**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Bruno Chianca Ferreira

# Energy-aware Gossip Protocol for Wireless Sensor Networks

November 2019

**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Bruno Chianca Ferreira

# Energy-aware Gossip Protocol for Wireless Sensor Networks

Master Dissertation
Master Degree in Informatics Engineering

Dissertation supervised by
**Professor João Marco C. Silva**
**Professor Vitor Francisco Fonte**

November 2019

## ACKNOWLEDGEMENTS

I would first like to thank my dissertation supervisors Professors João Marco Silva and Vitor Francisco Fonte who have always been available to guide this work towards the goal and gave extremely valuable insights and research guidance. Thank you also to all other Professors from UMinho from whom I had the pleasure to learn and work with.

I also would like to acknowledge my previous employees, specially my last two managers Øistein Martinsen and Miguel Gomes for being so comprehensive and supporting me on this new journey after so many years working with power systems.

Finally, I would like to thank my family who have always been supportive despite the geographical distances and my partner Anna for all the support and patience during this journey.

## STATEMENT OF INTEGRITY

## ABSTRACT

In Wireless Sensor Networks (WSNs), typically composed of nodes with resource constraints, leveraging efficient processes is crucial to enhance the network longevity and consequently the sustainability in ultra-dense and heterogeneous environments, such as smart cities. Epidemic algorithms are usually efficient in delivering packets to a sink or to all it's peers but have poor energy efficiency due to the amount of packet redundancy. Directional algorithms, such as Minimum Cost Forward Algorithm (MCFA) or Directed Diffusion, yield high energy efficiency but fail to handle mobile environments, and have poor network coverage.

This work proposes a new epidemic algorithm that uses the current energy state of the network to create a topology that is cyclically updated, fault tolerant, whilst being able to handle the challenges of a static or mobile heterogeneous network. Depending on the application, tuning in the protocol settings can be made to prioritise desired characteristics. The proposed protocol has a small computational footprint and the required memory is proportional not to the size of the network, but to the number of neighbours of a node, enabling high scalability.

The proposed protocol was tested, using a ESP8266 as an energy model reference, in a simulated environment with *ad-hoc* wireless nodes. It was implemented at the application level with UDP sockets, and resulted in a highly energy efficient protocol, capable of leveraging extended network longevity with different static or mobile topologies, with results comparable to a static directional algorithm in delivery efficiency.

***Keywords***—energy-aware, epidemic, IoT , routing, WSN

# RESUMO

Em Redes de Sensores sem Fios (RSF), tipicamente compostas por nós com recursos limitados, alavancar processos eficientes é crucial para aumentar o tempo de vida da rede e consequentemente a sustentabilidade em ambientes heterogêneos e ultra densos, como cidades inteligentes por exemplo. Algoritmos epidêmicos são geralmente eficientes em entregar pacotes para um *sink* ou para todos os nós da rede, no entanto têm baixa eficiência energética devido a alta taxa de duplicação de pacotes. Algoritmos direcionais, como o *MCFA* ou de *Difusão Direta*, rendem alta eficiência energética mas não conseguem lidar com ambientes móveis, e alcançam baixa cobertura da rede.

Este trabalho propõe um novo protocolo epidêmico que faz uso do estado energético atual da rede para criar uma topologia que por sua vez atualizada ciclicamente, tolerante a falhas, ao mesmo tempo que é capaz de lidar com os desafios de uma rede heterogênea estática ou móvel. A depender da aplicação, ajustes podem ser feitos às configurações do protocolo para que o mesmo priorize determinadas características. O protocolo proposto tem um pequeno impacto computacional e a memória requerida é proporcional somente à quantidade de vizinhos do nó, não ao tamanho da rede inteira, permitindo assim alta escalabilidade.

O algoritmo proposto foi testado fazendo uso do modelo energético de uma ESP8266, em um ambiente simulado com uma rede sem fios *ad-hoc*. Foi implementado à nível aplicacional com *sockets UDP*, e resultou em um protocol energeticamente eficiente, capaz de disponibilizar alta longevidade da rede mesmo com diferentes topologias estáticas ou móveis com resultados comparáveis à um protocolo direcional em termos de eficiência na entrega de pacotes.

***Palavras chave***—energy-aware, epidêmico, IoT, RSF, roteamento

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACRONYMS

**B**

| | |
|---|---|
| **BS** | Base Station. |

**C**

| | |
|---|---|
| **CSV** | Comma-separated Values. |

**E**

| | |
|---|---|
| **EAGP** | Energy Aware Gossip Protocol. |
| **EAGPD** | Energy Aware Gossip Protocol Digest. |
| **EEPROM** | Electronically Erasable Read Only Memory. |

**G**

| | |
|---|---|
| **GPS** | Global Positioning System. |
| **GUI** | Graphical User Interface. |

**I**

| | |
|---|---|
| **I2C** | Inter-Integrated Circuit. |
| **IOT** | Internet of the Things. |

**J**

| | |
|---|---|
| **JSON** | JavaScript Object Notation. |

**L**

| | |
|---|---|
| **LEACH** | Low Energy Adaptive Clustering Hierarchy. |

**M**

| | |
|---|---|
| **MAC** | Medium Access Control. |

| | |
|---|---|
| MCFA | Minimum Cost Forwarding Algorithm. |
| MECN | Minimum Energy Communication Network. |
| **O** | |
| OS | Operating System. |
| OSM | Open Street Maps. |
| **P** | |
| PDU | Protocol Data Unit. |
| PEGASIS | Power-Efficient Gathering in Sensor Information Systems. |
| PLUMTREE | Push-lazy-push Multicast Tree. |
| **Q** | |
| QOS | Quality of Service. |
| **R** | |
| RAM | Random Access Memory. |
| RF | Radio Frequency. |
| ROM | Read-Only Memory. |
| RTC | Real Time Clock. |
| **S** | |
| SDN | Software Defined Networks. |
| SOB | System on a Board. |
| SOC | System on a Chip. |
| SOM | System on a Module. |
| SPI | Serial Peripheral Interface. |
| SPIN | Sensor Protocols for Information via Negotiation. |
| **T** | |

| | |
|---|---|
| **TEEN** | Threshold-Sensitive Energy Efficient Sensor Network Protocol. |
| **TTL** | Time-to-Live. |
| **U** | |
| **UART** | Universal Asynchronous Receiver/Transmitter. |
| **W** | |
| **WSN** | Wireless Sensor Networks. |

# 1 INTRODUCTION

Wireless Sensor Networks (WSN) are composed of several nodes that have a specific purpose of monitoring and measuring diverse types of systems and physical phenomena. Each member of a WSN, called node, is usually very small and low cost equipment designed to run on limited energy source [Anastasi et al., 2009]. These nodes usually should not need maintenance for years, making it very important to find optimal ways to lower the required power to sense, store and forward the data to a central entity like a Base Station (BS). Connecting each node directly to the BS would add high energy cost to the network design, so, WSN are usually configured to relay data from one node to another until it reaches the final destination. The naïve way to solve this problem is to set each node to broadcast packets to each of its neighbours, and as a result each node would eventually receive all packets created by every node. That however would result in each node receiving the same packet several times coming from different paths. Many routing protocols have been designed to find ways of relaying packets without the need to sending all packets to every node, and thus saving resources. When energy resources are saved, the lifetime of the whole network is prolonged.

Several routing protocols for WSN have been designed and an encompassing taxonomy of them is described in [Al-Karaki and Kamal, 2004], which, regarding network structured-based protocols, is mainly divided in three groups: *flat-based* routing, *hierarchical-based* routing and *location-based* routing. In the *flat-based* routing scheme [Schurgers and Srivastava, 2001; Braginsky and Estrin, 2002; Al-Karaki and Kamal, 2004], each node has the same function whilst in the *hierarchical-based* routing [Intanagonwiwat et al., 2000; Al-Karaki and Kamal, 2004] some nodes acquire the role of cluster head and concentrate data before forwarding to another node and eventually the base station. *Location-based* protocols [Al-Karaki and Kamal, 2004; Kumar et al., 2017] use the sensor's geographical position to choose their role in the routing scheme. In some routing protocols, the nodes can assume a static role after being defined during a topology discovery stage. As a consequence, this might lead to energy depletion in specific nodes before others, creating an energy unbalance in the network, and even in some extreme cases completely isolating part of the network. Having this in mind, another approach should be also considered, adaptive protocols that take in consideration the current state of the network and automatically update the node's role.

Routing protocols can also be divided regarding the data traffic model. In the continuous delivery flow model, the sensor nodes report newly sensed data to a sink periodically. In the event-driven model, the sensing nodes only report the occurrence of pre-defined events, and if the sink node is interested in data related to the event they issue a query. The query-based model is used when the sink node requests data from a previously known node, independently of an event. The definition of the model depends ultimately of the application and not every protocol is suitable for all situations.

When controlling the node's behaviour in the network is required, it is important that the state of all nodes are taken in consideration, requiring a high delivery rate of messages. In such cases, epidemic[1] approaches usually yield positive results [Al-Karaki and Kamal, 2004; Leitao, 2007]. An epidemic protocol can create a big communication overhead when nodes receive repeated messages, which makes them less energy efficient. So, for applications that require a specific direction for the data flow, epidemic protocols might not be the best option. Many different routing protocols have been studied to enhance WSN schemes, each one prioritising a required characteristic, like high throughput, high delivery rate or low latency[Al-Karaki and Kamal, 2004]. Also, to increase network energy conservation, different approaches can be considered: duty cycling, data-driven schemes or mobility-based schemes [Anastasi et al., 2009].

The main motivation for this work is to design a continuous delivery epidemic routing protocol that solves the issue with data redundancy while still keeping a high delivery rate and good network coverage. It shall aim at not only prolonging the lifetime of a WSN by using energy more efficiently, but also by exploiting the remaining energy level as a factor for the topology definition. Another important aspect is that this protocol shall enable the network to reconfigure automatically in case of topology changes without the need of intervention.

## 1.1 OBJECTIVE

The objective of this dissertation is to design an energy-aware communication protocol capable of reducing and balancing energy consumption in WSNs. The main purpose of the proposed protocol is to create a new way of reducing traffic in an epidemic distribution with the following goals:

1. Lifetime of the network - Increasing the lifetime of the network is a main goal of the protocol, and since nodes in a WSN both create and forward sensed data, the algorithm complexity has direct impact in the computational and communication energy used by the nodes. The proposed protocol shall introduce a routing technique that

1 Also called gossip routing

can comply with all the goals while keeping computational complexity to a minimum in order to ensure the longevity of the network.

2. Delivery rate of packets - The proposed protocol must be able to ensure high delivery rate of packets. A trade-off between delivery rate and energy efficiency is acceptable as long as it is an application design choice. Other negative Quality of Service (QoS) aspects that are acceptable in order to ensure the lifetime of the network are high jitter and high latency.

3. Data redundancy - Duplication of delivered packets is an expected side effect of high coverage support, but it can have a negative impact in the network lifetime since nodes will eventually be relaying unnecessary duplicated data. The protocol shall implement features that can reduce data redundancy without affecting the delivery rate.

4. Fault tolerance - Nodes in a WSN can be deployed in a completely flexible and inconsistent distribution. Furthermore the nodes, even the sink, can also be mobile, and have the geographical position changed constantly. The protocol must ensure that this has no affect in the functionality, even if some nodes fail and stop participating in the network.

5. Data traffic - The time-driven continuous delivery data model shall be considered for this protocol. Meaning that nodes' sensors are cyclically reading new data and relaying to the network.

6. Scalability - The network performance and longevity should not be affected by the scale of the network. Hence, the protocol shall avoid the maintenance of complex and long tables, buffers and lists unless when used for keeping a list of neighbours.

## 1.2 METHODOLOGY

In order to fulfil the objectives of this proposal, this work is divided in two main stages: study and experimentation. The study phase covers the existing literature about routing protocols for WSN, identifying the diverse aspects and characteristics of some of them concerning energy consumption and efficiency. Also, since there are many different network simulators available, some will be tested and one chosen for the experimentation stage.

During the experimentation stage, the chosen simulators are used to implement the designed protocol and collect metrics about its behaviour, and later the best solutions are benchmarked. The protocols are tested in different scenarios and different topologies to explore characteristics under different well established conditions as described in details in Chapter 5. The main steps of them work are sumarized as:

- Study the existing routing protocols for WSN;
- Study the existing simulation tools available for WSN and choose one or more to be used throughout the development and evaluation phases;
- Define the main performance and efficiency metrics that will be chased as primary targets;
- Fine tune the routing protocol parameters to deliver best results for the chosen metrics;
- Measure and benchmark the protocol comparing to well established protocols described in literature.

## 1.3 SUMMARY

This chapter briefly introduced the problem this work aims to solve, the objectives and methodology to be adopted.

### 1.3.1 *Dissertation Outline*

For the next chapters, Chapter 2 presents presented basic concepts about wireless sensor and WSNs and related work in routing protocols for WSN, showing some of the protocols that are classified similarly to the proposed protocol. Chapter 3 describes the proposed protocol, presenting the design goals and constrains considered. Chapter 4 introduces the possible platforms for testing the implementation of the protocol while Chapter 5 presents the methodology that shall be used for testing. Chapter 6 presents the results obtained from testings and the considerations about the results. Finally, Chapter 7 presents the final conclusions stemming from this work and possible future work related to this dissertation.

# 2 RELATED WORK

This section will introduce related work in WSN routing protocols. An overall introduction about some protocols is presented and some aspects related to this work are highlighted. Since the number of existing protocols is too big, only the few that are more directly related to this work will be introduced. Other works have been made with the goal of classifying all the protocols and extensive lists and classifications can be found in [Al-Karaki and Kamal, 2004; Anastasi et al., 2009].

Routing protocols are vital to WSNs due to its distributed nature. Data created in one node need to be disseminated in the network to reach its final destination and the way this routing is implemented varies depending on specific applications and requirements. Some protocols are reactive, waiting for a data request in order to calculate the route towards the destination, others are proactive and maintain routing tables which are calculated in advance. The existing routing protocols for WSN are usually classified regarding their architecture as *flat-based*, *hierarchical-based* and *location-based* routing as introduced in [Al-Karaki and Kamal, 2004] and shown in Figure 2.1. Location-based protocols are usually also classified as flat or hierarchical and therefore, examples of such protocols will be presented inside those classifications.

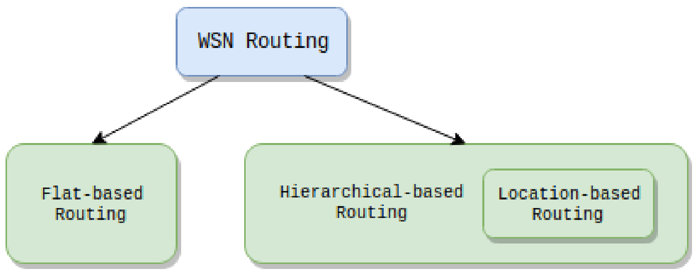


Figure 2.1.: WSN Routing Protocols Taxonomy

## 2.1 THEORETICAL BACKGROUND

Before discussing related work regarding existing routing protocols, an introduction about the wireless sensors and WSN is presented in this section.

### 2.1.1 *Wireless Sensors*

Wireless sensors are a special type of sensor that does not require to be physically connected to any other device in order to transfer the data it has acquired. It is usually comprised of a *compute*, a *sensing* (usually associated with an analogue to digital converter), a *memory* and a *communication* (radio) units as shown in Figure 2.2. Since these sensors are not connected to any other device and are often deployed in remote areas, the sensor's design often include energy harvesting from the environment, such as heat, solar, wind or vibration [Shaikh and Zeadally, 2016]. These sources of energy are usually intermittent, so it is important to equip the sensor with an energy buffer unit, such as a battery.

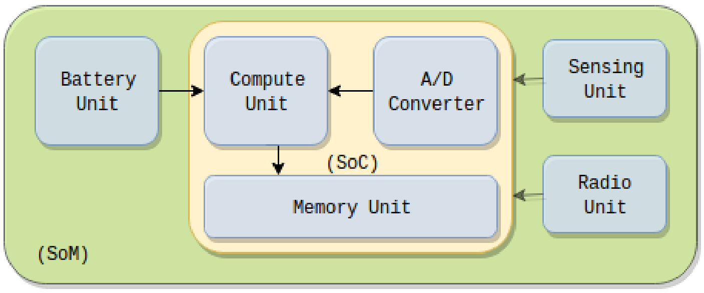


Figure 2.2.: Typical Wireless Sensor

- **Battery Unit** - The battery unit is usually the main power source for wireless sensors. It is commonly comprised of a battery, mainly Lithium Polymer batteries in the most modern devices, a protection circuit and a battery charger. The function of the battery charger is to provide a stable voltage and a controlled current for charging it, since exceeding current could lead to a temperature increase and possibly a fire. Each type of battery has a required safe charging voltage and current;
- **Compute unit** - The compute unit is where the main logic for controlling the other embedded hardware is located. It can be a micro-controller, a System on a Module (SoM) or a System on a Chip (SoC) for instance, but there are many other definitions sometimes created by manufacturers for marketing purposes. One important aspect when choosing the compute unit is to get enough computing power needed for the application, using as least power as possible [Malewski et al., 2018]. Hence, the system

should always aim to get just enough performance as it is needed, typically without any spare computing power, therefore saving more energy;

- **Micro-controllers** are a category of small factor computers usually composed of a 8 or 16 bit microprocessor and sometimes equipped with some accessories, like Random Access Memory (RAM), Read-Only Memory (ROM), Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI) or Inter-Integrated Circuit (I2C) interfaces. The modern versions of micro-controller are very energy efficient and depending of the manufacturer they can be equipped with wireless interfaces such as IEEE802. 11 [802.11, 2016], IEEE802.15.4 [802.15.4, 2015] or Bluetooth [BluetoothCore, 2019];
- **System on a Chip** are devices that can concentrate all the required modules to build a computer inside a chip. So the processor, RAM, ROM, and all the peripherals required for communication are housed in the same silicon chip;
- **System on a Module** devices are small footprint systems that can fit on a module that shall or not be connected to a carrier board that make more IO ports available to the user. So, the SoM are one step ahead of the SoC since they install a SoC on a small board with some extra hardware, usually USB ports, video outputs, buttons and LEDs. These devices are more commonly used for prototyping.

• **Sensing Unit** - The sensing unit is responsible for reading a physical phenomena and transforming into an eletrical signal that can be read by the computing unit. Most of the devices have built-in an analog to digital converter that can read the electrical signal and transform into a digital signal that the processor can understand. It is also possible that the sensing unit also has a embedded computer that can read the electrical signal, and provide the digital value to the computer via a communication protocol such as UART, SPI or I2C.

• **Memory Unit** - The memory unit is a temporary or permanent storage where the sensor can store its measurements before they are routed out to the base station. This memory can be volatile RAM or non-volatile EEPROM[1] for applications that require permanent storage of data even after the measurements have been delivered.

• **Communication Unit** - The communication unit contains the radio which is used for sending data to other nodes. This unit has one or more radios available for communication via modulation in pre-defined frequencies. Using the same radio, different protocols can be implemented in the MAC/PHY layer [Al-Sarawi et al., 2017] and, since some protocols share the same frequency, some devices, such as the nRF52840[2]

1 Electronically Erasable Read Only Memory (EEPROM)
2 `https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840`

for example, implement different protocols using the same radio simultaneously by using multiplexing.

### 2.1.2 *Wireless Sensors Networks*

Wireless sensors networks are made of nodes of wireless sensors distributed across an area and exchanging information. These nodes usually cannot reach a cloud or central server, so the data they collect is routed to a sink (or base station) on a multi-hop scheme. The sink on the other hand acts as border router and can be both in the WSN and in other network, being able to upload data collected by the sensing nodes to an external entity. Figure 2.3 shows a typical representation of a WSN.

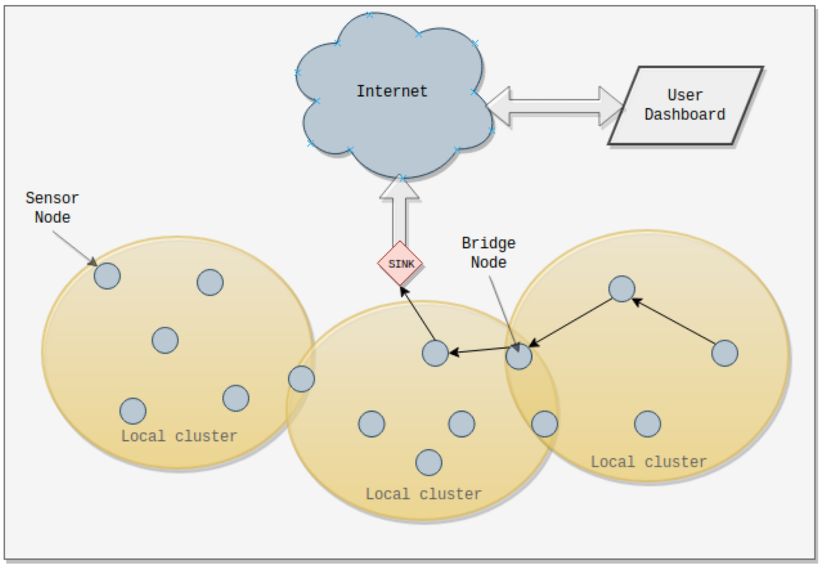


Figure 2.3.: Typical Wireless Sensor Network

It is possible that a cluster of nodes cannot reach a sink but there is a particular node that can reach both an isolated cluster and the cluster where the sink is located. This *bridge node* therefore routes the data between the clusters enabling the data to reach the sink. A node that serves as a bridge will mostly always consume more energy than its surrounding counterparts, and if it gets depleted of energy before the isolated cluster, the whole cluster ends up losing connectivity with the sink. This serves as an example of the importance of energy-awareness when designing routing protocols for a WSN.

### 2.1.3 *Energy Consumption in WSN*

As previously mentioned, since wireless sensors are deployed on remote locations, it is imperative that they consume the least possible energy in order to prolong the network's longevity. There are different ways that energy is consumed in a WSN. The sensor needs to read new data, which has to be processed and stored in memory and further transmitted. So basically, the main energy consumption topics are circuit power (sensing, computation and memory units), and radiated power (communication unit) [Niyazi, 2017].

The way each sensor consumes energy depends on each node itself, as the sensing sub-system for instance, depends much on the type of measurement it samples. Some types of sensing devices require more energy than others, not only depending on the physical phenomena being measured (some require pre-heating of elements before sensing, such as gas sensors for example) but also because some applications require more frequent sensing than others. Typically, the communication on a wireless sensor consumes more energy than the computation part [Anastasi et al., 2009], but if a sensor reading demands filtering or other computation intensive processing, that could change the scenario. It could also happen that routing protocols can consume too much computation energy if they make use of a complex algorithm, or if they need to keep looping through routing tables in memory.

The communication sub-system consumes power either when it is transmitting or receiving data. Most of the sensor nodes however have the ability to put the radio in *sleep* mode, reducing drastically the energy consumption and this option should be used as much as possible [Anastasi et al., 2009]. Since the communication in a WSN works mostly using a multi hop scheme, if one sensor node that is placed in a strategic position, as the bridge node in Figure 2.3, and for instance connect two local clusters runs out of battery, the whole network suffers a loss due to the loss of connection between the local clusters. A network that relies on a specific node for maintaining connectivity of a local cluster cannot afford the loss of such node.

### 2.1.4 *Energy Conservation in WSN*

In order to conserve energy, different techniques can be used depending on the application and design goals. As identified in Section 2.1.3, there are three main subsystems that consume energy, and different approaches can be used to reduce the power consumed in each subsystem or even in all of them simultaneously. As highlighted in [Anastasi et al., 2009], the main focus areas for power reduction in the communication sub-system are *topology management*, *duty cycling*, *medium access control* and *data driven management*.

### *Topology Management*

In order to keep full connectivity in the network some nodes can be assigned different duties. Finding out which node is more suitable for each activity in the network is referred as topology management, and that usually can only be achieved when network redundancy is available. In order to enable proper topology management, sensor nodes need to be aware of the current topology, available paths, state and path redundancy. The problem of discovering the surroundings can become very complex in epidemic WSN and is generally referred as topology discovery [Anastasi et al., 2009; Hao et al., 2018]. With the topology management, a sensor can change its role, by either reacting to an updated condition in the network or by means of an external command. The conditions that trigger updates in the topology configuration varies depending on the routing protocol in use.

### *Duty Cycling*

The idea of duty cycling or sleep cycle management is to reduce the energy consumed by the communication sub-system by turning the radio *ON* and *OFF* in pre-defined amount of times [Lazarescu, 2013].

Duty cycling power control consists in controlling the power *ON* and *OFF*, or toggling between different energy modes of the compute unit, such as *sleep/awake* during a pre-defined amount of time of a complete cycle as represented in Figure 2.4. Considering the full cycle as the time between transmissions, the system should sleep for an amount of time and be awake for the rest of the time. The awake time depends on how much work it is required for the protocol. The higher percentage of sleeping time, the more energy efficient the system is, and the awake time is commonly called duty cycle. This is applicable for every component of the node and even for the whole node itself.

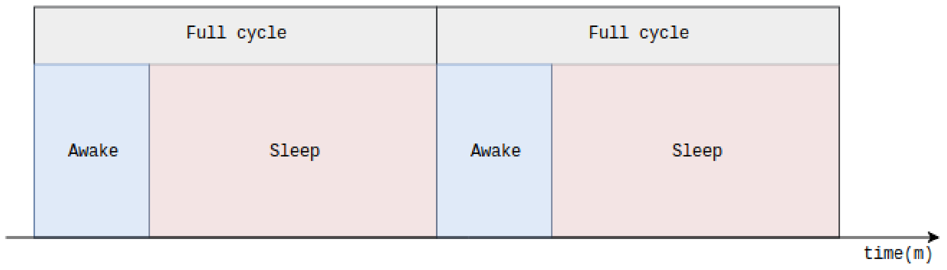


Figure 2.4.: Representation of a Duty Cycle

Duty cycling on the radio unit is usually performed by Medium Access Control (MAC) protocols, as used for instance by the ContikiMAC protocol [Dunkels, 2011]. Nowadays, there are several hardware designed specifically with the Internet of the Things (IoT) market as target. They introduce different ways of implementing low power consumption by switching *OFF* one or more components of the system in order to save energy. The sleeping time of a system can be severely increased when all nodes are synchronised, since in this case all nodes can be put in a *deep sleep* state, and only wake up when is time to work. When the nodes cannot be synchronised, they must be put in a *light sleep* state, in which the radio can still listen the medium and wake up the node via interrupt signal to execute some tasks. The problem of synchronising the nodes can become difficult when there is no Global Positioning System (GPS) global clock or similar available.

*Medium Access Control*

As any other type of network, WSN needs a medium in order to propagate data. Since WSN do not use wires, they need to use electromagnetic radio waves as medium, and the MAC controls the way the nodes access this Radio Frequency (RF) medium. Many MAC protocols exist today such as the IEEE 802.11 family, IEEE 802.15.4 and Zigbee [Al-Sarawi et al., 2017], but they are usually not directly applicable to WSN due to the aspect of energy constrain present on nodes [Schaefer et al., 2013]. A MAC protocol that is specialised in WSN needs to take in consideration and aim for:

- Reduced overhead - Protocols aim at reducing the most possible the amount of overhead added by the protocol;
- Deep sleep when idle - The nodes benefit of being able to stay in deep sleep state when not transmitting, saving energy for not being listening all the time;
- Auto reconfiguration - The protocol needs to enable the auto-reconfiguration of the network without direct intervention.

*Data Driven Management*

The focus on the data driven approaches is to reduce the energy consumption by changing the way the node treats data. The reduction takes place in different forms:

- Data reduction
  - Data compression - Nodes can compress the data before sending to a cluster head, or the data can be compressed by a cluster head before forwarding. The data needs then to be uncompressed or decoded at the sink node or final destination;
  - Data processing - Nodes can also aggregate or discard redundant data in the network before forward, avoiding unnecessary data redundancy at the destination;
  - Data prediction - By using statistics can also be predicted, avoiding unnecessary sensing.

- Data acquisition optimisation
  - Adaptive sensing - Different applications require different sensing frequency, but it does not need necessarily to be deterministic. The sensing frequency can be changed during execution based on how data is changing and application parameters [Silva et al., 2017];
  - Hierarchical sensing - Nodes that are cluster heads can take decision on the sensing rate of the other nodes and establish thresholds values to define when nodes will transmit new values after sensing [Manjeshwar and Agrawal, 2001];
  - Model driven sensing - Uses models based in probability to define how often the sensing will take place [DESHPANDE et al., 2004].

## 2.2 FLAT-BASED ROUTING

Multihop flat-based routing is a type of protocol in which every node or mote perform the same or similar role in the network. Considering that, and the large number of nodes in the network, all nodes cooperate with similar roles in order to achieve proper data distribution. Flat-based routing protocols can be proactive and constantly have the updated state of the network and in some cases routing tables, and in other cases they can be reactive, trying to discover the state and/or routes only when needed. The latter helps in saving energy but can increase the latency in the network, specially in large topologies due to the time spent calculating routes before the node can respond the request. Some of the flat-based routing protocols are now presented. A comprehensive list and the respective classification is presented in [Al-Karaki and Kamal, 2004].

### 2.2.1 *Gossip / Epidemic Routing*

Gossip or epidemic routing [Kuosmanen, 2000] uses a *one to many* forwarding behaviour where each node simply relays a received packet to its neighbours. This creates an exponential growth in the number of messages traversing network similar to the way a disease spreads, hence the name epidemic routing. On a naïve implementation of the gossip protocol, each node immediately forwards all messages received to all the neighbours in sight. Usually a gossip protocol randomly selected set of neighbours to whom packets will be relayed by using a *fan-out*[3]. As consequence, all nodes are expected to sooner or later receive all the packets being disseminated in the network [Carvalho et al., 2007]. However, they are also expected to receive several copies of the same packet, leading to *implosion*[4]. In order to

3 Fan-out is a technique of choosing only part of the neighbours to receive a packet.
4 A node receiving several copies of the same packet.

stop the message spreading, the protocol could either record the IDs of the latest packets shared and iterate through it for every received packet, or implement a Time-to-Live (TTL) for each packet. The former is more effective in reducing the entropy in the network, but requires more processing and the implementation of an additional buffer management algorithm [Leitao, 2007], while the latter requires a way of finding the optimal TTL to achieve maximum coverage with less forwarding rounds.

This type of algorithm is indicated to applications in which all nodes are required to receive all packets (full network coverage) and when high delivery efficiency is desirable. Both design goals require tuning on the TTL and *fan-out* in order to be achieved and these goals usually have a side effect of requiring more energy due to the high packet redundancy yielded [Al-Karaki and Kamal, 2004; Carvalho et al., 2007]. Due to the excessive number of packets exchanged in the network, it is expected that this protocol will result in rapid depletion of the node's energy store when applied to WSN.

### 2.2.2 *Sensor Protocols for Information via Negotiation*

The Sensor Protocols for Information via Negotiation (SPIN) protocol introduced in [Heinzelman et al., 1999] can be considered more as a set or a family of protocols than a protocol itself. In this family of reactive *negotiation-based* protocols, defined originally by Heinzelman [Al-Karaki and Kamal, 2004], the nodes disseminate data between their peers assuming originally that every node is a potential sink. As a consequence, the information is *virally* distributed to a large number of nodes, increasing the probability of a hit when a request for data is sent to any random node.

SPIN works as an epidemic protocol, but tries to address common issues by using negotiation and resource adaptation. Instead of disseminating all data, it spreads information with meta-data. Furthermore, another issue is sending data to areas that do not require it in the first place. In order to address these issues, SPIN introduced the concept of a three stage meta-data negotiation protocol with three types of messages:

- ADV - Used by a node to advertise new data;
- REQ - Used by the receiving node to request the advertised data;
- DATA - The actual data message.

When a node advertises its data with **ADV**, it sends meta-data information. So, considering the received meta-data, the neighbour can decide based in predefined parameters if it is interested. If so, it can request it with a **REQ** message, and receive it via **DATA** messages. The process is repeated in every node by advertising to its neighbours. Eventually, the whole area will be able to receive all the data they are interested in. While this process helps in reducing the data amount traversing the network when dealing with large payloads, it adds an unnecessary overhead when dealing with small payloads. The SPIN family consists of the following set of protocols as presented in [Al-Karaki and Kamal, 2004]:

- SPIN 1 - Works with the aforementioned 3 stage protocol;
- SPIN 2 - Introduces an energy-aware threshold. The node only participates in the three stages diffusion when it can finish the whole process without depleting stored energy below its predefined threshold. Defining a correct threshold value is there challenging, since reaching it in too many nodes will eventually block the network;
- SPIN-BC - Broadcast version;
- SPIN-PP - Point-to-point version;
- SPIN-EC - Similar to PP but with energy heuristic;
- SPIN-RL - Designed to solve issues with lossy communication channels.

Besides the overhead, another disadvantage of having a multi stage communication is that it increases the probability of failure in the since all three messages need to be delivered in order to exchange one data unit. **SPIN-RL** was created to solve this issue in networks where the probability of packet loss is higher.

### 2.2.3 *Directed Diffusion*

The event-driven directed diffusion algorithm [Intanagonwiwat et al., 2000] is characterised by being data-centric, in such way that data is aggregated and named according to attribute value pairs. The idea behind is to eliminate redundancy and save energy by aggregating data that have the same destination, and reinforcing lower cost paths. An example of a interest is given below, where the sink request sights of an animal in a specific area.

```
{
    instance: leopard
    rectangle: [0, 400, 0, 400]
    duration: 3600s
    interval: 5s
}
```

In this reactive protocol, the sink requests data based on its interest at the time, and by broadcasting this interest, the sink is dictating the network behaviour. When a sink needs specific information, it requests interest from a location by broadcasting a packet with the duration and the frequency that data should be sent back (initially set to a low value). Each node receiving the request, sets up a gradient containing the desired attribute and the direction back to the source. When a node that can answer the interest receives the packet, it sends data back to the nodes from which it received the request obeying the previously set gradient. The interest will follow different paths back to the sink as seen in Figure 2.5, and from all the possible paths the best (the one that arrived before at the sink) it reinforced. The reinforcement is made by reissuing the same interest with increased frequency only to the nodes in the best path.

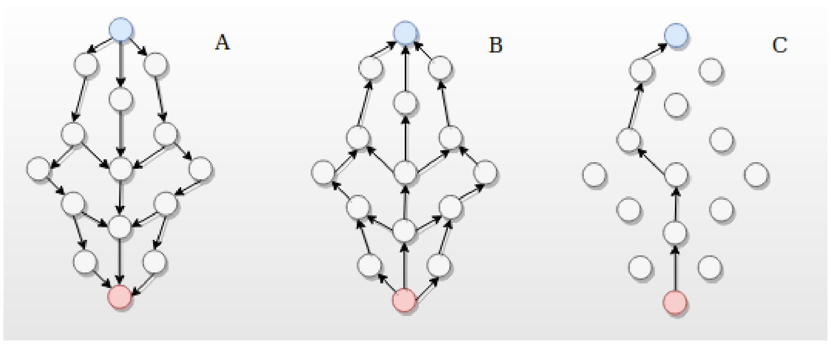


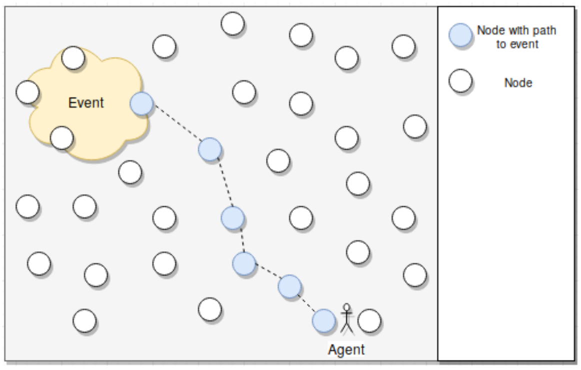
Figure 2.5.: Direct Diffusion [Al-Karaki and Kamal, 2004]

The steps of the direct diffusion protocol are listed below.

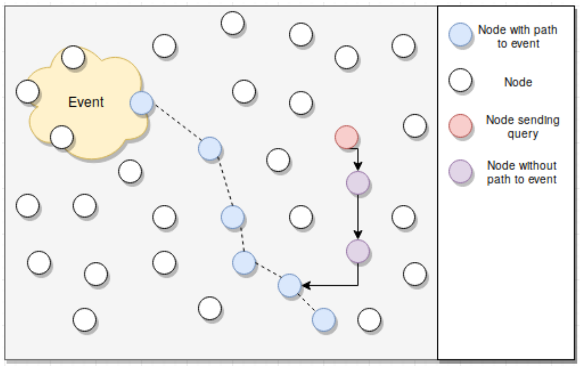
- A - Broadcasting or Interest Propagation: The sink broadcasts the interest from a specific location, flooding the network and establishing a state. Each node has a interest cache where it keeps interests and directions.
- B - Gradient Setting - Sets up the connection between the interests and the direction of the data towards the sink.
- C - Best Path - When the gradient has been set and all possible paths are known, the best path can finally be chosen between the requested location and the sink. The sink sends a new interest packet with a low interval setting to reinforce the path.

### 2.2.4 *Rumour Routing*

Rumour routing [Braginsky and Estrin, 2002] is an event-driven protocol designed for exchanging data in very large networks where the interest is usually not in a specific node, but on an area covered by several nodes. Those nodes observe an event that occurs in that specific area, and the sink node express interest in events. It works in a similar way to directed diffusion, but in directed diffusion the sink node express an interest in a specif geographical area, while in rumour routing the interest is expressed in terms of the event. In this particular situation the queries have to be injected in the entire network in a flooding manner. In addition, when the number of measuring events is small compared to the number of queries, the latter can be directed to particular nodes that have observed such events before, instead of flooding the whole network. To flood such events throughout the network the algorithm create agents, that are propagated in to other nodes and contain information about new local events to the far located nodes as seen in 2.6(a). Agents are special packets that have a high TTL when should cover the entire network. When a the agent reaches a node, this can update a table that relate events to routes towards them. In rumour routing the query packets follow a random walk through the network until it reaches a node that has that event in its event table as seen in Figure 2.6(b), it knows the exact route towards that event, reducing communication cost.



(a) Agent Advertises Event



(b) Node Request Event

Figure 2.6.: Rumour Routing

By reducing communication costs, this protocol can yield good energy saving results, but only when the number of events is small. When the number of events is large, the cost of maintaining large events and agents tables has a negative impact. This protocol require evenly distributed symmetrical topologies since its delivery efficiency relies on query packets finding nodes that have the been visited by the event's agent. If it is required high delivery, the events would need to issue many agents with high TTL to flood the network, which would increase the energy use.

### 2.2.5 *Minimum Cost Forwarding Algorithm*

The Minimum Cost Forwarding Algorithm (MCFA) is a protocol that creates a flat topology by calculating the minimum cost path from each node to the sink or base station [Ye et al., 2001]. The protocol has two distinctive stages, the topology discovery and the running. When sending a message to the next hop, the receiving node tests if the message is following the minimum cost route. If so, it forwards the message to its neighbours and the process is repeated until the message reaches the destination via the lowest cost path, since each node knows the shortest approximated path towards the sink.

During the topology discovery stage, in order to estimate the shortest path, the sink starts as an advertiser and sends a broadcast message to all nodes and set the cost to zero, which is incremented at every hop. The nodes, on the other hand, start as listeners and set their current cost towards the sink to infinity. So, when a node receives the broadcast messages from the sink, it compares with its current estimation. If the new cost is smaller, the current cost is updated. All the nodes that received the message from the sink repeat the same process by becoming advertisers and sending a similar message to their neighbouring nodes. It can happen that each node has neighbours with different costs, so in order to avoid calculating its cost by using not the shortest path to the sink, a back-off timer is implemented. Every time the node receives a packet during the topology discovery phase, it resets a timer and waits for a possible other packet with smaller cost. When the timer expires, meaning that all the neighbours have sent their setup packet, the node can finally pass to the running stage.

During the running stage, every new message is configured with the weigh[5] of the node which is creating the message, and a initial cost is set to zero. For every hop, the receiver adds the cost between the nodes to the initial cost and compares to its own weigh. If it is equal or smaller it means that the message is following the minimum cost path and it is forwarded, otherwise the message is dropped.

The MCFA is energy efficient since it does not require maintenance of routing tables and adds small overhead for each message. One disadvantage is the lack of adaptability in case of topology change in the network due to mobile nodes or if the node assigned as sink changes, and the impossibility to self healing in case a node along the minimum cost path dies before the others. One possible solution for that would be to force the network to return to the topology discovery stage periodically, which would increase the energy consumption.

### 2.2.6 *Push-lazy-push Multicast Tree*

The Push-lazy-push Multicast Tree (PLUMTREE) Leitao [2007] protocol is a bi-modal protocol that makes use of the simplicity of a tree-based protocol but introduces an epidemic overlay to increase the resilience of the network. In this protocol the epidemic broadcast is performed in two different modes, eager push or lazy push. The tree-based dissemination is performed by the eager nodes, while the remaining nodes perform a lazy push broadcast, that serves as a backup purely gossip dissemination, useful in face of failures in

5 The number of hops multiplied by the communication cost between the nodes.

the broadcast tree and that guarantees high delivery rate and tree healing in case of node failure. In this protocol is assumed that nodes can keep a log of distributed packets, so that in case of repetition, those packets are dropped. To build the broadcast tree, the protocol needs a partial view of the neighbouring network, and in order to keep a partial view of the network, this protocol uses a membership service based in *Peer Sampling Service*. The main functions of the PLUMTREE are:

- Tree Construction - module responsible to select the neighbouring nodes that will construct a link which will perform the eager pushes. The subset of nodes chosen for the eager pushes function in a similar way as pure gossip with fan-out. The remaining nodes will than operate using lazy pushes.
- Tree Repair - module responsible to ensure that even on the occurrence of failure in some nodes, all nodes remain reachable by the spanning tree.

In order for the protocol to ensure low latency in the network, it selects as eager nodes the peers from whom they receive the first messages. On a first moment the lazy nodes list is empty, and as the receiving node starts to received repeated messages, those nodes are moved to the lazy list. As soon as the network has lazy nodes, they start broadcasting digest messages with the list of available message IDs they have. If the eager broadcast tree fails, and a node has interest in a message received via the lazy push digest, it sends a crafted message requesting the desired message via the eager tree. This special message both broadcasts the interest in the message, but also heals the tree. The main goals of this protocol are therefore to ensure high delivery rate, network coverage and low latency.

## 2.3 HIERARCHICAL AND LOCATION BASED ROUTING

Hierarchical or cluster based routing stands for a set of techniques where different nodes perform different tasks based in predefined characteristics. In hierarchical routing, a two layer scheme is introduced, in which one layer takes care of controlling how each node will behave and the other to actually perform data routing. The idea behind these protocols is to divide the network in clusters and define cluster heads that are responsible for the routing whilst the other nodes are responsible for sensing events.

By performing data fusion and aggregation, hierarchical routing can reduce the energy burden within the clusters by reducing the overall number of messages to the sink. Most of the techniques in this category are more about how to choose the roles of each node and controlling adaptive features instead of routing itself [Al-Karaki and Kamal, 2004]. Some hierarchical protocols use location to build the hierarchical topology, hence, are also classified as location-based protocols.

### 2.3.1 *Low Energy Adaptive Clustering Hierarchy*

Low Energy Adaptive Clustering Hierarchy (LEACH) is a hierarchical-based protocol, in which a cluster head is randomly selected between the nodes. The protocol has two temporal stages: the first one deals with the topology discovery and control, and the second is the stead state. On the topology control stage, the cluster head is selected randomly based on a equation described in [Heinzelman et al., 2000]. During the stead state stage, nodes actually exchange data. Since it is a cluster based protocol, data can be aggregated, processed and compressed in the cluster before being forwarded towards the sink. Such processing is made by the current cluster head, that during this time will spend more energy than the other nodes. This action leads to energy unbalances, but since the cluster heads are chosen stochastically, given enough time it is expected that the unbalance disappears.

### 2.3.2 *Power-Efficient Gathering in Sensor Information Systems*

Power-Efficient Gathering in Sensor Information Systems (PEGASIS) is a chain-based protocol [Lindsey and Raghavendra, 2002] developed as an enhancement of the LEACH protocol. The protocol works in a way that each node only needs to talk to its closest neighbours and in order to equalise power drain, in each round one node takes the role of talking to the BS, restarting when all nodes have taken this role.

PEGASIS uses signal strength to find closest neighbours and also adjust it so that each neighbour can talk to only one node, thus avoiding the creation of clusters like in LEACH. Since this protocol uses the location in order to build the topology, it is also classified as a location-based protocol. It has the requirement that all nodes need to have enough radio range to reach the BS and that all nodes are capable of adjusting the radio power during run-time.

### 2.3.3 *Threshold-Sensitive Energy Efficient Sensor Network Protocol*

Threshold-Sensitive Energy Efficient Sensor Network Protocol (TEEN) is a hierarchical protocol introduced in [Manjeshwar and Agrawal, 2001], where the node that is selected as cluster head with the same random criteria as LEACH, send two threshold values to the other nodes:

- Hard threshold - Which is a range of values that govern when the sensor is allowed to transmit. Only when the sensed value is within the defined range, the node will transmit;

- Soft threshold - Which is a percentage value, stating how much the new value should change compared to the previously measured value to justify a new transmission.

Those threshold values are calculated based in a equation described in [Manjeshwar and Agrawal, 2001]. A node configured with the TEEN protocol has the transmission radio always off, but is always listening to the medium, in order to receive new threshold values transmitted by the cluster head or by a newly attributed cluster head. Since the nodes never transmit anything unless the two threshold values are reached, the network saves energy. A possible negative effect is the possibility of never listening to the two threshold values, which would cause the nodes to never transmit, or to be too conservative in those values, which could harm the measurements accuracy. To find optimal values for the thresholds is therefore the same as finding the balance between accuracy and energy efficiency. Another negative side of TEEN is having the receiving radio always on, and consequently the microcontroller never in *deep sleep*. Other protocols have been created based on TEEN aiming for different enhancements depending on the application:

- MODTEEN - Similar to TEEN but with a modified equation for calculating the threshold values [Pundir et al., 2018];
- APTEEN - Similar to TEEN but can issue warnings for critical value changes which can bypass the thresholds, enabling the protocol to be used in time critical applications [Manjeshwar and Agrawal, 2002];
- DAPTEEN - Makes use of geographical parameters to try to reduce data redundancy [Anjali et al., 2016], so the protocol can also be classified as location-based.

### 2.3.4 *Minimum Energy Communication Network*

Minimum Energy Communication Network (MECN) is a routing protocol introduced in [Rodoplu and Meng, 1999] that uses geographical data to build a hierarchical topology, and so can also be classified as location-based. The rational behind this protocol is to create local cluster or relay groups where nodes exchange data. A node, by knowing its surroundings, tune the transmission power to talk only to nodes closer to it by using low power GPS data. Which requires a hardware capable of doing so during run-time.

The protocol also has the capability of self-configuring, so it can heal the topology if the network changes. A negative side of this protocol is that it always assume that all nodes can talk to each other, which is not always true since physical barriers can be between nodes even though they are geographically close. To solve this issue, an extension was added to the protocol called SMECN, where the letter *S* stands for small.

## 2.4 OVERVIEW

Table 2.1 presents an overview of the protocols introduced in this chapter. Two continuous delivery flat protocols were introduced and will later be used for comparison with the proposed protocol during the simulations. The naïve gossip with and without fan-out have as main characteristic the delivery efficiency and network coverage, with the disadvantage of wasting too much energy with packet duplication. The proposed protocol aims at keeping the same delivery efficiency whilst reducing the duplication, thus increasing the energy efficiency. The MCFA on the other hand, is very efficient when delivery messages to a sink, guarantying high delivery efficiency but with low network coverage. Also, the proposed protocol can handle mobile nodes by definition, mobile sink nodes and multiple sink nodes. While the PLUMTREE protocol has similar goals as the proposed protocol, meaning high delivery rate and good network coverage, it focus on low latency. Due to added complexity in the PLUMTREE algorithm, and the additional requirement of relying on other algorithms, it is expected that this adds an computational and energy footprint. It also has capacity of performing a slow tree healing, which enables limited mobility to the protocol.

The query-based flat protocol SPIN has the advantage of saving energy with large payloads, since initially the nodes only advertise new metadata, and sends data only upon request. But, when the payload is small, the overhead of needing three packets for each data unit instead of one makes this protocol inefficient. Directed diffusion and rumour routing have a negative impact in energy efficiency during the initial moments of the network, when no gradient has been created in the directed diffusion and no agent has been issued in the rumour routing. After some time, all nodes will eventually have tables with gradients and routes to events, but keeping those add a negative footprint in computational cost for the protocols, cost that is limited in the proposed protocol.

Table 2.1.: Protocols Summary

| Protocol | Classification | Data Model | Routing | Mobility |
|---|---|---|---|---|
| Gossip | Flat | Continuous Delivery | Reactive | Yes |
| PLUMTREE | Flat | Continuous Delivery | Proactive | Limited |
| SPIN | Flat | Query Based | Reactive | Yes |
| Directed Diffusion | Flat | Query Based | Proactive | Limited |
| Rumour | Flat | Event Based | Proactive | Limited |
| MCFA | Flat | Continuous Delivery | Proactive | No |
| EAGP[6] | Flat | Continuous Delivery | Reactive | Yes |
| LEACH | Hierarchical | Continuous Delivery | Proactive | Fixed sink |
| PEGASIS | Hierarchical / Location | Continuous Delivery | Proactive | Fixed sink |
| TEEN | Hierarchical / Location[7] | Event Based | Proactive | Fixed sink |
| MECN | Hierarchical / Location | Continuous Delivery | Proactive | No |

The main problem with the hierarchical protocol LEACH, the energy unbalance, is solved by the PEGASIS protocol with the cluster head rotation. PEGASIS however, has a requirement that nodes need to be able to change radio power properties during run-time and need to have a fixed sink. The TEEN family of protocols are energy efficient in the way that nodes create less packets, since they only transmit new data when the threshold values are satisfied. That however, reduces the flexibility of the applications since only values within a range are reported. The protocol proposed in this work, shall render flexibility for the application and the problem with excess sensing could be solved by an adaptive sensing protocol. Finally, the MECN has high energy efficiency since it uses less power to transmit data, but requires the nodes to be equipped with GPS radios and to be able to change radio power during run-time. Also, it has a disadvantage of not compensating for the nodes' surroundings, relying solely on distances to set radio range.

## 2.5 SUMMARY

This chapter introduced different protocols covering the main elements of the WSN routing protocols taxonomy, which are the flat-based, the hierarchical-based and location-based protocols. Since each has positive and negative aspects depending on the situation, choosing one depends much of the application where it will be used. As this work will later compare the proposed protocol to others by means of experimentation, some protocols were chosen to be simulated. It is important that the simulations' results are comparable between protocols, so it was decided that all need be implementable following the same traffic model. Hence, it has been considered that they must follow the continuous delivery model, where the sensor data is updated constantly and propagated towards the sink immediately. Therefore, the gossip protocol will be implemented as a naïve baseline with both a simple version and a version with fan-out. Another protocol that can be implemented using the continuous delivery model is the MCFA, and since is not an epidemic protocol, will be a good comparison in terms energy efficiency and adaptability. None of the selected protocols is reactive, so they will also be comparable regarding the latency.

6 Protocol introduced by this work
7 DAPTEEN

# 3

# ENERGY-AWARE GOSSIP PROTOCOL

This chapter will introduce the Energy Aware Gossip Protocol (EAGP), which is the novel routing protocol proposed by this work, and its main goals, detailing the requirements and decisions considered for its design and implementation. The main goal of this protocol is to enable the extension of the lifetime of a WSN, ensure highly efficiency in data distribution whilst reducing data duplication.

## 3.1 DESIGN GOALS

As discussed in Chapter 2, different communication protocols forward data by emitting a broadcast packet to all its neighbours or choosing a fan-out to reduce traffic and the probability of data duplication. In this way, the information is disseminated on an exponential rate, enabling quick and effective propagation of the information to all nodes. The side effect is the amount of duplicated packets which can lead to fast energy depletion due to excessive use of the radio as exemplified in Figure 3.1.

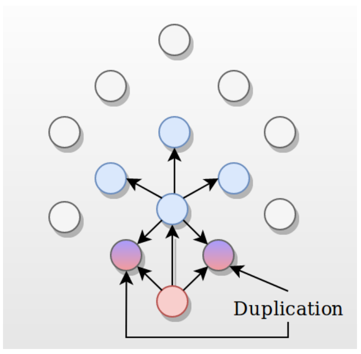


Figure 3.1.: Epidemic Packet Diffusion

Some protocols use an epidemic multicast or broadcast as a single stage on a multi-stage protocol. The rumour routing for instance, in order to reduce the amount of data during storms of epidemic dissemination, sends only meta-data about the events observed by the node. If the sink or another node is interested in such information, the node will receive a request and will answer using a different routing mechanism, using a directional route for example. That makes sense on a network where each node has a big amount of data to be transmitted, but usually in a WSN, where the data collected by each sensing event might be just a few bytes, that is less efficient. Hence, depending on the amount of data to be disseminated, the cost of sending meta-data and later the actual data might add an overhead and affect the QoS, that cannot be afforded by a resource constrained network.

The rational behind the proposed algorithm is to remain as an epidemic protocol, but also slow down the forwarding of the packets in some nodes proportionally to their current energy level. So, the idea is that nodes with high energy level will relay packets without delay, while the nodes with low energy level will hold the information longer, and relay data as a back-up path to ensure delivery efficiency. Details about the calculation of the timers will be presented in Section 3.2.3. Table 3.1 illustrates the new terminology introduced by the protocol and some other terminology that will be used in this document. More details will be further given.

Table 3.1.: EAGP Terminology

| | |
|---|---|
| Lazy | A node behaviour in which it schedules the relay of packets with $T_{MAX}$ |
| Eager | A node behaviour in which it schedules the relay of packets with $T_{NEXT}$ |
| $T_{MAX}$ | The maximum time a node can wait to forward each packet in its buffer |
| $T_{NEXT}$ | The time an eager node waits to forward each packet in its buffer |
| Node | Each sensor in the network made of a radio, sensor, computation and battery modules |
| Sink | The node that can forward the disseminated information to another network, usually a border router |
| Neighbour | Node that is within radio reach |
| Latency | Time difference between packet creation and delivery at the sink node |
| Storm | High number of packet dissemination from different nodes simultaneously |
| Implosion | Delivery of several copies of the same packet simultaneously or not from different nodes |

## 3.2 PROTOCOL IMPLEMENTATION DIRECTIVES

In order to define the protocol implementation, the Protocol Data Unit (PDU) and a pseudo algorithm are presented in this section. The algorithm was divided in two rounds: data creation and data propagation.

### 3.2.1 *Algorithm*

#### *Data creation round*

Every time a node wakes up from sleep mode, a callback function is called that changes the state of the node and perform a few steps. To reduce excessive computations, the first two steps are performed only if there is a variation in the node's own battery level.

1. Maintain the list of neighbours, removing nodes that are no longer visible;
2. After maintaining the neighbours list, each node can decide which mode it will assume thereafter, eager or lazy;
3. The node can now read a sensor for updated values and forward the information via multicast to all the neighbouring nodes, sending its own energy level, so that the others can refresh their list of neighbours.

The data creation round is executed less frequently than the data propagation round. So, the update in node behaviour and the maintenance of the list of neighbours is performed in this step to reduce the protocol's computational footprint. Figure 3.2 illustrates each step of this round.

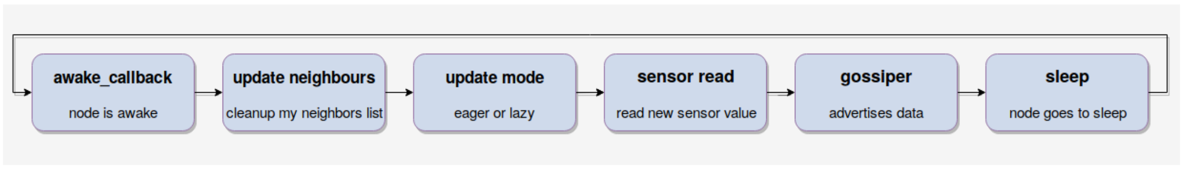


Figure 3.2.: Data Creation Round

#### *Data propagation round*

This round is composed of the following stages:

1. The node receives a packet and verifies if the packet is an echo of its own sent packet. If so discards it;
2. Check if the sender is already in the neighbours list. If so, refresh battery level, if not add;
3. Verify the TTL value. If expired, discards the packet;
4. Check if the *node id* of the previous hop was the current receiving node. If so, the packet is discarded. This avoids local looping between two nodes;

5. If the node is in eager mode, it adds the packet to the sender scheduler with the timer set to the latest $T_{NEXT}$ value according to Equation 2, which is calculated every time there is a change in the neighbours list. If the packet with same ID was already scheduled to be forwarded, the new packet is discarded and the current scheduled packet removed;

6. If the node is in lazy mode, it adds the packet to the sender scheduler with the timer set to the $T_{MAX}$ value. If the packet with same ID was already scheduled to be forwarded, the packet is discarded;

7. When each timer expires, the packet is broadcast with current energy level of the node.

The reason to discard the packets in *stages 5* and *6* is to reduce the number of duplicated packets in the network. When a packet is already scheduled to be forwarded, it is already flowing thought a faster path. This stage can be represented as an algorithm.

**Algorithm 1:** Data propagation

```
Result: Packet is forwarded
message_received();
if $(sender\_id$ and $prev\_id) \neq my\_id$ then
    if $sender\_id \notin visible\_list$ then
        $visible\_list \leftarrow sender$;
    end
    if $pkt\_ttl > 0$ then
        if $mode = eager$ then
            if $pkt\_id \notin sched$ then
                $sched \leftarrow pkt$
            else
                $sched \rightarrow pkt$
            end
        end
        else if $mode = lazy$ then
            if $pkt\_id \notin sched$ then
                $sched \leftarrow pkt$
            end
        end
    end
end
```

### 3.2.2 *Protocol Data Unit*

For the protocol it would be sufficient only one PDU, as presented in Figure 3.3.

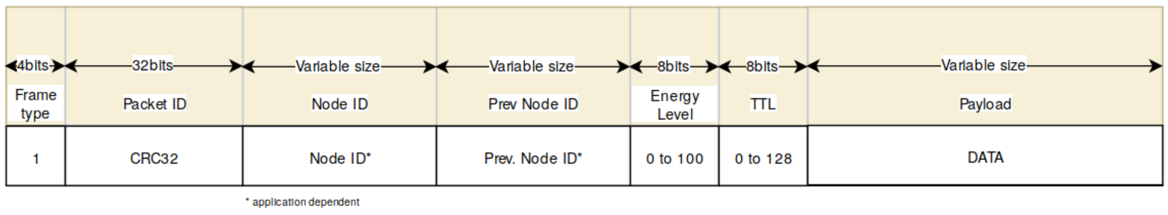


Figure 3.3.: Protocol Data Unit

The enumerated fields are the minimum required for an implementation of the protocol.

1. Frame type - For EAGP only one type of frame is currently required, but for the implementation of the a version with digest that will be discussed in 3.2.5, 4 bits are required.

2. Packet ID - Unique packet ID used for scheduling packets. It is enough to use CRC with 32 bits to avoid ID collision, but this could be an application design choice.

3. Node ID - ID of the node originating the packet transmission. Depending of the network stack used in the implementation, for the protocol it should be sufficient to use another type of identification for each node, such as MAC address or IP address. The need of a unique node ID might depend on the application.

4. Previous Node ID - ID of the node sending the packet in previous hop. Used to avoid local looping between two nodes.

5. Energy Level - The current battery level from 0 to 100 percent of the current node sending the packet. For a representation of 0 to 100, 8 bits are enough.

6. TTL - TTL set during packet creation. This TTL is used by the protocol and is independent of other TTL values used by the network stack. The choice of TTL is an application design choice. Setting to a value equal to the *sink radius*[1] ensures high delivery ratio towards the sink, but depending on the topology could lead to low network coverage when the goal is to deliver packets to all nodes. Reserving 8 bits for the TTL allows to a maximum of 128 hops. Since this is application dependent, the reserved size dependes ultimatelly on the scale of the network.

7. Payload - The data to be sent. The size of the payload is independent of the MTU used by the network stack.

1 Hop count from the sink to the farthest node.

Packet segmentation and reordering were not considered in this work, since it depends much on the implementation constrains and the other protocols in the network stack.

### 3.2.3 *Timers Calculation*

Every node in the network shall have a neighbours list, and with it, each node can decide which behaviour it will assume thereafter, eager or lazy. These behaviours are based on their current battery level comparing to the average level of the local neighbourhood. If each node's battery level is equal or greater than the average, it assumes an eager behaviour, otherwise it assumes a becomes lazy behaviour. Eager nodes will schedule packets to forward using $T_{NEXT}$, while lazy nodes will schedule packets with $T_{MAX}$. Making the $T_{NEXT}$ proportional to the battery level, as shown in Equation 1, has however a problem illustrated by Figure 3.4. The lower the battery level, $T_{NEXT}$ grows exponentially and no node would assume an eager behaviour.

$$T_{next} = \frac{T_{max}}{Bt_{node}} \tag{1}$$

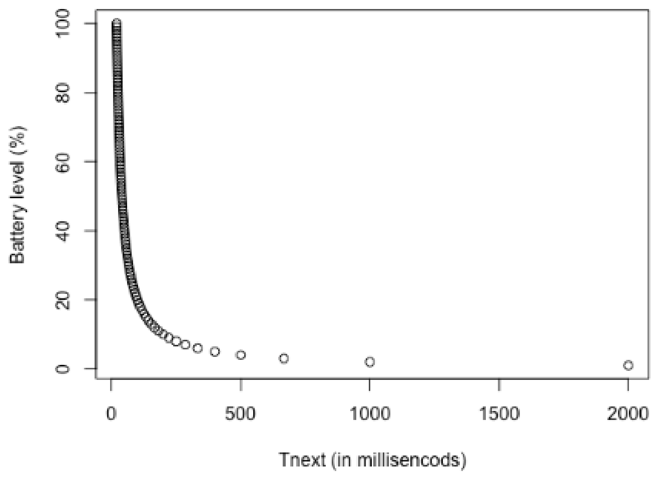


Figure 3.4.: $T_{NEXT}$ for $T_{MAX}$ = 2000ms

To solve this problem, the energy level of not only the node calculating $\mathbf{T_{NEXT}}$, but also the energy levels from its neighbours should be considered. A node that has higher level than its neighbours assumes an eager behaviour to forward the packets faster following the expiration of a $\mathbf{T_{NEXT}}$ timer, whilst its lower energy peers assume a lazy behaviour and hold the information until a $\mathbf{T_{MAX}}$ timer expires. If all nodes remain in lazy mode, it is expected that the latency grows proportionally to the number of hops between the sender and the sink by a factor of approximately $T_{MAX}$ times the number of hops. Hence, in order to achieve a lower latency without unbalancing the network, some of the nodes — the ones with more energy — should assume a eager behaviour. As a consequence, eager nodes will spend more energy and might eventually equalise to its neighbours, creating a tendency in the whole network to always equalise. An expected side effect is high jitter, which should be taken in account during the application design. Equation 2 governs the calculation of the $T_{\mathrm{NEXT}}$ timer. Considering the number of neighbouring nodes greater than zero,

$$T_{NEXT} = T_{MAX} - (T_{MAX} * \frac{Bt_{node} - min(Bt_{neighbours})}{max(Bt_{neighbours}) - min(Bt_{neighbours})}) \tag{2}$$

Where,
$Bt_{node}$= Energy level of the node in %
$Bt_{neighbours}$ = List with the energy levels of a node's neighbours
*min* = Function that returns the minimum battery level of the neighbourhood
*max* = Function that returns the maximum battery level of the neighbourhood

The rational behind the formula is to scale the visible nodes in a value from 0 to 1 using the feature scaling min/max equalisation technique.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{3}$$

Since the x' value will be always between 0 and 1, the nodes that have lower energy in the local neighbourhood will score closer to 0, and the ones with higher energy will score closer to 1. This value is than multiplied by $T_{\mathrm{MAX}}$, and $T_{\mathrm{NEXT}}$ will be $T_{\mathrm{MAX}}$ - $T_{\mathrm{MAX}}$*x'. So high energy nodes will get lower $T_{\mathrm{NEXT}}$ values and low energy nodes will get $T_{\mathrm{NEXT}}$ values closer to $T_{\mathrm{MAX}}$, solving the issue illustrated in Figure 3.4. It is possible to observe in this equation that when maximum and minimum values of the energy levels of the neighbours are the same, the equation yields a division by zero, which is not acceptable. So a verification must be made in the code before the calculation. Some constrains were defined to the protocol in order to ensure the main goals are achieved.

### 3.2.4 *Design Constrains*

With the main directives of the protocol defined, the design constrains were defined. The constrains listed here shall be considered during the implementation in order for the protocol to achieve the main goals previously defined.

- The protocol should behave in an epidemic manner;
- Each node in the network must know the energy state of all its neighbours and this information should be updated every round;

- Only the energy state of nodes currently in neighbouring reach should be considered;
- Packets are created with a TTL that is decremented for each hop. When TTL is zero, the packet is discarded;
- Each exchanged packet must have a unique identifier;
- The protocol shall not require persistent storage for historic data (list of routes, list of past packets and etc.), if a node reboots or a new node is added, the behaviour of the protocol is not affected.

### 3.2.5 *Digest Variation*

For testing purposes, a variation of the EAGP protocol is defined. In this variation, when a lazy node receives a packet that it has scheduled to send, it drops it and removes from the scheduler. All dropped messages are added in a circular buffer and a periodic digest of this buffer is sent to all nodes. The digest has its own independent scheduler, with periodicity set to ten times the $T_{MAX}$ to avoid saturation of the network.

The digest PDU is similar to a regular EAGP PDU, but it sends an *ADV* packet containing an array with the packet IDs it has in buffer. A sink when receives this *ADV* packet, checks in its delivery log if any packet is new. If so, it creates a new type of packet using a *REQ* PDU. This request PDU is also very similar to a regular EAGP PDU with a different identifier, but instead of sending a regular message it sends an array with the desired message IDs. This special PDU is disseminated among all nodes and whoever has one or more of the packets buffered send it as a regular packet with TTL reset again to the maximum usual value and the message is than removed from the buffer. Therefore, the protocol behaves for the lazy nodes, similar to the SPIN protocol mentioned in Section 2.2.2.

Since the TTL is reset to increase delivery rate, the hop count is expected to increase above the usual TTL, and also the entropy in the network is expected to increase due to the introduction of two new types of PDUs and as consequence more packets being disseminated. The complexity of the algorithm and the use of computation power to process more lists also increase. Therefore, this idea was not taken forward for the protocol definition but was kept for testing and benchmarking purpose.

## 3.3 SUMMARY

The EAGP introduces a deterministic way of deciding which nodes will relay the packets faster based in remaining energy level, and tries to solve the problem of lower efficiency in packet delivery by using the lazy nodes as a backup path to deliver packets to remaining nodes. It is also expected that some nodes will receive the messages almost immediately and others delayed by $T_{MAX}$. Hence, this protocol is applicable when epidemic coverage is needed, but higher average latency is tolerable. Since the information about the nodes energy level is transmitted together with every packet, the nodes can recreate the topology every round of data propagation, enabling a very effective protocol in self-healing for mobile nodes.

# 4

# SIMULATORS AND TESTING ENVIRONMENTS

This chapter covers some of the available simulators, emulators, hardware, operating systems and test beds that can be used to research and prototype with WSN. The proposed protocol is further implemented based on one of the simulators presented here.

## 4.1 WIRELESS NETWORKS SIMULATION AND EMULATION

Implementing a routing protocol using a simulator enables the possibility of analysing behaviour and collecting metrics before a possible implementation in hardware. A few free network simulators are available [Helkey et al., 2016; Bhattacharya, 2011] and some of them are introduced in this section.

### 4.1.1 *Mininet-Wifi*

*Mininet-Wifi* is a fork of a well know network simulator called *Mininet*. The *Mininet* simulator was created with the goal of studying Software Defined Networks (SDN), so its original version focuses only in wired networks. *Mininet-Wifi* was than forked from *Mininet* to add support to wireless networks. The program itself is a bundle of scripts programmed in Python allowing the creation of network topologies where each device can be accessed via its own virtual terminal [Fontes et al., 2015]. These topologies can be created manually via command line or can also be created via Python scripts to run different scenarios. The fact that the topology can be created and manipulated via Python scripts opens a large number of possibilities for simulations, since the script can be created with logics and automations to change the topology during execution. *Mininet-Wifi* also includes the possibility of changing the nodes position in three dimensions and the behaviour of the medium with different ways to simulating the wireless signal attenuation.

Since users have access to a virtual Linux terminal running its own network devices, they can create applications in any programming language that is able to run in the chosen host operating system. Also, any application compatible with the host Operating System (OS) can run seamlessly, being possible to run tools useful for this research such as Wireshark. The network interfaces created in each virtual node use the Linux kernel driver *mac80211_hwsim* to create a virtual wireless network interface [Rethfeldt et al., 2016]. This network interface emulate the behaviour of an interface with perfect radio, therefore all packets created in one node will arrive in the destination node as long as it is in range. This behaviour is usually not desired for most of the simulations, so another software simulates the wireless medium between the virtual nodes. The *wmediumd* project was originally created by a company called CozyBit[1] to work on top of *mac80211_hwsim* implementing a probabilistic model that was originally statically configured during run-time via a configuration file [Silvano et al., 2017]. The final architecture when using *wmediumd* is presented in Figure 4.1.

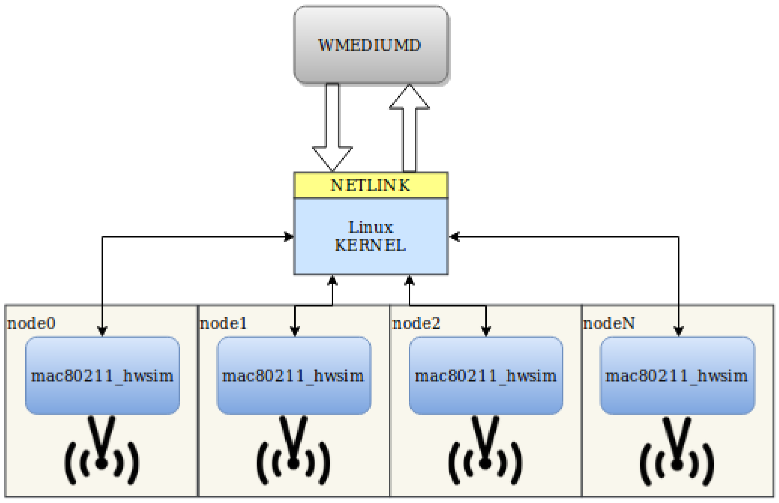


Figure 4.1.: *Wmediumd* Architecture

The user would need to create beforehand a matrix with all the nodes and the probability that packets in each channel and each bandwidth would fail to be delivered. Therefore, mobility, obstacles, shadowing and other aspects of wireless communications were not considered in the original project. Since the project is open source, several additions were made by subsequent forks of the project, and the following propagation and mobility models [2] were added:

- Propagation Models
  - Free Space
  - Log-Distance

1 The company is currently out of business but the project is open source.
2 https://github.com/ramonfontes/wmediumd/blob/mininet-wifi/wmediumd/config.c

  - Log-Normal Shadowing
  - International Telecommunication Union (ITU)
  - Tow-Ray Ground
- Mobility
  - Random Walk
  - Truncated-Levy Walk
  - Random Direction
  - Random Waypoint
  - Gauss-Markov
  - Reference Point
  - GPS Route

Even though *Mininet-Wifi* has the possibility of simulating software defined network components [Fontes et al., 2017], this feature will not be considered for this project since there will not be a need of switches in the topologies created for testing the protocol. It should be also taken into consideration that since it uses Linux drivers to emulate real network cards, and *wmediumd* also runs most of its software in kernel space, *Mininet-Wifi* is utterly dependent of the Linux scheduler and simulated applications that run in user space might have the performance affected by that.

### 4.1.2 *CORE*

CORE[3] stands for *Common Open Research Emulator* and is, in fact, an emulator, not a simulator. CORE works in a very similar manner as *Mininet-Wifi*, being a set of tools enabling the user to setup an emulated environment. When running a simulation on CORE, applications running on each node act as real network application. It is open source and was originally created by Boeing and released with BSD license.

CORE has two main components, *core-daemon* and *core-gui*. The *daemon* is responsible for creating the virtual environments for each node by using Linux network namespaces (same as used by *Mininet-Wifi*). When spawning a node using namespaces, each node has its own network stack and acts as a lightweight virtual machine where all nodes share the same resources and the same kernel. Each node can therefore create its own network interface that communicates to the host via network bridges. Wireless networks are emulated with *ebtables* rules and the medium does not take in account simulated losses caused by signal attenuation or obstacles. Likewise Mininet, *core-daemon* is written mainly in Python, making

3 Available at: https://github.com/coreemu/core

it easy to create scenarios with Python scripts as shown in Annex C.1. As an alternative to create scenarios, *core-gui* can be used. The overall architecture of CORE is presented in Figure 4.2 as seen on CORE's documentation[4].

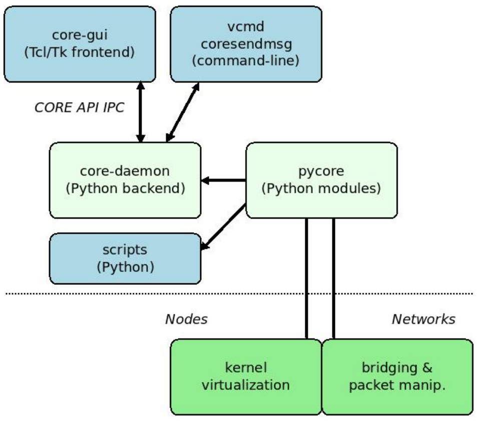


Figure 4.2.: CORE Architecture

*Core-gui* is made in TCL/TK and has an intuitive interface, making it easy to configure scenarios. It is however not very practical for simulations that require open terminals in each node. CORE's Python API enables the user to export a running scenario created in Python as XML file that can be open in *core-gui*, but not all configurations are exported. Figure 4.3 exemplifies the interface with one simulation imported from a XML file which was exported from a previous simulation created with Python scripts.

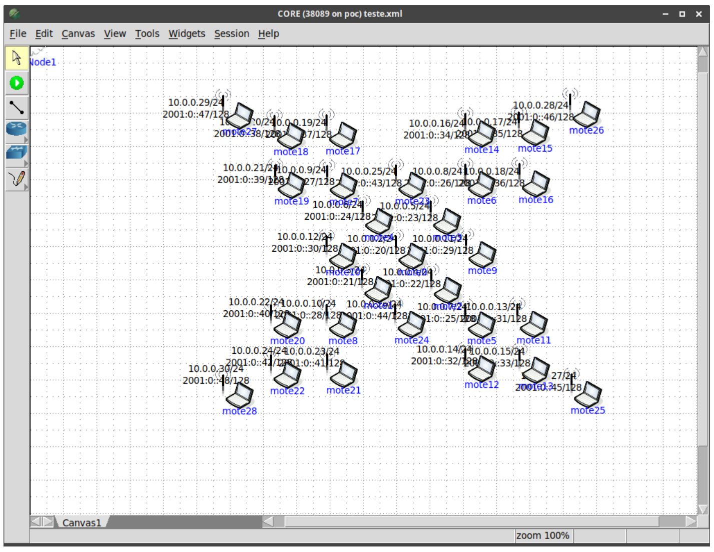

CORE (38089 on poc) teste.xml
File Edit Canvas View Tools Widgets Session Help
Canvas1
zoom 100%

Figure 4.3.: CORE Gui

4 http://coreemu.github.io/core/architecture.html

When running different radio or propagation models is required, EMANE can be used with CORE. EMANE (Extendable Mobile Ad-hoc Network Emulator)[5] is a MANET(Mobile Ad-Hoc Network) emulator that can be used together with CORE to emulate the MAC/-PHY layers [Ahrenholz et al., 2011]. It enables higher fidelity for layers 1 and 2 when required. It relates to CORE the same way *mac80211_hwsim* and *wmedium* relate to *Mininet-Wifi*. EMANE offers different radio models to be used:

- RFPipe
- IEEE 802.11 a/b/g
- TDMA
- LTE
- Bypass

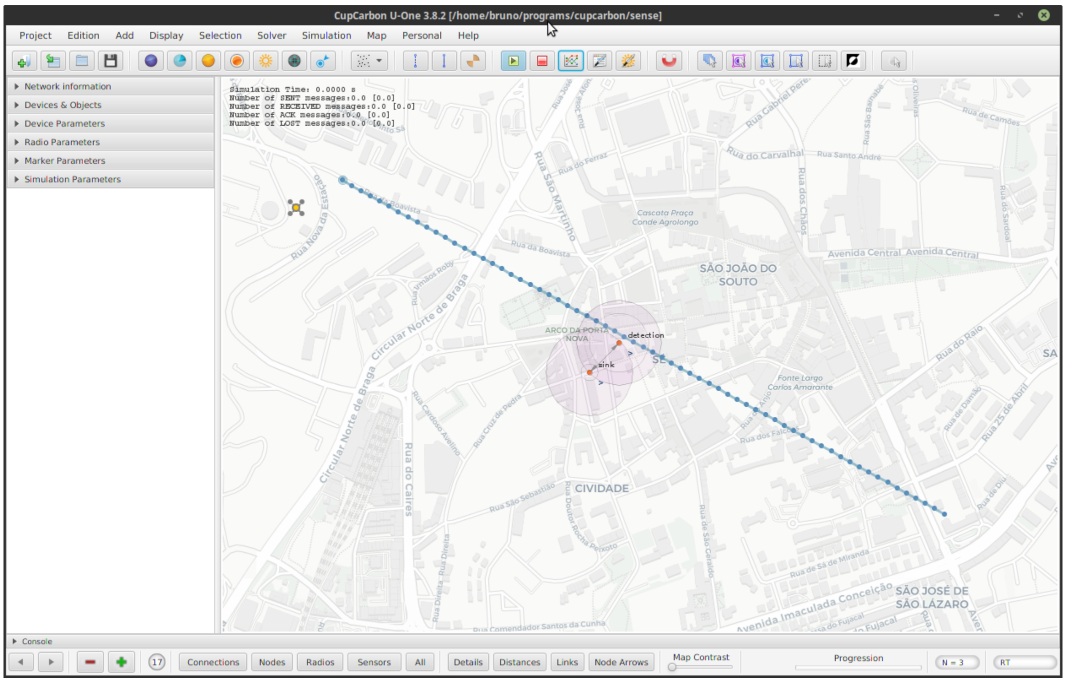
EMANE is open source and originally created by the US Navy. Both CORE and EMANE can be deployed in a distributed network to enhance the computing power for the scenario, thus enabling extensive scalability that is limited only by the available number of servers.

### 4.1.3 *CupCarbon*

CupCarbon [Bounceur et al., 2018; Mehdi et al., 2014; Lounis et al., 2017] is a smart city and IoT WSN simulator, open source and written in Java that has a user friendly interface. It allows the simulation of different types of nodes and protocols [Mehdi et al., 2014]. For the radio, the user can choose between 801.15.4 (ZigBee or 6LoWPan), WiFi and LoRa radio protocols.

In CupCarbon, the user has the option to create a route for the sensor node and save it as file with GPS coordinates. This file can be associated with one or more nodes and also be used in different simulations. The user has the option to use Open Street Maps (OSM) or *Google Maps* as background as shown in the Figure 4.4, so the GPS file with the route can be seen as an overlay layer on a real map representation.

5 Available at: https://www.nrl.navy.mil/itd/ncs/products/emane



```
Simulation Time: 0.0000 s
Number of SENT messages:0.0 [0.0]
Number of RECEIVED messages:0.0 [0.0]
Number of ACK messages:0.0 [0.0]
Number of LOST messages:0.0 [0.0]
```

Figure 4.4.: CupCarbon Main Window with OSM as background

It is also possible to use energy models of the sensors in order to follow how the stored energy decay along simulations, providing graphics and reports. However it is not possible to emulate and control the different energy modes that each specific hardware can achieve, such as radio sleep or deep sleep. CupCarbon is a simulator that aims at simulating the behaviour of mobile nodes in a wireless environment but it does not give access for the user to different network layers, hampering the desig and test of routing protocols. In order to allow the user to simulate nodes different actions, CupCarbon has its own scripting language called SenScript. A script can be created to control how the nodes interact with each other by using a simple set of commands. Finally, one extra feature of CupCarbon that can be useful for prototyping is the possibility of exporting the script associated with the nodes as Arduino[6] compatible C++ code, allowing uploads to real WSN nodes and testing of algorithms. This scripting language is adequate to simulate simple node's behaviour and access some variables, but it might however come short for prototyping different routing algorithms running at application level.

## 4.1.4 *Cooja*

Cooja is an open-source wireless network simulator usually bundled together with ContikiOS [Roussel et al., 2016]. It has a graphical interface somewhat similar but more simple

6 https://www.arduino.cc/

than CupCarbon, as seen in Figure 4.5. It also provides the possibility of changing the simulation speed, making it easier to simulate long lasting scenarios. It has also a packet dissector, where the developer can see the whole sequence of packet exchange. One feature is the ability of running simulations of binary code compiled with other OS than Contiki as long as the hardware is compatible with the platforms supported by Cooja. It is possible for example, to run binary code compiled with TinyOS and RiotOS [Roussel et al., 2016].

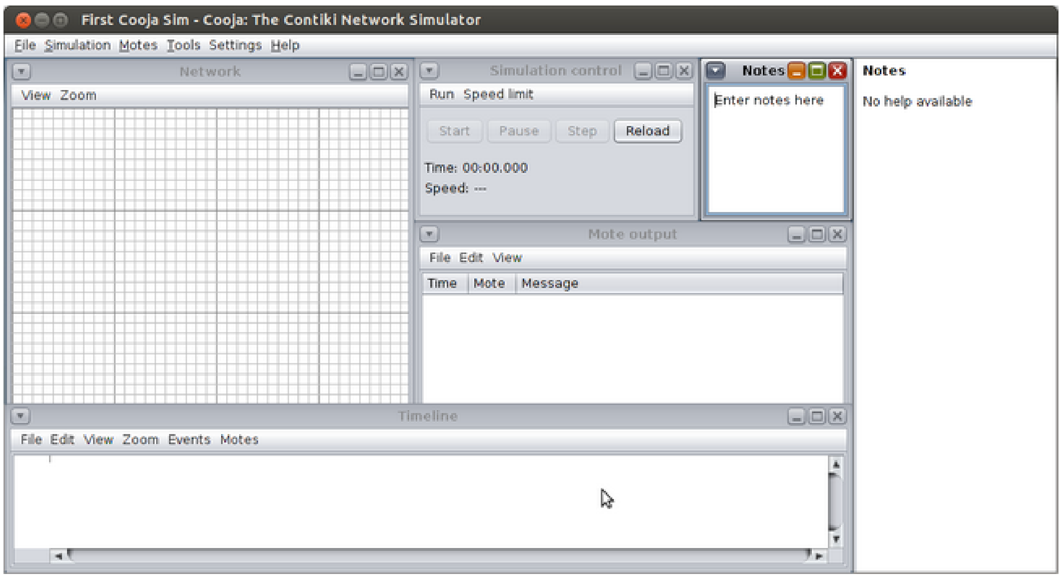
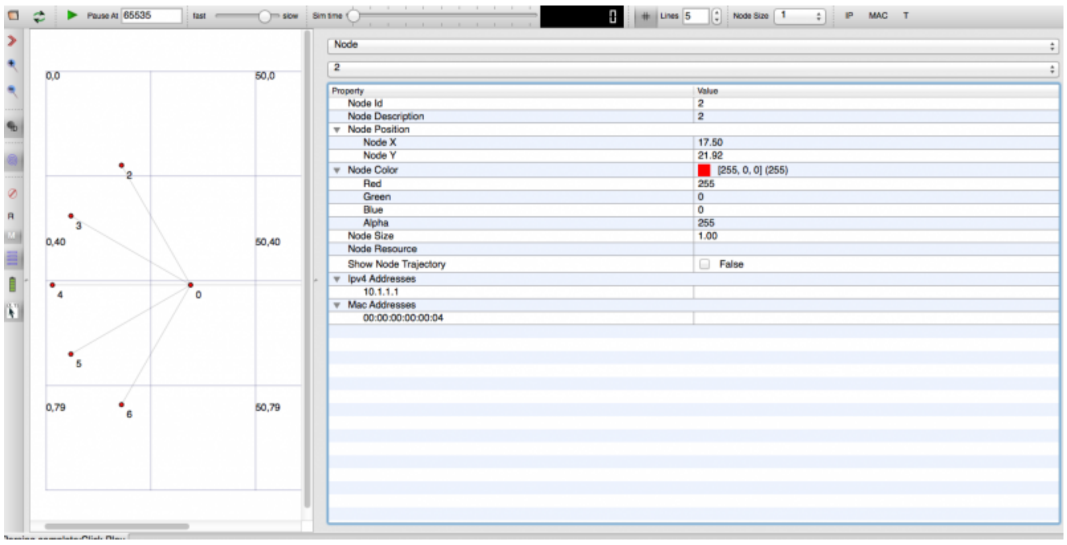


Figure 4.5.: Cooja Main Screen

Cooja also provides extra modules that can be added to the simulations, being one of them an energy monitoring module. This module however is only usable if the code being simulated was bundled with Contiki OS and compiled with support to energy metering. Unfortunately, like ContikiOS, the project has not been very active lately.

### 4.1.5 *NS-3*

The NS-3 [Kodali and Sarma, 2017] simulator was created as a successor to the NS-2 simulator, which is a simulator that has been extensively used in both private sector and academic world. This software is a modular simulator, so when building a simulation script, the user instantiates objects with specific functions according to the application need. It has many features, specially for reporting and generating traces and debug data. It is written in C++, and the simulation scripts can be run both in C++ or Python.



| Property | Value |
|---|---|
| Node Id | 2 |
| Node Description | 2 |
| Node Position | |
| Node X | 17.50 |
| Node Y | 21.92 |
| Node Color | 🟥 [255, 0, 0] (255) |
| Red | 255 |
| Green | 0 |
| Blue | 0 |
| Alpha | 255 |
| Node Size | 1.00 |
| Node Resource | |
| Show Node Trajectory | ☐ False |
| Ipv4 Addresses | |
| 10.1.1.1 | |
| Mac Addresses | |
| 00:00:00:00:00:04 | |

Figure 4.6.: NetAnim Interface

There is also a tool called NetAnim for the NS-3 that allows the user to have a Qt based graphical interface to visualise the simulation architecture. The interface of NetAnim is presented in the Figure 4.6 from the NS-3 website[7]. Besides being able to simulate WSN energy models [Wu et al., 2011], is also capable of handling large scale simulations with good performance [Weingartner et al., 2009]. Even though NS-3 has an energy model included , this model is focused mainly in wireless network interfaces, and not in WSN nodes with different energy operating modes.

## 4.2 TESTING PLANTFORMS

Nowadays, as presented in Section 4.1, there are many platforms that can be used to implement nodes for WSN. Another possibility is to simulate routing protocols for testing is to use test beds. Test beds are real hardware platforms that can be programmed directly with *bare metal* software, or with the help of an operating system depending on how much program memory is available and the software complexity. Some companies can also rent test beds for end users, and those can be remotely programmed. Choosing the platform depends mainly on the requirements of the project and its constrains such as connectivity, energy density and physical dimensions.

7 https://www.nsnam.org/wiki/File:Properties_panel.png/

### 4.2.1 *Devices Programming*

For programming sensor nodes, depending on the platform, there can be a few different software available when the goal is to generate *bare-metal*[8] binaries. Some manufacturers distribute IDEs together with compilers for their platforms and others toolchains to be used on Linux. The most common programming language for *uControllers* for example is C/C++ and Assembly, but for some of them Python[9] can also be used. Some of the manufactures sometimes also release libraries in order to reduce the development costs. Another way of programming the nodes is by using special OS designed for embedded systems.

#### *RiotOS*

The RiotOS is an open-source operating system with focus in wireless sensor networks created based on a previous OS called FeuerWhere [Rasool et al., 2017; Roussel et al., 2015]. The system has as main focus delivering real-time capabilities, low energy consumption and prioritise modern wireless networks. Those characteristics makes the RiotOS a promising platform for wireless sensor networks.

RiotOS is written in C and C++ and so are the applications intended to run on it, but a proper toolchain is required for each specific processor. It can use regular building tools such as *gcc* and *gdb* making it accessible to a bigger developer community. It is also compatible with a large hardware base, being compatible with 8, 16 and 32bit processors. The overview of Riot is presented in Figure 4.7 from the RiotOS website[10].

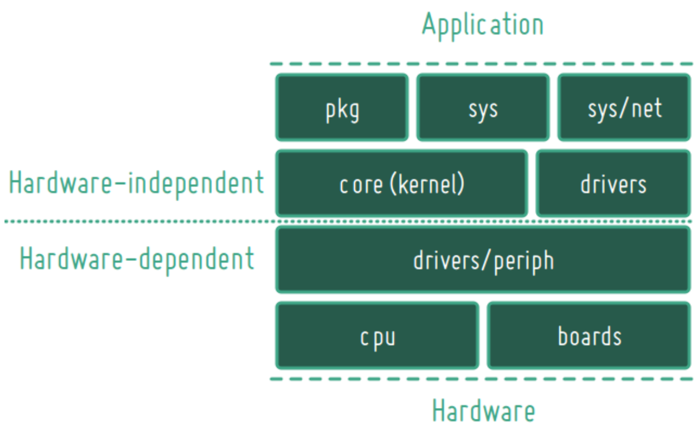


Figure 4.7.: RiotOS Overview

One significant advantage of RiotOS is that when building an application, since the OS is modular, only the necessary modules for such application to function will be compiled

8 Nodes executing instructions directly on logic hardware without an intervening operating system.
9 https://micropython.org/
10 https://riot-os.org/api/

and loaded making it resource and energy efficient. The developer is allowed to deploy as many threads as needed, being limited by the RAM memory and thread stack sizes [Baccelli et al., 2013]. RiotOS, like most of the real-time operating systems, has its software based in events which are triggered by timers or other interrupts. Since it is designed with focus on constrained wireless devices, it gives priority to supporting energy efficient technologies.

RiotOS allows the user to compile the same code to different target platforms as long as it is compatible with the resources available in the hardware. It also allows compiling the software using the host compiler, enabling part of the code to be tested and debugged in the host computer. In order to compile the code to different hardware, the user must before install the necessary cross-compiler. However, to make the development faster, a Docker[11] container was created with the cross-compiler for every supported platform pre-installed.

*TinyOS*

The TinyOS operating system is also an open-sourced OS designed for small wireless devices [Gao et al., 2011]. Unlike the RiotOS, the project is currently not very active, and not so many devices are having support being added. TinyOS is written in nesC, a dialect of C written exclusively to be used with TinyOS and to comply with its execution model.

The application software is structured around tasks, and each task can be fired based on events such as interrupts or timers. TinyOS applications can be compiled for a guest device on a host machine using a cross-compiler and simulated using a bundled simulator called TOSSIM, which is configured and set by means of a Python script.

As many of the open-source simulators, TOSSIM has been forked to create variations. One is the PowerTOSSIM, which was a fork aimed to add the capability of measuring with high accuracy the energy consumption of a TelosB board during the simulation [Perla et al., 2008]. PowerTOSSIM, has a very accurate energy estimation model, but unfortunately cannot be found anymore for download.

*FreeRTOS*

Another real-time operating system that aim the embedded systems market is the FreeRTOS. FreeRTOS[12] is a mature project which unlike the previously described ones, does not necessarily aim WSN devices. This project has the support of a lot of big companies with implementations made by Amazon and Espressif for example. Applications are written in C and are also based in tasks that can be fired based on interrupts or timers. FreeRTOS is a full real-time OS and programmers can implement complex scheduling schemes if needed.

11 Available at: https://www.docker.com/
12 Available at: https://www.freertos.org/

On Table 4.1, an overview about the main characteristics of discussed operating systems is presented.

Table 4.1.: OSes Overview

| | License | Main language | Simulator | RT | MT | IP4/IP6 |
|---|---|---|---|---|---|---|
| RIOT OS | LGLP 2.1 | C/C++ | - | yes | PT | Both |
| TinyOS | BSD 2 | nesC | TOSSIM | yes | no | Both |
| FreeRTOS | MIT | C/C++ | - | yes | yes | IPv4 |

RT - Real-Time, PT - Proto Threads, MT - Multi Threads

### 4.2.2 *Hardware*

In order to implement routing algorithms and to perform tests with actual devices, several cheap hardware implementations are available.

#### *ESP*

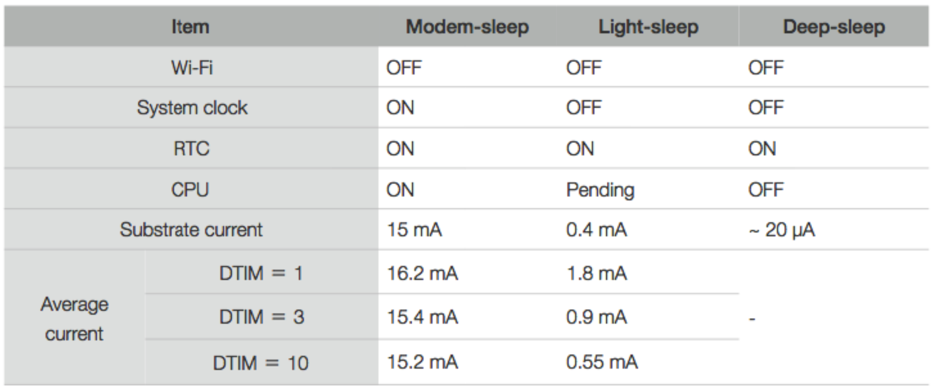
The ESP is a family of embedded hardware manufactured by a company called Espressif. It is a SoC containing a programmable micro-controller and a Wifi controller [Thaker, 2016]. The basic features that the ESP8266 offer are listed in Table 4.2:

Table 4.2.: ESP 8266

| | |
|---|---|
| Processor: | L106 32-bit RISC |
| Frequency: | 80MHz |
| Memory: | 32 kB instruction RAM<br>32 kB instruction cache RAM<br>80 kB user data RAM<br>16 kB ETS system data RAM |
| Wifi | IEEE 802.11 b/g/n |
| IO: | 16 GPIO Pins<br>SPI<br>I2C<br>UART |

This hardware is found in different formats and be also found as a System on a Board (SoB). The ESP8266 has different power modes with distinct characteristics. Figure 4.8, extracted from a brochure[13], shows the different operating modes and the expected current drawn in each of these modes.

13 https://www.espressif.com/sites/default/files/9b-esp8266-low_power_solutions_en_0.pdf



| Item | | Modem-sleep | Light-sleep | Deep-sleep |
|---|---|---|---|---|
| Wi-Fi | | OFF | OFF | OFF |
| System clock | | ON | OFF | OFF |
| RTC | | ON | ON | ON |
| CPU | | ON | Pending | OFF |
| Substrate current | | 15 mA | 0.4 mA | ~ 20 μA |
| Average current | DTIM = 1 | 16.2 mA | 1.8 mA | - |
| | DTIM = 3 | 15.4 mA | 0.9 mA | |
| | DTIM = 10 | 15.2 mA | 0.55 mA | |

Figure 4.8.: ESP 8266 Power modes

According to Figure 4.8, when the goal is to run this board for years, it should utilise the maximum as possible the deep sleep mode. When in deep sleep, the board can only be awaken via external interrupt or an internal interrupt connected directly to an external interrupt port, which is fired by a Real Time Clock (RTC) timer. There are several different boards sold with the ESP8266, each of them with distinct characteristics. Figure 4.9 shows the block diagram for the ESP8266 extracted from the datasheet[14].

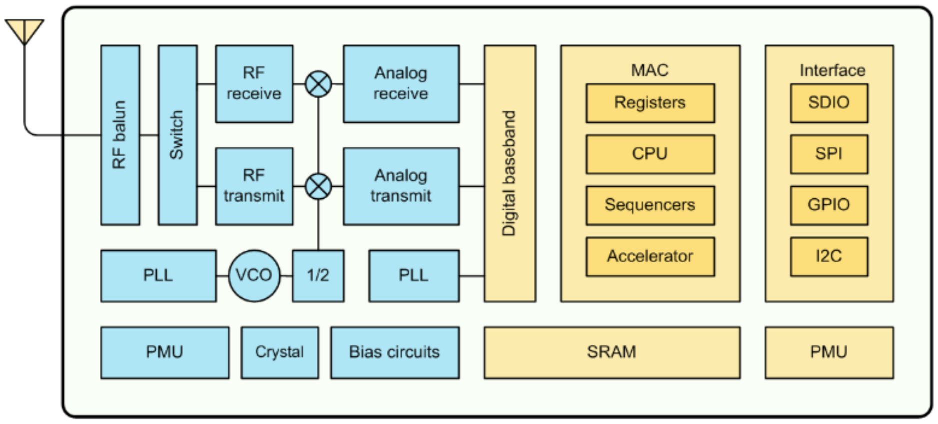


Figure 4.9.: ESP 8266 Block diagram

*TelosB*

As the ESP family, the TelosB is a hardware designed for WSN motes. It is equipped with a low power processor and has built in several additional hardware that can be used for sensor data acquisition. It has a IEEE 802.15.4 compliant wireless radio and is fully supported by TinyOS. In Figure 4.10 extraceted from the datasheet[15], a block diagram of the board is presented.

14 https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf
15 http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf

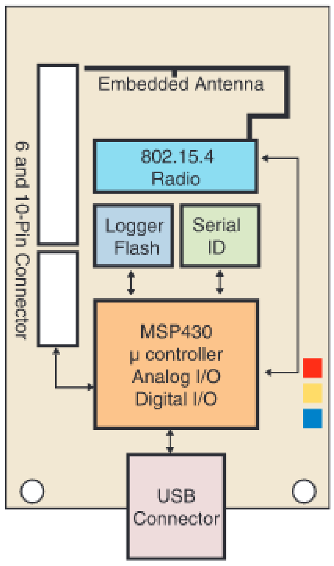


Figure 4.10.: TelosB TPR2400 Block diagram

The basic features are listed on Table 4.3:

Table 4.3.: TelosB TPR2400

| | |
|---|---|
| Processor: | TI MSP430 16-bit RISC |
| Frequency: | 8MHz |
| Memory: | 10 kB RAM<br>16 kB Configuration ROM<br>48 kB Program Flash Memory<br>1 MB external flash |
| Wireless radio | IEEE 802.15.4 b/g/n |
| IO: | 8 channel 12 bit ADC<br>2 channel 12 bit DAC<br>Digital I/O<br>SPI<br>I2C<br>UART |

TelosB's clock can also be used in lower frequencies such as 1 or 4 MHz for increased energy savings, and furthermore, it has different special sequential power saving modes, where in each of them embedded devices are turned *ON* or *OFF* so that the higher the level goes, more devices are disabled to save energy [Prayati et al., 2010] as presented in Table 4.4:

Table 4.4.: TelosB Power Modes

| Mode | Description |
|---|---|
| LPM0 | All hardware is activated |
| LPM1 | The CPU is disabled |
| LPM2 | The loop control for the fast clock is disabled |
| LPM3 | The fast clock is disabled |
| LPM4 | The DCO oscillator and its DC generator are disabled |
| LPM5 | The crystal oscillator is disabled |

### *MicaZ*

MicaZ [Ali et al., 2011] is a SoB made by the company Crossbow and like TelosB is compliant with IEEE 802.15.4. It is equiped with an Atmel micro-controller that became popular with the Arduino platform. In Figure 4.11, a block diagram extracted from the datasheet[16] is presented.

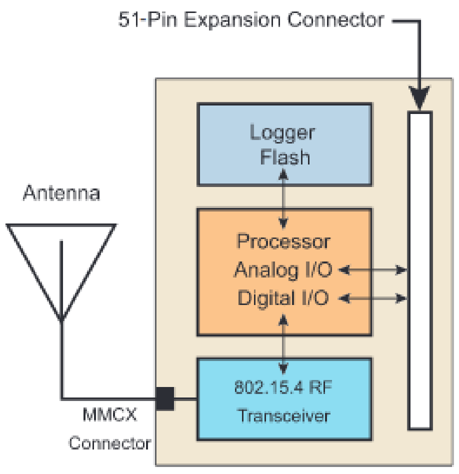


Figure 4.11.: MicaZ MPR2400 Block diagram

The micro-controller is energy efficient and runs in a low frequency. From all the hardware platforms presented, this is the one that has less available memory for the user. More details about the board are presented in Table 4.5.

16 http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf

Table 4.5.: MicaZ MPR2400

| | |
|---|---|
| Processor: | Atmel AVR ATmega128L 8bit |
| Frequency: | 8MHz |
| Memory: | 4 kB Data RAM<br>4 kB Data ROM<br>128 kB Program Flash Memory |
| Wireless radio | IEEE 802.15.4 |
| IO: | 8 channel 10 bit ADC<br>Digital I/O<br>SPI<br>I2C<br>UART |

Unfortunately due the diversity and heterogeneity of the available hardware for WSN, none of the operating systems at the moment of the writing of this document, are compatible with all technologies, processors and protocols. Table 4.6 gives an overall compatibility matrix.

Table 4.6.: OSes compatibility

| | ZigBee | 6LoWPan | 802.11 | TelosB | MicaZ | ESP8266 | ESP32 |
|---|---|---|---|---|---|---|---|
| RIOT OS | Yes | yes | no | yes | yes | yes | yes |
| TinyOS | yes | yes | no | yes | yes | no | no |
| FreeRTOS | no | no | yes | no | no | yes | yes |

### 4.2.3 *IoT Lab*

Another approach for testing all things related to WSN and IoT is to use test beds available to rent such as IoT Lab[17]. In this type of platform real hardware is deployed in a controlled environment and sometimes even mobility is available with sensors mounted on moving robots that can be remotely controlled as the one shown in Figure 4.12

17 https://www.iot-lab.info/

Figure 4.12.: Test bed Robot from IoT Lab

They usually have available different hardware platform, saving research institutions the trouble of procurement and maintenance.

## 4.3 SUMMARY

Since this work will include different simulation scenarios, five simulators were researched and tested, *Mininet-Wifi*, CORE, CupCarbon, Cooja and NS-3. From the simulators, *Mininet-Wifi* and CORE were considered the ideal choices for prototyping, since they allow the creation of native applications for the host operating system and also enable more freedom in metric collection and energy modelling. Those were the main reasons why those simulators were chosen for this work's test stage. There are other operating systems that are aimed at embedded wireless devices like Contiki OS, for example, but those mentioned are the most currently active and with more contributors. With some preliminary tests, RiotOS has shown to be a very complete and easy to use OS, since it has a very active community and new devices are added constantly to the supported list.

Another important aspect is that RiotOS and TinyOS generate smaller binaries to be downloaded to the devices, which indicated that only the necessary modules of the OS are included, while FreeRTOS creates large binaries that possibly will consume more energy. In the specif case of the ESP[18] devices, the compilation added drivers for devices from the board that were not used in code. More tests however are required to have precise measurements of energy consumption with different operating systems and same devices and algorithms. This dissertation however only focused on simulations, leaving the implementation in hardware as future work.

18 https://www.espressif.com/en/products/software/esp-sdk/overview

# 5 TESTING METHODOLOGY

In order to test the proposed protocol and gather results data for analysis, it was implemented to run on *Mininet-Wif*i and *CORE* simulators. The analysis was made comparing EAGP to other well-established protocols. All the different protocols chosen for the tests were implemented independently so that it is easy to switch between them during the tests. All the software, constraints, models and scenarios described along this chapter were equally applied to all tested protocols, so the conditions were always the same. More details about the test software can be found in Annex A.1

## 5.1 PROTOCOLS

From the protocols presented in Section 2 some were selected to be implemented and used as baseline for comparison with the proposed protocol:

- Gossip - Since the proposed protocol is an epidemic protocol, a purely gossip protocol was implemented and used as the main baseline;
- Gossip with fan-out - The gossip protocols are usually implemented with some sort of fan-out as an optimisation to reduce the packet duplication in the network, so a version with configurable fan-out was also implemented;
- MCFA - This protocol was chosen since it can also be implemented with the continuous delivery traffic model and represents a good comparison since it is not a gossip protocol;
- EAGPD - A different version of the proposed protocol was created based on the Plumtree protocol that had similar eager / lazy push behaviour described in [Leitao, 2007]. More details about this protocol can be found in Annex 3.2.5.

## 5.2 OBSERVED METRICS

Base on the protocol goals described in Section 3.1, some metrics were considered to be recorded and analysed during the simulations and used to compare the proposed protocol with other protocols [Yitayal et al., 2015].

- Network longevity - A WSN that implements an energy efficient protocol to distribute data is expected to live longer in the sense that it will take longer until the last node is able to deliver data to a sink. The idea is to record the time when the last packet was delivered at a sink.
- Delivery rate - Depending on the application, it might be crucial that all packets created on a node are delivered to all nodes or to the sink. During the simulations, a node will always be chosen as a sink, and taken as reference for message delivery. All messages created on a node have a unique random ID, so that in the end of each simulation it is possible to verify which messages were able to be delivered to the sink. Furthermore, it will be also recorded the percentage of the packets created by each node that were delivered in relation to other nodes.
- Data redundancy - Using the gossip dissemination process to send packets induce them to follow different paths towards the sink. Hence, the same packet might be received by the sink more than once. Redundancy can be good to ensure high reliability, but excess redundancy can lead to energy waste. Comparing the implosion with the reliability can indicate how efficient a routing algorithm really is.
- Energy efficiency - At the end of each simulation, with all the metrics collected, it is possible to assess how much energy was used and how many packets were delivered. It is therefore possible to calculate how much energy was used by the network to deliver each packet to the sink. The least energy used, more energy efficient the routing algorithm is to deliver packets.
- Latency - Even though low latency is not a design goal of the proposed algorithm, the time between the packet creation and delivery at the sink is also calculated to assess the latency impact of the protocol compared to other protocols.

## 5.3 TEST SCENARIOS

In all scenarios described in this section, every node's software starts and wait from a signal from the host to start the simulation, ensuring they start simultaneously. Time in the nodes is calculated based on the host timer, and all tasks are performed by a scheduler that is also based on the host timer. By doing so, it is possible to have a global clock for all nodes and calculate delays between actions.

### 5.3.1 *Steady State Scenario*

The steady state scenario aims to view how the algorithms perform during a predefined slice of time from the network's total lifetime. That represents how the protocol behave during the majority of the network's lifetime, when the all the nodes are participating and none dies before the end of the simulation. So, it is configured in a way that the nodes have a large amount of remaining energy available and configured to stop the execution after a predefined amount of time.

### 5.3.2 *End of Life Scenario*

The end of life scenario is the opposite of the steady state. The nodes are configured with a small amount of remaining energy, so that it is possible to view the difference in longevity of the network when using exactly the same configurations listed in Section 6.1. Since in this scenario the nodes have a very small battery, the simulation ends fast and it is possible to evaluate the behaviour of the protocols in the end of life of the network, when some nodes start to die before others.

### 5.3.3 *Mobility Scenario*

The mobility scenario is configured in the same way as the steady state, however mobility is added to the nodes to observe how they adapt to topology changes during execution. Since some algorithms have a start-up phase, the mobility is only introduced after a predefined amount of time. The mobility used for the tests is a simple random walk model where after every second, the geographical coordinates of the nodes are changed by random steps.

## 5.4 TOPOLOGIES

In order to evaluate the proposed protocol in different situations and conditions, different topologies were created. The main ones from whom results will be presented, are shown below. In order to speed up the topology creation, a graphical tool was created allowing the creation of topologies and to export them as a *Mininet-Wifi* or *CORE* python script, and a JavaScript Object Notation (JSON) object file. With the JSON file, the user can reopen the topology and make adjustments when needed. More information can be found in Annex

B.1. The number of nodes in each topology was limited according to the capabilities of the testing environment used.

### 5.4.1 *Symmetrical*

The symmetrical topology has a sink node (red node in Figure 5.1) in the middle and nodes equally distributed around it. This topology is a best case scenario for the protocols to achieve maximum efficiency. All nodes are more or less equidistant, symmetrically distributed in a perfect square and there is no overlap in signal coverage.

| Grid[1] | # of nodes | Radio range |
|---|---|---|
| 50m | 29 | 90m and 120m[2] |

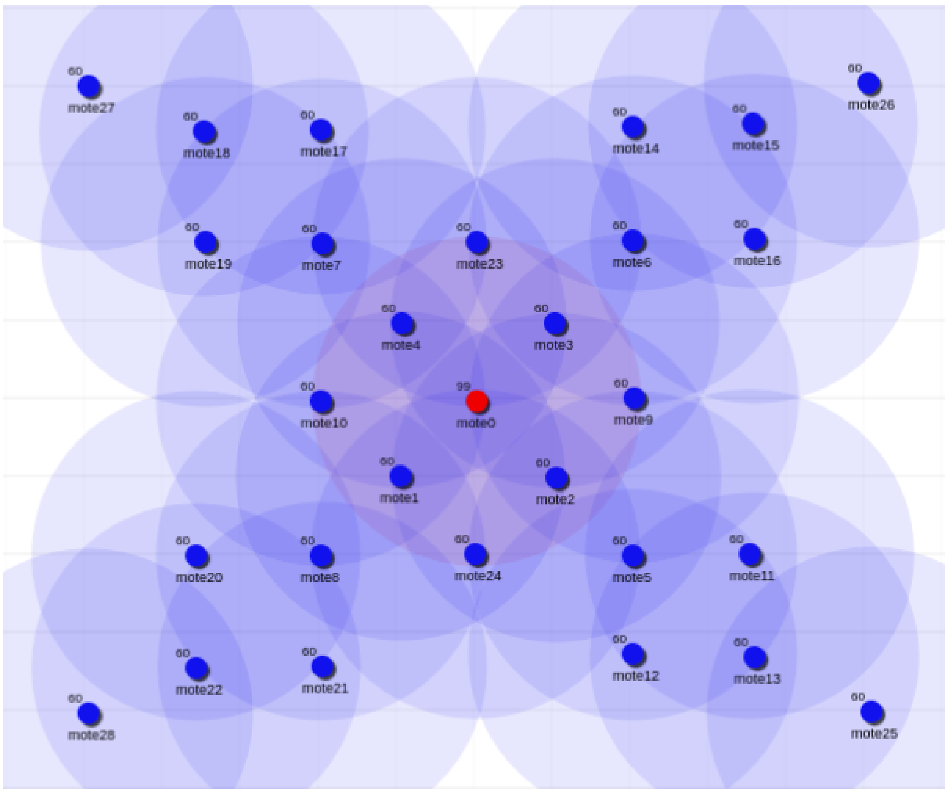


Figure 5.1.: Symmetric Topology

1 Grid seen in the background to assist the reader in distance assessment.
2 For the mobility scenario, the radius is increased to 120m. Since 90m is a borderline radio range considering the disposition of the nodes, when the nodes start moving they would immediately lose contact with each other.

### 5.4.2 *Asymmetrical*

This asymmetrical topology has a sink node in the extreme left and nodes distributed to the right. Some nodes are obvious paths in the way to the sink, like the node *mote5* for instance, and it is expected that the energy depletion in those nodes could isolate the whole network from the sink.

| Grid | # of nodes | Radio range |
|---|---|---|
| 50m | 19 | 120m |

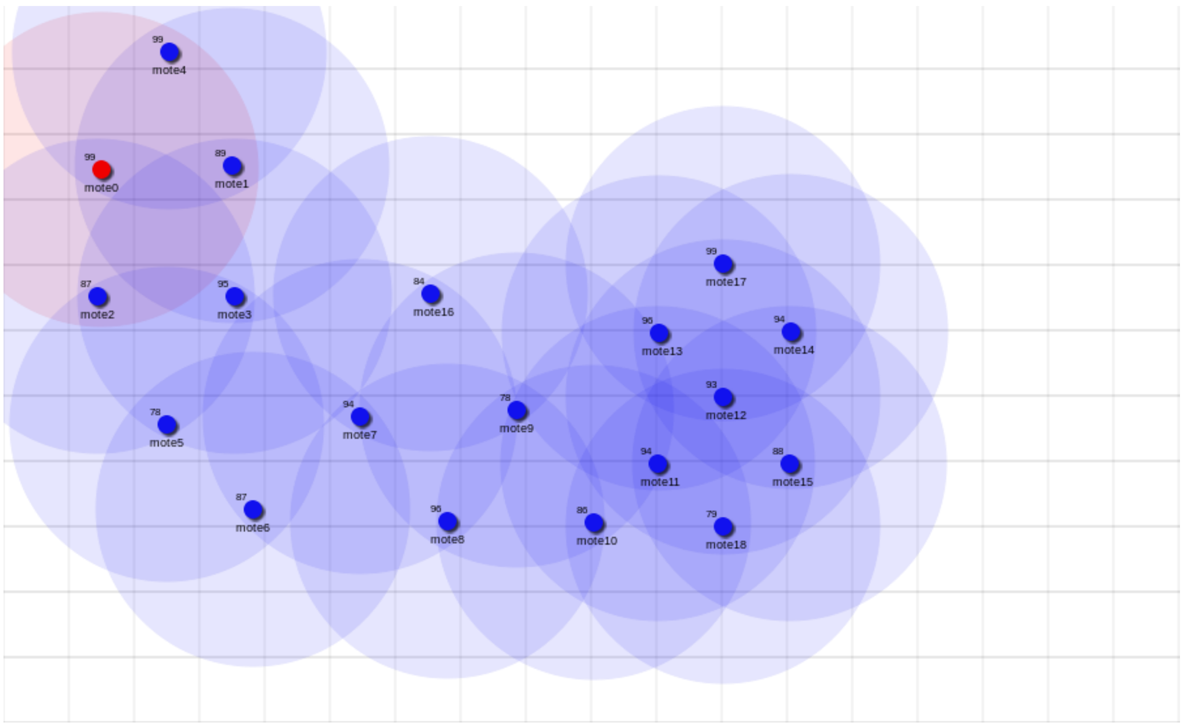


Figure 5.2.: Asymmetrical Topology

### 5.4.3 *Random*

This random topology was created using the random function of the scenario generator describe in Section B.2.2 and has a sink node in the middle and nodes randomly distributed around it. Node *mote3* connects a whole cluster on the left side of the map to the sink, so this node is essential to all nodes in this cluster. This topology is closer to a real case since was randomly created, and the proposed protocol has as a requirement to perform good in any topology. It has nodes with different distances and overlapping signal coverage.

| Grid | # of nodes | Radio range |
|---|---|---|
| 50m | 30 | 154m |

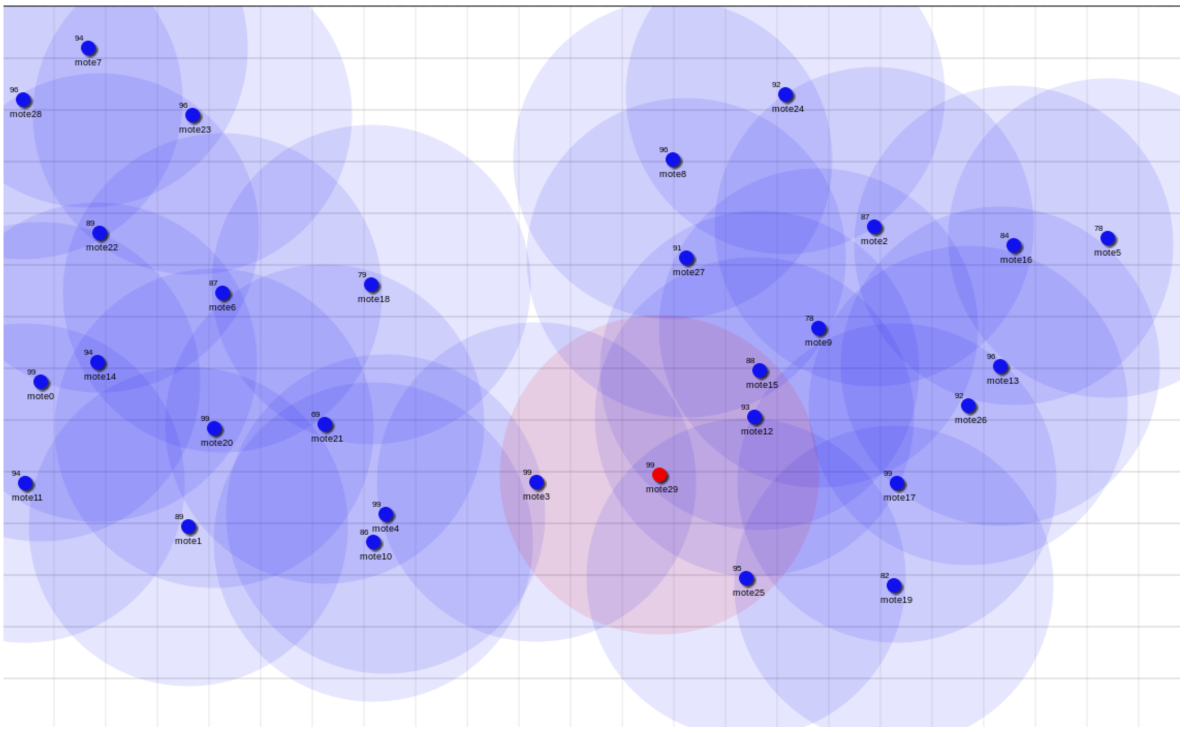


Figure 5.3.: Random Topology

## 5.5 IMPLEMENTATION

For testing purposes, an implementation of the protocols presented in Section 5.1 was created in Python, where they could be simulated using Mininet-Wifi or CORE. The idea is to simulate the routing protocol at the application level, using UDP network sockets as the network endpoints. For that, two main classes were defined to run the simulation, the **main** class and the **Node** class. Mode details can be found in Annex A.1.

### 5.5.1 *Requirements*

For the simulated implementation of the protocol some requirements have been defined:

- SR01 - The user must be able to run the simulation in real time or accelerated;
- SR02 - The user must be able to interact with each node to monitor activities, and consult status and parameters during the execution of the simulation via command prompt;
- SR03 - The simulation must have real-time properties available via RESTful API endpoints. The developed API is described in Section B.2.1;
- SR04 - The simulation results must be stored in Comma-separated Values (CSV) files, so that reports can be generated;
- SR05 - The user must be able to see the topology in real-time via a graphical interface. The developed API is described in Section B.2.2.

More information about the software can be found in Annex A.1.

### 5.5.2 *Simulations*

When running a simulation with Mininet or CORE the topology is set with help of a Python script. All the virtual wireless network interfaces are configured as *ad-hoc*, and receive an IP address corresponding to the node ID. Each spawned main class is than responsible to simulate each node in the network and each node has an open **UDP** socket listening the medium for packets. When a node wants to transmit, it creates a socket, binds to the *ad-hoc* network interface, and send data to the broadcast address and listening port. All nodes in range will receive the packet and the listening socket will handle it. The main classes overview is illustrated in Figure 5.4.

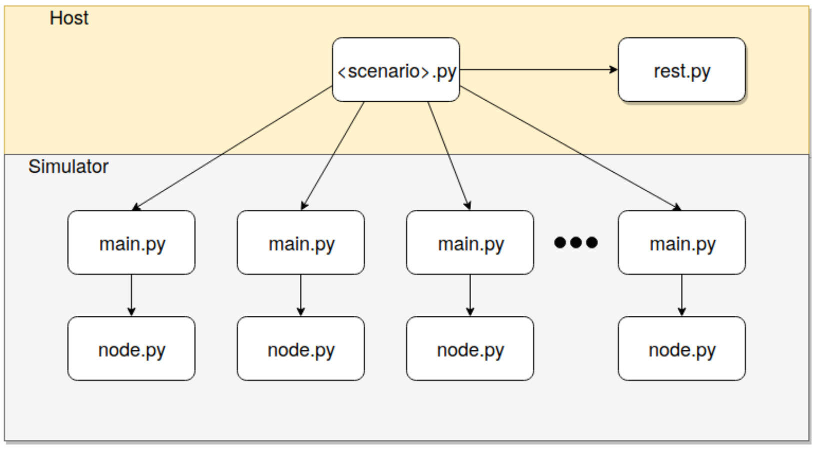


Figure 5.4.: Scenario Overview

### 5.5.3 *Energy model*

One of the most important aspects in the simulation of an energy-aware protocol is to be able to simulate the energy depletion of each node during the simulation duration. For that, a model of the energy consumption was created based on [Shnayder et al., 2005] and [Zhou et al., 2011] and a battery implemented as a class so that each node can have access to the battery level during run-time, and a report about the energy use in the end of the simulation. As mentioned in Section 2.1.3, the energy consumption on a WSN node is divided in three categories and the total energy is the sum of those as shown in Equation 4.

- Circuit Energy ($E_{\text{Circuit}}$) - Energy consumed by the circuit board and main processor of a node;
- Radio Energy ($E_{\text{Radio}}$) - Energy consumed by the radio module of the node;
- Sensor Energy ($E_{\text{Sensor}}$) - Energy consumed by the sensor installed on the node.

$$E_{Total} = E_{Circuit} + E_{Radio} + E_{Sensor}, \tag{4}$$

#### *Circuit Energy*

The circuit energy, or processor energy [Zhou et al., 2011] model is defined by the energy consumed by the processor during all possible cycles the processor can assume. Taking for instance a ESP8266 board[3], which will be later used as reference for the tests, the energy states the processor can assume is represented in the Figure 5.5.

3 This board is very cheap and accessible with plenty of information available, enabling easy verification

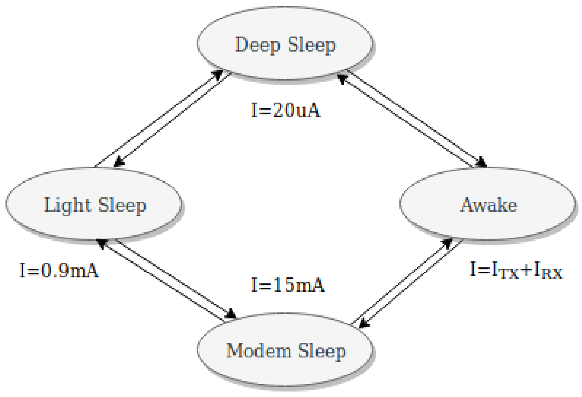


Figure 5.5.: Possible Energy States on a ESP8266

So when a ESP8266 board is not sleeping (deep or light sleep) but the radios are *OFF*, it is considered to be in *modem sleep*. When a board is receiving or transmitting data, it will consume the modem sleep current in addition to the current associated to the radio. Each state consumes the equivalent to the time spent in that state multiplied by the power[4] consumed in the state as show in Equations 5, 6 and 7.

$$E_{Light} = V * I_{Light} * t_{Light} \tag{5}$$

$$E_{Deep} = V * I_{Deep} * t_{Deep} \tag{6}$$

$$E_{Modem} = V * I_{Modem} * t_{Modem} \tag{7}$$

So, the total circuit energy is finally:

$$E_{Circuit} = E_{Light} + E_{Deep} + E_{Modem} \tag{8}$$

Several different types of boards have available a type of deep sleep or hibernation mode. On the particular case of the ESP8266 board and in most of the others, in this mode even the processor is de-energised, so there is not retention of data in volatile memory. Since no internal function is running in the processor, in order to wake up from a deep sleep cycle it needs an external interrupt. This interrupt can be triggered by a pulse generated by a RTC or by another hardware [Systems, 2019]. When the board is in this mode, to be able to communicate with other nodes, they would need to be synchronised and wake up at the same time, or another external ultra low power receive only radio would need to be added to the node just to wake up the main board. Hence, this mode was not considered in the simulations.

4 Voltage (V) multiplied by current (I)

One important aspect of the circuit energy is that for the same algorithm, different processors will spend different amounts of energy for computational calculations. The only way to estimate how much energy would be spent on calculations, would be to cross-compile the code to machine code for a specific processor and count how many instructions each section of the code has. Knowing how much energy each instruction consumes, makes it possible to estimate how much energy in total each section of the code consumes. This work has been previously done for the TelosB boards [Prayati et al., 2010], and the PowerTOSSIM simulator would use tracing to make accurate estimations of energy consumption by the processor. Unfortunately this simulator can no longer be found for download and has the disadvantage of not simulating a virtual battery. In order to make an approximation of power consumption during computation, the execution time for each part of the code is measured and multiplied by a *computational constant*. The expected result is that for different host machines where the simulations can be performed, the results will be different. Hence, they are comparable only when different algorithms simulations are run in the same machine. So the reason to try to estimate consumption in computational time is to know how the algorithms compare between each other, and not to have an absolute estimate of power consumption. An example on how to estimate power consumption with high accuracy is to use an external hardware that can be connected to the prototype board during execution as with a Nordic Semi hardware[5].

*Radio Energy*

The radio energy model is defined by the energy spent transmitting or receiving data. For most applications, the transmission radio can be constantly *OFF* until there is a need to transmit data. The receiving radio in the other hand, can be *ON* or *OFF* depending on the application or routing algorithm. In some applications, the radio can be turned *ON* and *OFF* rapidly in order to listen to the medium whilst saving energy, but this is usually taken care by the MAC protocol. The efficiency of the energy saving is proportional to how long the radio can be *OFF* without missing a peer calling. The total RX and TX energy can be than calculated by:

$$E_{TX} = V * I_{TX} * t_{TX} \tag{9}$$

$$E_{RX} = V * I_{RX} * t_{RX} \tag{10}$$

As an approximation for the radio energy, the time spent for RX and TX have been set to a fixed value independent of how much data is being transmitted.

5 http://www.nordicsemi.com/Software-and-Tools/Development-Kits/Power-Profiler-Kit

*Sensor Energy*

The sensor energy is the energy spent to supply the required power by a sensor to read data. For the simulation, a sensor was chosen to be the reference for all simulations based on the availability and low cost. The DHT22[6] has the characteristics presented in Table 5.1.

Table 5.1.: DHT22 - Electrical data

| Item | Condition | Min | Typical | Max | Unit |
|---|---|---|---|---|---|
| Power Supply | DC | 3.3 | 5 | 6 | V |
| Current | Measuring | 1 | - | 1.5 | mA |
| Time | Collecting | - | 2 | - | Seconds |

It was considered that the sensor is always *OFF*, and than turned *ON* for a reading request during two seconds to comply with the specification of typical collecting time.

$$E_{Sensor} = V * I_{Sensor} * 2 \tag{11}$$

The total energy consumed during the simulation is than:

$$E_{Total} = E_{Circuit} + E_{RX} + E_{TX} + E_{Sensor} \tag{12}$$

The total energy consumed is a sum of all the energy spent during a full cycle of reading the sensor, multi-casting the information, receiving information and forwarding buffered messages. Figure 5.6 represents the energy consumption during a full cycle. It must be noted that during the transition from one state to another, usually there are also transitory overshoots of energy [Perla et al., 2008] that have not been considered during the simulations.

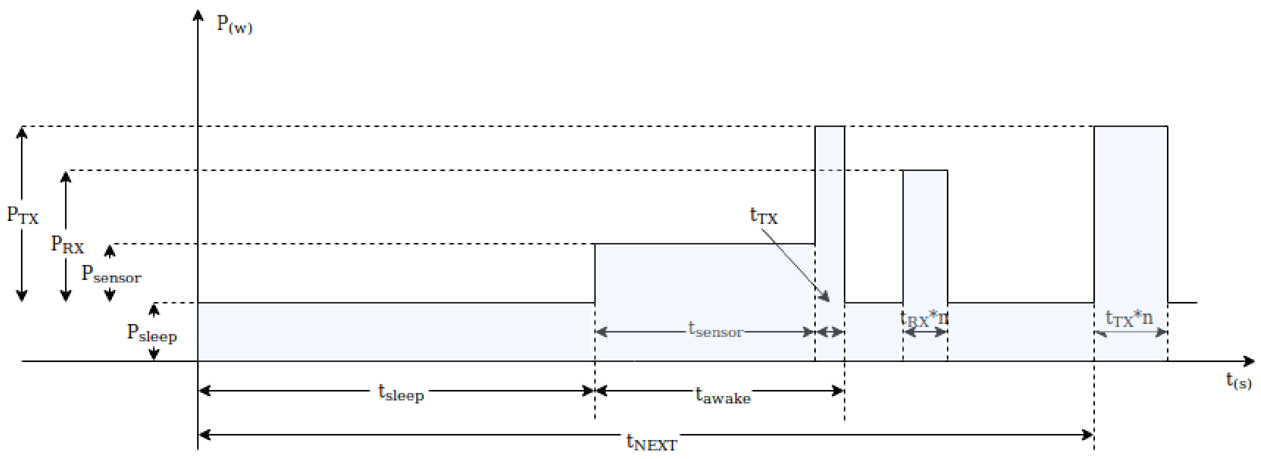


Figure 5.6.: Energy Cycle

6 https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf

## 5.6 SUMMARY

Using Python to prototype routing protocols enable accurate behaviour modelling and tracing, but estimating power consumption is a complex problem that require specific tools. The energy model developed for this work enables a comparison between protocols for tests run in the same machine, but results from different test beds are not comparable. All the approximations considered were implemented across all the different protocols simulated. The different topologies and scenarios used for the tests will show different positive and negative aspects of each protocol under different circumstances, since it is not expected that a specific protocol will be better or worse than the other under all topologies, constrains and scenarios.

# 6

# RESULTS AND DISCUSSION

In this chapter the results for each scenario and topology are presented. Although both *Mininet-Wifi* and *CORE* were suitable for the tests, and *Mininet-Wifi* chosen as the preferred platform, it started to present issues when the number of packets per second in the network started to increase, and as a consequence the simulated network kernel module would simply crash. *CORE* on the other hand, could handle all simulations running with time being simulated ten times faster, simulations with faster than that would yield inconsistency results.

## 6.1 GENERAL CONFIGURATIONS

All the simulations have a set of fixed configurations and a set of variable configurations. The fixed configurations are constant throughout all the simulations. They are listed here so that by using the same software, or creating a different software to implement the same protocols, similar results should be achieved. The results presented in this section are a mean of the results of five simulations excluding the higher and lower values (avoiding outliers).

### 6.1.1 *Test environment*

All the results are based on simulations and all the simulations were run in the same machine which specifications are listed in Table 6.1. Running the same simulations in a different machine would yield different results for the estimated computational footprint, as it would even if the protocols were implemented in real sensor nodes.

Table 6.1.: Test platform

| | |
|---|---|
| Processor Manufacturer: | AMD |
| Model: | Ryzen 7 1700 |
| Number of cores | 8 (16 HW Threads) |
| Microarchitecture | Zen |
| Cache L1 | 768KB |
| Cache L2 | 4MB |
| Cache L3 | 16MB |
| Memory | 16GB DDR4 |
| Operating System | Xubuntu 18.04 |
| Simulator | CORE 5.3.1 |

### 6.1.2 *Fixed Configurations*

Some initial setup is also made for the simulations. There settings reflect the simulated sensor node and the wireless channel. All results presented here were run with the setup listed in Table 6.2:

Table 6.2.: Fixed Configurations

| | |
|---|---|
| Battery Voltage: | 3.7V |
| Sensor sleep time (s): | 15 + random(0 to 35) |
| TX time (ms) | 30 |
| RX time (ms) | 40 |
| fan-out | 3[1] |
| Energy Model | ESP8266 |
| Wireless Model | BasicRange |
| Bandwidth (bps) | 54000000 |
| Jitter | 0 |
| Delay Between Nodes(us) | 5000 |
| Error | 0 |

### 6.1.3 *Energy Model*

For the energy model the ESP8266 was chosen as reference since it is an accessible and cheap wireless board. It has the parameters extracted from the hardware datasheet [Systems, 2019] presented in Table 6.3. The sensor energy was set to a value very low so that the sensing energy has a low affect in the final results, since sensing is not in the scope of this work.

1 Value chosen by experimentation

Table 6.3.: Energy Parameters

| | |
|---|---|
| Deep sleep current (A): | 0.00001 |
| Modem sleep current (A): | 0.015 |
| Awake current (A): | 0.081 |
| TX current (A): | 0.170 |
| RX current (A): | 0.056 |
| Computational Constant[2] | 250 |
| Sensor Energy (J): | 0.0000000111 |

## 6.2 RESULTS

### 6.2.1 *Steady State*

The results presented here are from simulations configured to run for 10000 seconds in sensor's time. The battery capacity for each node is set to 5000 mAh, and each node starts with different remaining charge to simulate a heterogeneous state. In all simulations, the initial charge configuration is repeated and the only parameter that is stochastic is the time the node sleeps between sensing events. The vector below illustrates the initial battery level of each node, starting from node *mote 0*.

battery_level = [ '99', '89', '87', '95', '99', '78', '87', '94', '96', '78', '86', '94', '93', '96', '94', '88', '84', '99', '79', '82', '99', '69', '89', '96', '92', '95', '92', '91', '96', '87' ]

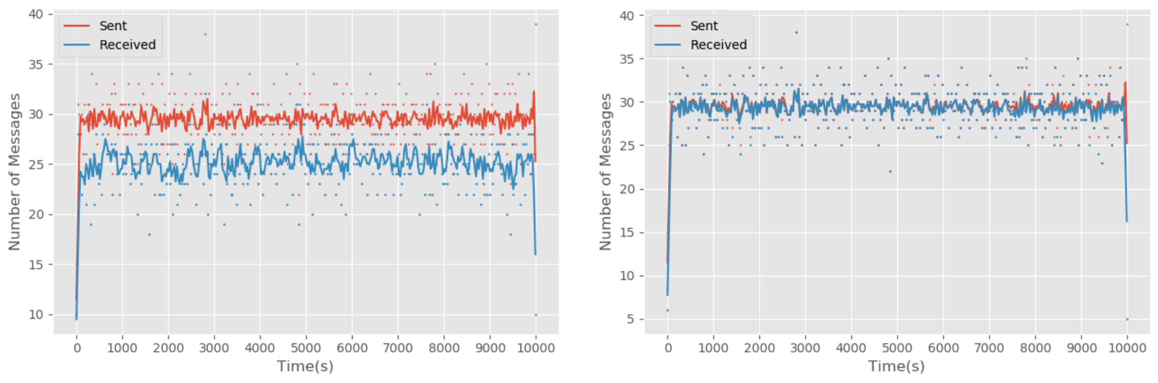
#### *Symmetrical Topology*

This topology, presented in Figure 5.1, has a hop radius of 5, so TTL was initially set to 5 in all simulations that use it to measure the parameters considering reaching from the farthest node to the sink as coverage goal. Measuring the hop radius could be made automatically by using a topology discovery algorithm, or it could also be manually set during pre-deployment configuration or even be set later via a configuration epidemic message. An example of topology discovery is made by the MCFA algorithm, as described in Section 2.2.5, and that is how it was calculated for this work.

The plots in Figure 6.1(a) and Figure 6.1(b) illustrate the run-time of the simulation. The red points represent the number of packets created during a window of 60 seconds and the lines are a moving average with period of 240 seconds. The blue points represent the number of packets delivered at the sink. It is possible to notice that compared to a gossip protocol with fan-out of 3, value that was chosen by experimentation, when using

2 Constant multiplier explained in Section 5.5.3

the proposed EAGP protocol the two curves almost overlap. The results for all protocols are presented in Table 6.5 whose headers are explained in Table 6.4.



(a) Gossip w/ fan-out of 3 Nodes (b) EAGP with $T_{MAX}$=10s

Figure 6.1.: Network Longevity

Table 6.4.: Results Table's Headers

| | |
|---|---|
| GOSSIP: | Naïve gossip implementation |
| GOSSIPFO: | Gossip implementation with fan-out |
| EAGP10: | EAGP with $T_{MAX}$ = 10s |
| EAGP20: | EAGP with $T_{MAX}$ = 20s |
| EAGP50: | EAGP with $T_{MAX}$ = 50s |
| EAGP100: | EAGP with $T_{MAX}$ = 100s |
| EAGP500: | EAGP with $T_{MAX}$ = 500s |
| EAGPD10: | EAGPD with $T_{MAX}$ = 10s |
| MCFA: | MCFA implementation |

When analysing all measurements for all protocols on Table 6.5 it is possible to see in the first line, regarding energy consumption[3], that the EAGP consumed 10% less energy than the gossip with fan-out. Furthermore, the energy usage tends to lower when $T_{MAX}$ increases. As expected, the energy consumption is much lower in the MCFA since there is no waste of energy sending messages in the wrong direction if the goal is to deliver packets to a sink node. The second line of the table presents the values of data duplication. Those are the average number of times each packet (verified by the unique packet ID) was received. The EAGP with $T_{MAX}$ of 10 seconds had 72% less duplicated packets at the sink than the pure Gossip protocol, and yielded a reduction of 31% when compared to a Gossip with fan-out.

3 Presented only communication and computational energy since sensor energy and the energy spent sleeping is the same.

The next lines of Table 6.5 show the average maximum and minimum hops needed to deliver each packet to the sink. The minimum is dictated by the topology, and was closely the same for all protocols. For the maximum, EAGP achieved a reduction of 6% when compared to the pure gossip protocol, and when compared to the gossip with fan-out the reduction was of around 4%. The average maximum number of hops has a direct impact in the energy reduction of EAGP due to the energy cost that every hop represents.

Table 6.5.: Symmetrical Summary

| | GOSSIP | GSP. FO | EAGP 10 | EAGP 20 | EAGP 50 | EAGP 100 | EAGP 500 | EAGPD 10 | MCFA |
|---|---|---|---|---|---|---|---|---|---|
| Energy (J) | 23854.59 | 8978.86 | 8137.76 | 8134.75 | 8101.98 | 7983.99 | 7427.65 | 8904.34 | 2422.26 |
| Duplication | 8.26 | 3.04 | 2.29 | 2.27 | 2.29 | 2.26 | 2.22 | 2.71 | 1.88 |
| Max Hops | 5.00 | 4.79 | 4.69 | 4.68 | 4.68 | 4.66 | 4.56 | 5.08 | 3.79 |
| Min Hops | 2.80 | 2.68 | 2.80 | 2.79 | 2.78 | 2.78 | 2.69 | 2.76 | 2.79 |
| Latency(s) | 0.01 | 0.01 | 7.69 | 15.20 | 36.16 | 73.02 | 347.18 | 14.97 | 0.01 |
| Efcy. (%) | 99.73 | 84.96 | 99.04 | 98.89 | 97.98 | 97.95 | 91.81 | 91.00 | 99.73 |
| Pck. Del. | 9828 | 8372 | 9760 | 9745 | 9655 | 9652 | 9047 | 8968 | 9626 |
| J / Pck. | 2.43 | 1.07 | 0.83 | 0.83 | 0.83 | 0.82 | 0.82 | 0.99 | 0.25 |

The latency on Table 6.5 is the average time measured from when the packet was sent by the sensing node to when it was delivered. As expected, the EAGP introduces a latency not present when using the other protocols, since the nodes schedule the forwarding of the packets with a timer value up to $T_{MAX}$, while in the other protocols the packets are forwarded *immediately*. Increasing $T_{MAX}$ increases the latency proportionally, but in this topology it grows slower than the $T_{MAX}$ itself as seen in Figure 6.2, where the nominal curve represents the configured $T_{MAX}$. After the latency, the tables shows the delivery efficiency of the protocols, which is the ratio of the created messages that reached the sink, and the total number of packets delivered.

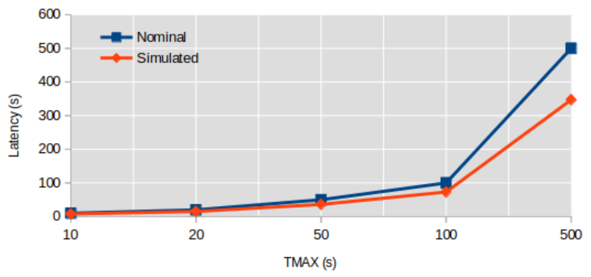


Figure 6.2.: EAGP Latency

The proposed protocol achieved an efficiency close to the pure gossip protocol, which is higher than the gossip with fan-out, but using much less energy. A protocol that uses less energy to deliver each packet is more energy efficient, which is represented in the table by the Joules per unique packet measurement. Figure 6.3 compares the energy efficiency of all protocols in relation to the efficiency in delivering packets. The more distant the two curves are the best, and the MCFA protocol has the best combination.

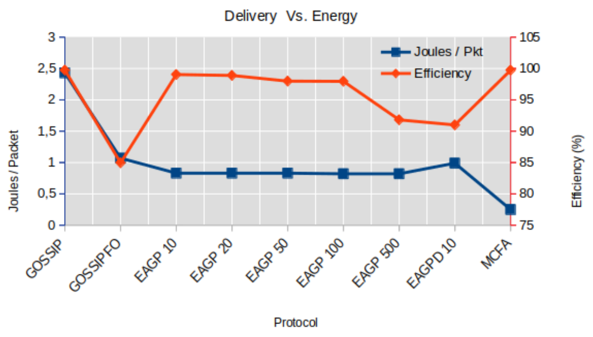


Figure 6.3.: Efficiency per Protocol

It is possible to see that both gossip protocols are in the worse position whilst the EAGP gets closer to the MCFA even tough it is a gossip protocol. In this particular scenario and topology there was no advantage in the EAGPD, since more packet exchange is added to the network, and it didn't increase the efficiency in delivering packets. It also worth notice that increasing the $T_{MAX}$ in the EAGP, reduces the energy consumption but also the efficiency, that is because since packets will stay longer scheduled, there is a bigger chance that the same message will be received as duplicate and dropped. So, besides using less energy, the EAGP with $T_{MAX}$ of 500 seconds is ultimately less efficient in delivering packets.

Previously the TTL was set to 5 because the results were being evaluated regarding the delivery to a sink which was in the middle of the topology. When the goal is to cover packet distribution to the whole network, the value of TTL has to be set to at least 10. It was therefore measured the average network coverage considering each node as the starting point as shown in Figure 6.4.

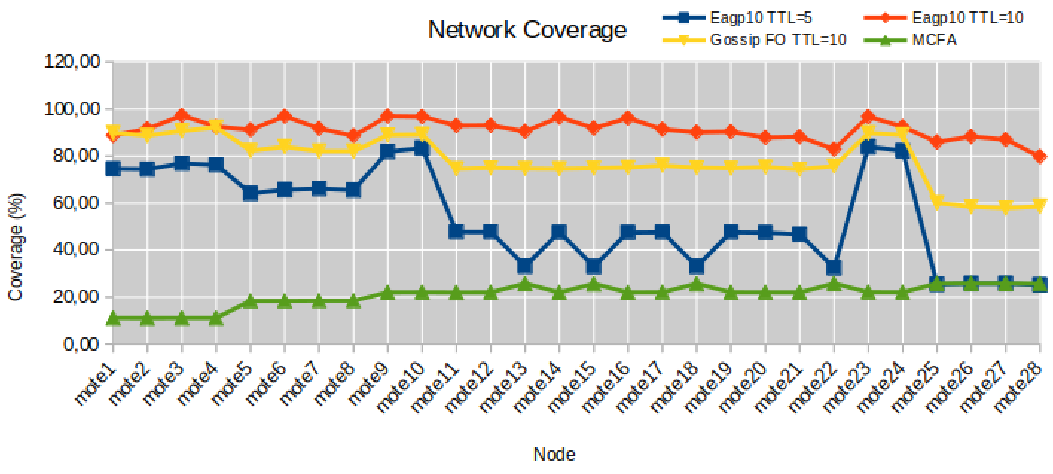


Figure 6.4.: Network Coverage by Node

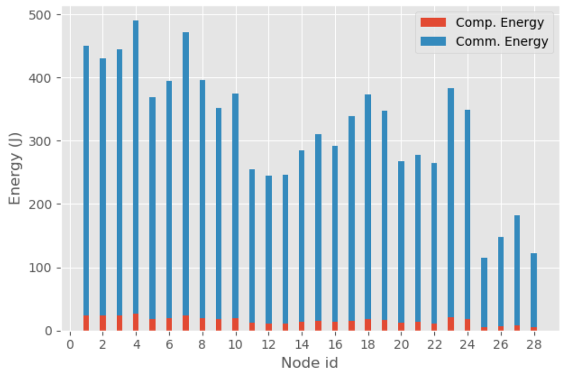
These results represent the average percentage of the nodes that were reached by each packet created by the nodes (excluding the sink node). Taking as reference the EAGP with

$T_{MAX}$ of 10 seconds, when increasing the TTL from 5 to 10 resulted in a network coverage better than the gossip with fan-out. For example, for the EAGP with $T_{MAX}$ of 10 seconds and TTL of 5, the packets created by *mote25* where able to reach only 20% of the nodes. Increasing the TTL to 10, the packets created by the same node were able to reach in average 85% of the nodes, but with the gossip with fan-out, and the same TTL configuration, the packets created by the same node were able to reach only 60% of the nodes. That means that in order to achieve the same network coverage as the EAGP, the gossip always need higher TTL value, and as a consequence consume even more energy. The MCFA, since it is not an epidemic protocol, yielded very poor results in this regards, and packets created in the border of the topology reached more nodes as expected, which is exactly the opposite of when a epidemic protocol is used. Increasing TTL increases the coverage, but also increases packet redundancy and consequently energy consumption. Fine tuning TTL, yields optimal results for energy efficiency and is related to the application needs.

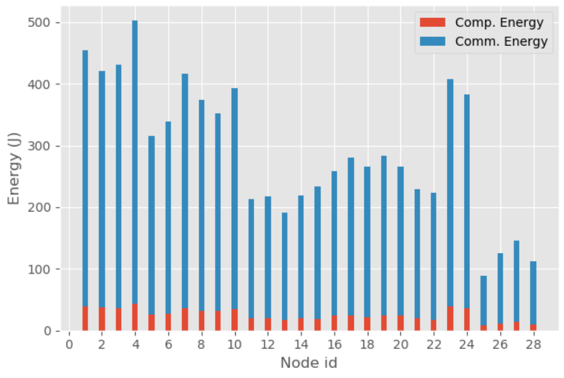
Designing a routing protocol that is too complex can increase the energy use for general computation. The gossip protocol with fan-out choose random neighbours to forward the packets, while in the proposed algorithm there is an additional computational footprint of maintaining a list of neighbours[4], updating the node's state, calculating $T_{NEXT}$ and maintaining a scheduler for the message forwarding. It is than expected an addition in energy not related to communication or sensing.

When plotting the energy profile of the nodes during the simulation, it is noticeable how similar the pure gossip with fan-out, in Figure 6.5(a), and the EAGP, in Figure 6.5(b), are regarding the overall energy consumption per node. Although the amount of energy used by each node is similar, the EAGP has a higher cost in computational energy than the two other protocols. The MCFA, in Figure 6.5(c), on the other hand has an overall lower cost and a profile in the shape of a slope, explained by the fact that nodes closer to the sink forward more packets than the others away from the sink, hence consume more energy. The MCFA uses more computational resources only during the topology discovery phase. In the running phase it only adds the node's weight to the packet and when a packet is received, compares to its own weight.

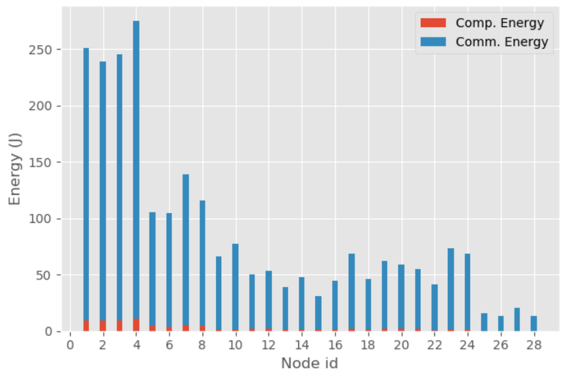
4 Also present in the gossip with fan-out.



(a) Gossip With fan-out of 3



(b) EAGP with $T_{MAX}$=10s



(c) MCFA

Figure 6.5.: Energy Profile

*Asymmetrical*

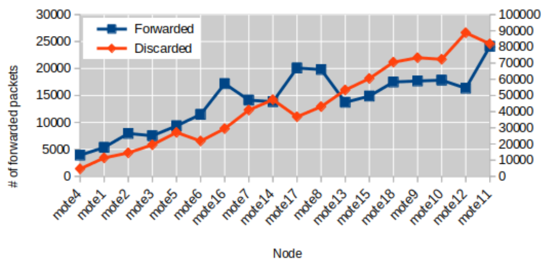
This topology, presented in Figure 5.2, has a hop radius of 9, so TTL was set to 9 in all simulations that use it.

This asymmetrical topology has a common path before the sink where all packets must flow before reaching it. It is expected the nodes in this path will create a bottleneck, and since the TTL is higher than the topology before, more entropy is expected when using the gossip protocols. The results for this topology are presented in Table 6.6.

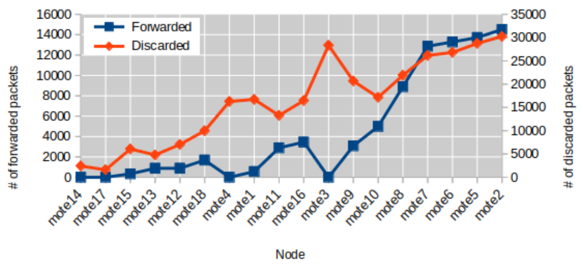
Table 6.6.: Asymmetrical Summary

| | GOSSIP | GSP. FO | EAGP 10 | EAGP 20 | EAGP 50 | EAGP 100 | EAGP 500 | EAGPD 10 | MCFA |
|---|---|---|---|---|---|---|---|---|---|
| Energy (J) | 771238.78 | 101212.79 | 17454.45 | 17716.52 | 17329.38 | 17359.56 | 15157.16 | 9297.59 | 5216.91 |
| Duplication | 139.82 | 11.68 | 2.33 | 2.29 | 1.91 | 2.03 | 2.06 | 1.76 | 4.35 |
| Max Hops | 8.99 | 8.65 | 7.22 | 7.24 | 6.95 | 7.07 | 6.66 | 6.44 | 6.06 |
| Min Hops | 4.40 | 4.44 | 5.08 | 5.12 | 5.07 | 5.13 | 4.70 | 4.76 | 5.06 |
| Latency | 0.01 | 0.01 | 21.53 | 42.88 | 105.00 | 211.61 | 920.37 | 23.28 | 0.01 |
| Efcy. (%) | 86.82 | 70.61 | 87.30 | 87.73 | 86.89 | 87.84 | 77.25 | 75.30 | 99.71 |
| Pck. Del. | 4771 | 4276 | 5431 | 5458 | 5406 | 5465 | 4806 | 4685 | 5954 |
| J / Pck. | 161.65 | 23.66 | 3.21 | 3.24 | 3.20 | 3.17 | 3.15 | 1.98 | 0.87 |

The results were similar to the previous topology, but this time the EAGP with the digest for lazy nodes had higher energy efficiency (lower Joules per delivered packet) than the base one. The reason is because more repeated packets are dropped from the buffer, and the expected negative side effect is lower efficiency in delivery. MCFA, for a static scenario, had again lower energy consumption since the packets flow only in one direction, towards the sink.



(a) EAGP with $T_{MAX}$=10s



(b) MCFA

Figure 6.6.: Packets Forwarded vs Discarded

When comparing the number of packets forwarded and discarded per node in EAGP in Figure 6.6(a) and MCFA in Figure 6.6(b), the latter, as expected, had the nodes closer to the sink more active. The opposite is observed in the EAGP, where nodes closer to the sink, that is located in the less dense area of the network, handle less packets. It is important to recall that in a gossip network, the packets flow in a chaotic manner, so there is no way to induce the direction towards the sink, therefore most of the packets will *live* bouncing in the most dense area of the network. In a topology like this, it is expected that when using a directional protocol like MCFA, the nodes closer to the sink will work more, and as a consequence use more energy and run out of battery before. If nodes closer to the sink run out of battery earlier, the whole network will die, since no packets will ever reach the sink again.

Previously, the TTL was set to 9 because the results were being evaluated regarding the delivery to a sink, but for the network coverage tests it was set to 18. It was therefore measured the average network coverage considering each node as the starting point as shown in Figure 6.7.

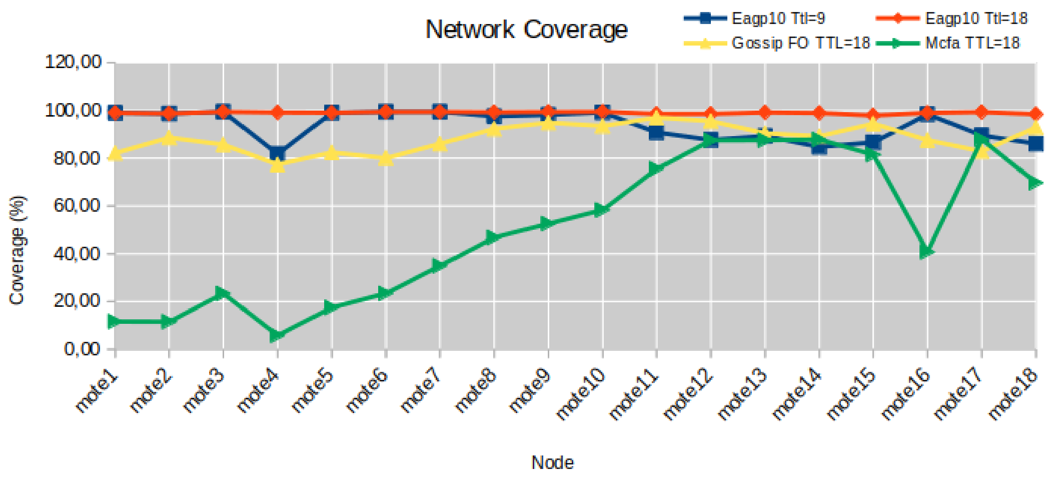


Figure 6.7.: Network Coverage by Node

In this case, the coverage using the EAGP even with a very conservative TTL, had good results, and increasing the TTL brought the results closer to 100%. The results for EAGP with TTL of 9 were even better than the **gossip with fan-out** configured with TTL of 18.

### *Random*

This topology, presented in Figure 5.3, has a hop radius of 11, so TTL was set to 11 in all simulations that use it. In this topology there is a bottleneck in one node close to the sink that connects the whole left side of the network.

In this topology the results presented on Table 6.7 were similar to the asymmetric and the EAGP achieved a high delivery efficiency, but lower energy efficiency than the digest variation. The average latency also in this topology gets closer to the $T_{MAX}$ value defined. Regarding the delivery efficiency however, the Energy Aware Gossip Protocol Digest (EAGPD) protocol achieved a result similar to the EAGP while having an energy efficiency 46% better. The results for packet duplication were also in EAGPD close to the MCFA which is good considering it is an epidemic protocol.

Table 6.7.: Random Summary

| | GOSSIP | GSP. FO | EAGP 10 | EAGP 20 | EAGP 50 | EAGP 100 | EAGP 500 | EAGPD 10 | MCFA |
|---|---|---|---|---|---|---|---|---|---|
| Energy (J) | 1437968 | 186740.43 | 25209.64 | 25071.99 | 25108.65 | 24506.28 | 22015.26 | 12980.53 | 3971.89 |
| Duplication | 308.63 | 34.27 | 5.00 | 4.86 | 4.90 | 4.78 | 4.52 | 2.48 | 2.21 |
| Max Hops | 9.36 | 9.92 | 9.18 | 9.08 | 9.08 | 9.03 | 8.50 | 7.18 | 4.66 |
| Min Hops | 3.24 | 4.47 | 3.64 | 3.65 | 3.64 | 3.63 | 3.55 | 3.67 | 3.66 |
| Latency | 0.05 | 0.01 | 12.46 | 24.56 | 60.88 | 120.49 | 573.53 | 15.05 | 0.01 |
| Efcy. (%) | 70.66 | 96.88 | 94.82 | 94.83 | 94.43 | 94.03 | 89.88 | 90.55 | 99.72 |
| Pck. Del. | 6414 | 10084 | 9869 | 9870 | 9829 | 9787 | 9355 | 9425 | 10103 |
| J / Pck. | 224.19 | 18.51 | 2.55 | 2.54 | 2.55 | 2.50 | 2.35 | 1.37 | 0.39 |

The random topology, as the asymmetric topology, benefited more the EAGPD than the EAGP when the main goal is the energy efficiency. This topology is denser than the previous ones, and for that is expected that the gossip without fan-out will struggle more deliver the packets due to the high entropy in the network. Packets are excessively exchanged in chaotic directions, and using fan-out to reduce the amount of exchanged packets results in less packet duplication and consequently higher delivery rate as observed in the results.

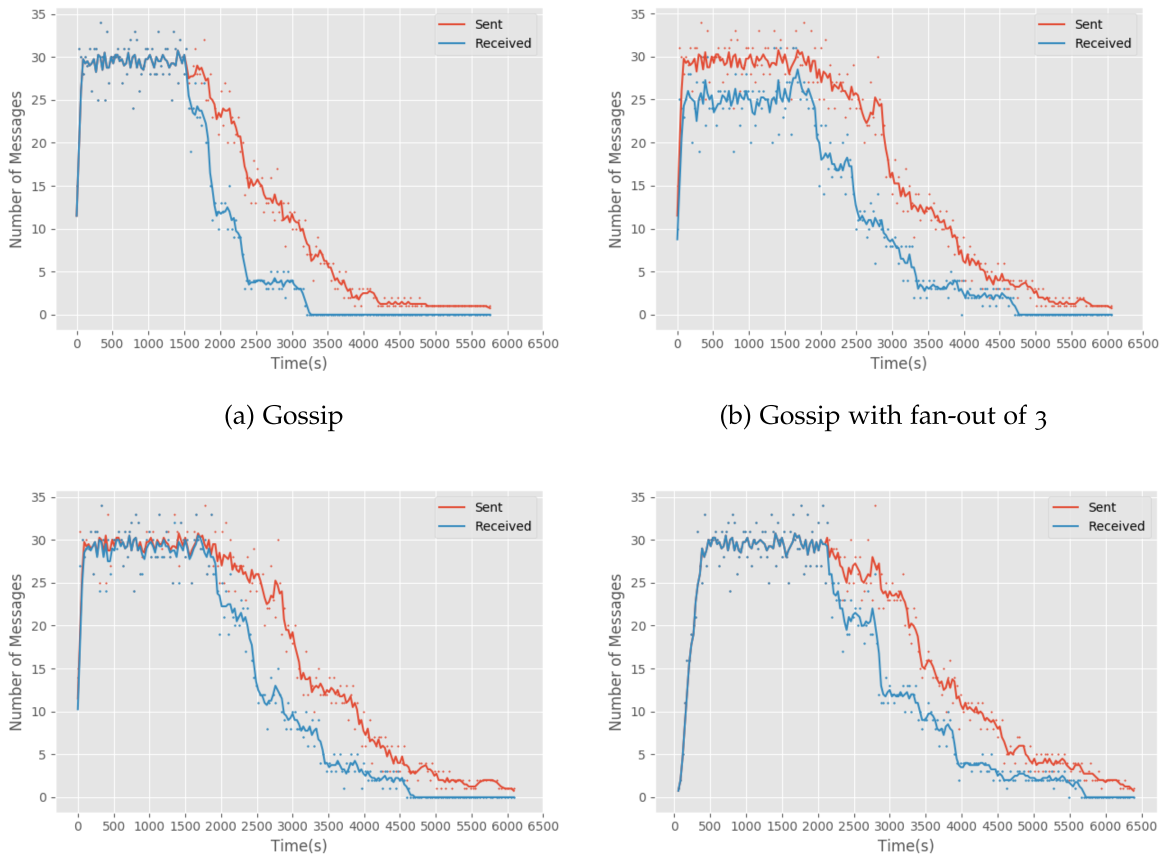
The excess in packet exchange in a denser network work can also penalise the EAGP due to the cancellation of packet scheduling when repeated packets are received, and that is partially solved by the EAGPD packet digest. This is shown in the results on Table 6.7, since the EAGPD in this topology resulted in half packet duplication and half energy use while keeping the delivery efficiency close to EAGP. A possible solution for the duplication in EAGP would be to implement an optional alternative of introducing an energetic fan-out for denser networks, where the nodes would choose from it's neighbours list nodes with more energy to receive the packets, the ones with less energy would just discard the packets. This would not only reduce the overall energy use, but also help in balancing the nodes' energy.

### 6.2.2 *End of Life Scenario*

In the end of life scenario the nodes are configured to have a very small battery with 50mAh of full capacity. The initial state of each node's battery level is the same described before for the steady state scenario. The simulation runs until the network is dead, which is when the sink stops receiving packets.

#### *Symmetrical*

In Figure 6.8 results for different protocols are presented. The pure gossip protocol had shortest lifetime as shown in Figure 6.8(a), in which the network totally collapsed around 3250s, and adding the fan-out as shown in Figure 6.8(b), increased the lifetime significantly allowing the network to run until around 4600s. The EAGP protocol with $T_{MAX}$ set to 10 seconds shown in Figure 6.8(c), achieved a lifetime similar to the gossip with fan-out without being penalised in delivery rate to the sink. For the MCFA, represented in Figure 6.8(d), it is noticeable that the received messages form a sudden drops in steps, that is because when certain nodes stop working due to battery depletion, a whole path is blocked, while with the epidemic protocols packets can find other ways around the dead node.



(a) Gossip

(b) Gossip with fan-out of 3

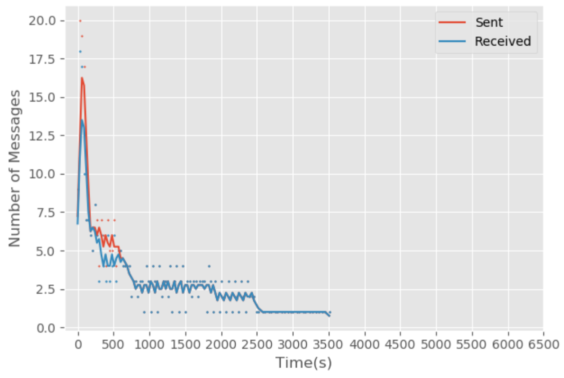
(c) EAGP with $T_{MAX}$=10s

(d) MCFA

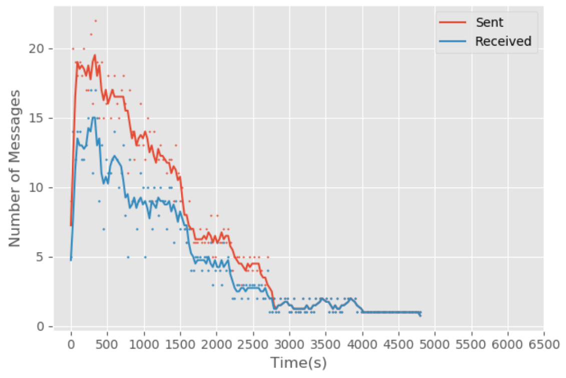
Figure 6.8.: Network Longevity - Symmetrical

*Asymmetrical*

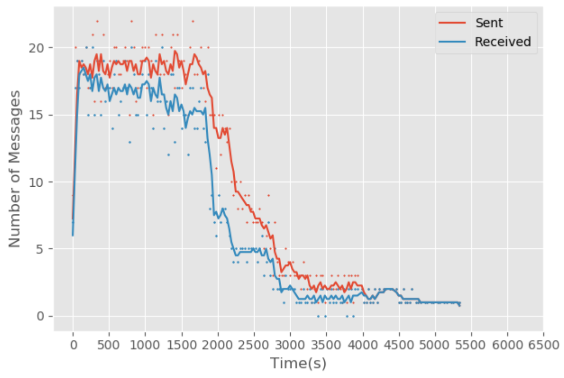
The lifetime representation of the nodes configured with the asymmetric topology is shown in Figure 6.9. Having TTL set to 9 increases the amount of exchanged messaged in the network when compared to the symmetric topology. For that reason, when configured with the same initial battery capacity of 50mAh, the gossip protocol in Figure 6.9(a) has a much shorter lifetime than when compared with the symmetric topology. However, the EAGP in Figure 6.9(c) protocol does not suffer equally due to ability of cancelling duplicated messages not present in the purely gossip protocol. With the MCFA in Figure 6.9(d) the loss of a key node had a big impact in the network as seen after 2500s of simulation.



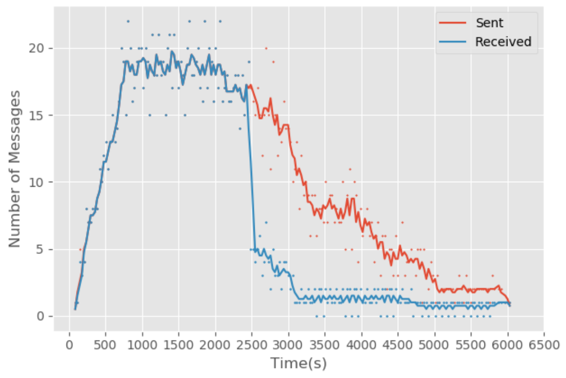
(a) Gossip



(b) Gossip with fan-out of 3



(c) EAGP with $T_{MAX}$=10s



(d) MCFA

Figure 6.9.: Network Longevity - Asymmetrical

### 6.2.3 *Mobility*

For the mobility scenario, the nodes are set to change position every second, ultimately creating always a random topology. The initial state was the symmetrical topology with TTL set to 5. Since the radius of 90m is too short for mobility, it was increased to 120m. Figure 6.10 represents a snapshot during the execution of the mobility scenario and symmetric topology. The nodes represented in green are the nodes that at a specific time were in eager behaviour, while the blue ones where in lazy behaviour.

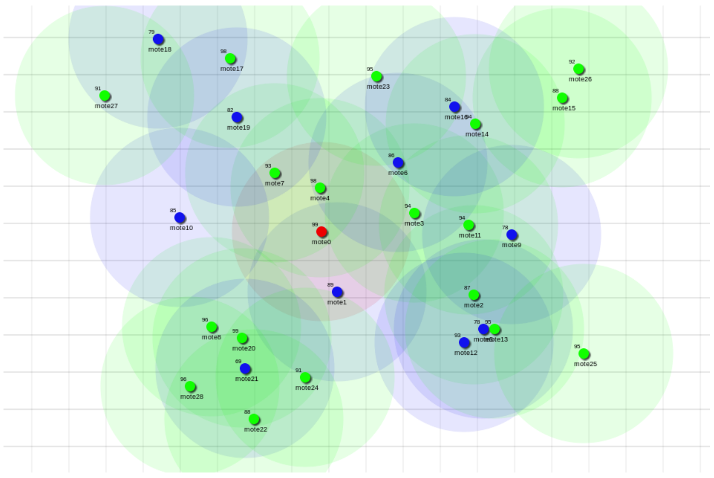
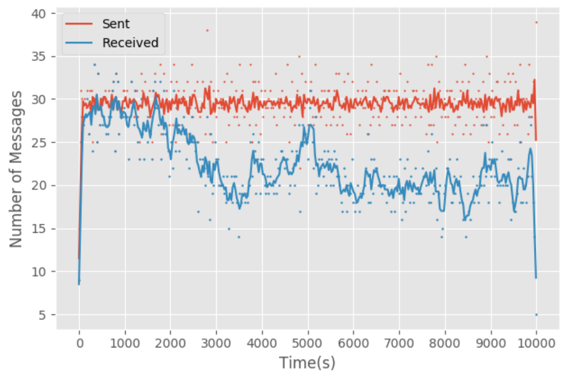


Figure 6.10.: Topology After Mobility Starts

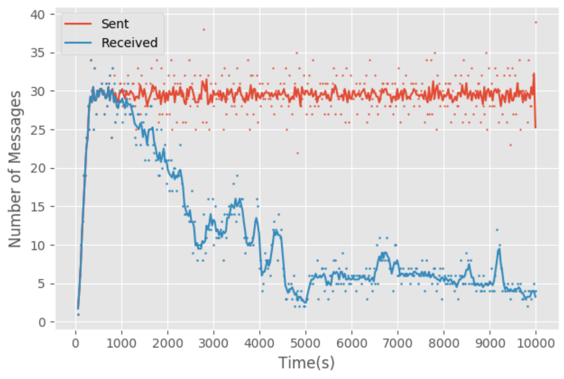
Table 6.8.: Mobility Summary

| | GOSSIP | GSP. FO | EAGP 10 | EAGP 20 | EAGP 50 | EAGP 100 | EAGPD 10 | EAGPD 20 | MCFA |
|---|---|---|---|---|---|---|---|---|---|
| Energy (J) | 166525.10 | 23856.12 | 13592.17 | 15440.83 | 13460.21 | 12230.26 | 10850.25 | 10499.09 | 2120.80 |
| Duplication | 102.81 | 14.94 | 7.01 | 7.18 | 7.41 | 6.02 | 4.72 | 4.87 | 2.58 |
| Max Hops | 4.87 | 4.91 | 4.78 | 4.84 | 4.80 | 4.65 | 5.25 | 5.31 | 3.01 |
| Min Hops | 2.36 | 2.38 | 2.45 | 2.49 | 2.50 | 2.47 | 2.60 | 2.64 | 1.79 |
| Latency | 0.01 | 0.01 | 6.23 | 14.14 | 36.56 | 74.84 | 12.31 | 23.83 | 0.01 |
| Efcy. (%) | 56.40 | 65.80 | 67.89 | 74.76 | 60.81 | 63.38 | 67.93 | 70.83 | 46.50 |
| Pck. Del. | 5558 | 6484 | 6690 | 7368 | 5993 | 6246 | 6694 | 6980 | 4506 |
| J / Pck. | 29.96 | 3.67 | 2.03 | 2.09 | 2.24 | 1.95 | 1.62 | 1.50 | 0.47 |

The movement of each node during the mobility scenario is completely random, so one simulation is never equal or similar to another. The results presented are an average of five executions of each protocol, and even tough they do not provide an accurate forecast of how each protocol will behave for each possible random situation, it can provide a performance estimation. Most of the results were as expected, with the MCFA losing its previous outstanding delivery performance as shown in Figure 6.11(b), due to the fact that it doesn't recalculate the optimal path during execution like the EAGP that is periodically updating the neighbours list. It might be even expected that given enough time, the delivery rate of the MCFA will tend to zero. It could however be implemented a mobility version of the MCFA that periodically rebuilds the topology, but even building the topology with the nodes moving could yield bad results, because during the topology rebuilding stage no packet could be delivered, and when the process would be finished the topology could be already different. However, for a scenario where the node's position is static but the sink is alternated, it could be implemented that every sink rotation, the nodes would restart the topology discovery. The EAGPD on the other hand, made better use of energy besides having similar delivery rate to the EAGP, as shown in Figure 6.11(a). The EAGP had much more repeated messages increasing the energy consumption, and resulting as a consequence, in lower energy efficiency.



(a) EAGPD $T_{MAX}$=20s



(b) MCFA

Figure 6.11.: Delivery Rate

Figures 6.11(b) and 6.11(a) show typical executions illustrating the difference between the amount of packets sent and received during a mobility scenario.

## 6.3 SUMMARY

This section presented the simulation results for different protocols, scenarios and topologies. On a static scenario, where epidemic packet distribution is not required to ensure high delivery rate, the MCFA protocol is more efficient in both delivery and energy wise. When epidemic distribution is required, such as in situations with mobile nodes, or when the sink node changes throughout the network lifetime, or when the distribution of the packets should aim in covering the whole network and not only towards a sink, the EAGP protocol performed better than a regular gossip with fan-out and the MCFA. Moreover, when mobility is required, the EAGP protocol and its digest variation achieved an overall better result, which was expected considering that the MCFA has a topology discovery step that is not repeated when the topology changes due to mobility of nodes.

# 7 CONCLUSIONS

This chapter presents the conclusions after the analysis of the results and possible future work to continue with the investigation in this subject.

## 7.1 CONCLUSIONS

This work had the primary goal of designing a routing protocol for WSN that is energy efficient and compatible with resource constrained devices. Designing a routing protocol for WSN involve different factors, and a crucial one is to be able to run simulations. Finding a simulator that would cover all aspects have shown to be a difficult task, specially if the code running in the simulator is to be reused for other simulators or reused for implementations in hardware. On the beginning of this work, the Mininet-Wifi simulator was being used, but the machine in which the tests were being performed could not handle too many nodes simultaneously and the kernel module used for the simulated network interface would crash. After running the same tests with the CORE simulator, the results were similar but the same machine could handle much more nodes, and all the results from this work are from simulations.

The verification and analysis performed with the EAGP protocol yielded promising results, and had better performance in almost every aspect when compared to a pure gossip protocol with fan-out. It achieved in average 10% less packet duplication while achieving in average 16% higher delivery efficiency. For network coverage, the protocol could achieve for a symmetrical topology, a coverage around 10% higher. In the asymmetric topology, the energy efficiency of the EAGP was around 7 times higher than the pure gossip with fan-out. When compared to a purely gossip protocol the disadvantage comes for time critical applications, since there is an average latency proportional to the chosen $T_{MAX}$, while in a purely gossip protocol the latency is typically low. The fact that some packets will ultimately follow different paths, with some following a fast path while others a slow path, could create an issue when segmented packets are being transmitted. One technique that was created for testing but is not documented in this dissertation and could help in the application with time critical constrains was the creation of a special *fast-track* PDU. These special packets would be forwarded immediately, bypassing the $T_{NEXT}$ calculation and the ignoring the eager/lazy state of the node.

The proposed protocol has also been shown as more efficient than a common gossip protocol, requiring less energy from the network for delivering each packet. Meaning that, in average, the whole network will always consume less and, as a consequence, have longer lifetime. Moreover, this advantage comes without the cost of a trade-off in delivery efficiency, since this aspect is not penalised. It is also noticeable that when the topology is unbalanced and there are bottlenecks, the digest variation of the proposed protocol demonstrates better energy efficiency, and that must be taken into consideration depending on the application.

When comparing with non-gossip protocols, like MCFA, the results have shown that it is more efficient in any aspect to establish a route for the packets, since there is no epidemic dissemination with exponential growth. However, it is not always achievable for all applications, and specially not applicable when nodes are moving and in which the topology needs to be updated constantly. The waste of energy in purely gossip protocols happens because packets can follow ways totally contrary of what is expected, but this characteristic is desired when the application requires packet delivering to all nodes.

There are many applications that require that a network is capable of delivering packets to all nodes, even thought, the architecture is based in a *nodes to sink* or *many to one* topology. That epidemic distribution could be used for firmware upgrade for instance, or to update settings. Another application is where nodes work as state machines and their state or mode of operation need to be changed externally. On a best case scenario point of view, nodes that have enough resources could have the ability to change protocols during runtime. So when there is an application need of epidemic dissemination, the nodes would change to a gossip protocol, and when there is the need to deliver messages only towards a sink, change to a directional protocol. In situations where resources in the nodes are extremely constrained and this is not feasible, the application needs to be carefully studied so the correct protocol is chosen.

It is possible than to conclude that all the goals of this work were achieved. The protocol, when compared to a purely gossip implementation, has presented an energy efficiency in static scenarios around one order of magnitude higher, without adding too much computational and complexity burden to the nodes. Another outstanding characteristic is the ability of rebuilding the topology by finding a node's neighbours, choosing the eager/lazy mode and calculating $T_{NEXT}$ without the need of extra PDUS or states. For that, only the addition of the battery level to the packet at each hop was needed, and for the zero to one hundred representation only 7 bits are required.

## 7.2 Contributions

Besides the contribution of the implementation and testing of the proposed protocol, this work also contributed with the creation of tools to help researchers and developers to create scenarios for testing WSN protocols and applications with CORE and Mininet-Wifi simulators. With the Python code base created, it allowed the implementation of other routing protocols for testing, to be very fast and easy, requiring only small changes in the code.

## 7.3 Prospect for future work

There is a need for the development of a simulator that can accurately predict energy use in each aspect of the application and is not limited for a specific hardware platform. The problem of heterogeneity could be easily solved if every manufacturer would create emulators for their own products that could calculate energy consumption so end users could compile their code for specific platforms and analyse with each emulator. By using operating systems like RiotOS or FreeRTOS, the users could even reuse the same code for different platforms and compare the results before testing in real hardware. To consolidate the results, implementing the protocol in real hardware would allow real world case studies for the protocol.

For the EAGP protocol, more tests could be made with the digest and the fast-track features, in order to verify the possibility of achieving higher packet delivery and analysing the applicability to time constrained applications. Also, there is a need to analyse the behaviour when packet segmentation and reordering are required.

Even though the consumption of the communication radios are still high, the nodes spend most of the time doing other tasks than transmitting or receiving data. So, implementing ways to reduce the energy consumption of the nodes when they are not communicating is crucial. Most of the hardware available in the market today have ultra low power modes available that enable the nodes to run for years without changing batteries, but this mode usually cannot be used if the nodes have to stay awake waiting for another node's transmission. Finding optimal ways to allow the nodes enter these *deep sleep* modes will help to increase the lifetime of the networks even further, allowing the design of WSN which require very low maintenance.

# BIBLIOGRAPHY

802.11, I.
2016. Ieee 802.11-2016 - ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. https://standards.ieee.org/content/ieee-standards/en/standard/802_11-2016.html. Accessed: 2019-09-01.

802.15.4, I.
2015. Ieee 802.15.4-2015 - ieee standard for low-rate wireless networks. https://standards.ieee.org/standard/802_15_4-2015.html. Accessed: 2019-09-01.

Ahrenholz, J., T. Goff, and B. Adamson
2011. Integration of the CORE and EMANE network emulators. In Proceedings - IEEE Military Communications Conference MILCOM.

Al-Karaki, J. N. and A. E. Kamal
2004. Routing techniques in wireless sensor networks: a survey. IEEE Wireless Communications, 11(6):6–28.

Al-Sarawi, S., M. Anbar, K. Alieyan, and M. Alzubaidi
2017. Internet of Things (IoT) communication protocols: Review. In 2017 8th International Conference on Information Technology (ICIT).

Ali, N. A., M. Drieberg, and P. Sebastian
2011. Deployment of MICAz mote for wireless sensor network applications. In ICCAIE 2011 - 2011 IEEE Conference on Computer Applications and Industrial Electronics.

Anastasi, G., M. Conti, M. D. Francesco, and A. Passarella
2009. Energy conservation in wireless sensor networks: A survey. Ad Hoc Networks, 7(3):537 – 568.

Anjali, A. Garg, and Suhali
2016. Distance Adaptive Threshold Sensitive Energy Efficient Sensor Network (DAPTEEN) protocol in WSN. In Proceedings of 2015 International Conference on Signal Processing, Computing and Control, ISPCC 2015.

Baccelli, E., O. Hahm, M. Günes, M. Wählisch, and T. Schmidt
2013. Riot os: Towards an os for the internet of things. Proceedings - IEEE INFOCOM.

Bhattacharya, H. S. H. L. V. K. D. M. A. &. P.
2011. Wireless Sensor Network Simulators A Survey and Comparisons. International Journal of Computer Networks.

BluetoothCore
2019. Bluetooth core specifications. https://www.bluetooth.com/specifications/bluetooth-core-specification/. Accessed: 2019-09-01.

Bounceur, A., L. Clavier, P. Combeau, O. Marc, R. Vauzelle, A. Masserann, J. Soler, R. Euler, T. Alwajeeh, V. Devendra, U. Noreen, E. Soret, and M. Lounis
2018. CupCarbon: A new platform for the design, simulation and 2D/3D visualization of radio propagation and interferences in IoT networks. In CCNC 2018 - 2018 15th IEEE Annual Consumer Communications and Networking Conference.

Braginsky, D. and D. Estrin
2002. Rumor routing algorthim for sensor networks. Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, Pp. 22–31.

Carvalho, N., J. Pereira, R. Oliveira, and L. Rodrigues
2007. Emergent structure in unstructured epidemic multicast. In Proceedings of the International Conference on Dependable Systems and Networks.

DESHPANDE, A., C. GUESTRIN, S. MADDEN, J. HELLERSTEIN, and W. HONG
2004. Model-Driven Data Acquisition in Sensor Networks. In Proceedings 2004 VLDB Conference. -.

Dunkels, A.
2011. The ContikiMAC Radio Duty Cycling Protocol. SICS Technical Report T2011:13 , ISSN 1100-3154.

Fontes, R. D. R., M. Mahfoudi, W. Dabbous, T. Turletti, and C. Rothenberg
2017. How Far Can We Go? Towards Realistic Software-Defined Wireless Networking Experiments. Computer Journal.

Fontes, R. R., S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg
2015. Mininet-wifi: Emulating software-defined wireless networks. 2015 11th International Conference on Network and Service Management (CNSM), Pp. 384–389.

Gao, R., H. Zhou, and G. Su
2011. Structure of wireless sensors network based on tinyos. -, Pp. 1–4.

Hao, H., D. Silvestre, and C. Silvestre
2018. Source localization and network topology discovery in infection networks. 2018

37th Chinese Control Conference (CCC), Control Conference (CCC), 2018 37th Chinese, P. 1915.

Heinzelman, W. R., A. Chandrakasan, and H. Balakrishnan
2000. Energy-efficient communication protocol for wireless microsensor networks. -, Pp. 10 pp. vol.2–.

Heinzelman, W. R., J. Kulik, and H. Balakrishnan
1999. Adaptive protocols for information dissemination in wireless sensor networks. In Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom '99, Pp. 174–185, New York, NY, USA. ACM.

Helkey, J., L. Holder, and B. Shirazi
2016. Comparison of simulators for assessing the ability to sustain wireless sensor networks using dynamic network reconfiguration. Sustainable Computing: Informatics and Systems.

Intanagonwiwat, C., R. Govindan, and D. Estrin
2000. Directed diffusion: A scalable and robust communication paradigm for sensor networks. Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, Pp. 56–67.

Kodali, R. K. and N. V. Sarma
2017. Simulation analysis of multi-dimensional WSNs using NS-3. In IEEE Region 10 Annual International Conference, Proceedings/TENCON.

Kumar, A., H. Shwe, K. Juan Wong, and P. Chong
2017. Location-based routing protocols for wireless sensor networks: A survey. Wireless Sensor Network, 9:25–72.

Kuosmanen, P.
2000. Classification of Ad Hoc Routing Protocols. Structure.

Lazarescu, M. T.
2013. Design of a WSN platform for long-term environmental monitoring for IoT applications. IEEE Journal on Emerging and Selected Topics in Circuits and Systems.

Leitao, J. Pereira, J. R. L.
2007. Epidemic broadcast trees. 2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007), Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on, P. 301.

Lindsey, S. and C. S. Raghavendra
2002. Pegasis: Power-efficient gathering in sensor information systems. -, 3:3–3.

Lounis, M., A. Bounceur, R. Euler, and B. Pottier
2017. Estimation of energy consumption through parallel computing in wireless sensor networks.

Malewski, M., D. M. Cowell, and S. Freear
2018. Review of battery powered embedded systems design for mission-critical low-power applications. International Journal of Electronics.

Manjeshwar, A. and D. P. Agrawal
2001. TEEN: A routing protocol for enhanced efficiency in wireless sensor networks. In Proceedings - 15th International Parallel and Distributed Processing Symposium, IPDPS 2001.

Manjeshwar, A. and D. P. Agrawal
2002. APTEEN: A hybrid protocol for efficient routing and comprehensive information retrieval in wireless. In Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2002.

Mehdi, K., M. Lounis, A. Bounceur, and T. Kechadi
2014. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques.

Niyazi, Lama B. Chaaban, A. D. H. A.-N. T. Y. A. M.-S.
2017. Energy-aware sensor networks via sensor selection and power allocation. 2017 IEEE 86th Vehicular Technology Conference (VTC-Fall), Vehicular Technology Conference (VTC-Fall), 2017 IEEE 86th, P. 1.

Perla, E., A. Ó. Catháin, R. S. Carbajo, M. Huggard, and C. Mc Goldrick
2008. PowerTOSSIM z: Realistic Energy Modelling for Wireless Sensor Network Environments. -.

Prayati, A., C. Antonopoulos, T. Stoyanova, C. Koulamas, and G. Papadopoulos
2010. A modeling approach on the telosb wsn platform power consumption. Journal of Systems and Software, 83(8):1355 – 1363. Performance Evaluation and Optimization of Ubiquitous Computing and Networked Systems.

Pundir, S., A. Bhalla, A. Bakshi, A. K. Singh, and D. P. Singh
2018. MODTEEN: Modified threshold sensitive energy efficient network in Wireless Sensor Network. In Proceedings - 2017 3rd International Conference on Advances in Computing, Communication and Automation (Fall), ICACCA 2017.

Rasool, I. U., Y. B. Zikria, A. Musaddiq, F. Amin, and S. W. Kim
2017. RIOT-OS: Firmware for futuristic internet of things. Far East Journal of Electronics and Communications.

Rethfeldt, M., H. Raddatz, B. Beichler, B. Konieczek, D. Timmermann, C. Haubelt, and P. Danielis
2016. ViPMesh: A virtual prototyping framework for IEEE 802.11s wireless mesh networks. In International Conference on Wireless and Mobile Computing, Networking and Communications.

Rodoplu, V. and T. H. Meng
1999. Minimum energy mobile wireless networks. IEEE Journal on Selected Areas in Communications.

Roussel, K., Y.-Q. Song, and O. Zendra
2015. RIOT OS Paves the Way for Implementation of High-Performance MAC Protocols. -.

Roussel, K., Y.-Q. Song, and O. Zendra
2016. Using Cooja for WSN Simulations: Some New Uses and Limits. EWSN 2016 — NextMote workshop.

Schaefer, F. M., T. Gro??, and R. Kays
2013. Energy consumption of 6LoWPAN and Zigbee in home automation networks. In IFIP Wireless Days, Pp. 13–15.

Schurgers, C. and M. B. Srivastava
2001. Energy efficient routing in wireless sensor networks. 2001 MILCOM Proceedings Communications for Network-Centric Operations: Creating the Information Force (Cat. No.01CH37277), 1:357–361 vol.1.

Shaikh, F. K. and S. Zeadally
2016. Energy harvesting in wireless sensor networks: A comprehensive review. -.

Shnayder, V., M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh
2005. Simulating the power consumption of large-scale sensor network applications. -.

Silva, J. M. C., K. A. Bispo, P. Carvalho, and S. R. Lima
2017. LiteSense: An adaptive sensing scheme for WSNs. Proceedings - IEEE Symposium on Computers and Communications, Pp. 1209–1212.

Silvano, G., I. Silva, L. Oliveira, M. Pinheiro, and B. Ferreira
2017. A hybrid architecture for experimentation in wireless sensor networks. In Brazilian Symposium on Computing System Engineering, SBESC.

Systems, E.
2019. ESP8266EX Datasheet. Espressif Systems. Rev. 6.2.

Thaker, T.
2016. Esp8266 based implementation of wireless sensor network with linux based web-server. 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Pp. 1–5.

Weingartner, E., H. vom Lehn, and K. Wehrle
2009. A performance comparison of recent network simulators. 2009 IEEE International Conference on Communications, Pp. 1–5.

Wu, H., S. Nabar, and R. Poovendran
2011. An energy framework for the network simulator 3 (ns-3). Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, Pp. 222–230.

Ye, F., A. Chen, S. Lu, and L. Zhang
2001. A scalable solution to minimum cost forwarding in large sensor networks. -, Pp. 304–309.

Yitayal, E., J. M. Pierson, and D. Ejigu
2015. Towards green networking: Gossip based balanced battery usage routing protocol to minimize energy consumption of MANETs. In IEEE AFRICON Conference.

Zhou, H.-Y., D.-Y. Luo, Y. Gao, and D.-C. Zuo
2011. Modeling of Node Energy Consumption for Wireless Sensor Networks. Wireless Sensor Network.

# A

# SIMULATION SOFTWARE

## A.1 SIMULATION SOFTWARE

The simulation, as show in Figure A.1, starts when a scenario python script is run. The script for Mininet or CORE is very similar, and the other scripts for the rest API, main, and node are the same independently if Mininet or CORE is being used.

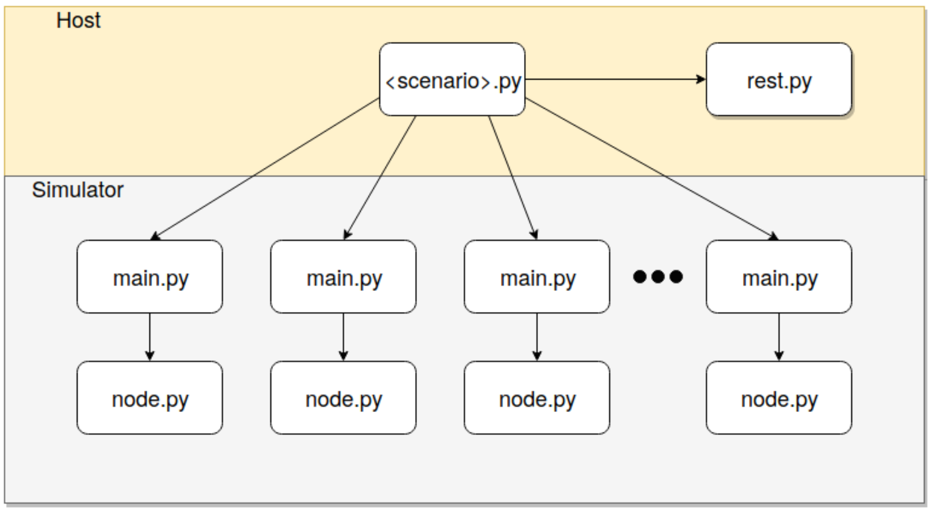


Figure A.1.: Simulation Overview

The scenario script, after starting the simulator, opens a terminal in each node and starts the main.py script. This script instantiates the Node class and starts a main background scheduler the governs the whole simulation. Figure A.2 shows the overview of the main.py script, while Figure A.3 shows the overview of the node.py class.

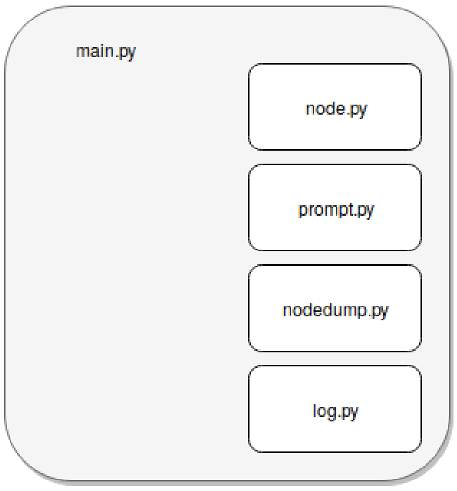


Figure A.2.: main.py Overview

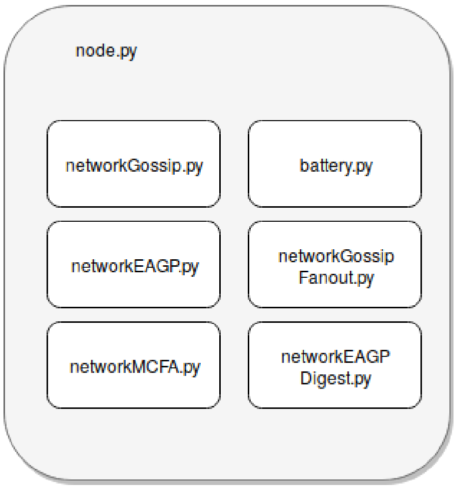


Figure A.3.: node.py Overview

### A.1.1 *main.py*

The classes instantiated by main.py are described as follow:

- node.py - Main class related to the node itself
- prompt.py - Class that enables the user to have a command prompt
- nodedump.py - Class that dumps information about the nodes during the simulation that can be used by the rest API
- log.py - Logging and tracing functions

### A.1.2 *node.py*

The classes instantiated by main.py are described as follow:

- battery.py - Class the control the energy model
- networkGossip.py - Class that implements a pure gossip routing protocol
- networkGossipfan-out.py - Class that implements a pure gossip routing protocol with the option of using fan-out
- networkEAGP.py - Class that implements the proposed protocol
- networkEAGPDigest.py - Class that implements the proposed protocol with the digest modification

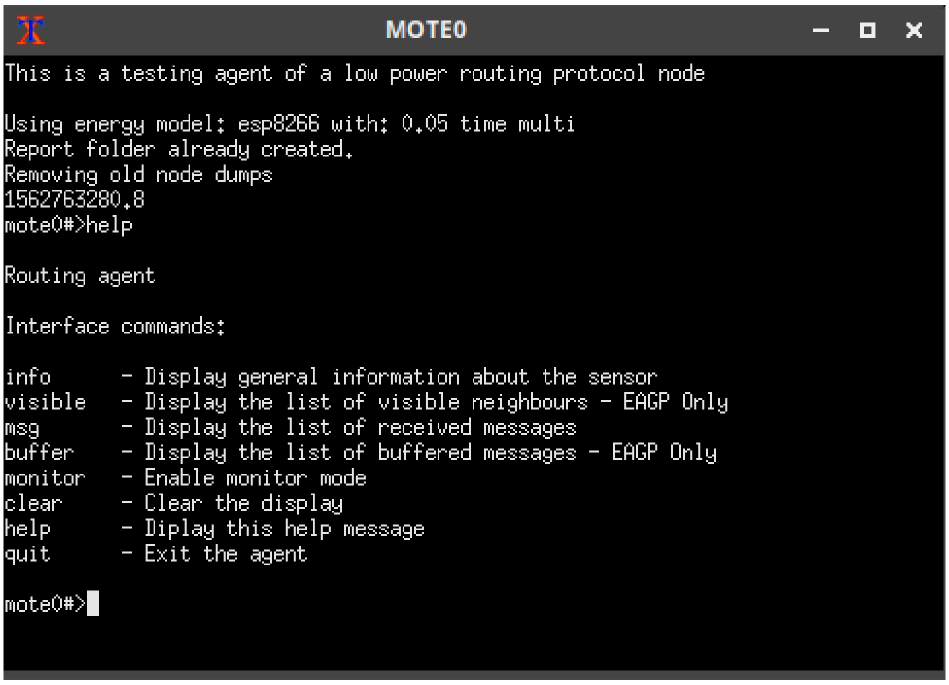
- networkMCFA.py - Class that implements the MCFA routing protocol

As shown before, some requirements were set for the simulation software:

- SR01 - The user must be able to run the simulation in real time or accelerated
- SR02 - The user must be able to interact with each node to monitor activities and consult status and parameters during the execution of the simulation via command prompt
- SR03 - The simulation must have real-time properties available via a RESTapi end-points
- SR04 - The simulation results must be stored in CSV files
- SR05 - The user must be able to see the topology in real-time via a graphical interface

To comply with requirement SR-01 the user can send via command line argument a time multiplier value that is used in the scheduler to change the timer configuration. The idea is that for every second passed in the simulation, the corresponding time in real world will be 1 second times the value of the multiplier. So for instance if the sleeping time is set to 240 seconds, and the multiplier is set to 0.05, the scheduler will set the timer for the main cycle to 12 seconds. The same is valid to all temporal tasks, as the tasks triggered by an interrupt are note changed.

For being able to interact with each node during execution and comply with SR02, a prompt command was implemented to run in each node with as illustrated in Figure A.4. For coding organisation the prompt was implemented as a separated class.



```
This is a testing agent of a low power routing protocol node

Using energy model: esp8266 with: 0.05 time multi
Report folder already created.
Removing old node dumps
1562763280.8
mote0#>help

Routing agent

Interface commands:

info       - Display general information about the sensor
visible    - Display the list of visible neighbours - EAGP Only
msg        - Display the list of received messages
buffer     - Display the list of buffered messages - EAGP Only
monitor    - Enable monitor mode
clear      - Clear the display
help       - Diplay this help message
quit       - Exit the agent

mote0#>
```

Figure A.4.: Command Prompt for an Agent

In the command prompt the following commands are available:

- help - Print help message
- info - Display general information about the sensor
  - Node name
  - Node role (mote or sink)
  - Sleep time configured
  - Simulation elapsed time in virtual simulation time
  - Simulation elapsed time in real world time
  - Node position
- visible - Display a table with the current visible neighbours
  - Neighbour IP address
  - Last time it was seen in simulated time
  - Neighbour battery level in 0-100%
- monitor - Turns verbosity on displaying the received messages in real time
- clear - Clears the display

- buffer - Display the buffer scheduler
- battery - Display information about the battery and energy use
- network - Display information about the routing protocol
- quit - Stops the simulation of the node and exits.

In order to make more practical and enable the creation of more advanced graphical interfaces for the simulation, a restful API was implemented to read the current status of each node and make it available via endpoints assuring that requirement SR03 is fulfilled. The RESTapi was implemented using the *Flask*[1] Python library.

For complying with SR04, two different types of logs were created. First during execution every predefined time, a task is run by the scheduler to save the current state of the node in a CSV file. For coding organisation the log creation is managed by a separate class. Below is a list of the information logged.

- Simulation seconds - Simulation internal clock counted in seconds starting from 0
- Battery level in %
- Neighbours average battery level in %
- Node mode
- Number of neighbours
- TMAX
- TNEXT
- # of Created messages
- # of Forwarded messages
- # of Delivered messages
- # of Discarded messages
- Energy spent in computation (J)
- Energy spent in communication (J)
- Energy spent in sleep (J)

1 http://flask.pocoo.org/

- Energy spent in sensor reading (J)
- Node X coordinate
- Node Y coordinate

CORE has a built-in graphical interface to display the nodes in real-time, and that should be enough to comply with SR05, but when running the simulations via python scripts that cannot be used. So, a replacement web application was developed using the Vue.JS[2] framework. More information about this web application can be found in Annex B.

2 https://vuejs.org/

# B

# WEB APPLICATION

## B.1 WEB APPLICATION

Both Mininet-Wifi and CORE do not have a good Graphical User Interface (GUI) available for debug the protocols during simulation. Because of that, there was the need of developing a GUI with specific aspects for the EAGP protocol. Since both simulators are portable, and also the implementation of the protocol in Python, it would be desirable that the application would also be. Nowadays the fastest way of building a portable application is to build for WEB. So, it was decided to create a responsive web application that could be used in any device that can run a modern browser. The application is the counter part software for a RESTful API that runs together with the simulations, supplying the information consumed by the front-end interface.

## B.2 ARCHITECTURE

The overview architecture of the application is presented in Figure B.1.

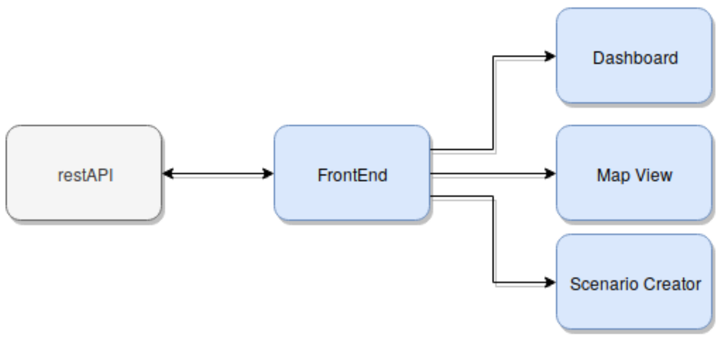


Figure B.1.: WebApp Overview

### B.2.1 *RESTful API*

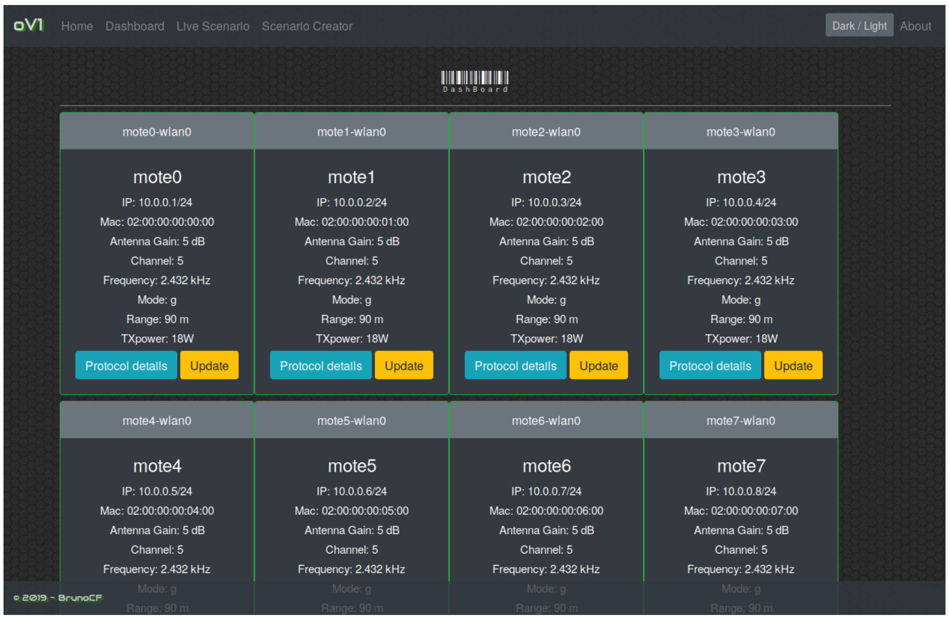
Since the RESTful API runs together with the simulator, it was developed in Flask so that it could be easily integrated with the simulation scripts. The protocol application exchange data to the API via Unix Domain Sockets and files. The API than has some endpoints available that are consumed by the front end application.

### B.2.2 *Front End*

The front end application was built using a JavaScript framework know as Vue.JS[1]. Vue is similar to React and Angular, allowing extensive code reuse due to the creation of components. To enable the application to be responsive, another framework called Bootstrap[2] was used. Finally, to consume the API using promises, Axios[3] was used.

#### *Dashboard*

The dashboard of the application shows an overview of all nodes in the simulation and some data about the wireless interface being used. This dashboard shown in Figure B.2, currently only support Mininet-Wifi simulations.



mote0-wlan0
mote0
IP: 10.0.0.1/24
Mac: 02:00:00:00:00:00
Antenna Gain: 5 dB
Channel: 5
Frequency: 2.432 kHz
Mode: g
Range: 90 m
TXpower: 18W

mote1-wlan0
mote1
IP: 10.0.0.2/24
Mac: 02:00:00:00:01:00
Antenna Gain: 5 dB
Channel: 5
Frequency: 2.432 kHz
Mode: g
Range: 90 m
TXpower: 18W

mote2-wlan0
mote2
IP: 10.0.0.3/24
Mac: 02:00:00:00:02:00
Antenna Gain: 5 dB
Channel: 5
Frequency: 2.432 kHz
Mode: g
Range: 90 m
TXpower: 18W

mote3-wlan0
mote3
IP: 10.0.0.4/24
Mac: 02:00:00:00:03:00
Antenna Gain: 5 dB
Channel: 5
Frequency: 2.432 kHz
Mode: g
Range: 90 m
TXpower: 18W

mote4-wlan0
mote4
IP: 10.0.0.5/24
Mac: 02:00:00:00:04:00
Antenna Gain: 5 dB
Channel: 5
Frequency: 2.432 kHz
Mode: g
Range: 90 m

mote5-wlan0
mote5
IP: 10.0.0.6/24
Mac: 02:00:00:00:05:00
Antenna Gain: 5 dB
Channel: 5
Frequency: 2.432 kHz
Mode: g
Range: 90 m

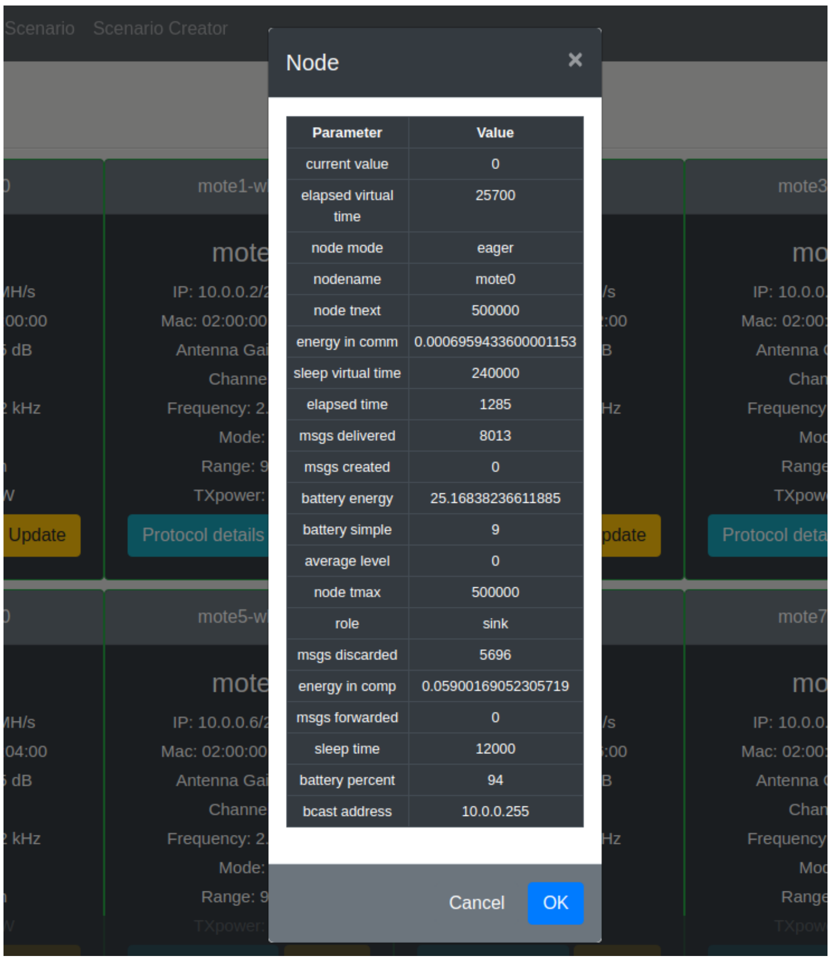
mote6-wlan0
mote6
IP: 10.0.0.7/24
Mac: 02:00:00:00:06:00
Antenna Gain: 5 dB
Channel: 5
Frequency: 2.432 kHz
Mode: g
Range: 90 m

mote7-wlan0
mote7
IP: 10.0.0.8/24
Mac: 02:00:00:00:07:00
Antenna Gain: 5 dB
Channel: 5
Frequency: 2.432 kHz
Mode: g
Range: 90 m

Figure B.2.: Dashboard Overview

1 https://vuejs.org/
2 https://bootstrap-vue.js.org/
3 https://github.com/axios/axios

When the user clicks on **Protocol Details** for a node, some information is show about that node regarding protocol statistics as show in Figure B.3



| Parameter | Value |
|---|---|
| current value | 0 |
| elapsed virtual time | 25700 |
| node mode | eager |
| nodename | mote0 |
| node tnext | 500000 |
| energy in comm | 0.0006959433600001153 |
| sleep virtual time | 240000 |
| elapsed time | 1285 |
| msgs delivered | 8013 |
| msgs created | 0 |
| battery energy | 25.16838236611885 |
| battery simple | 9 |
| average level | 0 |
| node tmax | 500000 |
| role | sink |
| msgs discarded | 5696 |
| energy in comp | 0.05900169052305719 |
| msgs forwarded | 0 |
| sleep time | 12000 |
| battery percent | 94 |
| bcast address | 10.0.0.255 |

Figure B.3.: Protocol Details

*Map View*

The map view page, allows the user to visualise the distribution of the nodes in the topology and to know which are in eager (green) or lazy (blue) modes. It also shows the id of each node as well as it's current battery level in percentage as seen in Figure B.4.

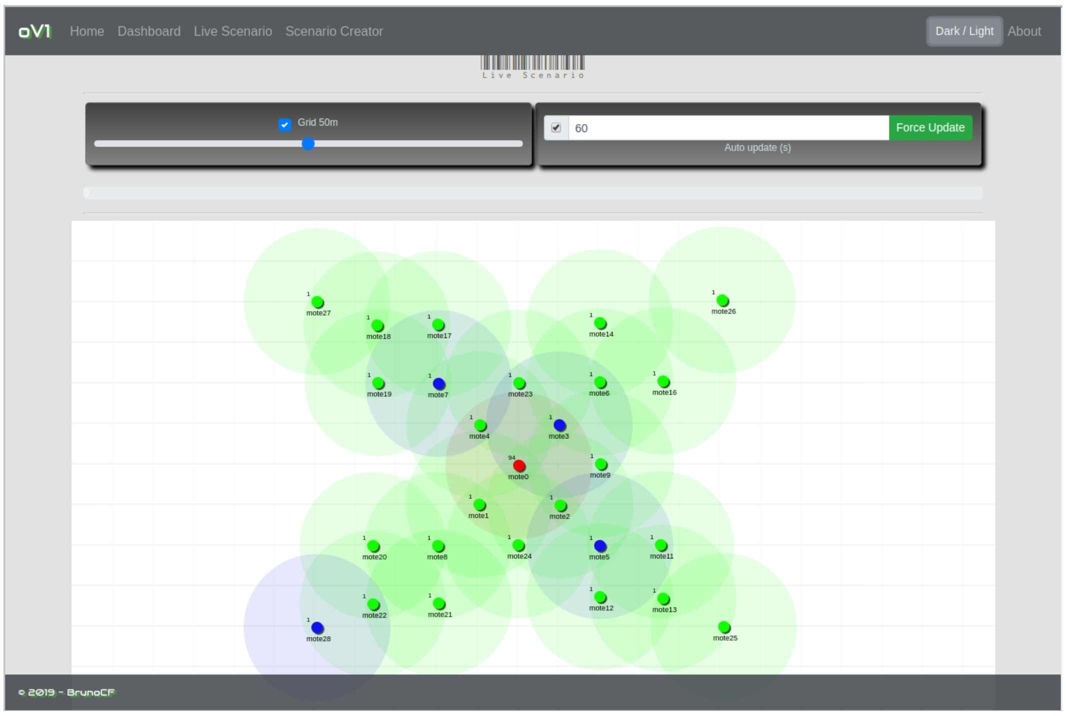
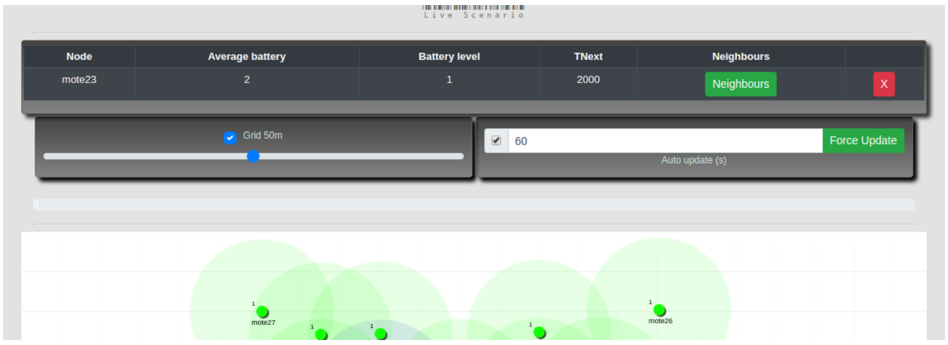


Figure B.4.: Map View

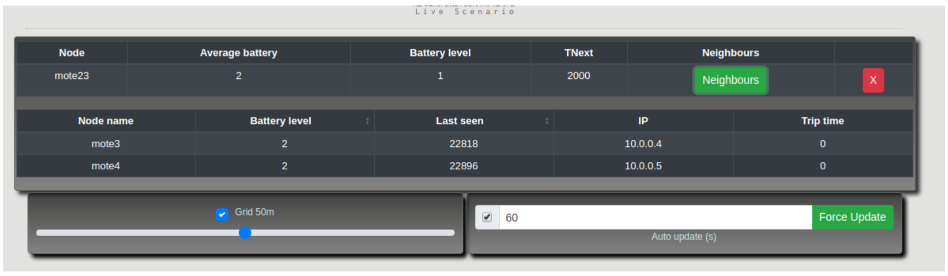
When the user clicks on a node, information about the node is displayed as shown in Figure B.5



| Node | Average battery | Battery level | TNext | Neighbours | |
|---|---|---|---|---|---|
| mote23 | 2 | 1 | 2000 | Neighbours | X |

Figure B.5.: Node View

And if the user needs to view information about the current neighbours in site, they can click on the button Neighbours and the information is show as in Figure B.6


Live Scenario

| Node | Average battery | Battery level | TNext | Neighbours | |
|---|---|---|---|---|---|
| mote23 | 2 | 1 | 2000 | Neighbours | X |

| Node name | Battery level | Last seen | IP | Trip time |
|---|---|---|---|---|
| mote3 | 2 | 22818 | 10.0.0.4 | 0 |
| mote4 | 2 | 22896 | 10.0.0.5 | 0 |

✓ Grid 50m

✓ 60 Force Update
Auto update (s)

Figure B.6.: Neighbours View

It is also possible to toggle the visibility of a grid, and the size of the grid can also be changed. To avoid the need of refreshing the page updates, the page updates itself after a configurable interval if desired.

### *Scenario Creator*

Using python scripts for creating scenarios is very practical, but creating diverse scenarios for repetitive testing might be time consuming. So, in the page scenario creator the user has a GUI that makes the process visual and the scripts are created automatically.

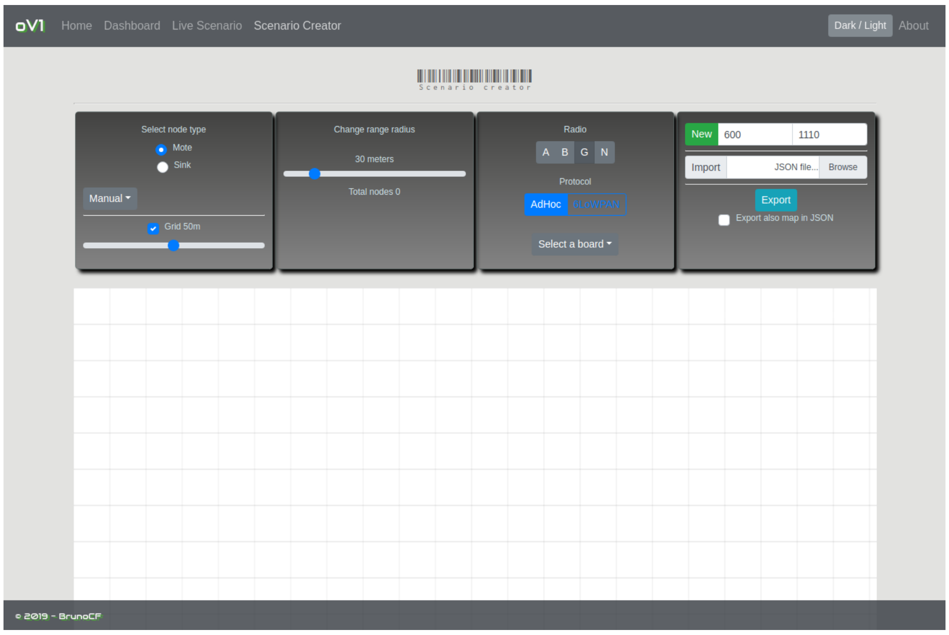


Figure B.7.: Scenario Creator

When adding nodes to the scenario, it is possible to choose between manual adding as in Figure B.8 and adding random nodes as in Figure B.9. For manual adding, the user just need to choose if it is a sink or mote node as in Figure B.10, and click on the map to add the node. To remove a node, the user needs to drag and drop the node outside the map area. Is is possible at any moment to move a node by just dragging and dropping. After adding the nodes, the user can change the radius of the radio in meters using the slider as seen in Figure B.11.

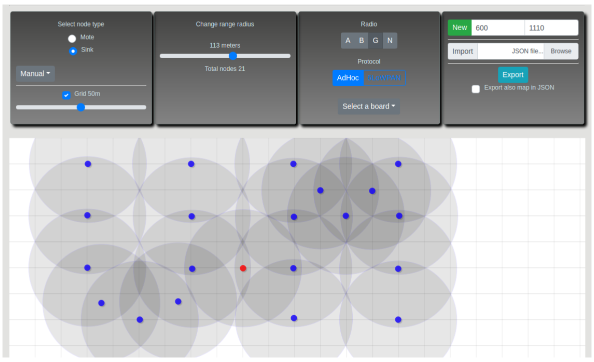


Figure B.8.: Manual Adding

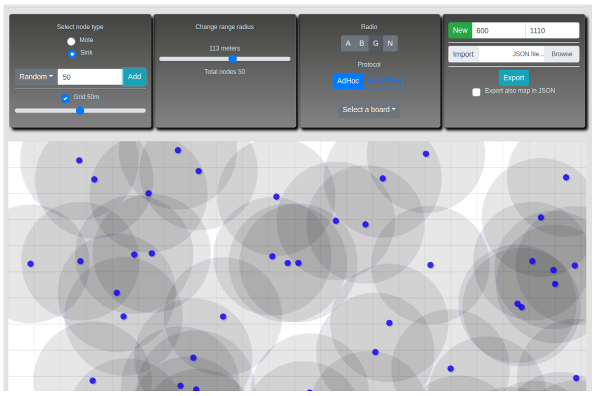


Figure B.9.: Random Adding

The network options can be set in B.12. The selection of a board is needed for the energy model that will be used for the simulation.

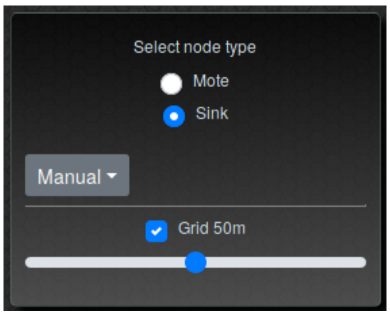


Figure B.10.: Manual / Random Adding

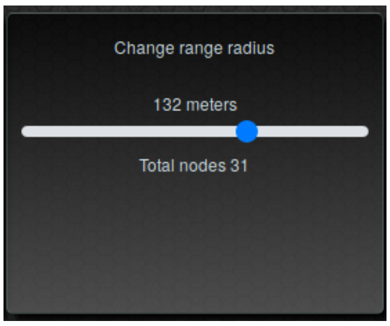


Figure B.11.: Radius Setup

The user has the option when exporting the python script, to also export a JSON object as seen in Figure B.13. The JSON object can be imported later if some adjustment is needed.

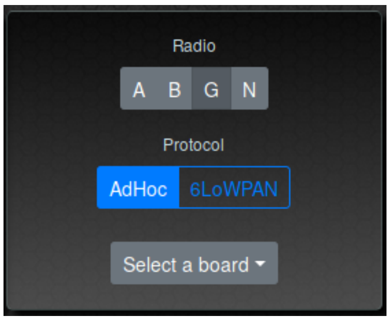


Figure B.12.: Network Config

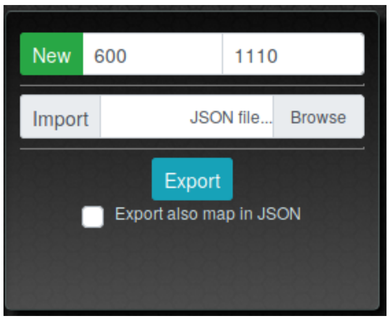


Figure B.13.: Import and Export

# C

# CORE QUICK GUIDE

## C.1 CORE PYTHON SCRIPTS BASICS

Here is a list of basic commands to be able to run a CORE simulation using Python scripts. There is not so much documentation on how to run CORE from Python scripts, so the best way is to look on existing examples.

Import of basic libraries:

```
import logging
from builtins import range
from core import load_logging_config
from core.emulator.coreemu import CoreEmu
from core.emulator.emudata import IpPrefixes, NodeOptions
from core.emulator.enumerations import NodeTypes, EventTypes
from core.location.mobility import BasicRangeModel
from core import constants
```

Load initial configuration for the logger library:

```
load_logging_config()
```

Set the network ip range:

```
prefixes = IpPrefixes("10.0.0.0/24")
```

Create emulator instance for creating sessions and utility methods:

```
coreemu = CoreEmu()
session = coreemu.create_session()
```

Must be in configuration state for nodes to start

```
session.set_state(EventTypes.CONFIGURATION_STATE)
```

Create the wlan node. The settings for the medium can changed here

```
wlan = session.add_node(_type=NodeTypes.WIRELESS_LAN)
session.mobility.set_model(wlan, BasicRangeModel,config={'range':50, '
    bandwidth': 54000000, 'jitter':0, 'delay': 5000, 'error': 0})
session.mobility.get_models(wlan)
```

Create node options for a node

```
node_opt=append(NodeOptions(name='mote0'))
```

Add the node to the session.

```
mote=session.add_node(node_options=node_opt))
```

Create a network interface and connect it to the wlan

```
interface = prefixes.create_interface(mote)
session.add_link(mote.id, wlan.id, interface_one=interface)
```

Instantiate session

```
session.instantiate()
```

Opens a terminal in the node, and run commands with attributes

```
sink=session.get_node(2)
mote.client.term_cmd("bash","command",['attibutes'])
```

Shutdown simulation

```
coreemu.shutdown()
```