wireless Mesh Networks, where (+++) - highest, (++) - moderate, (+) - low and (−) - bad.

Table 2.1 compares WMNs discussed above in terms of operating frequency bands with the number of channels, scalability, data rate, number of hops, range, and message transmission mechanism (routing or flooding). Thread, Zigbee, and Wirepas can use both routing and flooding. BLE Mesh only uses flooding, thus making it unsuitable for this project as flooding can consume more energy and hence is excluded. Although Wirepas operates at 2.4 GHz frequency, it has 40 channels, where nodes can choose any available channel, enabling transmission in parallel. Wirepas has the highest data rate compared to other mesh networks except for BLE, which has already been eliminated. Furthermore, unlike Thread (16352) and Zigbee Mesh (65000), Wirepas can support more than one million nodes with a single gateway in the network reducing infrastructure and overall cost, depending on the availability of hardware memory [37]. Thus, after analyzing the available mesh networks and looking at the specifications for each, it was decided to use the Wirepas Massive network to achieve the scalability of EADS.

2.2 Energy-Aware Scheduling in battery-less devices

The lifetime of IoT devices can be improved with low-power technologies and modeling energy consumption. However, there is still a problem when the available energy is spent. Therefore, battery-less devices such as EADS use energy harvesting to operate and increase the energy efficiency and lifetime of the devices. Battery-less devices harvest energy from their environment and store it in capacitors. Given the behavior of capacitors, the energy stored is unpredictable and inconsistent. As these capacitors are of small size, they cannot store a large amount of energy which leads to the erratic behavior of these devices [28]. To reduce the risk of EADS shut down, it is essential to keep track of available energy. The scheduling algorithms can be classified into three main categories:

- ***Dynamic Voltage and Frequency Scaling (DVFS)***
  The power consumption of the sensor hardware depends on the supplied voltage and also on the operating frequency. Hence, these two parameters can be controlled to reduce power consumption. Using DVFS on a sensor can help in maximizing the performance given the energy budget or minimize the energy consumption [30]. However, DVFS algorithms are not suitable and challenging to implement in energy-harvesting sensors due to the resource-constrained hardware [7]. Therefore, alternate task scheduling is required to work without additional overhead in terms of resources and energy. Hence the proposed algorithm will not consider the DVFS category of task scheduling.

- ***Decomposing and combining of tasks***
  In this category, the tasks can be decomposed, i.e., split into atomic subtasks. Without this division, it would be challenging to execute energy-intensive tasks continuously, and there would not be enough energy to execute future tasks. Although tasks can be decomposed or composed to save energy, the harvested energy is insufficient to run all-ready tasks. In this case, a controlling mechanism executes these tasks depending on the priority [30]. Similarly, the proposed algorithm splits tasks into atomic subtasks that are executed based on available energy and priority, thus preventing the system from shutting down during the execution of large energy-intensive tasks.

- ***Duty Cycling***
  The duty cycling mechanism is one of the most common methods for minimizing energy usage in battery-less devices. In most of the traditional IoT protocols, duty cycling is based on the number of tasks, their energy consumption, and remaining energy of the node [14] [30]. This approach cannot be used in battery-less devices due to dynamic harvested energy. Therefore, in battery-less devices, duty cycling is based on the incoming harvested energy to efficiently utilize the energy for task execution on a node [7]. Furthermore, the time a node spends in sleep mode depends on the harvested energy in the future to plan the consumption of incoming energy for required operations. In the proposed algorithm, the duty cycling mechanism will be employed that allows the node to sleep intelligently to harvest more energy that will aid in the execution of future tasks.

Based on the description provided for task scheduling in energy harvesting devices, duty cycling and task decomposition mechanisms will be used for the EADS device task execution. Alpaca [22], Mayfly [16], InK [38] and Optimal energy-aware scheduling [7] are the state-of-the-art task-based schedulers. Alpaca and Mayfly only consider static task flows, and if a task cannot be executed due to energy level, it will be executed again. However, if a specific task is unable to execute or the energy conditions of the harvester change, any other task will starve, waiting for the current one to be executed over and over again. To address this issue, InK considers a dynamic and priority-based scheduler that allows the application to adapt to changes in available energy. However, InK is not energy-aware and does not consider deadlines. Optimal energy-aware scheduling considers task priorities, deadlines, and available energy to decide which task should be completed first, making it more energy-efficient.

It also uses task decomposition and duty cycling approaches to prevent the system from a complete shut down. The proposed algorithm is inspired by this, where available energy and task priorities will be considered before executing any task. Additionally, the proposed algorithm will also estimate the sleep-time and be implemented in a real-time scenario. The results obtained for the optimal algorithm are based on a simulated scenario. As the energy harvested in the supercapacitor is dynamic, many factors may influence the stored energy and the charging rate, thus making it more challenging to implement in a real scenario as is the case for EADS device.

Chapter 3

Working of Wirepas Mesh

Scalability is one of the essential requirements for IoT applications. It defines the ability of a network to integrate the growing number of nodes in the network with an already existing one. As this thesis focuses on the minimum power consumption of EADS device, a protocol capable of achieving low-power EADS scalability is - *Wirepas Massive*. The motivation for choosing this protocol is elaborated in Section 2.

Wirepas Network basics

Wirepas Massive supports decentralized mesh wireless communication protocol. Decentralized means that each node in the network selects its role maintains, and optimizes the connections autonomously based on its environment. The devices that run Wirepas Massive monitor the interference and automatically choose the best frequency channel available for communication. Additionally, the devices with Wirepas Massive can change channels without affecting the rest of the network. Wirepas Massive offers throughput up to 150 packets per second with 102-byte payload with 1 Mbps bandwidth per gateway [24]. However, the network throughput can be increased further by connecting multiple gateways to the network, allowing automatic load balancing.

Security and Reliability

Wirepas Massive network is resilient and reliable because there is no single point of failure. If one node fails, the rest of the nodes re-organize automatically to find an alternate route to a gateway. Even if a gateway breaks down, the nodes can find a route to another existing gateway. Wirepas Massive network is reliable as each packet can be sent point-to-point to the next hop with acknowledgment and automatic re-transmissions. This feature assures 99.9% packet reliability in end-to-end transmissions in most of the use cases [24]. Wirepas Massive network is secure as it uses industry-standard AES128 encryption in all messaging. An unauthorized party cannot join the network, modify messages, or read the message content. Different encryption keys can be set to each separate network. Furthermore, the Wirepas Massive network uses cost-based routing, enabling each node to find the best path to the gateway. As the routing is cost-based, no routing tables are needed to store in the device's memory, thus providving support for extensive networks. Communication between nodes is also split into time slots to optimize the channel usage of multiple nodes while minimizing power consumption.

Scalability and Density

Wirepas Massive is designed to work in a large-scale or dense environment. It uses the same physical layer as Bluetooth Low Energy (BLE) in the 2.4 GHz spectrum as shown in Figure 3.1. Also, Wirepas Massive uses the same modulation (Gaussian Frequency-Shift Keying (GFSK)), and the same channel width (2 MHz). However, unlike BLE that uses only three channels of the spectrum, Wirepas uses a total of 40 channels, where one is allocated to the network discovery channel and the remaining 39 are available to be used by the routers. Each router selects the channel locally and independently. The independent channel measurement allows optimized spectrum usage and interference avoidance. Communication between nodes can also be split into time slots to optimize the channel usage of multiple nodes while minimizing power consumption.

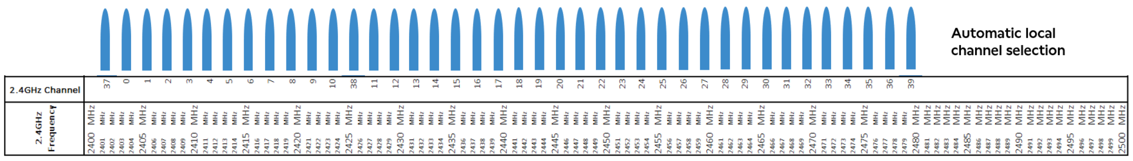


Figure 3.1: Wirepas Massive and BLE frequency channels [24]

Network Organization

A Wirepas Massive network consists of gateways, sink node, and nodes as shown in Figure 3.2.

- *Gateway*:
  The gateway acts as a bridge between the mesh network and the backend (internet). If the system has multiple gateways, the mesh network organizes automatically around them making Wirepas Massive a multi-gateway system. The gateway can use one or more sink nodes to connect to the network.

- *Sink Node*:
  The sink node is responsible for transmitting the data between the mesh network and gateway. The sink node is connected to the gateway via UART or SPI to route the backend and Wirepas Massive network traffic. If a new gateway/sink is added to the network, the nearest nodes find the "cheapest" path, thus balancing the network load automatically to maximize the throughput without any user interaction.

- *Nodes*:
  The nodes of a Wirepas Massive network can act as a router node or a non-router node. By default, nodes are in automatic role selection called Auto Role. In this role, the nodes select autonomously if they route traffic or not.

Communication services

Wirepas Massive supports communication in three directions (as seen in Figure 3.3): (a) *Uplink*: from nodes to back end, (b) *Downlink*: from back end to nodes, and (c) *Intra-network*: between nodes inside the network (only allowed with CSMA-CA mode). Furthermore, Wirepas supports three addressing modes: (a) *Unicast* messaging to a single node, (b) *Multicast* messaging to a group of nodes and (c) *Broadcast* messaging to all nodes.

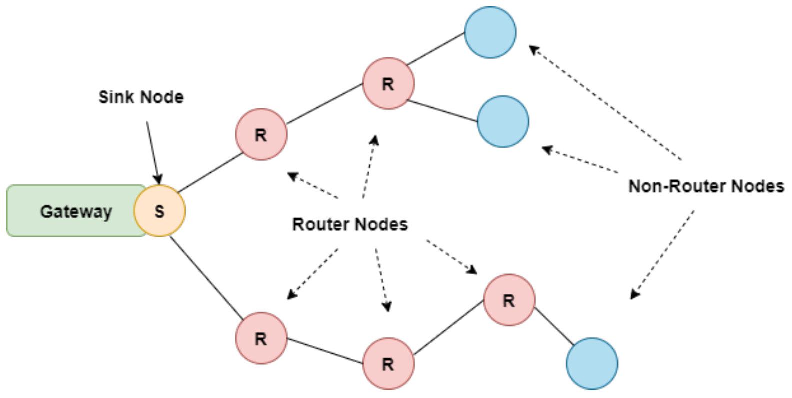


Figure 3.2: Wirepas Massive network components

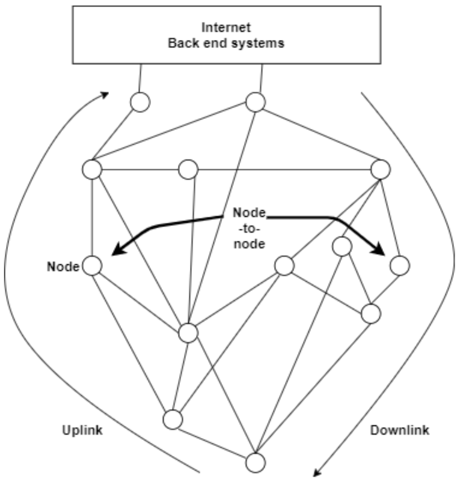


Figure 3.3: Communication directions supported by Wirepas

Operating Modes

Wirepas offers two operating modes as shown below. Additionally, it is also possible to have a network with both operating modes. For instance, a lighting network can use a combination of both operation modes.

1. *Multi-channel CSMA-CA mode:*
   This operating mode, known as Low-Latency mode, is used in applications requiring low latency and/or high throughput. Here, the nodes are always *on* to provide as low latency as possible. For instance, gateways are in low latency mode most of the time to send the information from the network to the backend or vice-versa.

2. *Multi-channel time-slotted mode:*
   This operating mode, also known as Low-Energy mode, is generally used for battery-operated devices. The communication in this type of network happens in synchronous time slots and the devices are asleep the rest of the time to save energy. An example of this mode can be an asset tracking system where all the sensors are battery-operated. Only the node that sends the data can be active while other nodes can be asleep thus conserving energy.

Open joining and Provisioning

To join a Wirepas network, the nodes should have network and device parameters. Network parameters are network-wide and only the nodes with the same network parameters can connect. The network parameters a node should possess are as follows:

- *Network address*: Network identification
- *Network discovery channel*: To use one channel among 40 channels for node discovery and connection.
- *Security keys*: AES128 encryption key

Similarly, the device parameters are for the nodes and are as follows:

- *Node address*: A 32-bit address is used to identify each device and should be unique in the network.
- *Application Area ID*: Used to separate different devices' firmware during the firmware update process. The ID is the same for the given hardware/software combination.
- *Bootloader keys*: Used to secure update images and are common to a manufacturer or a network operator

If a node does not have matching network parameters, it cannot join the existing network with the normal connection. Wirepas offers a Provisioning protocol to be able to join a provisioned network. When Joining is enabled, the router nodes are set to send an open joining beacon transmitted on the same configurable channel with the same network address. The beacon includes the minimal parameters of the network for the joining node to connect. The joining node is set to receive these beacons on the joining network channel. If the joining node receives the beacon, it sends a handshake signal that the stack transmits to the gateway and the backend. The backend can then decide to include the new node in the existing network. If approved, the node applies the network parameters and boots, thus joining the network. Figure 3.4 shows the provisioning message flow of a joining network.

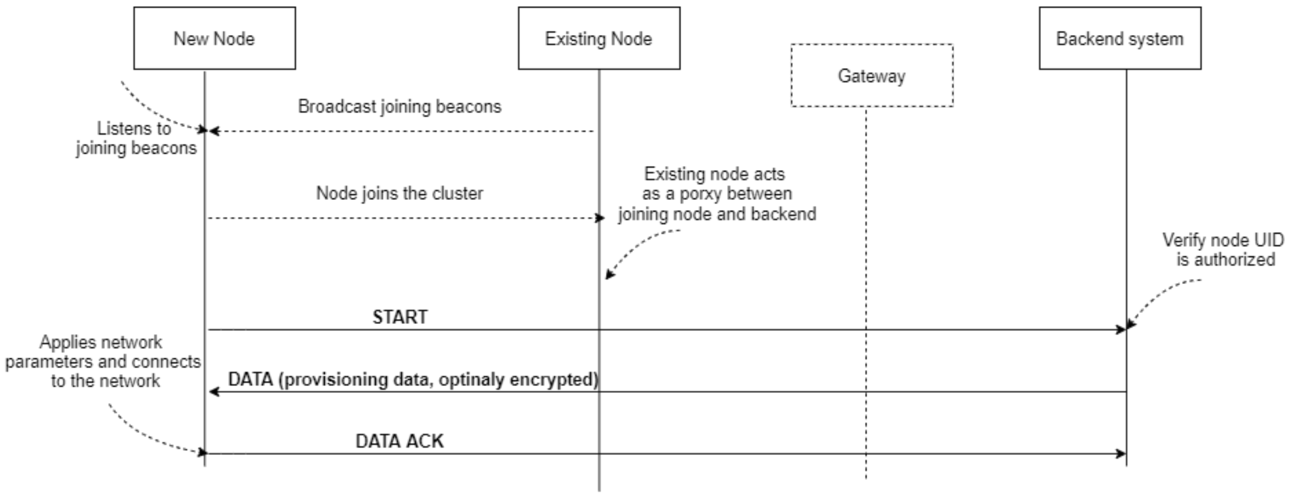

New Node
Existing Node
Gateway
Backend system
Broadcast joining beacons
Listens to joining beacons
Node joins the cluster
Existing node acts as a porxy between joining node and backend
Verify node UID is authorized
START
Applies network parameters and connects to the network
DATA (provisioning data, optinaly encrypted)
DATA ACK

Figure 3.4: Provisioning message flow [12]

Wirepas scheduler

Wirepas Stack includes Wirepas Massive communication stack and Wirepas scheduler for enabling the application operating on the same MCU. Wirepas Massive scheduler provides priority-based cooperative scheduling i.e., all the tasks are run to completion. The tasks are scheduled based on their priorities and their execution times. The mesh stack has strict real-time requirements such as accurate synchronization of messaging which has the highest priority. Therefore, the application tasks such as display tasks should be executed when no Wirepas tasks are executed. The maximum time given for a single application task should not exceed 100 ms. If the application task exceeds this time it would affect the whole system performance and may result in incorrect operation. As the stack scheduler cannot preempt a task, it should be guaranteed that a task finishes on time.

Experimental setup

In this work, a Wirepas Massive network with a gateway, sink node, and router and non-router nodes will be established to support the scalability of EADS devices. Various experiments will be carried out to observe the behavior of EADS under different betwork configurations. For instance, the EADS client will be configured as a router and non-router nodes, and the energy consumption will be measured. Furthermore, the node configurations will be modified to Low-Latency and Low-Energy mode to determine the accurate difference in energy consumption. Additionally, the EADS client will use *Uplink* and *Downlink* communication directions to send/receive data from the backend and vice-versa. In this thesis, *Tree topology* will be used as the calendar information has to be sent from the gateway to the nodes, whereas Flat topology does not have a gateway. To form a Wirepas Massive network, all the network and device parameters will be configured before the nodes join the network. Furthermore, a description of how Wirepas Massive network is established and the configurations are varied will be explained in the following sections.

Chapter 4

Methodology

This chapter formulates the concept and design of EADS. The architecture and modules of this system are defined and the design choices made in the selection of hardware are identified. The software architecture is developed in C and Python programming languages.

4.1 Overview of the proposed system

The thesis aims at creating a scalable battery-less meeting room display system. The system comprises various components such as a server, solar panel, supercapacitor, power management IC, processing unit, and display. The server is known as an EADS server, whereas other hardware modules (solar panel, supercapacitor, power management IC and processing unit) are together termed as EADS client. EADS operates on the energy stored in the supercapacitor from the energy harvester i.e., indoor solar panel. The power management IC regulates the solar panel voltage and supercapacitor voltage. Once the supercapacitor is charged to the minimum voltage required by the processing unit to turn on, the processing unit can transmit/receive messages to/from the server. The EADS server connects with the Office 365 cloud server, residing in the cloud, that consists of organization email accounts. This cloud server requests calendar information for every room located at the Capgemini Engineering office in Eindhoven. Later, the EADS server requests this calendar information via an Internet connection from the Office 365 cloud server. The EADS server further sends this information to a processing unit via a wireless mesh protocol (for scalability). Furthermore, the display communicates with the EADS client through the Serial Peripheral Interface (SPI) protocol to display the calendar information such as meeting name and meeting time. Figure 4.1 shows the overview of the proposed system. As seen in the figure, the EADS client will be placed at the entrance of the meeting rooms (Meeting rooms A and B) at the Capgemini Engineering Eindhoven office. And the EADS server will be deployed at an optimum central location in the office building. Furthermore, it is also important to monitor the behavior of these devices when distributed over wide-area such as an entire building. The EADS mesh network will be finally geographically mapped for easy maintenance and monitoring. Section 4.2 and Section 4.3 give a detailed explanation of how the functionality and goal of the proposed system will be achieved.

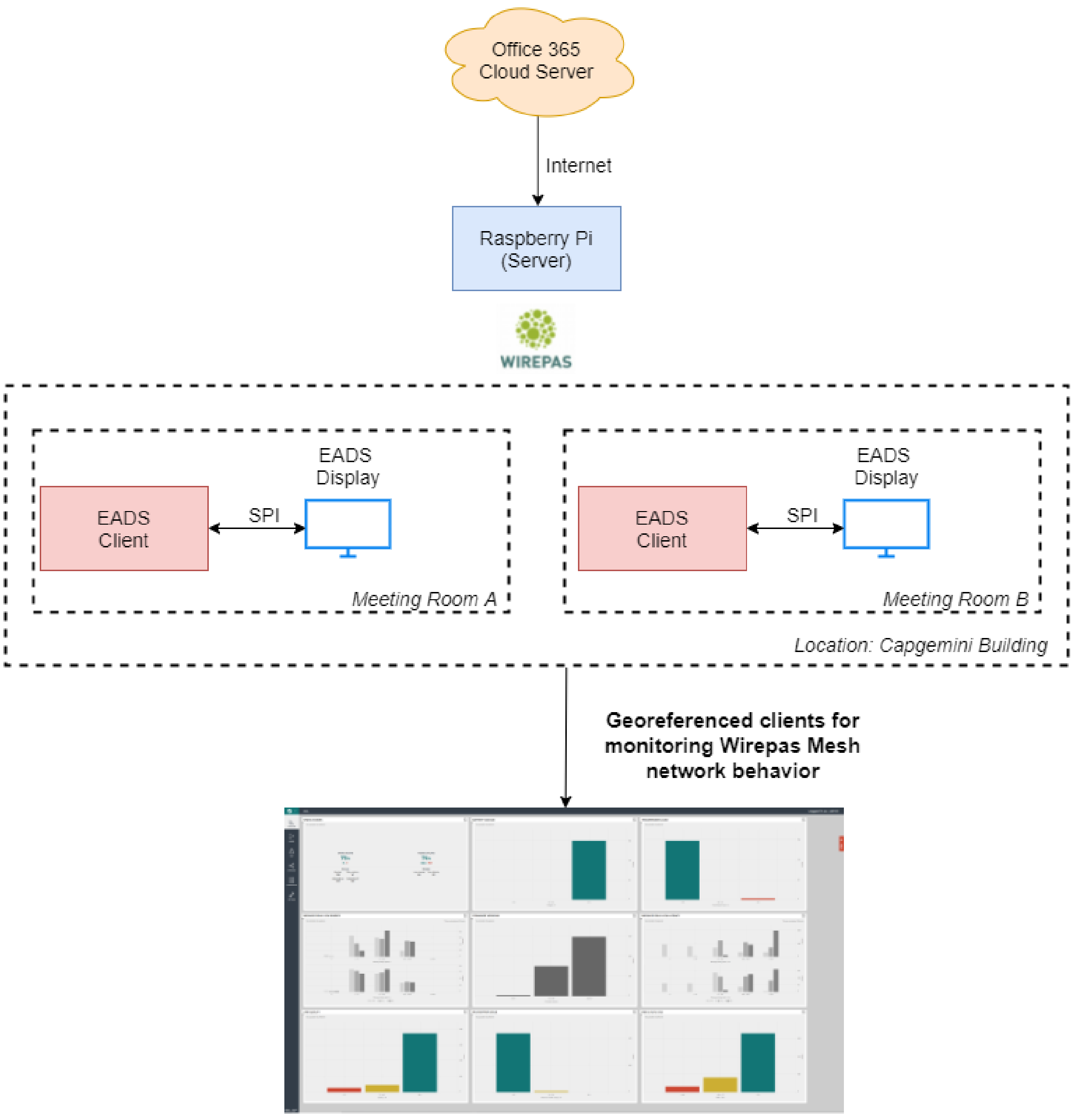


Figure 4.1: Schematic diagram representing the proposed system

4.2 Scalability of EADS

Scalability is one of the crucial factor of an IoT network as it defines the ability of a network to integrate the growing number of nodes in the network [15]. One of main focus of this thesis is to address the scalablity issue of EADS. To form a high density network with multiple EADS devices spread across an entire building, a wireless mesh protocol should be utilized. Mesh protocol is an interconnection of all nodes connected with all other nodes in the network. Nodes between the source and destination can be involved in forwarding the messages to destination node. In contrast to other IoT topologies, such as star topology where each node has only path, in mesh topology, each node can be reached through multiple paths in a network. Thus, a mesh network is more reliable as single node failure would not effect the entire network. As discussed in Section 2.1, the communication between the EADS server and the EADS client is through Wirepas Massive protocol. The EADS server and EADS client can only communicate via Wirepas if they are equipped with a working Wirepas stack that includes Wirepas Massive communication stack.

The framework created for the communication between EADS server and EADS client should have two functionalities as follows.

- EADS server sends calendar information to the nodes (EADS client) every 15 minutes.
- When a node receives the calendar information, it sends *1* on successful reception.

To analyze the behavior of growing EADS network on energy, a set of experiments have to be conducted. Practically, it would be challenging to show behavior of 50 or 100 EADS clients inside a building. Therefore, the number of clients in the network will be limited to 4 as it would be expensive to have more than 4 nodes for a prototype testing. Initially, a single node N_1 will be connected to a network where it receives calendar information from the gateway at a distance d_1 and forwards it to the display (see Figure 4.2). Secondly, an additional node N_2 will be added to the already existing network of N_1. The distance d_2 between these two nodes will be varied and changes (if any) in the energy consumption will be observed. This procedure will be repeated for a maximum of four nodes. Furthermore, Wirepas Massive nodes supports two configurations: router and non-router nodes. The router node is responsible for relaying the information to the destination nodes, whereas non-router nodes do not route any messages and can only receive and transmit messages destined to itself. This functionality of Wirepas will be employed to see how the node configurations impact the energy consumption on the nodes.

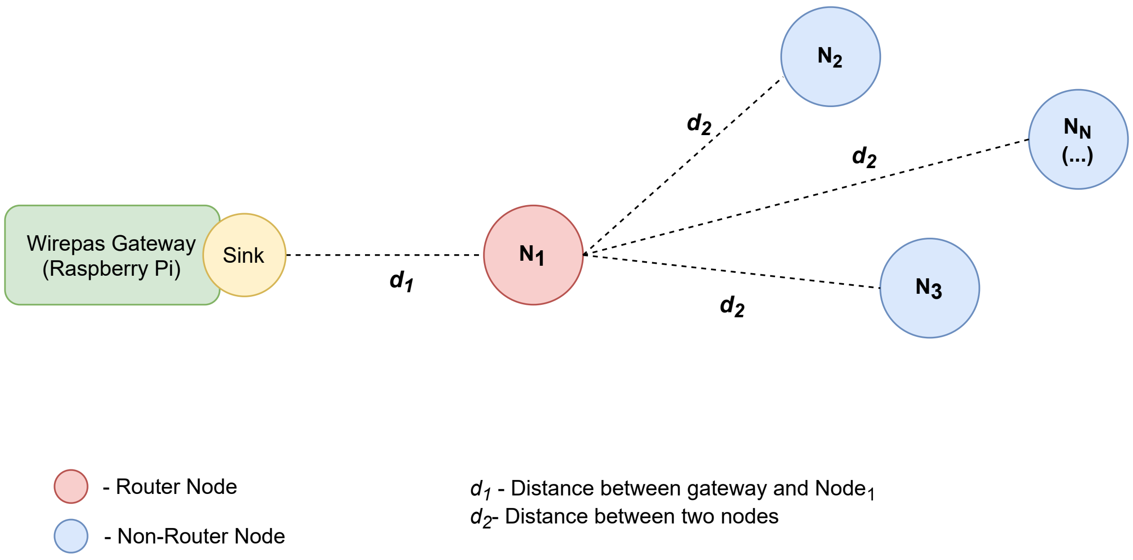

Wirepas Gateway (Raspberry Pi)
Sink
d1
N1
d2
N2
d2
NN (...)
d2
N3
- Router Node
- Non-Router Node
d1 - Distance between gateway and Node1
d2- Distance between two nodes

Figure 4.2: Schematic representation showing distances d_1 and d_2, number of nodes *(N)* and node configurations (router & non-router).

When the number of nodes in the network increase, depending on the distance between the nodes, the number of hops may increase or decrease. The increase in the number of hops may also lead to increase in the energy consumption of a node involved in relaying the message to the destination node. Therefore, the total energy consumption is the sum of energy consumed by Wirepas stack and the energy consumed for SPI communication. Additionally, the node configurations may also have an impact on the energy consumption and should also be considered. Initially, all the nodes in the network will be configured as router node. The main reason for selecting this configuration is because all the relaying node may not have sufficient energy to relay the message destined to some other node. As a result, this may lead to increase in the latency of message reception on the destination node with limited router nodes in a battery-less device network.

Various combinations of router and non-router nodes will be explored to observe the behavior on the energy consumption of the nodes. The results for the proposed hypotheses for variations in energy consumption obtained by varying the parameters - number of nodes (N), distance between sink node and router node (d_1), distance between two nodes (d_2) and node configurations will be discussed in further sections.

4.3 Energy-Aware Scheduling Algorithm

Energy-Autonomous Display System (EADS) is a battery-less device and it is important to keep track of available energy given the intermittent nature of energy storage (supercapacitor). EADS server sends the calendar information every 15 minutes to the nodes to update the display with new information. This information should be further sent to the display where the node is unaware of the available energy. When the node is executing a task, the system may shut down halfway during the task execution due to minimum charge in the energy storage. Furthermore, this interrupted task execution also leads to wasting of scarce available energy on the task chains that cannot be completed on time anyway. Hence, to avoid system failure due to low-power and efficiently utilize the energy stored, an intelligent Energy-Aware Scheduling (EAS) algorithm is essential. With EAS, EADS can keep a track of available energy, input power and energy consumption of selected tasks. With this information, EAS can determine whether a function can be executed without the risk of system failure due to low power. The scheduler keeps a track of available energy before and after execution of each task. The task is executed only when the available energy is greater than the energy required to run that task. Furthermore, the available energy should also be greater than the threshold voltage to ensure system does not fail due to low power. When the task is executed, the scheduler updates the energy available. Furthermore, if the energy stored is insufficient to execute the task, scheduler will estimate the required sleep time.

EADS tasks selection

To create such a framework, the first step would be to know which tasks are performed by the MCU and the energy required per task. For instance, some of the tasks that can be considered are: forwarding the received data on node to display and refreshing the display. It is also important to note that Wirepas Massive network discourages the user to control the Wirepas stack tasks, thus limiting the number of tasks to be scheduled by the proposed scheduler. Therefore, only the application tasks on the node such as display tasks can be managed by the scheduler. Depending not only on the energy available and energy consumption per function, but also task priority, the proposed scheduler will determine which task should be executed and when. The main goal of this scheduler is to execute maximum number of tasks, prioritizing its priority status. The tasks are modeled as atomic tasks, where every task is defined with a priority and energy required for its execution. These tasks are also characterized by an order. For example, Task 3 can only be executed when Task 1 and Task 2 have been executed (see Figure 4.3).

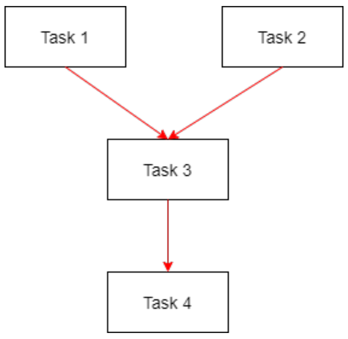


Figure 4.3: Overview of atomic tasks model

4.4 Pseudocode for Energy-aware Scheduling algorithm

The pseudo code and execution flow for achieving the energy-efficient task scheduling for EADS can be seen below.

1. Initially, EADS server retrieves calendar information from Office 365 server through internet. EADS server forwards this received information to the nodes. It should be noted that the energy consumption on the server is not considered as it will be connected to the main supply. This information is received by the node through Wirepas. Upon successful reception, the node sends 1 and on unsuccessful reception the node sends 0 to the server.

2. As the tasks to be scheduled are the static display tasks, before scheduling a task, the node and display SPI pins should first be initialized. Furthermore, foreground and background colors are set and the display driver is initialized.

3. Next, the display tasks are sorted according to their priorities where the first task has the highest priority. The tasks are given priority depending on the order of their execution. For instance, getting the information to display cannot be done after updating the display.

4. For every task (*TASK*) in total number of tasks (*No._of_Tasks*), the following functions should be executed:

    - To determine the available energy in the energy storage, the ADC value on the analog pin of node must be first read. The function `read_available_energy_adc()` returns the ADC value.
    - The returned ADC value is then converted to a voltage value and later to energy in Joules by the function `calculate_available_energy()`.
    - The energy required per function is then calculated to be compared with `calculate_available_energy()`.
    - Furthermore, a threshold value is set to avoid the risk of system failure during task execution. Task *TASK* can only be executed if the available energy is greater than energy required for TASK and set threshold value. After task execution, the available energy is reduced as the task has consumed some energy. Therefore, `update_available_energy()` updates the available energy after every task execution.
    - If the available energy is less than energy required for *TASK*, `estimate_needed_sleeptime()` function estimates the sleep time given the stored energy.

Hence, given the working of the EAS algorithm, this algorithm can be proved efficient if the task success rate improves compared to the operation of EADS without EAS. Section 5.4 will provide further explanation on task prioritization, energy measurement and effectiveness of proposed algorithm.

Algorithm 1: Energy-Aware Scheduling algorithm

```
1: procedure
2:     send_calendar_info_command()                                  ▷ from gateway to node
3:     send_response()                                               ▷ from node to gateway
4:     sorted_tasks = sort(No_of_Tasks) based on high priority
5:     for $TASK$ in sorted_tasks do
6:         read_available_energy(Analog_Pin);
7:         calculate_available_energy();
8:         check required_energy_per_task($TASK$);
9:         if (calculate_available_energy() > required_energy_per_task()) && (calculate_available_energy() > Threshold) then
10:            execute $TASK$;
11:            update_available_energy();
12:            return 1
13:        else
14:            estimate_needed_sleeptime();
15:            return 0
```

4.5 Design choices

This section explores various choices made during the selection of hardware modules used for the EADS device.

- **Solar panel**
  As mentioned in the previous sections, EADS harvests its own energy from the environment. Hence, the EADS must be energy-sufficient to operate without any additional battery. EADS uses solar panel also known as photo-voltaic cells, that convert the photons from indoor light into electrons generating electricity. Depending on the requirement for area and the ability to generate maximum power with limited lux, Powerfilm LL200 2.4-75 was chosen. This particular solar panel with an area of 65.51 cm^2, generates power of 47.646 μW. The generated power with this Powerfilm is larger compared to other solar panels such as Powerfilm LL200 3-37 and Panasonic AM-1815CA generating power of 25.5 μW (41.95 cm^2) and 33.031 μW (28.24 cm^2) respectively.

- **Power Management Integrated Circuit (PMIC)**
  A PMIC ensures that the energy from a solar panel is stored in a supercapacitor. This stored energy is then converted into a usable output to power a system. Efficiency is one of the main parameters for choosing a PMIC and can be calculated with equation 4.1.

$$\eta = \frac{P_{out}}{P_{in}} * 100\% \tag{4.1}$$

  The efficiency for the chosen solar panel (Powerfilm LL200 2.4-75) combined with Analog Devices ADP5091 (PMIC) was 86.6%. This efficiency was greater compared to other PMIC and solar panel combinations such as AEM10941 + LL200 2.4-75 and BQ25570 + LL200 2.4-75 giving efficiency of 76% and 79.8% respectively. Furthermore, it is desirable to have to two different outputs from PMIC as the supply voltage for other hardware varies. One of the output is for the processing unit as the minimum supply voltage required is 1.8 V. And the other output is to the display as it operates at a minimum voltage of 3.0 V.

- **Supercapacitor**
  The minimum storage capacity has to be calculated to find which capacitor is the most suitable for EADS. The nominal voltage for most of the capacitors between 100 mF and 1 F is 5.5 V. However, it was chosen to charge the supercapacitor to a maximum of 4.5 V so that the lifespan is longer. Additionally, supercapacitor drains fastest when charged to its maximum voltage. The lower voltage limit is set to 2.0 V as ADP5091 PMIC requires a minimum voltage of 2.0 V on the supercapacitor to provide an output voltage.

  A typical working day averages from 9.00 to 17.00 hours, limiting the lighting availability to only 8 hours as the lighting will be off for 16 hours. EADS device cannot generate any energy when the lighting is off. The average current used by the system is 15 μA [3]. The equation 4.2 can be used to calculate the value of capacitance.

$$\begin{gathered} C = \frac{15\mu * (16 * 60 * 60)}{(4.5 - 2.0)} \\ C = 0.35F \end{gathered} \tag{4.2}$$

  Based on the capacitance value calculated, it was decided to use a 0.47 F supercapacitor. Furthermore, the total time that the system can operate on a full supercapacitor can be calculated using the equation 4.3.

$$\begin{gathered} t_{back-up} = \frac{C * (V_{max} - V_{min})}{I_{back-up}} \\ t_{back-up} = \frac{0.470 * (4.5 - 2.0)}{15\mu} \\ t_{back-up} = 21.76 hours \end{gathered} \tag{4.3}$$

  Leakage current and voltage drop must be taken into account to determine the exact duration usable capacity of the supercapacitor. When the supercapacitor capacity decreases, leakage current also decreases. Furthermore, energy can be saved outside the office hours. If no one is present at the office, no more bookings will have to be made and therefore no one to read the screen. With this, the energy can be saved if the display is turned off and the server no longer sends calendar information to processing unit during night. Considering these factors, the capacity can be recalculated with equation 4.4.

$$\begin{gathered} C = \frac{3.5\mu * (16 * 60 * 60)}{(4.5 - 2.0)} \\ C = 0.081F \end{gathered} \tag{4.4}$$

  Therefore, based on two different capacitance values calculated from equation 4.2 and equation 4.4, it was decided to use both 0.47F and 0.081F supercapacitors to check the working of EADS.

- **Processing unit**
  The processing unit is the heart of EADS, responsible for monitoring the energy consumption, receiving/sending messages from/to server with radio and providing SPI peripheral to display. The most popular ARM Cortex MCUs with radio commercially available are EFM32GG11 (Silicon Labs), STM32L4S7 (ST Microelectronics) and nRF52840 (Nordic Semiconductors). As EADS should have minimal energy consumption, the energy consumption of CPU in active mode was considered. It was observed that nRF52840 CPU has the lowest energy consumption (52 μA/MHz) compared to EFM32GG11 (77 μA/MHz) and STM32L4S7 (110 μA/MHz). Moreover, only nRF52840 and EFM32GG11 provide support for the radio that was chosen in Section 2.1.

Ergo, given that nRF52840 has the least energy consumption compared to EFM32GG11 , it was decided to chose nRF52840 as EADS processing unit. Table 4.1 shows nRF52840 specifications.

Feature	**Specification**
Processor	32-bit ARM Cortex M4
RAM	64 KB
Flash Memory	512 KB
Radio	2.4 GHz, -20 to +8 dBm
Clock Frequency	64 MHz
Supported Protocols	Wirepas, BLE 5, Proprietary Protocols

Table 4.1: nRF52840 Specifications

- **Display**
  There are several displays available for displaying information through SPI such as LCD, OLED and E-Ink. One of the most important features a display should posses is that it should have a low static energy consumption and a low refresh energy consumption. Table 4.2 shows a brief comparison of the four categories of displays. It is interesting to see that E-Ink does not consume any static energy. However, because of high refresh energy, it is less suitable for use in EADS. Moreover, the table shows Memory in pixel has the least energy consumption among the other three. Hence, boostxl128x128, a memory in pixel display, is used for EADS. Additionally, the display will be used in only black and white contrast as it reduces the time to send the data, thus reducing the energy consumption. As mentioned in previous sections, the processing unit sends the data received from the server to the display through SPI communication. The display is powered by the minimum voltage provided from one of the outputs from PMIC.

Type	**Static Energy**	**Refresh Energy**
TFT LCD	200mW	200mW
OLED	738mW	738mW
E-Ink	$0\mu A$	10mA
Memory in Pixel	$14\mu W$	$123\mu W$

Table 4.2: Energy consumption of various displays

Finally, Table 4.3 gives the summary of chosen EADS hardware modules based on the motivation described in the above paragraphs.

Number	**Hardware Modules**	**Description**
1	Powerfilm LL200 2.4-75	Solar panel
2	ADP5091	Power management IC
3	100 mF, 470 mF	Supercapacitor
4	nRF52840	Processing unit
5	Boostxl128 x 128	Display

Table 4.3: Summary of chosen hardware EADS modules

Chapter 5

Implementation of Proposed Framework

5.1 Hardware setup

The EADS device consists of an energy harvester (solar panel), energy storage (supercapacitor), power management IC, processing unit, and display. In this section, connections between these hardware modules are explained. Solar panel and supercapacitor have two terminals - positive and negative, which are connected to the power management IC. The *Vout* pin of power management IC is connected to *Vdd* pin on the processing unit to provide supply voltage. The *Batt* pin of power management IC which represents the energy stored in the supercapacitor is connected to the analog pin *P0.03* of the processing unit. The minimum and maximum voltage for the processing unit operation are 1.8V and 3.3V respectively. It should be noted that the energy stored in the supercapacitor can be greater than the maximum input voltage of the processing unit and the processing unit may fail to read the voltage value. Hence, a hardware circuit must intervene to ensure the voltage supplied to the processing unit does not exceed the maximum voltage. The circuit shown in Figure 5.1, can act as a voltage divider between the energy storage, the analog pin, and the transistor switch. This circuit will prevent the current flow when the ADC_PIN_MCU is not in use by switching between the NPN and PNP transistors with the help of the DIGITAL_PIN_MCU. There is no leakage current when the ADC is not being used and the output voltage of the hardware circuit can be set so that at least half the voltage is used as the input voltage. The only drawback is that an extra digital pin is required on the processing unit. But since there is no shortage of available digital pins on the processing unit, this is not an issue. Moreover, the average energy consumption of this circuit was measured, and it was ensured that the value is under 1μW ($\sim$0.148μW).

Charging time of supercapacitor under different illuminations

The time taken for a supercapacitor depends on the illumination on the solar panel. For instance, generally, if high voltage is supplied to the supercapacitor, the faster the supercapacitor charges. As the illumination in the office environment may vary, it is important to examine the charging time of supercapacitors under illuminations. The solar panel was exposed to two different illuminations - 300 lux and 500 lux. The illumination exposed on the solar panel was measured using *Thunderboard sense* [20] which has a built-in lux meter that can be read via an app through Bluetooth. Figure 5.2 shows the setup used to measure the charging time of supercapacitors under different illuminations. The LED ring acts as a source to provide varying illuminations and is controlled with an Arduino UNO board. The code for this operation can be found in Appendix A.2. The distance d = *1m* is the distance between the source (LED Ring) and solar panel.

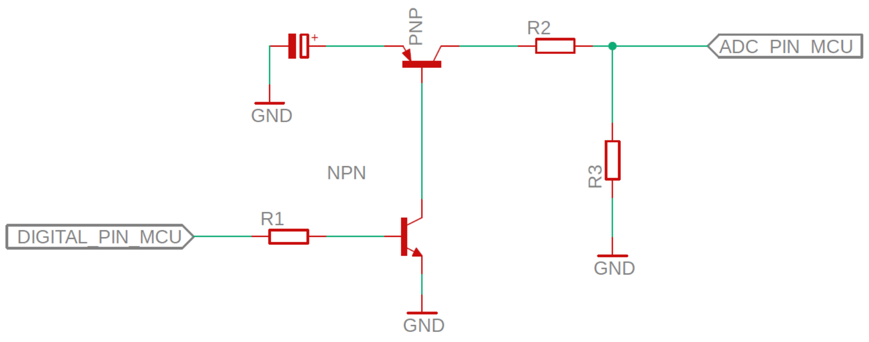


Figure 5.1: Voltage divider between energy storage, analog pin and transistor switch [3]

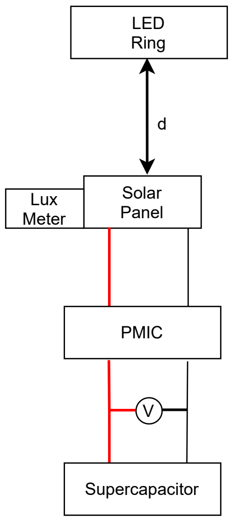


Figure 5.2: Setup to measure supercapacitor charging time

Figure 5.3 and Figure 5.4 show the voltage across the supercapacitor at different time instances for 100 mF and 470 mF respectively. The voltage values are noted down after every 15 minutes for the next four hours. As observed, the initial time taken for a 100 mF capacitor to reach a minimum operating voltage of 1.89V is 60 minutes at different illuminations. Similarly, the time taken by 470 mF capacitor is 120 minutes and 60 minutes under 300 lux and 500 lux respectively. The reason 470 mF takes a much longer time compared to 100 mF is because of the large capacitor size. As the capacitance size increases, the time required for the charge the capacitor also increases due to large area between the capacitor plates. Although larger capacitance requires extra time to charge compared to the smaller one, the large capacitor can store much more current.

Leakage current measurement

Additionally, the leakage current of the supercapacitors should also be taken into account as the leakage current is directly proportional to the size of the supercapacitor. Leakage current has a lot of influence on the time that the supercapacitor can provide the system with energy. There are several manufacturers of 470 mF supercapacitor, and three different series of supercapacitors were tested for leakage current.

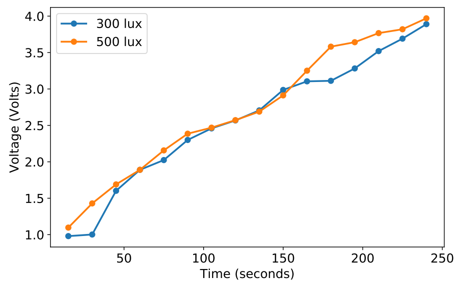


Figure 5.3: Charging time of 100mF supercapacitor at different illuminations

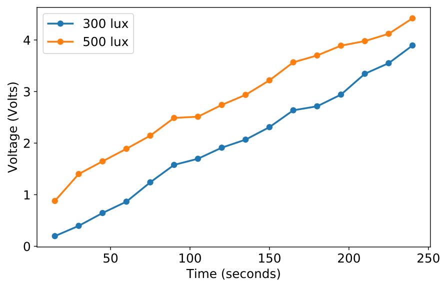


Figure 5.4: Charging time of 470mF supercapacitor at different illuminations

To estimate the leakage current of the supercapacitors, it was decided to charge them to 4.0V. The voltage that is still across the supercapacitor is then measured every hour for one day. This voltage value gradually decreases with which the leakage current can then be calculated with the equation 5.1.

$$I_{lek} = \frac{C * (V_{start} - V_{stop})}{\Delta t} \tag{5.1}$$

where:

C	= capacitance value (100 mF or 470 mF)
V_{start}	= voltage when the test was commenced (4.0V)
V_{stop}	= final voltage measured after one day.
Δt	= time in seconds (86,400 seconds)

Table 5.1 shows summary of the leakage current for one 100 mF capacitor and three 470 mF capacitors. As can be seen from the table, KEMET FS0H474ZF gives the least leakage current compared to other 470 mF supercapacitors. Therefore, this supercapacitor will be used in further experiments.

Supercapacitor	**Capacitance (mF)**	V_{start}	V_{stop}	**Leakage Current (μA)**
Eaton PB-5R0H104-R	100	4.03	3.43	0.69
Eaton KR-5R5C474-R	470	4.01	2.56	7.72
AVX SCMQ14D474PRBB0	470	4.03	3.02	5.49
KEMET FS0H474ZF	470	4.03	3.39	3.48

Table 5.1: Leakage current for different series of supercapacitors

Energy measurement setup

Figure 5.5 shows the energy measurement setup for the EADS device while communicating with the server. The highlighted box in red is the Energy-Monitor (ST X-NUCLEO-LPM01A) provided by ST Microelectronics. The measurements on nRF52840 are taken with the supply voltage of 1.8V as this is the minimum voltage required for the node operation. This energy monitor uses an STM Cube monitor software tool that can generate a real-time graph to show the current consumption for a specified time interval. The node is set to *nRFonly* mode as it disconnects the power supply, external memory, and LEDs of the interface MCU. It also disconnects the signal lines between nRF52840 System on Chip (SoC) and the interface MCU using analog switches.

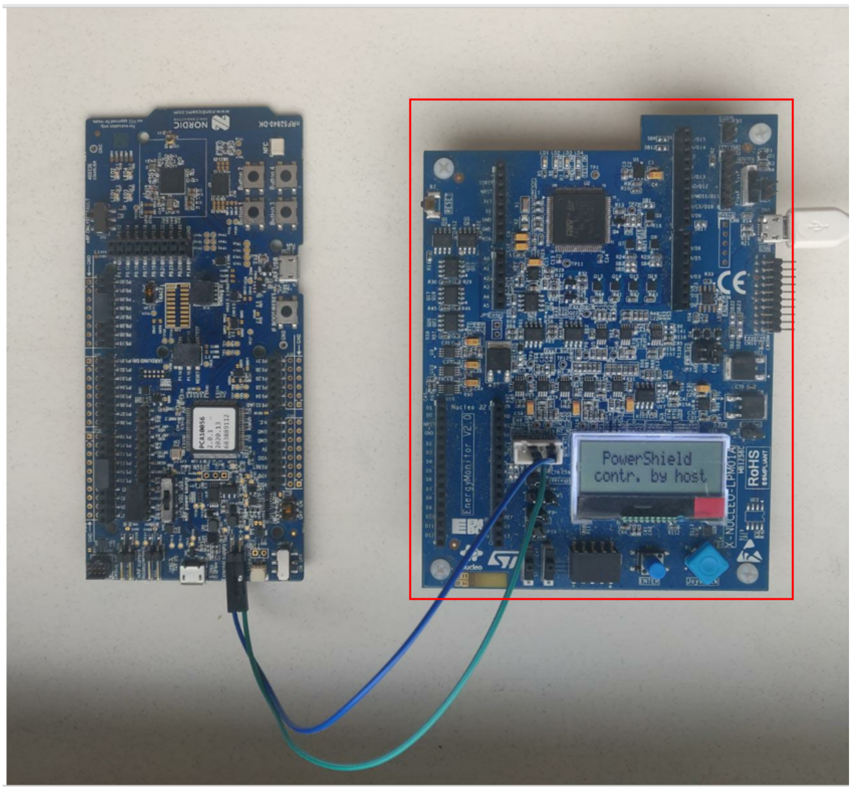


Figure 5.5: Energy measurement setup between the node and energy monitor

5.2 Communication framework with Wirepas Massive

This chapter describes in detail the setup and implementation of creating a Wirepas communication channel between the EADS server and the EADS client. The client and server can only communicate with Wirepas if they are equipped with Wirepas stack and Section 5.2.1 and Section 5.2.2 describe how this can be achieved. Furthermore, Section 5.2.3 explains the framework created for Wirepas communication between the EADS server and the EADS client.

5.2.1 Wirepas stack on Server

EADS server, also known as Wirepas gateway, is responsible for retrieving calendar information from the Office 365 cloud server through the Internet. Calendar information is received by the server at periodic intervals so that the information is up-to-date. This is because the server is unaware of any changes that may happen after it has received the most recent information. After receiving the calendar information, the server does not store this information if the node does not have enough energy to receive it. The reason is, as it is hard to predict when sufficient energy is available on the client, the information may no longer be *up-to date*.

Gateway and sink node

The gateway acts as a bridge between the Wirepas network and the backend. The sink node is responsible for routing the traffic between the backend and the Wirepas Massive network. The sink node is connected to the gateway via a serial interface such as UART or SPI. In this project, the sink node is a hardware module connected to the gateway and communicates with the gateway through UART as shown by the highlighted box in Figure 5.6.

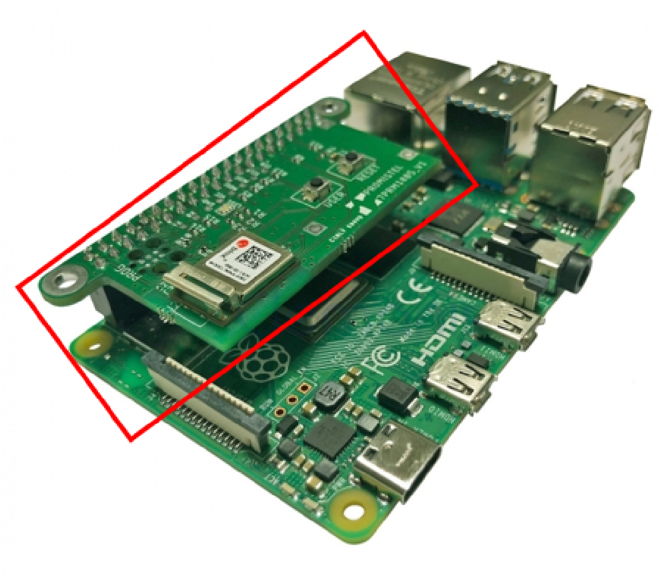


Figure 5.6: Wirepas 5.0 Raspberry Pi HAT [34]

Gateway to backend communication

To send data from a gateway to the backend, the gateway must be compliant with an Application Programming Interface (API). The backend to gateway API is based on a set of MQTT topics where messages are encoded as Protocol Buffers [1]. This API has four major services: (a) Gateway status, (b) Gateway configuration including sink node, (c) Send/receive data to/from a Wirepas node(s), and (d) To manage Over the Air Updates of Wirepas Massive networks (OTAP). The gateway supports two services: sink service and transport service. The sink service is responsible for interfacing locally with Wirepas devices. The transport service packs network messages on protocol buffers and publishes them on top of MQTT according to API. Every gateway must generate a unique ID that is used inside MQTT topics. Furthermore, multiple sink nodes can be attached to a single gateway and the gateway is responsible for their identification. Hence every sink node will also have a unique ID.

Wirepas Massive on Gateway setup

Wirepas Massive offers Wirepas prebuilt images for Raspberry Pi to function as a gateway. The gateway will allow the connection of the local Wirepas Massive network to the backend using the MQTT interface. Raspberry Pi needs to be connected to the internet for initial setup and can be accessed with TCP ports. One among the two available ports - backend (`8883`) and local MQTT Broker(`1883`) can be used. The prebuilt image consists of two configuration files for gateway (`gateway.env`) and sink node (`sink.env`) respectively. The `gateway.env` file is used to configure gateway drivers - *sink service* and *transport service*. In *sink service*, bitrate for UART transmission can be set as the sink node is connected to the gateway via UART. In *transport service*, the gateway ID, MQTT hostname, user name, and password should be configured.

[1]https://developers.google.com/protocol-buffers

As sink node forwards the local mesh data to the gateway, the `sink.env` file should be configured with network parameters such as node address, the network address, and network channel. Once these files are configured, the prebuilt image can be booted up on the Raspberry Pi. Now, the EADS server aka the gateway is capable of receiving the local data from the network and sending the backend data to the network.

5.2.2 Wirepas stack on EADS client

To receive the meeting information from the server and display it on the display, the client should also run the Wirepas stack. Wirepas stack on the client is made available with Software Development Kit (SDK) [2] provided by Wirepas. Wirepas application developed for sending the calendar information to the node will be discussed in Section 5.2.3. The steps involved in flashing the developed Wirepas application and Wirepas stack on the node are as follows:

- Wirepas Massive SDK is based on GNU ARM toolchain and has a few dependencies such as GCC ARM toolchain, maketool, and python 3.x to be resolved. Thus, a docker build environment is used as it allows containerized build environment where all dependencies are solved.
- The SDK can be cloned from a git repository and Wirepas Massive stack can be downloaded by the licensees from the Wirepas portal.
- Once the SDK folder and docker are ready, the application can be built by generating the application binary using docker's build environment. Once the build is complete, the hardware board (nRF52 MCU) can be connected to the host PC through USB.
- To program a Nordic MCU, nRF5x Command Line Tool is needed which will enable the programming environment for the device. Now that the programming environment is ready, the target device can be finally flashed with the Wirepas application (including all the necessary Wirepas dependencies).
- For a node to join a Wirepas Massive network, it should have the same network parameters as that of the existing network. The network parameters such as network address and network channel can be modified with *configure.mk* file for a particular application.

Communication between Wirepas Massive and node application

The Single-MCU operation allows an application to run on the same chip with Wirepas Massive stack. Figure 5.7 shows the overview system architecture of Wirepas SDK. The application firmware includes the application logic. The same network can have multiple applications i.e. different kinds of devices. and SDK Hardware Abstraction Layer (HAL) includes various software components for peripheral usage such as sensors and communication interfaces. Application-specific HAL allows various software components for peripheral usages, such as sensors/actuators and communication interfaces. Wirepas Massive provides Single MCU API for applications to use stack services and run tasks on the MCU. Wirepas stack includes Wirepas communication stack and Wirepas scheduler to enable application operation in the same MCU. All the hardware abstractions and drivers needed by the stack, such as the radio driver, are included in Wirepas Massive HAL. Bootloader binary initializes the hardware and also handles the processing of stored scratchpad in the flash received by the stack during an OTAP. The bootloader is also responsible for flash partitioning and flash management. The available flash memory for Nordic MCU is limited by the size of the memory area commonly used for application and scratchpad images. If the application size is too large, a scratchpad image will override the application image. Hence, the default maximum size of the application is set, so that it is always safe to use scratchpad images containing both firmware and application. The recommended maximum size of flash memory for an application is *40kB*. Furthermore, the available RAM for the Nordic chip is *188kB*.

[2]SDK is a set of software tools with libraries that allow building applications for specific platforms.

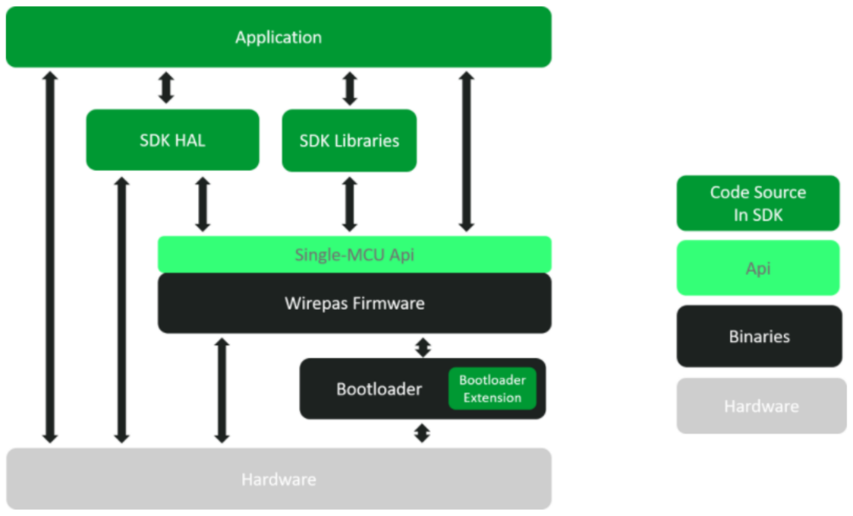


Figure 5.7: Wirepas SDK system architecture [26]

5.2.3 Communication between gateway and node

Now, with the Wirepas stack on both the server and the client, a communication channel can be established between the two. The server will send calendar meeting information to the client through Wirepas Massive. Additionally, when the client receives the meeting information, this information should be further sent to display via SPI communication. The information sent to the display is the result of the execution of a sequence of tasks such as (a) sending the information to the display; (b) updating the display content with the date just sent to it.

Application development for Node (C)

The node application (in C) allows discovering the Wirepas Massive network capabilities to generate and process messages received on or sent from the node. The application API developed has the following functionalities:

- To send and receive data on the node and supports unicast addressing mode and will only process messages intended for a specific node.
- Integrates the endpoint functionality of Wirepas which corresponds to an application channel (for example, if the node has multiple sensors different endpoints to send/receive each sensor's data can be used). In this software, all messages have a dedicated source and destination endpoint fixed to the value 1.
- Receives and generates internal Wirepas Massive network packet format with a specific payload format (<Message ID> <Message specific data format>) for each message.

Application development for Gateway (Python)

The Python script allows interaction with the nodes running the node application discussed above using Wirepas-Gateway-API for the nodes that are part of the network connected to the Wirepas gateway. In this configuration, a remote communication channel is be established via MQTT protocol as commands sent are published on dedicated MQTT topics from which gateway can read and parse to send them to the Wirepas Massive network. By subscribing to dedicated topics messages coming from nodes running the application can be obtained.

The script is built on several python modules listed below:

- paho-mqtt: To handle everything related to the MQTT protocol, broker, and client (i.e client creation and connection to the broker, topic publishing/subscribing).
- Wirepas-Mesh-Messaging: To implement the Wirepas gateway API to communicate with a Wirepas gateway (i.e encode and decode packets).
- Cmd: To create an interactive shell which allows to list all script commands, their help and enter commands to send to the network.

To be able to receive and send messages at the same time, the script needs to be run in two separate shells. One will only act as a viewer (i.e it receives, decodes, and display messages coming from a Wirepas Massive network) and the other as a controller (i.e it asks user input, parses the given command, encodes it, and sends the command to a Wirepas Massive network). To receive the messages from the node, *enable_messages_reception* and *calendar_response* commands can be used that send periodic messages and calendar responses to the gateway respectively. These commands subscribe the MQTT client to the "gw-event/send_data/+/+/+/1/1" topic where messages sent by nodes running the Calendar app are published. To send messages to the network *send_calendar_info* command can be used with the gateway ID, sink node ID, and node ID for better identification as there can be multiple gateways, sink nodes, and nodes in a network. The execution flow of the script is as follows:

- get MQTT client parameter from command line with the help of `mqtt_client_get_parameters()` function.
- create an MQTT client
- connect to the specified MQTT broker
- starts an interactive shell:
  If *send_calendar_info* command (server to node) is selected, encode the message and publish it to the correct topic. If *enable_messages_reception* command (node to server) is selected, subscribe to the topic where messages are published and wait for `on_message_event_data_callback()` function to be called. In this function, all the message handling is done i.e decoding with wirepas-mesh-messaging module and some custom decoding code to parse the data payload.

Message description

There are two types of messages generated in the EADS network for the developed node application.

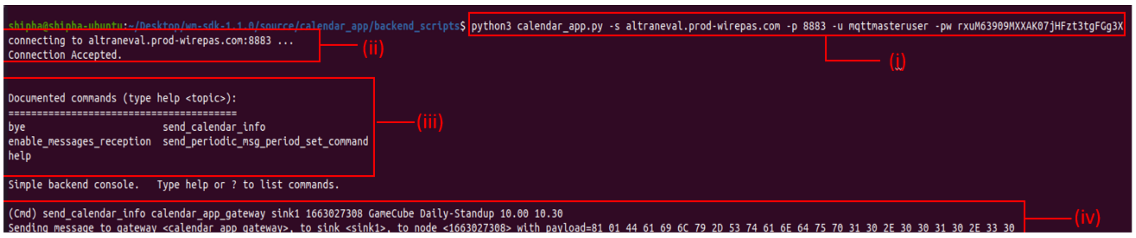
- *Server to node:*
  The message format for this message is as follows:
  <Message ID> <Meeting Room> <Meeting Name> <Start time> <End time>

  where, *Message ID* → Message identifier with decimal value 129
  Meeting Room → Name of the meeting room for instance, Daily-Standup and Sprint Review
  Meeting Name → Meeting name such as GameCube and SuperPong,
  Start and End time → Start and end time of a meeting

- *Node to server:*
  The message format for a response to server is as follows:
  <Message ID> <State>

  where, *Message ID* → Message identifier with decimal value 3
  State → Reception state - 1 (successful)

Depending on the information provided in the above paragraphs, Figure 5.8 shows Wirepas communication between the server and the client. Initially, a connection with the MQTT broker has to be established which is done with the port number, MQTT username, and password (see (i)). Next, after the connection is established (see (ii)), a set of commands are displayed that can be used to send or receive messages to and from the network (see (iii)). Finally, the *send_calendar_info* command is used to send the information containing the *gateway ID*, *sink node ID*, *node ID*, *room name*, *start* and *end meeting* time every 15 minutes. This message is converted in the form of a payload as a byte of hex strings (see (iv)). Furthermore, Figure 5.9 shows the response of the received message from the server. Here the response is 1 indicating that the information is successfully received.



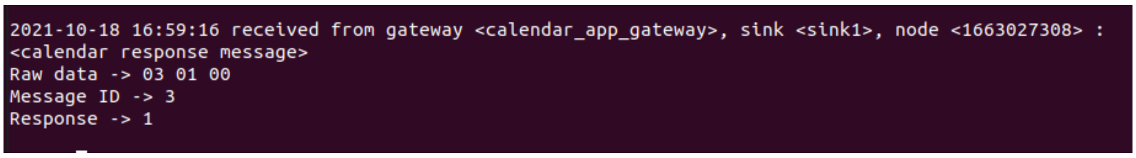
```
shipha@shipha-ubuntu:~/Desktop/wm-sdk-1.1.0/source/calendar_app/backend_scripts$ python3 calendar_app.py -s altraneval.prod-wirepas.com -p 8883 -u mqttmasteruser -pw rxuM63909MXXAK07jHFzt3tgFGg3X
connecting to altraneval.prod-wirepas.com:8883 ...
Connection Accepted.

Documented commands (type help <topic>):
========================================
bye                         send_calendar_info
enable_messages_reception   send_periodic_msg_period_set_command
help

Simple backend console.   Type help or ? to list commands.

(Cmd) send_calendar_info calendar_app_gateway sink1 1663027308 GameCube Daily-Standup 10.00 10.30
Sending message to gateway <calendar_app_gateway>, to sink <sink1>, to node <1663027308> with payload=81 01 44 61 69 6C 79 2D 53 74 61 6E 64 75 70 31 30 2E 30 30 31 30 2E 33 30
```

Figure 5.8: Gateway sending calendar information to node



```
2021-10-18 16:59:16 received from gateway <calendar_app_gateway>, sink <sink1>, node <1663027308> :
<calendar response message>
Raw data -> 03 01 00
Message ID -> 3
Response -> 1
```

Figure 5.9: Node responding to the message sent by gateway

5.3 Connection between node and display

This section explains the communication between the node and the display. Figure 5.11 shows the connections between the node and display. The node sends the received calendar information from the server to the display through SPI communication. The SPI on nRF52840 includes a TXD register for sending data and an RXD register for receiving data. The SPI supports four different modes regarding clock polarity and clock phase. In this thesis, SPI_MODE0 is used as the display supports reading data on the rising edge of the clock. Three different SPI pins are used as follows.

- *MOSI*: This pin is the Master Output Slave Input (MOSI), where the node is the master and the display is the slave device.
- *SCK*: This is a Clock signal where the data is read during the rising edge.
- *CS*: This pin provides active-low chip select signal to the display. The CS pin is useful when there are multiple SPI slave devices connected to a single master (node). This pin is set to LOW to select the display, and HIGH otherwise.

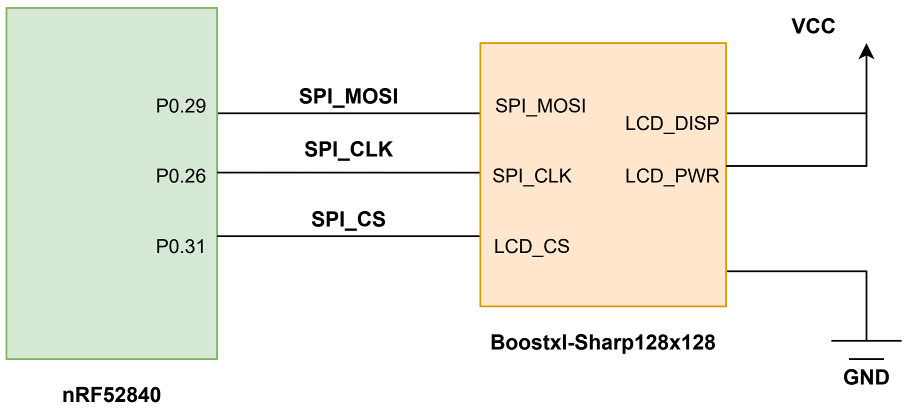


Figure 5.10: Schematic connection between node and display

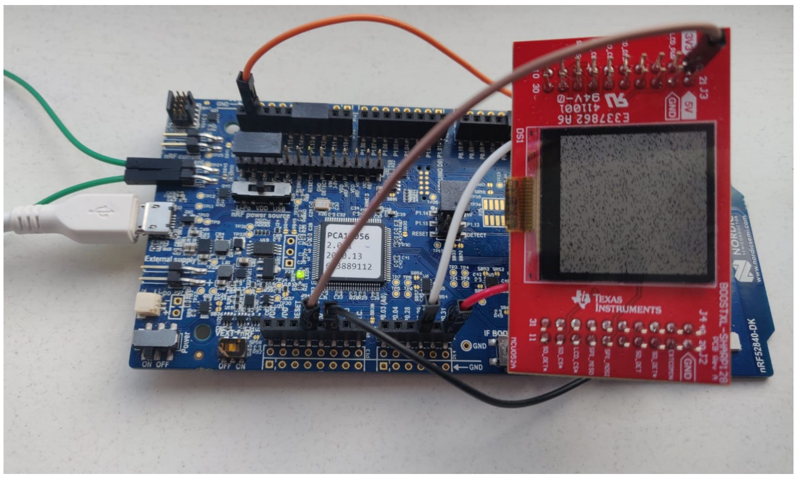


Figure 5.11: Connection between node and display

5.4 Energy Aware Scheduling

As discussed in the previous sections, EADS will use the EAS algorithm to efficiently utilize the available energy. The EAS algorithm will execute tasks based on available energy and priority. The MCU is powered by the energy stored in the supercapacitor. The output of the supercapacitor is connected to one of the analog pins (ADC) on the MCU. To determine the available energy, the ADC must be first read. When the ADC value has been determined, the ADC value is converted to input voltage *(V_i)* with the equation 5.4. $V_{forward}$ is the voltage drop across the supercapacitor and ADC voltage divider network (see Figure 5.1). The parameters - *Value*, V_{Ref}, *Scaling* and *Resolution* are used for the ADC conversion.

$$V_i = \frac{Value.V_{Ref}.Scaling}{Resolution} + V_{forward}$$

Furthermore, from this voltage value, the energy in Joules is determined by using the equation 5.2; where energy is in μJ, capacitance is in μF. Minimum voltage is when there is no operation and maximum voltage is the available energy that can be used.

$$Energy = \frac{1}{2}.C.(V_{max}^2 - V_{min}^2) \tag{5.2}$$

The average energy consumption for each function is determined to calculate the required energy per function with the equation 5.3

$$Avg\_Energy\_Usage = \frac{\sum Energy\_Usages\_Measurements}{\#\_of\_Measurements} \tag{5.3}$$

To estimate the sleep time (in seconds) required for a given amount of energy, each time the EADS client wakes up from sleep mode, the average harvested energy is determined with the equation 5.4

$$Avg\_Harvested\_Energy(persecond) = \frac{\sum Energy\_Measurements}{\#\_of\_Measurements.Sleeptime} \tag{5.4}$$

To estimate the required sleep time per amount of energy, using the stored average energy expenditure per function and average harvested energy in sleep time is determined with the equation 5.5. *Margin* is similar to worst-case energy that will be utilized in certain situations to perform a task.

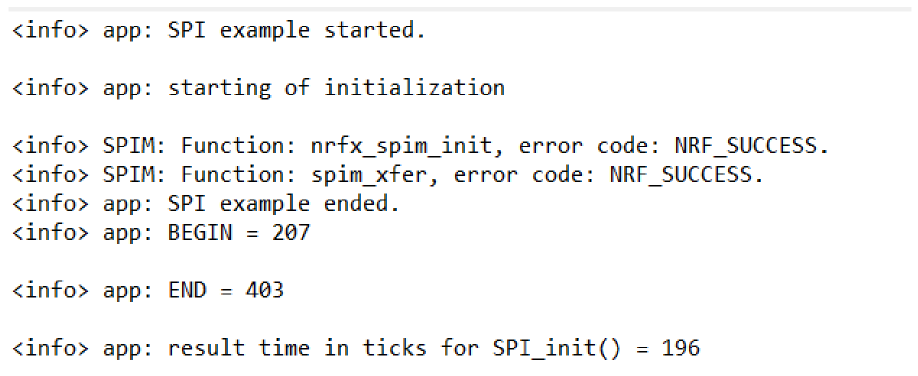
$$Required\_Sleep\_Time(seconds) = \frac{Avg\_Energy\_Usage + Margin}{Avg\_Input\_Power\_in\_Sleepmode} \tag{5.5}$$

To determine whether sufficient energy is available to perform a function, it is calculated using the stored average energy consumption per function and average harvested energy as shown by the equation 5.6

$$Sufficient\_Energy = Energy\_Available - Average\_Energy\_Usage - Margin \tag{5.6}$$

Task execution time and energy per task calculation

Since Wirepas tasks cannot be explicitly controlled, other node tasks related to the display can be considered for scheduling. To schedule the display tasks, it is crucial to measure the energy consumption and execution time of these tasks as they form the basis for the EAS algorithm. Additionally, it is also important to ensure that the execution times for each task do not exceed 100 ms to not interfere with Wirepas stack task execution. The execution time of each task is calculated with a Real-Time Counter (RTC) peripheral on the node. An application timer is created which is included before and after a specific task and the difference is computed. The `app_timer_cnt_get()` function is read before and after every task. And then the was difference computed with function `app_timer_cnt_diff_compute()`. The difference value obtained is represented as the number of ticks on RTC. Figure 5.12 shows the number of ticks required to execute initialization functions - `SPI_init()` and `Graphics_initContext()`.



```
<info> app: SPI example started.

<info> app: starting of initialization

<info> SPIM: Function: nrfx_spim_init, error code: NRF_SUCCESS.
<info> SPIM: Function: spim_xfer, error code: NRF_SUCCESS.
<info> app: SPI example ended.
<info> app: BEGIN = 207

<info> app: END = 403

<info> app: result time in ticks for SPI_init() = 196
```

Figure 5.12: Calculation of task execution time

The number of ticks is then converted to milliseconds using equation 5.7. Prescaler [3] is set to 0 and TIMER_CLOCK_FREQUENCY is set to the value of low-frequency clock i.e., 32768 Hz.

$$Time(milliseconds) = \frac{\#Ticks * (PRESCALER + 1) * 1000}{TIMER\_CLOCK\_FREQUENCY}$$
$$\begin{aligned} Time(milliseconds) &= \frac{196 * (0 + 1) * 1000}{32768} \\ &= 5.981ms \end{aligned} \tag{5.7}$$

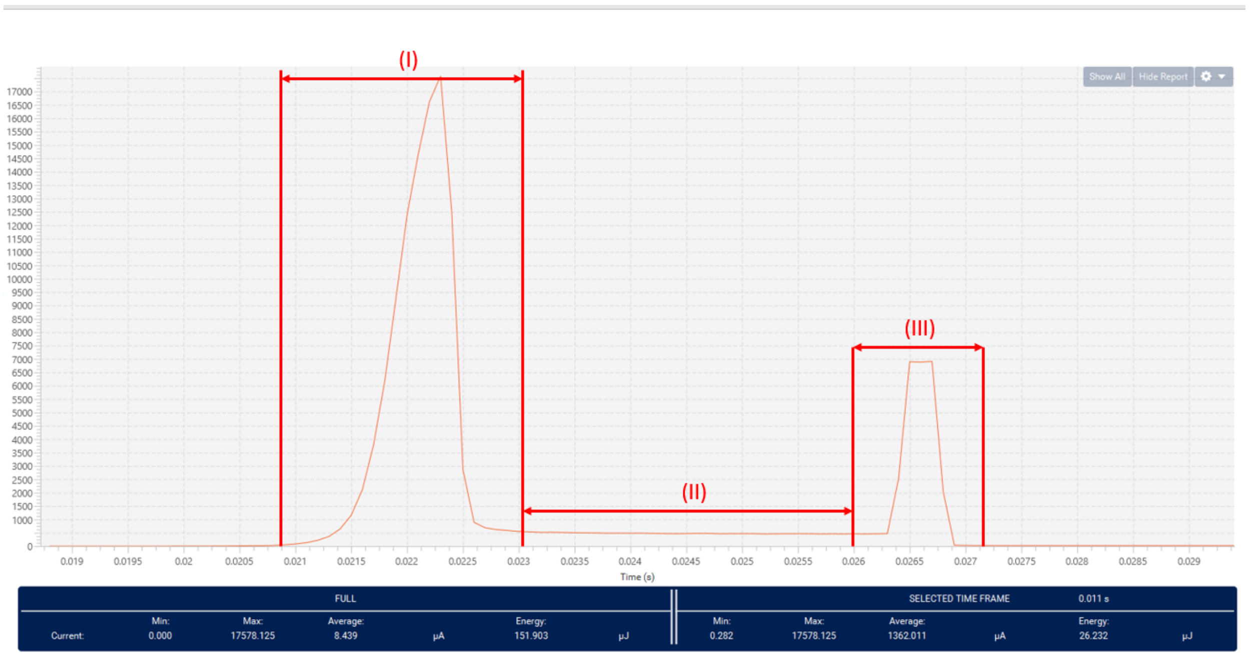
Furthermore, the energy required by each task was calculated with the energy monitor with similar configurations as discussed in Section 5. Firstly, the energy consumed by the empty application i.e., with only the `main()` function was calculated. Later, the initialization function for SPI pins (`SPI_init()`) and display drivers (`Graphics_initContext()`) were included with the empty application. Then, the energy obtained was subtracted with the energy for `main()` function. Similarly, to clear the display, initialization (SPI pins and display drivers) is necessary. Hence the total energy required for `clear_Display()` will be the sum of the `SPI_init()` and `Graphics_initContext()`, and `clear_Display()` functions. Similar procedure is repeated to calculate the energy consumption for the other two remaining tasks. Additionally, to conserve as little energy as possible, the SPI on the node was enabled and disabled before and after the tasks that utilize SPI transfer function to get the most appropriate energy consumption value. This is because, if the SPI functionality is not disabled explicitly, then the node consumes energy as the CPU is providing clock for SPI transmission even when it is not in use. Figure 5.13 shows the energy consumption for the execution of `main()` function along with `SPI_init()` and `Graphics_initContext()` functions. The highlighted (I) shows the energy consumption for CPU start-up and initialization of the timers on the node. The highlighted (II) shows the energy consumption for passing the execution control for the initialization function. The highlighted (III) shows the energy consumed by the initialization functions for SPI peripherals `SPI_init()` and display drivers `Graphics_initContext()`.

Table 5.2 shows the execution time and corresponding energy required for each of the display tasks that will be scheduled with the EAS algorithm. The values shown in the table are the average of the energy measurements taken for three iterations.

Display Tasks	**Description**	**Energy Consumption (μJ)**	**Execution Time (ms)**	**Priority**
SPI init and Display init	Initialize SPI pins and display drivers	6	6	1
Clear display mode	Clears memory internal data and write White (0xFF)	9	5	2
Display mode (Graphics_drawString, Graphics_drawLineH)	Maintains memory internal data	30	4	3
Data update mode	Update data on the display	71	26	4

Table 5.2: Atomic subtasks considered for EAS algorithm

[3] An electronic counting circuit is used to reduce a high-frequency electrical signal to a lower frequency by integer division. It takes the basic timer clock frequency and divides it by some value before feeding it to the timer.



	FULL						SELECTED TIME FRAME			0.011 s	
	Min:	Max:	Average:		Energy:		Min:	Max:	Average:		Energy:	
Current:	0.000	17578.125	8.439	µA	151.903	µJ	0.282	17578.125	1362.011	µA	26.232	µJ

Figure 5.13: Energy consumption measurement per task

5.5 Wirepas Massive network monitoring

Now that the Wirepas Massive network has been established, the network should be monitored for easy maintenance as the nodes are spread across the entire building. This functionality is provided by Wirepas Network Tool (WNT), a tool for monitoring and analyzing Wirepas Massive network operation. It provides individual node behavior and visualization of the logical topology of a Wirepas Massive network.

The clients will be placed in different meeting rooms at the Capgemini Eindhoven office. These rooms were geo-referenced from the Google map in the WNT client to get the approximate location of the rooms. The clients can be easily monitored through the WNT client application on the user's personal computer. WNT works by gathering the data from the network and relaying that information via a gateway to the backend server. WNT backend is hosted on a cloud service or local server installation. WNT client will communicate with the backend server over TCP/IP using ports 8811, 8812, 8813, and 8886. WNT provides client application information such as the number of nodes online/offline, link quality, and message delay. The overall operation of WNT is shown in Figure 5.14. Furthermore, Figure 5.15 shows an example of the setup network with geo-referenced nodes with the network number, decimal address of each node, and corresponding name, node roles, operating mode, and battery voltage. This information helps in easy maintenance of the Wirepas network even if the number of nodes in the network increase.

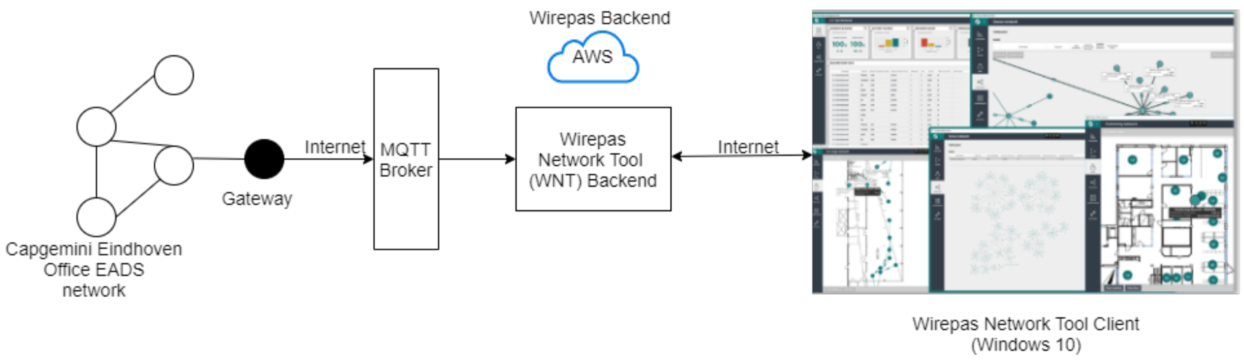
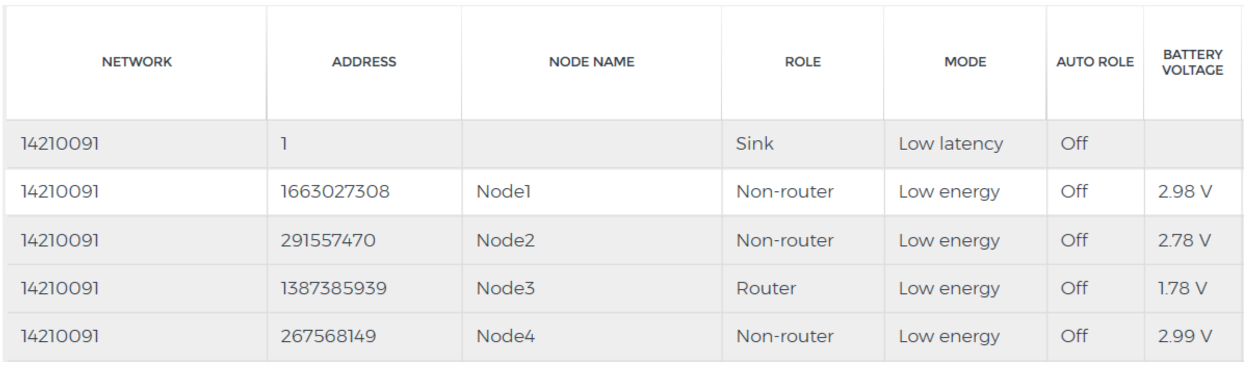

Wirepas Backend
AWS
Internet
MQTT Broker
Wirepas Network Tool (WNT) Backend
Internet
Gateway
Capgemini Eindhoven Office EADS network
Wirepas Network Tool Client (Windows 10)

Figure 5.14: Wirepas Network Tool (WNT) Overview



NETWORK	ADDRESS	NODE NAME	ROLE	MODE	AUTO ROLE	BATTERY VOLTAGE
14210091	1		Sink	Low latency	Off	
14210091	1663027308	Node1	Non-router	Low energy	Off	2.98 V
14210091	291557470	Node2	Non-router	Low energy	Off	2.78 V
14210091	1387385939	Node3	Router	Low energy	Off	1.78 V
14210091	267568149	Node4	Non-router	Low energy	Off	2.99 V

Figure 5.15: Wirepas Network monitoring with WNT

Chapter 6

Evaluation and Results

The chapter aims to discuss the performance and evaluation of the EADS with a mesh network and the efficient energy utilization of EADS with the EAS algorithm.

6.1 Scalability of EADS through Wirepas Massive network

One of the main aims of this thesis is to achieve low-power scalability of EADS through the Wirepas Massive network. To evaluate the Wirepas Massive performance on a battery-less device such as EADS, the energy consumption of EADS will be compared with other popular wireless protocols such as BLE and BLE Mesh. Table 6.1 shows the baseline energy consumption on nRF52840 for transmission and reception of a message through Wirepas Massive.

No.	**Mode**	**Energy Consumption (μJ)**	**Power Consumption (μW)**
1.	Start up and join the network	49637	2489
2.	Router node	1516	76
3.	Non-router node	681	34
4.	Sleep mode	18	1
5.	Wake up	56	3

Table 6.1: Energy consumption of EADS client with Wirepas Massive for 20 seconds interval

To observe the effect of energy consumption of EADS under various settings, a set of experiments are carried out, and the results obtained are discussed in the following sections. It should be noted that all the experiments could only be demonstrated with a maximum of four nodes due to the limited resources. Additionally, apart from Wirepas Massive, WiFi and BLE also operate at 2.4 GHz frequency. This may lead to interference in the signals being transmitted by the nodes or the server of the Wirepas Massive network. Therefore, all the measurements in the following sections are taken in presence of WiFi signals to represent a real-time worst-case scenario. Figure 6.1 shows WiFi interference measurements at a specific location with Wirepas Massive nodes.

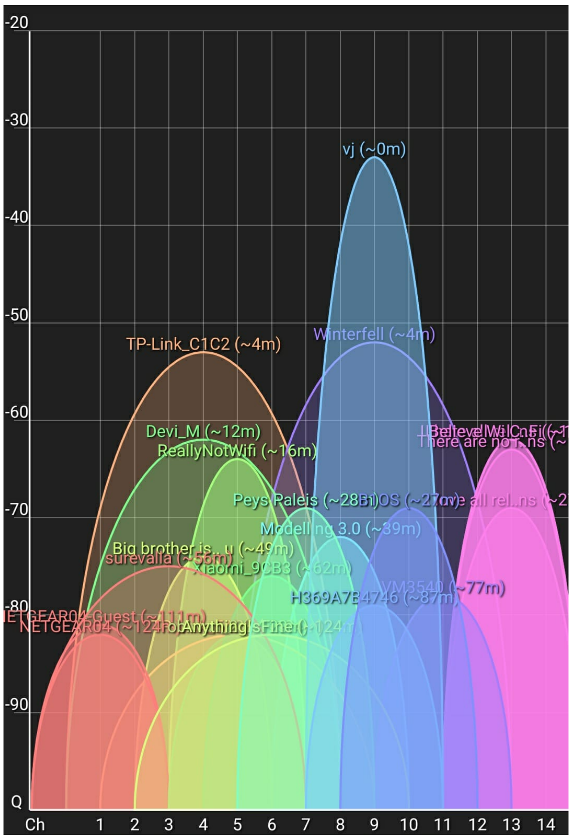


Figure 6.1: WiFi interference

6.1.1 Comparison with Bluetooth Low Energy (BLE)

BLE standard provides advertising and scanning operations to implement the communication between BLE devices. Initially, the communication between the EADS client and the server was through BLE protocol. The advertising device (EADS client) sends a message in one of the 3 channels with a repetition period known as advertising interval. The scanner (EADS server) listens to the channel for a duration known as a scan window. Table 6.2 shows the energy consumption for different states of the EADS client running BLE protocol. These energy measurements were taken when the EADS client is advertising and trying to receive the calendar information from the scanner i.e. EADS server. In *BLE-advertising (System ON sleep mode)*, BLE (EADS) device is advertising and can be found by the server. Furthermore, with System On sleep mode enabled, the CPU will be in sleep mode when no events occur. As observed from the table, BLE advertising consumes 67913.207 μJ for 20 seconds, whereas Wirepas consumes 1515.526μJ (see Table 6.1) for the same duration. Moreover, the energy consumed for wake up and sleep for BLE is 200% and 25% respectively more than Wirepas. However, BLE energy consumption can be reduced with two methods: (a) using the EADS client in *BLE-advertising (System ON sleep mode)* which consumes 4.5 times less energy than *BLE-advertising (System ON sleep mode off)*, (b) BLE advertising can be disabled when the calendar information has been received from the server and enabled again after a period with a timer which can reduce the power consumption by 25%. Despite these methods to reduce BLE energy consumption, Wirepas consumes 95% less energy compared (router node) to BLE in *BLE-advertising (System ON sleep mode)*. As energy consumption is the most important constraint for EADS and given the BLE energy consumption, Wirepas Massive shows significantly better results for the same duration (20 seconds).

Apart from the energy consumption, another reason why Wirepas Massive can be preferred over BLE is due to BLE's inability to support the increasing number of BLE devices in the network. BLE can only support a maximum of eight nodes per server at any given time. As scalability is one of the main issues addressed in this thesis, and given the BLE performance in terms of scalability and energy consumption, it can be concluded that Wirepas Massive out performs BLE in terms of both energy as well scalability.

No.	Mode	Energy Consumption (μJ)	Power Consumption (μW)
1.	BLE-advertising (System ON sleep mode)	33914	1696
2.	BLE-advertising (System ON sleep mode off)	176492	8825
3.	BLE-advertising disabled	27180	1359
4.	Sleep mode	450	25
5.	Wake up	11400	570

Table 6.2: Energy consumption measurements for EADS client with BLE for 20 seconds interval

6.1.2 Comparison with BLE Mesh

As BLE can only support eight nodes per server, BLE cannot be preferred for EADS scalability. Hence it is reasonable to compare Wirepas Massive with another popular mesh protocol such as BLE Mesh. BLE Mesh uses a flooding mechanism to exchange messages in a mesh network, thus ensuring each intermediate nodes relay messages to the destination node. The mesh nodes have a 100% duty cycle for scanning incoming packets on the advertising channels. This indicates that the mesh nodes are continuously scanning the packets on advertisement channels and the radio is never turned off. On the contrary, in BLE, the radio is on during transmission and reception only for a couple of milliseconds each connection interval.

A BLE mesh network supports three features - *Relay feature*, *Proxy feature* and *Friendship feature*. To avoid inefficiencies that may arise due to the flooding mechanism, *Relay feature* requires only relay-enabled nodes to forward received messages in the BLE network. *Proxy feature* can be used by a native BLE device such as a smartphone to connect with a BLE Mesh network. Furthermore, as the nodes use 100% duty scan for advertising, this reduces the low energy aspect of BLE advertising. When a power-limited device wants to join the mesh network, other nodes in the network can assist this node so that it can still be a part of the network. Both nodes establish a friendship relation as a low power node and the other node acting as a friend node. The MCUs provided by Nordic Semiconductors (such as nRF52840) support the BLE Mesh network. Some of the examples from SDK provided by Nordic Semiconductors were used to measure the energy consumption of three different features supported by BLE Mesh. Further details on the energy measurement setup for BLE mesh can be found in Appendix B. Table 6.3 shows the energy consumption of various features. As can be seen from the table, the Low-power node which wakes up only during transmission or reception consumes 52% more energy compared to the Wirepas non-router node. And the BLE mesh relay node consumes almost 300% more energy compared to Wirepas router node. Since energy is one of the major factors for EADS, Wirepas shows better performance compared to BLE mesh, and is most suitable for low-power battery-less devices such as EADS.

No.	**Feature**	**Energy Consumption (μJ)**	**Power Consumption (μW)**
1.	Low-power node	1038	52
2.	Friendship node	366980	18349
3.	Relay node	355004	17750

Table 6.3: Energy consumption of various BLE mesh features for 20 seconds interval

6.1.3 Varying number of nodes in network and node operating modes

As discussed in Section 3, Wirepas nodes can operate in two modes - router node and non-router node. The router node is awake for most of the time if the number of nodes in the network increase as it is involved in relaying the messages to other nodes. The non-router node is only awake during the time of transmission or reception. Hence, the energy consumed by the router node for 20 seconds is 1515.526 μJ, whereas for a non-router node is 680.983 μJ (see Table 6.1). As expected, the energy consumed by the router is 122.54% more than that of the non-router node.

As the energy values for router node and non-router node have been determined, the next step would be to see if the energy consumed by the router node varies with the number of non-router nodes. To examine this, a total of four nodes were used where one of them was configured as the router node and the remaining were configured as non-router nodes. Initially, one router node and one non-router node were added to the Wirepas Massive network consisting of a gateway and sink node. The nodes were placed at a distance of 6m from each other and the router node was placed at a distance of 2m from the sink node, The energy consumed for this setup was noted down. Similarly, non-router nodes were added to the network one after the other at distances 10m and 15m respectively from the router node. Table 6.4 shows the results of this experiment with varying numbers of non-router nodes in the network. As observed from the table, the energy consumed by the router node does not vary largely with the number of transmissions i.e., relaying messages to other non-router nodes in the network. However, there are some minute variations in the energy consumed, the reason for this could be that the router node is sending other messages such as diagnostic network data to the server at different intervals. Although the energy consumption of the router does not vary largely with three non-router nodes, the energy consumption may vary significantly when the number of nodes increase further, for instance, 10 nodes. The reason being, the more non-router nodes connected to single router node, the longer the router node has to stay awake to forward the messages.

No.	**Number of nodes**	**Energy Consumption of router node (μJ)**	**Power Consumption (μW)**
1.	1	1516	76
2.	2	1413	71
3.	3	1554	78

Table 6.4: Router node energy consumption with varying non-router nodes for 20 seconds interval

Wirepas Massive operating modes

Wirepas Massive also supports two modes of operation - Low-Energy mode and Low-Latency mode. As the name suggests, the Low-Energy mode is designed to achieve the lowest possible energy consumption. This mode uses time-slotted multi-channel access for synchronization with the mesh network. On the other hand, the Low-Latency mode is intended to achieve low latency and higher throughput on the network. To achieve this, router nodes are listening during their idle time allowing the nodes to transmit to the next hop immediately. Here, the trade-off is high energy consumption.

An experiment was conducted to see the variations in the energy consumption by the nodes configured with different operating nodes. Table 6.5 shows the energy consumed by both router and non-router nodes in the Low-Energy and Low-Latency operating modes. As observed from the table, the energy consumed by the nodes in Low-Latency is much more than the energy consumed by nodes in Low-Energy. Therefore, it can be concluded that the nodes in the Low-Energy operating mode should be used for the scalability of EADS devices due to low energy consumption.

Node	**Low-Energy Mode (μJ)**	**Power Low-Energy Mode (μW)**	**Low-Latency Mode (μJ)**	**Power Low-Latency Mode (μW)**
Router	1516	76	319764	15989
Non-Router	680	34	157410	7871

Table 6.5: Energy consumption for different operating modes for 20 seconds interval

6.1.4 Correlation between Supply voltage and RSSI

Since the node supply voltage is obtained from the energy stored in the supercapacitor, the supply voltage may vary given the unpredictable behavior of the supercapacitor and energy harvester. These variations in the voltage can also affect signal strength which is measured using Received Signal Strength Indicator (RSSI). RSSI is an estimated measurement of how well a device can detect and receive signals [4]. RSSI also helps to determine and know if a signal is sufficient to establish a wireless connection. With the increase in the distance, the wireless signal becomes weaker and the bandwidth of the connection becomes slower. Hence, it is important to observe how the signal strength is affected by variations in supply voltage. A higher absolute value in dBm means better signal quality whereas, a lower dBm represents a worse signal quality. For instance, -90 dBm is poor, whereas -60 dBm is a good signal. The nRF MCUs use DC/DC regulators to reduce the overall power consumption. The DC/DC regulator was enabled and the supply voltage was varied to examine the effect of varying voltage on the RSSI. Figure 6.2 shows the change in RSSI values concerning the change in battery voltages. As can be seen from the figure, RSSI is barely affected by the changes in the supply voltage with DC/DC regulator enabled and stays between the range of -53 dBm to -56 dBm. It was also observed that with DC/DC regulator enabled, the current consumption of the MCU is reduced by a factor of two under the same supply voltage. Hence, it can be concluded that the signal strength is unaffected by the varying supply voltage with DC/DC regulator enabled. And as the EADS device may experience varying supply voltage over time, the device should be used with DC/DC regulator to reduce the impact on signal quality.

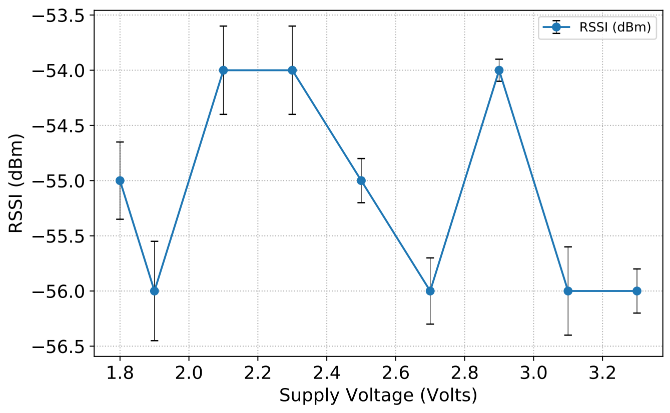


Figure 6.2: Correlation between Supply voltage and RSSI

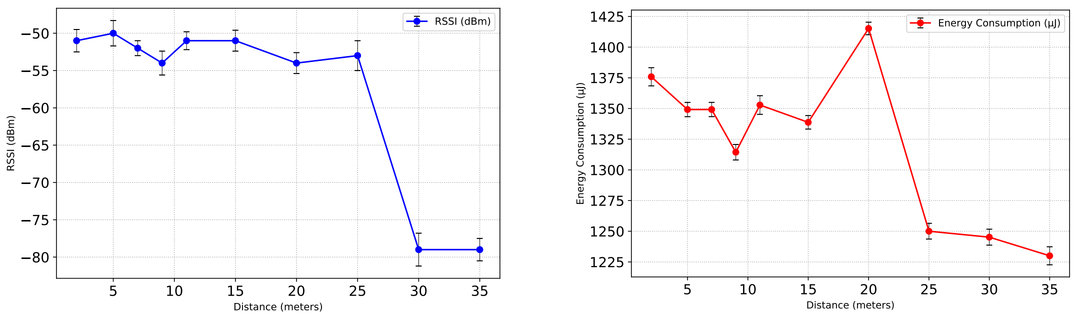


Figure 6.3: Varying distance between router node and non-router node

6.1.5 Varying distance between sink node, router node and non-router node

In an office environment, other wireless communication protocols such as WiFi and BLE are also used. Given that Wirepas operates in 2.4 GHz frequency which is a similar frequency compared to WiFi and BLE, the Wirepas signals may experience some interference. This interference may reduce the signal strength further, thus limiting the maximum distance between the nodes. Hence given the distance and interference, it is important to determine the maximum distance between the nodes ensuring a good signal quality. The distance between a router node and a non-router node as well the distance between sink node and router node was varied to find an optimum distance preserving the signal quality. Figure 6.3 shows the varying RSSI value with increasing distance between the router node and non-router node. As can be seen from the figure, the energy consumption as well as the RSSI of the non-router node vary slightly and are in the range of -50 dBm to -55 dBm and 500 μJ to 580 μJ respectively until 25 m. However, when the non-router node reaches a distance of 30 m, the energy drops by 105 μJ to 400 μJ and the RSSI value reaches -79 dBm. This is because the non-router node is no longer able to communicate with the router node. Hence as result, the RSSI value goes down indicating the poor signal quality as well the energy consumption of the non-router node decreases. Similar behavior is observed in Figure 6.4 that shows the results for the varying distance between sink node and router node. After the distance between the router node and sink node reaches 30 m, there is a decrease in the energy consumption as well as the signal quality. Hence, with this experiment, it can be concluded that the maximum distance between the router node and non-router and router node and sink node should be not greater than *25 m* given the interference of other protocols such as WiFi.

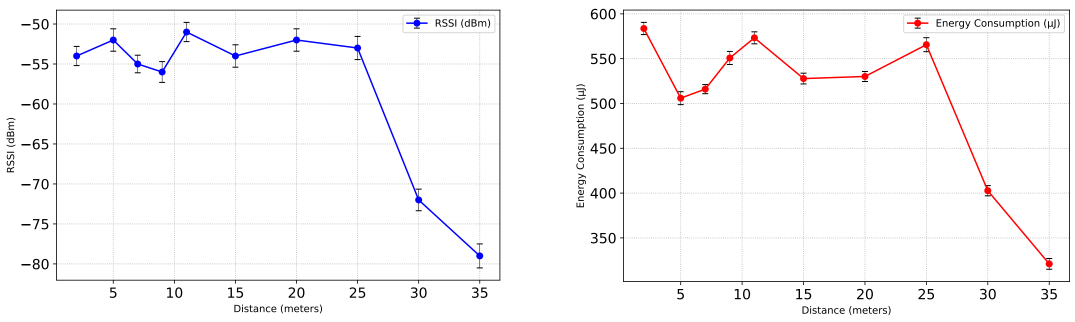


Figure 6.4: Varying distance between sink node and router node

Maximum buffer usage
Wirepas Massive reserves some area in the node memory known as buffer. The maximum buffer usage indicates the usage of RAM buffer on a specific node. A larger buffer value indicates a heavy load on the network. It was observed that when the distance between the nodes was varied and when the non-router node could no longer send/receive messages to/from the router node, the node's buffer usage increases. For instance, if the non-router node is too far away (30 m) to send an acknowledgment message, the node stores the message in its buffer. If the buffer usage increases and reducing the distance between the nodes is no longer an option, an additional router node in the proximity of a non-router can be added. This solution can prevent the additional buffer usage as well as reduce the latency by sending an acknowledgment message as soon as possible.

6.2 Effectiveness of EAS Algorithm

The main aim of using the EAS algorithm is to efficiently utilize the energy stored in the supercapacitor. Some experiments were conducted to analyze the effectiveness of EADS functionality with the EAS algorithm. The EAS algorithm ensures no tasks are lost and also avoids the risk of system shut down due to insufficient energy. Without the EAS algorithm, EADS may shut down in-between executions of a certain task, thus exceeding the task deadline and the wasting the scarce energy utilized for incomplete task execution. The hardware modules used by EADS device were setup as shown in Appendix C to observe the behavior of EAS algorithm.

The total power budget required for the EADS device to execute all the required functions continuously is obtained as follows:

- Supercapacitor leakage current is 1μA.
- Reception of calendar information and transmission of an acknowledgment message upon successful transmission on a router node = 76μW.
- Forwarding the received calendar information to the display through SPI = 15μW.
- EAS hardware circuit power consumption = 0.3μW.

At 300 lux with 100 mF supercapacitor, it was observed the node could only receive the calendar information and send an acknowledgment signal back. The system shuts down when before starting the display tasks. Furthermore, under the same illumination and capacitance with EAS algorithm, the EADS device executed all the tasks and did not shut down during execution of a task.

Chapter 7

Conclusions

This chapter provides summary (Section 7.1) of this thesis and some future directions (Sections 7.2) on proposed design and implementation.

7.1 Summary

Internet of Things (IoT) is getting more attention nowadays, as it provides tens of billions of interconnected devices to communicate with each other over the Internet. However, since their inception, batteries have been one of the main drivers of these devices. Batteries are harmful to the ecosystem as they contaminate the environment when disposed of carelessly. To reduce the effects of batteries on the environment, energy harvesting can be used where nodes harvest their energy from the surroundings. Capgemini Engineering aims to create a sustainable vision for IoT by developing an Energy-Autonomous Display System (EADS) that can operate without any batteries. EADS consists of an energy harvester (solar cells), energy storage (supercapacitor), power management IC, processing unit, display, and server. The server (Raspberry Pi/Gateway) retrieves calendar information from Office 365 cloud server and sends this information to the node (processing unit) through Wirepas Massive protocol. The node then forwards the received data to the display through SPI communication.

One of the main contributions of this thesis is to address the scalability issue of a battery-less device such as EADS. To allow the scalability of EADS devices, Wirepas Massive protocol was used. The energy consumption of Wirepas Massive when compared with other popular IoT protocols BLE and BLE Mesh, Wirepas router node consumes 95% (*BLE-advertising (System ON sleep mode)*) and 99.5% (relay node) less energy respectively. The scalability of EADS was evaluated with various experiments such as increasing the number of nodes, varying node configurations, and distance between nodes. These experiments were conducted to observe the variations on the energy consumption when the number of nodes increase, and when the distance is varied. The results obtained demonstrated that given the interference, the optimum distance between a router node and non-router node as well as a router node and sink node is *25m*. Furthermore, it was also observed that the energy consumption of a router node does not vary much with an increase in the number of transmissions i.e, relaying messages to other non-router nodes. The nodes were configured with two different operating modes supported by Wirepas Massive and it was concluded that Low-Energy mode consumes significantly lower energy than Low-Latency mode. Additionally the correlation between the battery voltage and RSSI was also examined and it was observed that with DC/DC regulator enabled, varying the supply voltage will not have any influence on the signal strength.

As EADS is a battery-less device, it can shut down when insufficient energy is available given the unpredictable behavior of energy storage. Therefore it is important to keep a track of available energy during the device operation. To overcome this, an intelligent algorithm was proposed that drives the device's operation by keeping track of available energy. The functionality of EADS was divided in terms of tasks and these tasks were executed depending on the priority, available energy, and energy required per task. The task was executed when the available energy is greater than the energy required for that task as well as the set threshold value. If the available energy was less, then the task is not executed and the algorithm determines the estimated sleep time on the energy stored. The results show that EADS functionality is improved by using the EAS algorithm.

7.2 Limitations and Future Work

This work has opened the doors for some future directions of research. Some of the possible limitations and future works are described below:

- *Energy-Aware Scheduling with Intermittent computing:*
  The proposed Energy-Aware Scheduling algorithm decides task execution depending on the available energy and task priority. This algorithm prevents the risk of system shut down during the task execution due to insufficient energy. The EAS algorithm also decides how long the system (display) should be in sleep mode (turned off) by estimating the needed sleep time based on the average energy consumption. One of the limitations of the proposed algorithm is that it cannot store the system state if the system shuts down during operation.

  Given the intermittent behavior of the energy harvester and energy storage, there still exists a possibility where EADS may shut down due to insufficient energy when harvester stops harvesting the energy. For instance, let's consider a scenario where the corridor lights, source of energy harvester, are damaged and have to be replaced. In this situation, the current state of EADS is lost and EADS cannot turn on as the energy storage does not have any energy. To avoid the risk of losing the current state, the node can start storing the current state in the Non-Volatile Memory (NVM). This feature is known as Intermittent computing. In this way, the state of the system is preserved before system shut down and then later restored once available energy exceeds the set threshold, thus conserving energy.

- *Over-the-Air-Programming (OTAP):*
  Wirepas Massive supports OTAP that allows updating the software running in a device via the mesh network. OTAP uses secure and reliable updation of new software versions to an existing network and to update devices. It can be used to update both the application and the Wirepas Stack. During OTAP, the stack capabilities are terminated and the gateway and node can no longer communicate with each other. Therefore, it is important that the process is completed as early as possible. The total energy consumption and time for a successful OTAP operation are 325 mJ and 800 seconds (see Appendix D for more details). A 100 mF capacitor cannot store this amount of energy. Although a 470 mF capacitor can store more energy than required for the entire OTAP operation, the voltage stored by this capacitance after 900 seconds is 0.197V at 300 lux. Furthermore, as the current system is incapable of storing current system state, the energy spent on incomplete OTAP process is wasted if the energy stored is depleted during the process and has to be started again. Hence to ensure a successful OTAP operation, two options can be employed - (i) a larger capacitance or 470 mF capacitor with increased illumination can be used to specifically perform the OTAP operation, and (ii) a software can be implemented that triggers the OTAP process when the capacitor reaches its full storage potential, and ensure successful completion.

Bibliography

[1] Apriorit. Connecting iot devices with mesh networking. https://www.apriorit.com/dev-blog/673-mobile-mesh-networking-for-iot.

[2] Mathias Baert, Jen Rossey, Adnan Shahid, and Jeroen Hoebeke. The bluetooth mesh standard: An overview and experimental evaluation. *Sensors*, 18(8):2409, 2018.

[3] Daniel Boon. Energy autonomous display system. https://gitlab.acidspace.nl/student/2020-1/meeting-room-display-2/-/tree/Final_version.

[4] Speed Check. Received signal strength indicator. https://www.speedcheck.org/wiki/rssi/.

[5] Antonio Cilfone, Luca Davoli, Laura Belli, and Gianluigi Ferrari. Wireless mesh networking: An iot-oriented perspective survey on relevant technologies. *Future Internet*, 11(4):99, 2019.

[6] Joshua Curry and Nick Harris. Powering the environmental internet of things. *Sensors*, 19(8):1940, 2019.

[7] Carmen Delgado and Jeroen Famaey. Optimal energy-aware task scheduling for batteryless iot devices. *IEEE Transactions on Emerging Topics in Computing*, 2021.

[8] Digi. Zigbee wireless mesh networking. https://www.digi.com/solutions/by-technology/zigbee-wireless-standard.

[9] Digi-Key Electronics. Wireless modules operating in the sub-ghz bands. https://www.digikey.nl/nl/articles/wireless-modules-operating-in-the-sub-ghz-bands.

[10] Mahmoud Elkhodr, Seyed Shahrestani, and Hon Cheung. Emerging wireless technologies in the internet of things: a comparative study. *arXiv preprint arXiv:1611.00861*, 2016.

[11] Ericsson. Internet of things forecast. https://www.ericsson.com/en/about-us/company-facts/ericsson-worldwide/india/authored-articles/ushering-in-a-better-connected-future.

[12] Vincent Genevès. Online joining and provisioning. https://wirepas.com/wirepas-go-2020-online/technical-track/.

[13] Nicholas Liew Long Guang, Thillainathan Logenthiran, and Khalid Abidi. Application of internet of things (iot) for home energy management. In *2017 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, pages 1–6. IEEE, 2017.

[14] Lakshmikanth Guntupalli, Jorge Martinez-Bauset, Frank Y Li, and Mary Ann Weitnauer. Aggregated packet transmission in duty-cycled wsns: Modeling and performance evaluation. *IEEE Transactions on Vehicular Technology*, 66(1):563–579, 2016.

[15] Anisha Gupta, Rivana Christie, and PR Manjula. Scalability in internet of things: features, techniques and research challenges. *Int. J. Comput. Intell. Res*, 13(7):1617–1627, 2017.

[16] Josiah Hester, Kevin Storer, and Jacob Sorber. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–13, 2017.

[17] Joan. Joan 6. `https://getjoan.com/shop/joan-6/`.

[18] GGKWMSIR Karunarathne, KADT Kulawansa, and MFM Firdhous. Wireless communication technologies in internet of things: a critical evaluation. In *2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)*, pages 1–5. IEEE, 2018.

[19] Katarzyna Kosek-Szott, Janusz Gozdecki, Krzysztof Loziak, Marek Natkaniec, Lukasz Prasnal, Szymon Szott, and Michal Wagrowski. Coexistence issues in future wifi networks. *IEEE Network*, 31(4):86–95, 2017.

[20] Silicon Labs. Thunderboard sense. `https://www.silabs.com/development-tools/thunderboard/thunderboard-sense-two-kit`.

[21] Ryan Lester. Scalability – what it means and why it's so critical in the iot? `https://www.itproportal.com/features/scalability-what-it-means-and-why-its-so-critical-in-the-iot/`.

[22] Kiwan Maeng, Alexei Colin, and Brandon Lucia. Alpaca: Intermittent execution without checkpoints. *arXiv preprint arXiv:1909.06951*, 2019.

[23] Zephyr Project Members. Zephyr project. `https://docs.zephyrproject.org/latest/introduction/index.html`.

[24] Wirepas Mesh. Our product - wirepas mesh. `https://wirepas.com/what-is-wirepas-mesh/`.

[25] E Dalila Pinedo-Frausto and J Antonio Garcia-Macias. An experimental analysis of zigbee networks. In *2008 33rd IEEE Conference on Local Computer Networks (LCN)*, pages 723–729. IEEE, 2008.

[26] Gwendal Raoul. Wirepas mesh sdk and github. `https://wirepas.com/wirepas-go-2020-online/technical-track/`.

[27] Herman Roebbers. Ultra low power: How to reach energy efficiency in connected product design? `https://www.altran.com/nl/en/news_press_release/ultra-low-power/`.

[28] Adnan Sabovic, Carmen Delgado, Dragan Subotic, Bart Jooris, Eli De Poorter, and Jeroen Famaey. Energy-aware sensing on battery-less lorawan devices with energy harvesting. *Electronics*, 9(6):904, 2020.

[29] Farzad Samie, Lars Bauer, and Jörg Henkel. Iot technologies for embedded computing: A survey. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–10. IEEE, 2016.

[30] Muhammad Moid Sandhu, Sara Khalifa, Raja Jurdak, and Marius Portmann. Task scheduling for simultaneous iot sensing and energy harvesting: A survey and critical analysis. *arXiv preprint arXiv:2004.05728*, 2020.

[31] Nordic Semiconductors. Generic onoff model. `https://infocenter.nordicsemi.com/`.

[32] Nordic Semiconductors. nrf5 mesh sdk. `https://www.nordicsemi.com/Products/Development-software/nRF5-SDK-for-Mesh`.

[33] Srikanth Sistu, Qingzhi Liu, Tanir Ozcelebi, Esko Dijk, and Teresa Zotti. Performance evaluation of thread protocol based wireless mesh networks for lighting systems. In *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–8. IEEE, 2019.

[34] Clemanis Matrix Technology. Wirepas 5.0 raspberry hat. `https://store.clemanis.com/en/iot/1187-raspberry-hat.html`.

[35] Ishaq Unwala, Zafar Taqvi, and Jiang Lu. Thread: An iot protocol. In *2018 IEEE Green Technologies Conference (GreenTech)*, pages 161–167. IEEE, 2018.

[36] Wirepas. Is wirepas ble mesh? `https://wirepas.com/blog/you-ask-we-answer-is-this-a-ble-mesh`.

[37] Wirepas. Wirepas mesh protocol. `https://developer.wirepas.com/support/solutions/articles/77000484367-wirepas-mesh-concepts`.

[38] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 41–53, 2018.

Appendix A

Arduino Code for LED Ring

This appendix describes the code used to make the WS2815 digital LED ring adjustable to 300 lux and 500 lux. The *FastLED* library is used to control the LEDs.

A.1 For 300 lux

```
#include <FastLED.h>
#define NUM_LEDS 16
#define LEDPIN 7
#define brightness 68


CRGBArray<NUM_LEDS> leds;

void setup() {
  FastLED.addLeds<NEOPIXEL, LEDPIN>(leds, NUM_LEDS);
}

void loop() {
  FastLED.setBrightness(brightness);
  leds[0] = CRGB::White;
  leds[2] = CRGB::White;
  leds[4] = CRGB::White;
  leds[6] = CRGB::White;
  leds[8] = CRGB::White;
  leds[10] = CRGB::White;
  leds[12] = CRGB::White;
  leds[14] = CRGB::White;
  leds[16] = CRGB::White;

  FastLED.show();
}
```

A.2 For 500 lux

```
#include <FastLED.h>
#define NUM_LEDS 16
#define LEDPIN 7
#define brightness 80

CRGBArray<NUM_LEDS> leds;

void setup() {
  FastLED.addLeds<NEOPIXEL, LEDPIN>(leds, NUM_LEDS);
}


void loop() {
 FastLED.setBrightness(brightness);
 for (int i = 0; i < NUM_LEDS; i++){
   leds[i] = CRGB::White;
  }
 FastLED.show();
}
```

Appendix B

Energy measurements for BLE Mesh

Bluetooth Mesh utilizes a mesh topology to support communication between numerous devices running Bluetooth Mesh protocol. Unlike BLE, Bluetooth Mesh supports some features such as friendship feature and proxy feature to enable a mesh ecosystem. It is important to measure the energy consumption of these features to comprehend the minimum energy required by each feature for a specific event as energy consumption has a major affect on EADS. The nRF5 Mesh SDK provides some examples that can be leveraged to demonstrate the Bluetooth Mesh network and estimate the energy consumed. A ”Light Switch” example [32] from the SDK was used that was exploited to measure the energy consumption for low-power node, friend node and relay node. This example uses GATT for provisioning and instantiates a Generic OnOff model that can be used to control light switch servers. The GATT provisioning allows a device without support for the advertising bearer to provision and/or communicate with the mesh network through a device using a GATT interface [31]. The Generic OnOff model implements the message based interface required to set the OnOff value on the server [31]. This interface API is responsible for validating the packet formats and field values.

When configured to interact with a device with a Generic OnOff Server model, the device running this example turns on the LED on the light switch server device upon a button press, which emulates triggering the occupancy sensor. It also sends an off message to the light switch server device after five seconds to emulate inactivity.

The nRF5 SDK does not support the friend feature for Bluetooth Mesh. Hence pre-compiled version of the Friend node from the Zephyr RTOS [23] to fulfill the Friend device role was used. The nRF Mesh mobile app is used for provisioning of low-power node. Furthermore, as the low-power node uses a friend node, both must be provisioned for the same mesh network. Once the friend node and low-power node are configured and provisioned, the low-power node enters the idle state. The friendship establishment process is started by pressing button 3 and the device starts searching for an appropriate Friend in the mesh network. The energy consumption was measured with a similar setup as discussed in Section 5 with STM32 power shield for each feature.

Appendix C

Hardware setup for Energy-Aware Scheduling

Figure C.1 shows the hardaware connection with the processing unit and EAS hardware. The SUPERCAP_PIN is the *Batt* pin of PMIC, connected to the hardware circuit consisting of resistors and transistors. The output of the voltage divider circuit is connected to the analog pin *P0.03*. The OUTPUT_VOLTAGE_PMIC is the *Vout* pin of PMIC which is connected to *Vdd* pin of the processing unit.

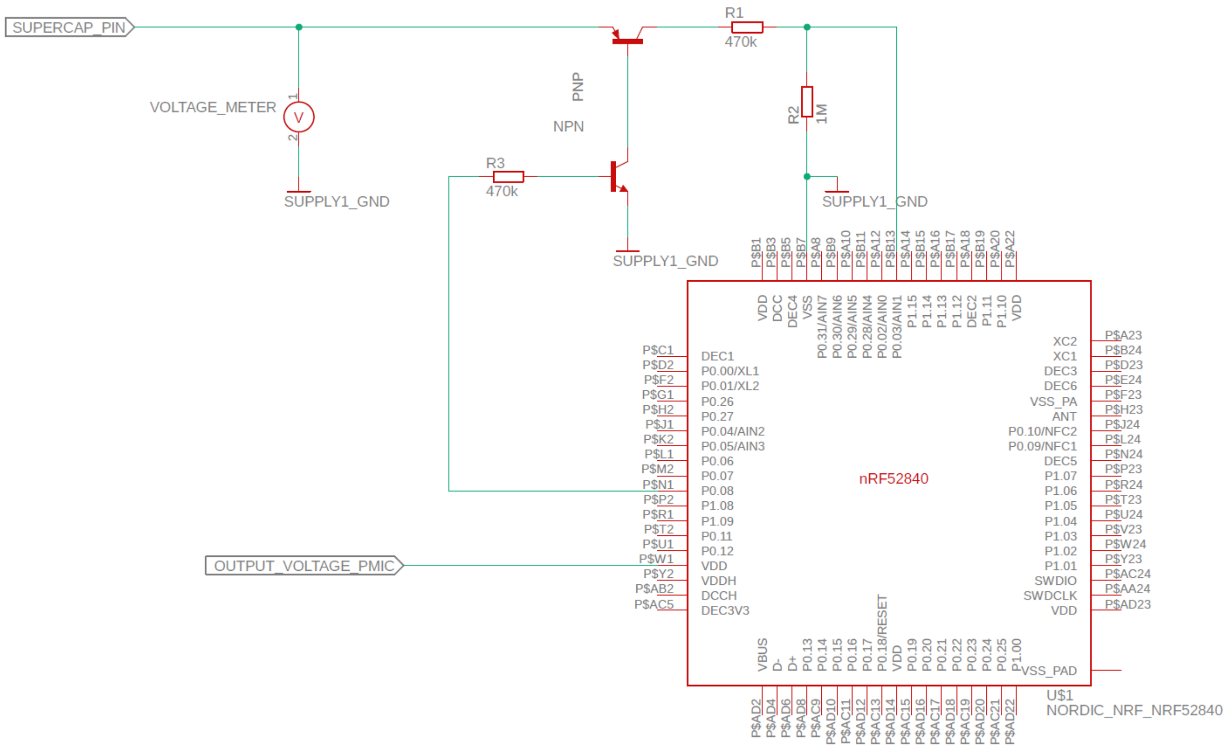


Figure C.1: Hardware setup for Energy-Aware Scheduling

Appendix D

Over-the-Air-Programming (OTAP)

As described in Section 7.2, Wirepas Massive OTAP allows updating the software running in a node via the mesh network. OTAP can be used to update both user applications and the Wirepas stack. OTAP needs to be done as fast and as efficiently as possible. To allow this, nodes use a special point-to-point communication mode with a high data rate to exchange scratchpad images. Ergo, the normal network operations are disrupted during the update process. Wirepas OTAP mechanism relies on three main steps:

- **Propagate** : To spread a new update image (called a scratchpad) to all nodes in a given network. As soon as a node sees a newer scratchpad, this new image is automatically exchanged using an optimized peer-to-peer communication between nodes and stored in the node's scratchpad memory. This mechanism ensures that all nodes will receive the new image when the image is present in the network.
- **Trigger** : During this phase, the backend is used to query the information from all nodes via the Wirepas Massive network to ensure they all have the proper scratchpad in the memory. Furthermore, the backend triggers the OTAP by sending a command to all nodes when all the information is correct. At the end of this step, all nodes have scratchpad and are ready to process it.
- **Process and apply the update** :
  After the OTAP process is triggered, the node reboots and transfers execution to the bootloader. The bootloader verifies the integrity of scratchpad images and checks if the scratchpad contains valid binaries to be processed. If yes, the bootloader decrypts, decompresses, and writes the binary to the node's flash, otherwise, it just ignores the scratchpad and set it as processed. At the end of the process, the bootloader reboots the node and the new version is executed.

As per the description above, performing OTAP is a long and tedious process. To ensure that the entire operation is executed successfully, the nodes should have sufficient energy to stay awake during the entire OTAP process. Hence, an experiment was conducted to examine the total energy required by a single node to be a part of the OTAP process. The entire OTAP operation was divided into tasks that reflect the above described OTAP steps. Table D.1 shows the energy and time required for each action to complete the OTAP process. A network of three router nodes placed at a distance of 10m from each other was considered to perform the OTAP process. The total time required to complete the OTAP process for a network of two nodes is around 800 seconds. This time varies with the increase in the number of nodes. As observed, the total energy consumed is 321.012 mJ, which exceeds the storage capacity of the 100 mF supercapacitor used i.e, 162 mJ. However, 470 mF supercapacitor may support the OTAP operation, the voltage stored by this capacitance after 900 seconds is 0.197V at 300 lux. Moreover, the entire OTAP process needs to be done expeditiously as the normal Wirepas stack operations are temporarily terminated. Given the unpredictable behavior of the energy storage and energy harvester, the current EADS prototype cannot support the OTAP process. In the future, a larger capacitance such as above 500 mF can be used to sustain EADS until the OTAP process is completed as it can provide energy of 810 mJ (for 1.8V). Additionally, the supercapacitors used currently 100 mF (470 mF) can only produce a voltage of 0.98V (0.197V) at 300 lux and 1.109V (0.879V) at 500 lux respectively for around 900 seconds. This voltage is not sufficient as the minimum voltage required for the node to turn on is 1.8V. Furthermore, a software functionality that triggers the OTAP process when the capacitance storage is at its full potential can be implemented.

No.	**OTAP Tasks**	**Time (seconds)**	**Energy Consumption (μJ)**
1.	Join network and get status	20	59597
2.	Loading scratchpad to WNT	60	14532
3.	Loading scratchpad to node	339	117906
4.	Activation	325	128979
	Total	**745**	**322mJ**

Table D.1: OTAP tasks energy consumption