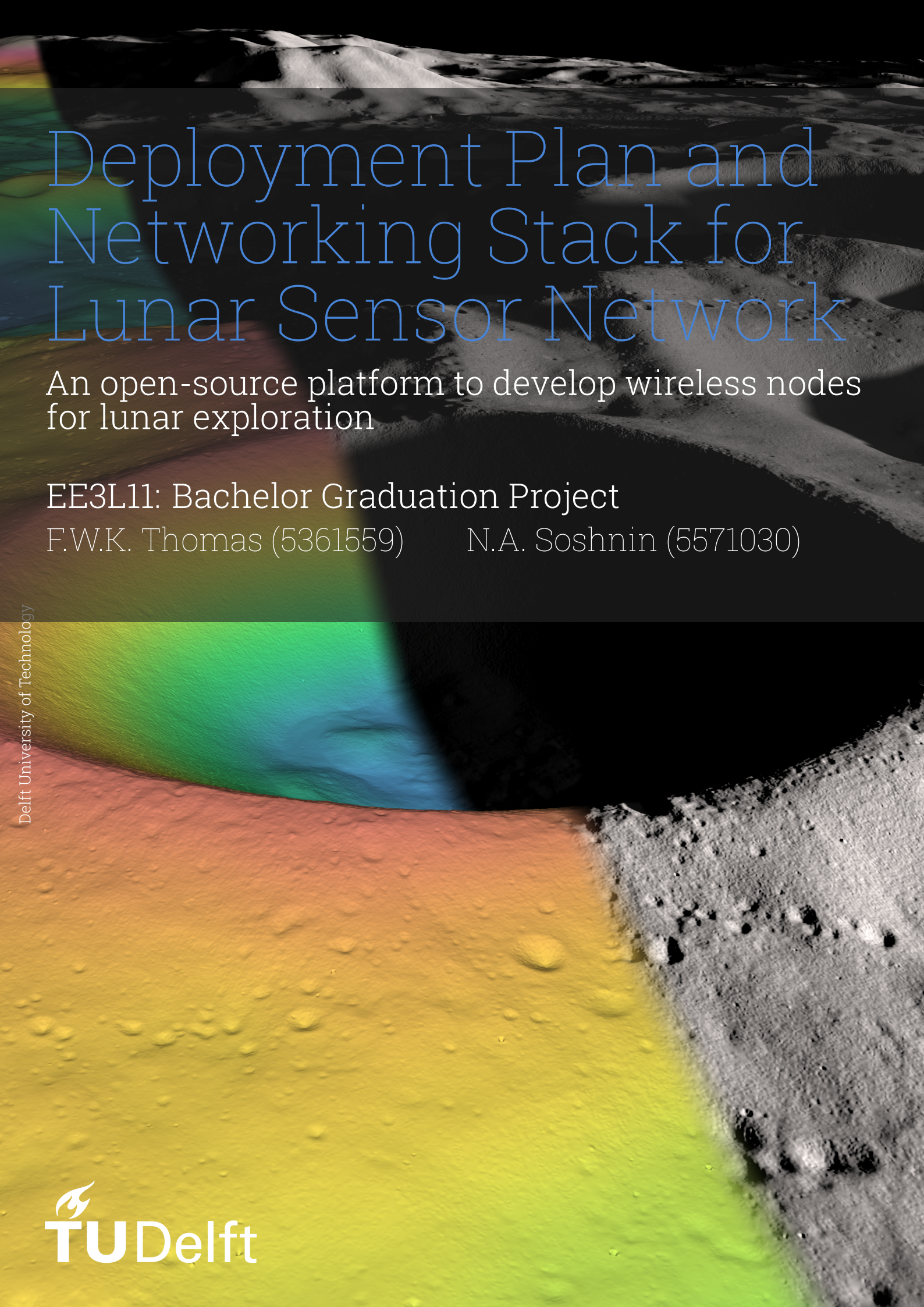


Deployment Plan and Networking Stack for Lunar Sensor Network

An open-source platform to develop wireless nodes for lunar exploration

EE3L11: Bachelor Graduation Project

F.W.K. Thomas (5361559) N.A. Soshnin (5571030)

Delft University of Technology

TU Delft

Deployment Plan and Networking Stack for Lunar Sensor Network

An open-source platform to develop wireless nodes for lunar exploration

Supervisor: F. Sebastiano

Project Duration: May, 2025 - June, 2025

Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover: A visualization of Shackleton crater.
Image Credit: NASA's Scientific Visualization Studio
Source: https://svs.gsfc.nasa.gov/4716

TU Delft

Abstract

This bachelor thesis presents the work of the sub-group ”Mesh Networking” of the larger project ”An Open-Source Platform to Develop Wireless Nodes for Lunar Exploration”. The aim of this project is to design and develop an open-source software to simulate, deploy, and visualize a mesh network to measure radiation on the Shackleton crater. The Shackleton crater is located on the South pole and has a high interest for lunar missions. Given the Moon’s rough terrain and obstacles, a measurement using a rover will be inefficient. By using a wireless mesh network, a robust, scalable network can be created to measure radiation across a wide area.

The developed software allows the user to deploy a mesh network with customizable inputs, such as connection radius, number of nodes deployed, and location. It simulates the number of hops from each node to the sink node, the lifetime, and network dynamics. On top of that, the user can input a script with a failure simulation. Additionally, the inter-connection protocols based on IPv6, 6LoWPAN, and RPL were studied and integrated.

Preface

This project was designed as a bachelor end project (BAP) for the BSc in Electrical Engineering at TU Delft. This report is written by the Mesh Networking sub-group, one of the three sub-groups working on the development of open-source wireless sensor nodes for lunar exploration. Our team mainly concentrates on the development of the deployment software and the connection between these nodes.

We chose the Shackleton crater as the primary deployment area, because of the upcoming interest in lunar missions, and the scientific need to understand the radiation near this crater. During this project, we not only gained theoretical insights into mesh networking with energy efficiency in mind, but also practical considerations for exploring the moon.

We would like to thank our supervisor, and fellow group members for their collaboration and valuable feedback. This project has taught us to use our gained knowledge from the bachelor in a valuable real-life project.

Contents

1

Project Setup

1.1. Motivation and context

Space agencies, such as National Aeronautics and Space Administration (NASA) and the Aueropean Space Agency (ESA), and private companies are striving for permanent human stay. The current longest duration of a lunar mission is 74 hours, and in order to increase this to a permanent stay, certain questions have to be answered beforehand. Is it safe to stay there for a longer duration? What is the most optimal place? Are there resources available?

The moon has a very harsh environment posing serious challenges, including extreme temperature variations, lunar dust, the presence of unfiltered cosmic radiation, and long lunar nights (with a duration of around 14 Earth days). One of the least-studied of these factors is the present of ionizing [1] radiation present on the moon, which poses a danger for human astronauts and electronic equipment alike.

Unlike Earth, the moon has no atmosphere or magnetic field. As a result, the surface is directly exposed to the solar particle events (SPEs) and galactic cosmic rays (GCRs). Prolonged exposure can damage equipment and seriously damage an astronaut's health. Therefore, in order to ensure the safety of future lunar missions, both manned and unmanned, it is important to map out how much radiation is present on areas of intereset on the moon.

The most optimal place currently identified for long-term human survival is determined to be the lunar south pole. The amount of sunlight received by areas along the rims of crators on the south pole ranges widely, with some areas staying lit up to 90 % of the time and others being permanently shadowed. It is these permanently shaded regions which may turn out to be the cornerstones of future lunar missions, since the frozen water contained in them can be used for a variety of purposes ranging from life support for astronauts to creating fuel for vehicles.

One notable crater that has drawn interest from researchers is Shackleton, located near the south pole. While much research has focused on the presence of water there, it's also important to study the radiation levels in the area. Understanding how much radiation is present is essential before Shackleton can be considered a viable landing site for future missions, particularly long-term ones [2].

To help achieve that goal, we will design a wireless sensor network (WSN) to perform radiation measurements on the rim of the Shackleton crater. This network consists of between 500-1000 miniature (on the scale of a few millimeters), low-powered sensor nodes that communicate with each other to pass the collected radiation data to a central node, which relays it back to Earth.

1.2. Purpose of the project

The main goal of our project is to design an open-source software to deploy mesh nodes on the lunar surface, and to define the protocols used for a reliable connection between them.

The Shackleton crater rim features rough terrain that makes traditional rover-based exploration impossible. To overcome this, we propose a deployment system using a small rocket to deploy each sensor

node. This approach allows wide-area, low-risk data collection.

To attack this problem as effectively as possible, the project is divided into three smaller semi-independent but integrated sub-groups: Mesh Networking, Mixed-signal Open-Source EDA, and Hardware Design.

1.2.1. Subgroup Division

The group has been divided into the following subgroups:

Mesh Networking

Given the rough terrain of the Shackleton crater, it is impractical to rely on a sensing system with one centralized node. Instead, we aim to design a mesh network with multiple clusters, which will result in a self-healing network aimed at covering obstacles. This network allows for efficient data transmission to the main hub.

This subgroup will develop an open-source software in which the mesh network can be deployed and simulated. The software will also show a visual representation of the node statistics.

Mixed-Signal Open-Source EDA

The Mixed-Signal Open-Source EDA subgroup will focus on developing an open-source tool that automates the mixed-signal integrated circuit (IC) design process. The tool will take schematic-level designs for analogue blocks and RTL-level descriptions for digital blocks as input files, and generate the final foundry-ready layout in GDSII format.

Hardware Design

The Hardware Design subgroup will focus on the development of the sensor node hardware. The main goal of this group is to define the hardware architecture and lay the foundations of its implementation.

1.3. state of the art analysis

There are also several advanced mesh networking simulators available. However, most of these tools operate at a very low level of abstraction by simulating specific networking stacks (examples include Cooja [3], which emulates the mesh node hardware, and `meshnet-lab` [4] which virtualises the nodes' networking stack using Linux namespaces). Researchers looking for higher-level tools that model node connectivity without elaborating the details of specific protocols are left to develop custom solutions, such as [5].

Despite the availability of both mesh network simulators and physical modeling libraries, there is currently no solution that effectively integrates the electrical and physical aspects of a mesh network deployment.

Furthermore, we aimed to create a ready-to-use tool, rather than a library. Although interactive mesh networking tools exist, such as the aforementioned Cooja, they are few and often difficult to use. This presents a barrier for users who want to interactively explore mesh networking strategies without a steep setup curve. We have designed our tool with this usability gap in mind, opting to build an interactive, ready-to-use simulator.

1.4. Synopsis

This thesis contains a detailed overview of the mesh simulation software that we developed, along with details of the hardware to be used for a potential deployment. Key engineering decisions for each of the two deliverables are explained with references to relevant literature. Screenshots and graphs are also present.

1.5. Programme of requirements

This Bachelor thesis focuses on the mesh networking sub-group. The main goal of our bachelor end project is to develop an open-source software to deploy, simulate and visualize node distribution on the Shackleton crater. Not only the software will be designed, also the inter-node connection protocols will be defined.

To ensure that our project meets scientific specifications, we set up a programme of requirements.

1.5.1. Functional requirements

1. Simulation software
    - The simulation software must be capable of deploying a multi-cluster mesh network.
    - The deployment must simulate an effective distribution method.
2. Visualization of network statistics
    - The software should display the amount of connected and disconnected nodes.
    - There must be a clear interface of the node locations.
3. Meteor shower/failure simulation
    - The software must have an option to implement a script to simulate a meteor shower or node failure over time.
4. Inter-node communication
    - The protocols for each layer of the Transmission Control Protocol/Internet Protocol model(TCP/IP) should be defined and clearly reasoned.
5. Energy Usage
    - An energy-efficient micro controller should be selected which is optimal for our application.
    - The energy usage of the micro controller must be calculated for relevant operation modes.

1.5.2. Non-functional requirements:

1. Open-Source
    - The software and code must be developed open-source and publiced on github.
    - There is an available online download for the deployment software.
2. Easy-to-use interface
    - The installation and use of the platform should be easy-to-use and accessible for everyone.
3. Performance
    - The deployment of up to 1000 nodes should be smooth and without lag.

2

Architecture overview

The lunar surface is characterized by craters, boulders, and steep elevation, and this rough terrain makes it difficult for a rover to explore. Traditional single-point measurements pose challenges to measure in a large area. Especially when the goal is to measure radiation, the amount of radiation can vary significantly over short distances. A wireless mesh network solves all these challenges and creates a lot of benefits:

- If one node fails, the nearby coherent nodes will take over. The network is self-healing.
- The data can move around obstacles such as boulders or debrees.
- New sensors can be added as the number of missions increases, creating a scalable and adaptable network.
- A sleep mode and duty cycle can be implemented, creating a very low-power, energy-efficient mesh network.
- The radiation can be measured and mapped across a lot of different points, this will show how radiation varies with location, amount of sun, and terrain.

A scalable, modern, and durable architecture is required to realize this mesh-based network. This chapter will introduce the use of the Open Systems Interconnection(OSI) model.

2.1. TCP/IP model

For our system the, Transmission Control Protocol/Internet Protocol model (TCP/IP) model is used, as described in [6]. The TCP/IP model is a communication model divided into four layers that establish a reliable connection between different devices, sensor nodes in our case. Each data layer relies on the layer below and above them, guaranteeing a smooth data transfer. Certain protocols were selected with energy efficiency as main priority.

– **Application layer:** This layer forms the link between the interface and the application, in our case this is this linked to the radiation sensor.

– **Transport layer:** Ensures reliable data transfer.

– **Internet layer:** Assigns IP addresses to nodes. It also fragments, and routes the packages to the correct address.

– **Link layer:** This layer handles the actual transmission of data over a network medium.

2.2. Transport layer

For the transport layer, the User Datagram Protocol (UDP) was selected. UDP provides a fast, and lightweight way to send data across a network. UDP does not establish a connection beforehand and does not provide feedback, making it very energy-efficient [7]. The other option for this layer is the Transmission Control Protocol. TCP is build for end-to-end communication over stable links. For TCP to require stable links it consumes a lot of energy, which is limited in space, it requires an acknowledgement (ACK) of the received data, and retransmits lost packets. While ensuring a reliable transmission, this does consume a lot of energy, which is not optimal in our case.

Studies [8] also confirm that the throughput of UDP drops less than TCP in the event of link failure, which refers to the fact that the connection between nodes is temporarily or permanently interrupted, which is likely to happen in space.

2.3. Internet layer

The network layer is responsible for packet routing and adressing. IPv6 via Low-Power Wireless Personal Area Networks (6LoWPAN) will be used. IPv6 is the newest version of the internet protocol and will give each node his unique address. IPv6 over 6LoWPAN will the IPv6 header from 40 bytes to around 4 bytes [9]. Routing Protocol for Low-Power and Lossy Networks(RPL) is used for routing data [10]. RPL uses Destination-Oriented Directed Acyclic Graph(DODAG) to establish the most optimal path for data transmission, minimizing the amount of hops, thus minimizing the amount of energy usage. Additionally RPL supports self-healing by calculating new routes when a node fails. Since our goal is to measure radiation in a lunar environment this makes it an ideal choice.

2.4. Link layer

The data link layer handles addressing between neighboring nodes, which then communicates with the Physical layer which actually transmits the individual bits. IEEE 802.15.4 standard is used to ensure low-power data transfer [11]. It offers a good balance between power usage and data transfer. A data frame can be up to 127-byte and the radio operates at 2.4 GHz.6LoWPAN is used to shrink the packets into frames smaller than 127 bytes.

A simplified block diagram of the OSI model is presented in figure 2.1.

2.5. Operating system

Contiki-NG is chosen as the designated operating system, it offers great benefits with energy efficiency and open-source embedded systems. Contiki-NG is specially designed for low-power, resource-constrained systems. On top of that it also supports IEEE 802.15.4, IPv6, RPL and 6LoWPAN making it highly fitting for our application [12]

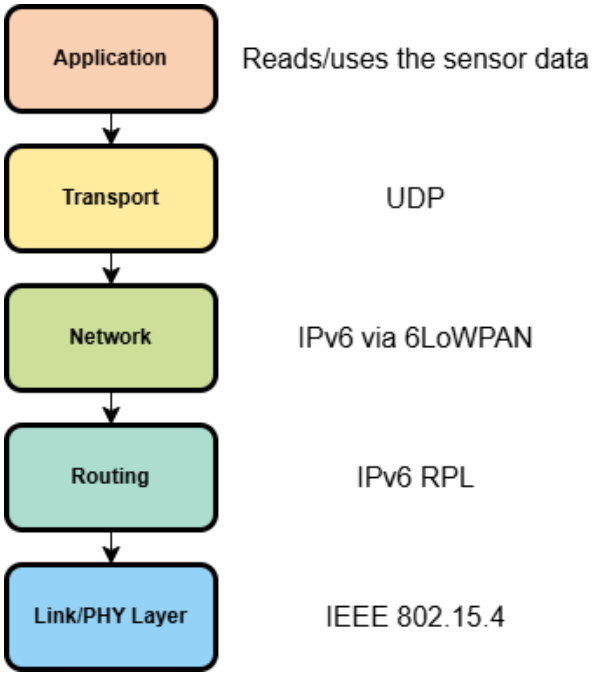


Figure 2.1: OSI model of a microcontroller

3

Hardware constraints

During this project the hardware design focuses on an IC design implementation of the sensor node. The hardware block diagram is illustrated in figure 3.1. The IC design is focused on optimizing power consumption and the ability to withstand harsh lunar conditions. Using our architecture decisions the hardware constraints for the microcontroller can be determined.

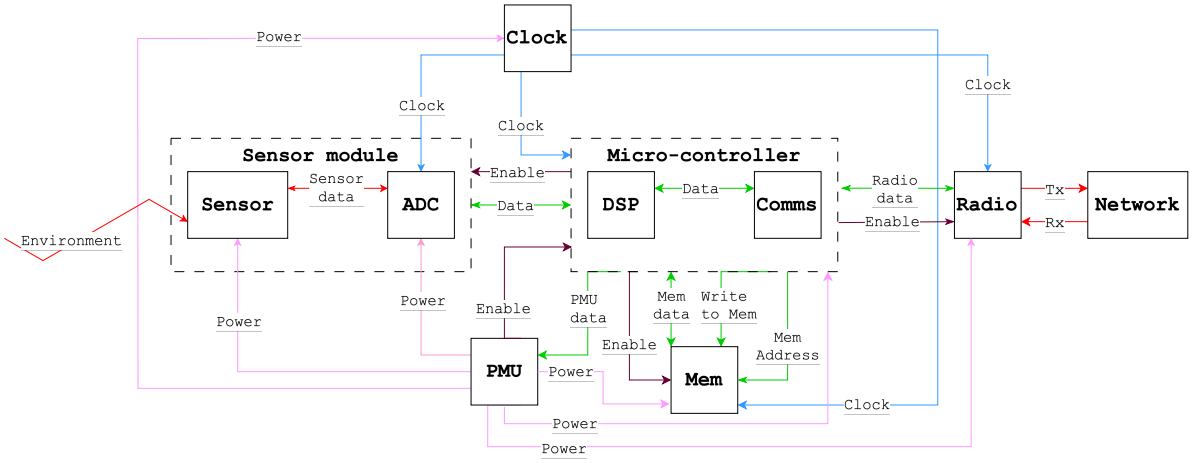


Figure 3.1: Hardware block diagram of the sensor node. Adapted from [13]

The explained in the previous section chapter 2 the sensor node will run Contiki-NG with the use of IPv6, RPL and 6LoWPAN. The sensor node will need a certain amount of RAM and flash storage to store the operating system and the network stacks.

A complete build of Contiki-NG takes around 100kB of flash storage and 10kB memory[12]. In order to allow application code to run on the nodes, the actual flash and RAM sizes should be bigger.

Although the sensor nodes will be based on a custom-designed mixed-signal chip, we note that there are existing commercially available microcontrollers that can host our networking stack. One such microcontroller is the The Texas Instruments CC2652R7 (shown in figure 3.2), which features IEEE 802.15.4 networking capabilities in the 2.4GHz band, in addition to bluetooth 5.2. This microcontroller also has 352 kB of flash memory and 88 kB of RAM (8 kB of cache SRAM and 80 kB of ultra-low leakage SRAM). The operating range is between -40 °C and 105 °C, which is not sufficient for the harsh lunar conditions which range from -253 °C to 120 °C. To ensure an operating range capable of withstanding these conditions, radiation hardening and thermal protection have to be applied. Although there are no plans to use this specific microcontroller, it can be used as a reference to design the digital part of the mixed-signal sensor node chip.

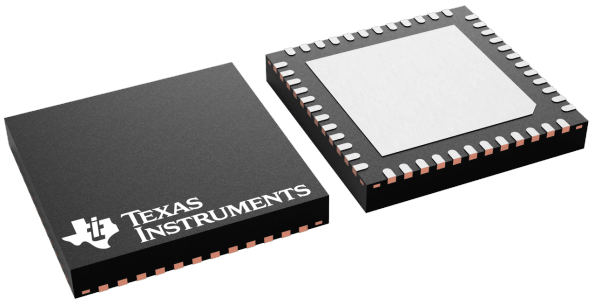


Figure 3.2: The Texas Instruments CC2652R7 Microcontroller. Adapted from [14]

3.1. Antenna

The antenna used for communication is one of the main defining factors of the capabilities of a sensor node. In applications where size and power are a primary concern, the best option is to use integrated on-chip antennas as opposed to discrete antennas [15].

There have been substantial developments in designing on-chip antennas and even complete on-chip transceivers specifically for the 802.15.4 signaling protocol, with communication ranges as far as 10 meters [16].

Apart from facilitating inter-node communication, the antenna is used to determine the location of each wireless sensing node. Although our networking stack is capable of routing data with no knowledge of the physical structure of the mesh, geographical position of each sensor is nevertheless a crucial component for analyzing the recovered data. To that end, Kim et al [16] has demonstrated that by using time-of-arrival (ToA) and angle-of-arrival (AoA) measurements, position can be estimated to within a few centimeters.

4

Deployment

The deployment of sensor nodes for lunar exploration is an important step in mesh networking. Due to the rough lunar terrain compromising large height differences and objects, the connectivity plays a crucial role. The method in which the nodes are deployed decides the number of nodes connected. Each node should be connected to minimal one node and optimal four nodes. Nodes that are connected to only one other node are more likely to lose contact in the future. [17] describes an optimal deployment method.

This deployment method is optimal for our situation for a few reasons. This method is designed with redundancy as main goal. The ”positive n-edge cross deployment” method is used to deploy the sensor nodes in symmetrical rings around the cluster head ensuring connectivity with multiple nodes, with multiple neighboring nodes, it reduces the risk of a node failure because of a failed connection. Since nodes have multiple neighboring nodes the mesh network can adapt to the uneven lunar terrain, using RPL the mesh network can route the data around the boulders and uneven terrain. The deployment starts in space, allowing precise locationing around obstacles. Also by using this method, the main cluster head can be a different type of node focused on processing the data, and sending it back to earth.

The deployment method describes a multi-cluster deployment plan for a wireless sensor network. The deployment involves a spacecraft, in the paper mentioned as ”prober”, and involves a four step deployment scheme.

1. **Prober seperation**: The spacecraft releases , with a maximum of 9, cluster heads at a predetermined location in 3d space.
2. **Cluster deployment**: Each cluster moves to its designated location in 3d space and deploys the sensor nodes utilizing a method called the ”positive n-edge cross deployment”. This method deploys the sensor nodes in 5 rings, each with the same number of nodes, around the cluster head, ensuring good connectivity and localization.
3. **Cluster head landing**: Each cluster head lands directly in a vertical position below the cluster deployment.
4. **Network operation**: The cluster head receives the data,radiation in our case, and transmits it back to earth for processing.

4.1. Mesh Network Simulation for Lunar Deployments

The Mesh Networking subgroup developed a comprehensive simulation software designed to inform and facilitate the deployment of mesh sensor networks on the lunar surface. The simulation addresses the unique challenges of lunar environments, offering robust capabilities for analyzing network performance and resilience.

The simulation software is called BAPMESIM and is licensed under the GNU General Public License

version 3 only. Source code and binary builds for Windows and Linux are available on `https://github.com/maybeetree/bapmesim`.

4.1.1. Simulation Capabilities

The simulation software is capable of modeling both single-cluster and multi-cluster deployments. It collects a range of network statistics, including the number of hops from each node to the root node, and the number of connected and disconnected nodes.

4.1.2. Simulation backend

At the core of the BAPMESIM simulation tool lies a k-d tree, a spatial data structure that efficiently stores the locations of the nodes in the mesh network. A k-d tree enables fast nearest-neighbor searches in 3D space, which is an essential function for our simulation, as determining which nodes are within communication range of each other is fundamental to computing the connectivity of the mesh.

A naive approach (namely, a brute-force scan of all node pairs to check whether they are in range), would run in $O(n^2)$ time, where each operation would require computing the euclidean distance between two 3D points.

By employing a k-d tree we instead reduce the complexity of building a connectivity graph of the network to $O(\log n)$ time [18], which allows for much faster operation.

Once neighbor relationships are established, the simulation uses the `networkx` library [19] to analyze the resulting communication graph.

4.1.3. Design Choices

In designing the BAPMESIM simulator, one of our first major decisions was to adopt a Cartesian XYZ coordinate system for node placement and distance calculations. While the Moon is approximately spherical, and formal lunar surveys typically use planetocentric or planetographic coordinate systems, we opted for XYZ coordinates due to their computational simplicity. Survey coordinates can be easily converted into Cartesian coordinates, making this decision both practical and compatible with existing datasets. Additionally, XYZ coordinates are more suitable for performance-optimized algorithms like the k-d tree; implementing a k-d tree over latitude, longitude, and altitude data would be significantly more complex and less efficient. Furthermore, many popular GIS tools internally operate using Cartesian representations, which makes our codebase more familiar to contributors and simplifies potential integration with other tools in the spatial computing ecosystem.

The simulator is implemented in Python to ensure accessibility for other researchers, particularly those in academic or interdisciplinary fields who may not have experience with lower-level languages. However, performance remains a priorityparticularly for vector and matrix operationsso the core numerical computations leverage numpy. This library allows for vectorized operations that are orders of magnitude faster than equivalent Python loops, especially when simulating large numbers of nodes.

These design choices form the foundation of the simulators backend. As for the frontend, we prioritized usability and long-term accessibility. To this end, we use `pyinstaller` [20] to generate binary builds of BAPMESIM. This allows users to run the simulator without needing to install Python or any dependencies – something that can be a barrier for those unfamiliar with Python development tools like `pip`. Additionally, packaging the simulator as a standalone executable ensures that it remains usable even if the Python ecosystem changes or services like PyPI become unavailable. Much academic and research software is notoriously difficult to install and use, often requiring complex manual setup that discourages adoption. By taking a more consumer-software-inspired approach to distribution, we aim to make BAPMESIM more accessible and user-friendly from the outset.

4.1.4. User Interfaces for Flexible Operation

To cater to various stages of deployment planning, the simulation software offers two distinct, yet complementary, user interfaces:

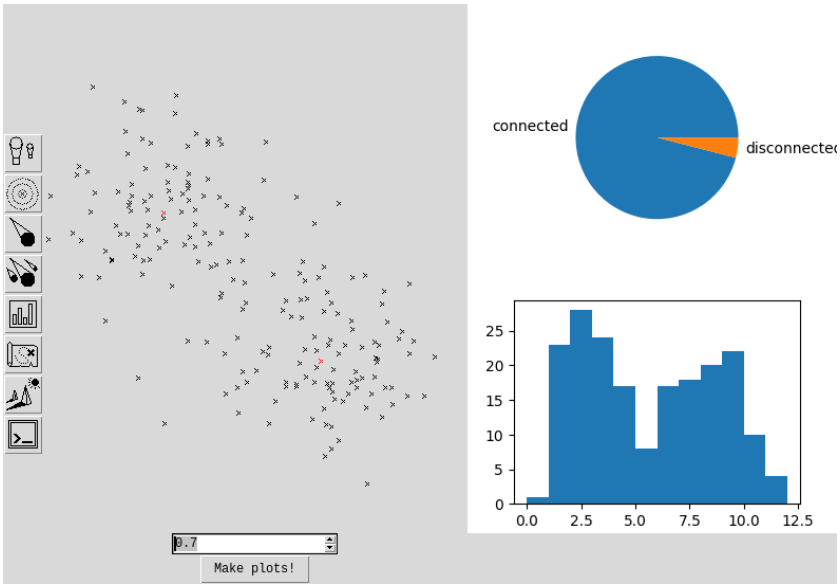


Figure 4.1: Screenshot of BAPMESIM, the simulation software developed by the Mesh Networking subgroup, demonstrating the toolbar and network metrics plots. Nodes marked in red are clusterheads.

4.1.5. Interactive Graphical Interface

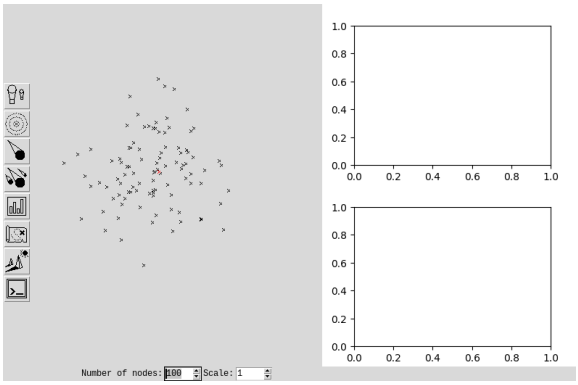
For the early stages of planning and interactive exploration, the software features a user-friendly graphical interface (shown in Figure 4.1). This interface incorporates a toolbar inspired by common image editing programs, aiming to provide a familiar and intuitive user experience. Key tools within this interface include:

– The **node scatter** tool, which enables users to efficiently set up diverse node deployments across the simulated lunar terrain.

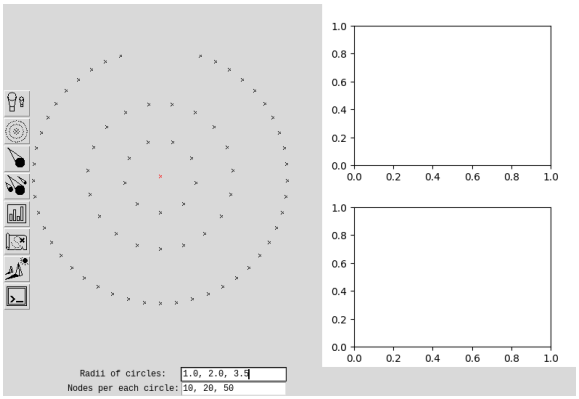
– The **hillshade** tool, vital for determining which nodes receive solar power at different times of the lunar day, crucial for energy management in power-constrained environments.

– The **meteor** tool, which simulates the impact of meteor strikes, allowing for analysis of network resilience when impacted nodes are taken offline. This interactive approach facilitates rapid prototyping and visualization of network configurations, enabling engineers to quickly assess different deployment strategies.

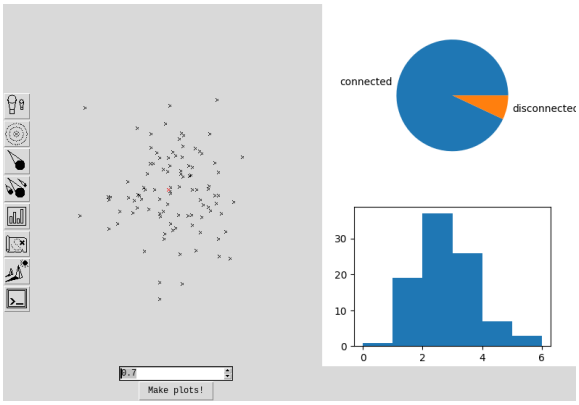
A visual demonstration of the features of BAPMESIM is given in Figure 4.2.



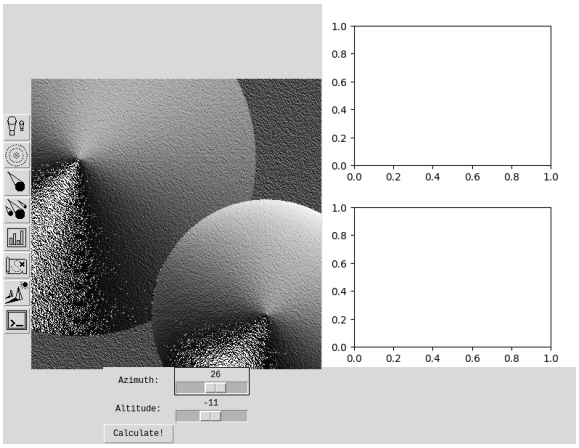
(a) A gaussian mesh deployment.



(b) An alternative "circle"-style mesh deployment. Radii and number of nodes can be entered in the toolbar at the bottom.



(c) Graphs displaying number of connected nodes and number of hops to sink node for a gaussian mesh deployment



(d) A demonstration of the terrain modeling features of BAPMESIM. A sample terrain file has been loaded in using the *terrain* tool, and sunlight illumination has been simulated using the *hillshade* tool. The position of the sun in the sky can be chosen with the sliders at the bottom of the window (or entered directly if using the scripting API). The terrain used in this example is not a real terrain, but, thanks to the use of `rasterio` package [21], real-world lunar terrain can also be used.

Figure 4.2: Simulation demonstration.

4.1.6. Non-Interactive Scripting API

For later stages of planning, which often involve more complex analyses and automated tasks, the simulation provides a simple yet flexible non-interactive (scripting) API. This API empowers users to create sophisticated scripts for various purposes, including:

- Collecting statistics over time, such as monitoring network connectivity as nodes degrade due to wear and tear or unexpected events like meteor strikes. Setting up complex node deployments, which can then serve as a basis for further interactive simulation or in-depth analysis. This allows for the programmatic generation of intricate network layouts that might be difficult to achieve manually.
- Collecting complex network statistics, such as quantifying the number of alternative routes between two clusters within a network, which is critical for assessing redundancy and fault tolerance.
- Implementing complex deployment patterns that deviate from standard distributions, such as non-Gaussian node distributions, offering greater realism for specific lunar scenarios. The scripting API significantly enhances the automation and scalability of simulation tasks, making it an indispensable tool for rigorous network design and validation.

4.1.7. Extensibility and Maintainability

Recognizing the need for adaptability in long-term research and development, the simulation tool is designed with extensibility and maintainability as core principles. Our approach to achieving this goal is threefold:

Free Software Principles: The tool is released as Free Software, granting users the legal freedom to modify, distribute, and utilize the software as they see fit. This open-source philosophy fosters community collaboration and ensures the longevity of the project.

Pythonic Implementation: The software is written in Python, leveraging widely adopted scientific libraries such as SciPy and NetworkX. This choice of programming language and libraries significantly lowers the barrier to entry for potential contributors, as Python is renowned for its readability and ease of use, and is already prevalent in academic and scientific computing. This technical accessibility empowers researchers to modify and extend the tool to suit their specific research needs.

Self-Contained Binaries: To guarantee long-term usability and mitigate potential issues arising from future software decay, we distribute self-contained binaries of the tool built using PyInstaller. This packaging approach ensures that the simulation remains functional ”out of the box,” even if its dependencies or the build infrastructure (such as PyPI) become unmaintained or obsolete. This foresight ensures the enduring utility of the simulation for future lunar missions and research endeavors. This comprehensive approach to the simulation software’s design and implementation ensures it is not only a powerful tool for current lunar mesh network planning but also a robust and adaptable platform for future research and deployments.

5

Energy usage

For lunar exploration, the use of energy is very important, there is only limited solar power available, and the sensor node has only a small battery. Reliable data transfer with minimal energy usage is required. With the use of the TI CC2652R7 datasheet [14] it can be estimated if our plan is feasible, using this datasheet the energy usage of a microcontroller with the required specifications can be calculated as illustrated in table 5.1

Mode	Current Draw (mA)	Duration (ms)	Energy (µJ)
Radio transmission @ 0 dBm	7.3	4	86.0
Radio transmission @ 5 dBm	9.7	4	116.4
CPU processing (6LoWPAN + IPv6 + UDP stack)	3.1	2	18.6
Radio receiving	6.4	4	76.8
ADC measurement	3.1	1	9.3
Standby / Sleep (24 h)	0.0009	86 400 000	233 280.0

Table 5.1: Power Consumption of Various Operating Modes of the CC2652R7 (Assuming 3V Supply)

Notice that the number for standby/sleep is extremely low, this is because the datasheet only states the low-power usage and does not include a clock, wake-up timer or any other components which are always active. These always active components will increase the standby/sleep energy usage significantly. The timer chosen by the Hardware design group draws between 2-4 µW. When still assuming a 3V battery, the clock draws between 0.667-1.334 µA. The clock will consume between 0.17289 J and 0.34578 J per day. So the microcontroller will draw between 0.40608 J and 0.57888 J per day assuming that its operating mode is standby/sleep for a full day.

The data presented in table 5.1 indicates that transmission consumes the most energy. The duracell Lithium CR2032 coin cell battery has a capacity of 210 mAh [22], which is about the average capacity of a coin cell battery, will be used ass example for energy availability. While this will not be the final battery, it will be used to illustrate a realistic perspective.

$$\mathrm{E} = 210 \times 10^{-3} \times 3 \times 3600 = 2268\,\mathrm{J} \tag{5.1}$$

The battery contains 2538 joule, this would mean that a standard coin cell battery could power the sensor node for an extended duration.

$$\text{Days} = \frac{2268}{0.40608} = 5585\,\text{days} \tag{5.2}$$

$$\text{Days} = \frac{2268}{0.57888} = 3917\,\text{days} \tag{5.3}$$

If the microcontroller unit would permanently be in sleep mode, it would theoretically be able to last between 3917 and 5585 days on a single coin battery. This is without battery discharge, losses and actually transmitting and receiving data, realistically this number is a lot smaller.

A node will not just be in sleep mode all the time, it can execute multiple operations. The initial operation involves gathering the data and afterwards transmitting the data to the next node in the mesh network. The next node then has to receive this data, process it by its CPU, and transmit it to the next node, the hop-through operation. The energy required per each operation is presented in table 5.2.

Operation	**Energy (mJ) @ 0 dBm**		**Energy (mJ) @ 5 dBm**	
	Min	**Max**	**Min**	**Max**
One hop (receive + CPU + transmit)	0.181	0.181	0.212	0.212
One measurement + send	0.114	0.114	0.144	0.144
Sleep (24 h)	406.1	578.9	406.1	578.9

Table 5.2: Estimated energy consumption per operation mode (assuming 3V supply).

6 Discussion

The main goals of our subgroup were to A. develop a simulation software for a lunar mesh sensor network, and B. create a sample deployment plan for such a mesh sensor network. To address the first goal, we have developed a free and open source python-based simulation software called BAPMESIM, which can compute connectivity statistics of simulated sensor nodes. As for the second goal, we've created a deplpoyment framework that addresses the networking software stack for use in the nodes, as well as a selection of hardware which can be used as a reference for eventually developing the real chip. The system uses Contiki-NG as operating system, RPL for self-healing routing, IPv6 communication using 6LoWPAN, and IEEE 802.15.4 for transmission. These protocols offer great compatibility with IP-based systems, making our system long-term viable, in comparison to other popular, but less widely supported, zigbee.

The major limitation in the TI CC2652R7 microcontroller is the lack of protection against radiation, the microcontroller is not radiation hardened. It is also uncertain whether the microcontroller can perform at extreme lunar temperature. Further design should focus on radiation shielding and thermal insulation which the Hardware subgroup is working on [13].

The energy consumption was estimated based on the specifications of the TI CC2652R7 and its operating modes. Although the energy usage is very low in low-power mode, the transmission and receiving of data requires a lot of energy. The implemented protocols contribute a lot to the energy usage. The Contiki-NG dutycycle protocol decides how often the node wakes up, transmits, receives, or does a measurement. The use of RPL may result in message exchange at irregular intervals. These protocols define the time a node has to be awake or asleep.

Due to the difficulty in calculating an exact duty cycle with our protocol choice, we were not able to calculate it during this bap. For future work, the protocol choices in combination with the software, which shows the amount of solar power the sensor node receives, and the power usage should be combined into an effective duty cycle.

6.1. Future Work: Simulation

Having discussed the foundational design choices behind BAPMESIM, we now turn to possible directions for further development.

6.1.1. Speed

One major area for improvement is performance optimization, particularly for simulations involving large node counts. While BAPMESIM already uses specialized data structures, such as the k-d tree, to accelerate node-related computations, we observe that the simulator begins to slow down significantly when running with more than 10,000 nodes. Although the software performs well under moderate loads, this scalability issue becomes a bottleneck for experiments involving denser or larger-scale mesh networks, which may be necessary for realistic lunar deployment scenarios.

To understand the root cause of this slowdown, a detailed performance profiling pass is needed. This would help pinpoint which parts of the codebase are contributing most to the lag. One promising hypothesis is that the slowdown is largely due to the overhead incurred by the current dependency stack, particularly the interaction between `networkx` and `scipy`, which requires a lot of (slow) conversions between different Python objects.

A potential solution is to replace these dependencies with a purpose-built C extension module [23] that integrates the k-d tree logic directly with a shortest-path algorithm implementation. By shifting these operations to C, we can eliminate the overhead of converting between the different datastructures used by Scipy and `networkx`. Additionally, writing more of the core logic directly in C could potentially benefit from compile-time optimizations, which are not possible with the current implementation. This approach would not only improve runtime performance but also enhance the maintainability of the software. By decoupling from external Python libraries, future compatibility issues due to dependency version changes can be avoided, leading to a more stable and self-contained codebase.

6.1.2. Packaging

In addition to improving performance, another promising area for further work is enhancing how BAPMESIM is packaged and distributed. As noted earlier, we currently use pyinstaller to create binary builds of the simulator. This has already made the software significantly more accessible, especially for users unfamiliar with Python development environments. However, there is still room for improvement in ensuring the longevity and broad compatibility of these builds.

One enhancement would be to produce static binaries. While current `pyinstaller` builds bundle most of the application's dependencies, they still rely on various low-level system libraries being present on the user's machine, such as the linker and a specific implementation of the C standard library [24].

This can lead to compatibility issues, particularly across different Linux distributions or on future versions of Windows where library support may change. Static binaries, which include all necessary libraries internally, would eliminate these concerns. A tool like `staticx` could potentially be used to transform our pyinstaller builds into fully self-contained executables, capable of running on a much wider range of systems without modification.

Beyond static binaries, broader distribution methods could also be explored. Packaging BAPMESIM for platforms such as Flatpak would allow for sandboxed, cross-distribution Linux support with simplified installation. For Windows users, submitting builds to the Microsoft Store could further lower the barrier to entry, making the simulator more discoverable. These improvements in packaging would help align BAPMESIM with the expectations of modern software users, both inside and outside academic research, by delivering a tool that is not only powerful but also accessible.

6.1.3. Tooling

To ensure the long-term maintainability of BAPMESIM, further development should also include improvements to the internal tooling used during development. One key addition would be the use of mypy [25], a static type checker for Python. By adding typing hints and enforcing them with mypy, we can catch subtle bugs that would be difficult to encounter at runtume. Moreover, type annotations serve as a form of documentation that makes the behavior and expectations of each function and class clearer to potential future contributors.

Complementing static type checking, adopting a modern linter such as `ruff` [26] would help enforce consistent coding style across the project. `Ruff` can automatically detect and correct a wide variety of style violations, reducing review overhead and making the codebase cleaner and more readable.

The broader goal of introducing these tools is to make BAPMESIM more inviting for external developers who may want to contribute. A clean, well-typed, and consistently formatted codebase not only improves reliability but also lowers the learning curve for new contributors, laying the groundwork for a sustainable and collaborative future for the project.

6.2. closing words

In conclusion, this project lays the foundation for the use of wireless mesh networks for lunar exploration. Future work should test the product under the stress of lunar conditions, make it even more energy efficient, and support large-scale node deployment.

7

Conclusion

This thesis aimed to develop an open-source software to measure radiation on the Shackleton crater, and the communication protocols between the sensor nodes. Since this project focuses on lunar exploration the emphasis is on energy efficiency since resources are limited on the moon. With the use of Contiki-NG as an operating system, IPv6 can be used to address packets using 6LoWPAN to shrink packets, and IEEE 802.15.4 can create an energy efficient, robust network.

With the use of RPL as routing method the deployed mesh network is suitable for the harsh lunar environment. The use of DODAG creates a self-healing mesh network optimal for communication in the lunar terrain.

BAPMESIM uses the ”positive n-edge cross deployment” creating circles around the cluster head also ensuring a robust network for communication. The software also shows the amount of hops it takes each node to reach this sink node, and simulates the amount of solar light reaching the nodes.

Since the project is open-source our work can be continued. In the future the functions of the amount of hops in combination with the amount of sun can be combined into a duty cycle making the system more energy efficient.

This project lays the foundation for robust, wireless radiation measurements on the moon, supported by an open-source software to simulate and deploy the mesh network.

References

[1] J. E. Mazur, W. R. Crain, M. D. Looper, *et al.*, "New measurements of total ionizing dose in the lunar environment," *Space Weather*, vol. 9, no. 7, 2011. DOI: https://doi.org/10.1029/2010SW000641. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2010SW000641. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2010SW000641.

[2] NASA, *Shackleton crater's illuminated rim and shadowed interior*, Accessed: [Insert Date], 2022. [Online]. Available: https://science.nasa.gov/resource/shackleton-craters-illuminated-rim-shadowed-interior/.

[3] T. Mehmood, *Cooja network simulator: Exploring the infinite possible ways to compute the performance metrics of iot based smart devices to understand the working of iot based compression & routing protocols*, 2017. arXiv: 1712.08303 [eess.SP]. [Online]. Available: https://arxiv.org/abs/1712.08303.

[4] *Emulate huge mobile ad-hoc mesh networks using Linux network namespaces.* 2025. [Online]. Available: https://github.com/mwarning/meshnet-lab.

[5] C. Sevgi and A. Koçyi□it, *Optimal deployment in randomly deployed heterogeneous WSNs: A connected coverage approach*, 2014. DOI: 10.1016/j.jnca.2014.09.004.

[6] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th. Pearson Education, 2010, ISBN: 9780132126953.

[7] J. Postel, *User datagram protocol*, RFC 768, 1980.

[8] A. Gupta, K. Mamgain, M. Singh, and A. Chhabra, "Ns2 – based experimental analysis of throughput for tcp and udp traffic during link failure of the network," *Iraqi Journal of Science*, pp. 401–413, Jan. 2024. DOI: 10.24996/ijs.2024.65.1.33.

[9] Z. Shelby, C. Bormann, G. Stuber, S. Chakrabarti, E. Nordmark, and J. Hui, *RFC 6282: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*, https://www.rfc-editor.org/rfc/rfc6282.html, Internet Engineering Task Force (IETF), 2011.

[10] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Routing protocols for low power and lossy networks in internet of things applications: A review," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 465–10 481, 2020.

[11] *IEEE Standard for Low-Rate Wireless Networks*, IEEE Computer Society, Feb. 2020. DOI: 10.1109/IEEESTD.2020.9129949. [Online]. Available: https://standards.ieee.org/standard/802_15_4-2020.html.

[12] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The contiki-ng open source operating system for next generation iot devices," *SoftwareX*, vol. 18, p. 101 089, 2022, ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2022.101089. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711022000620.

[13] R. J. Bithray and D. Stavrov, "Designing hardware for a lunar wireless sensor network," Delft University of Technology, Tech. Rep., Jun. 2025.

[14] Texas Instruments, *CC2652R7 SimpleLink™ Multiprotocol 2.4 GHz Wireless MCU Datasheet*, https://www.ti.com/lit/ds/symlink/cc2652r7.pdf, Rev. B, published March 21, 2023, 2023.

[15] R. Karim, A. Iftikhar, B. Ijaz, and I. Ben Mabrouk, "The potentials, challenges, and future directions of on-chip-antennas for emerging wireless applications—a comprehensive survey," *IEEE Access*, vol. 7, pp. 173 897–173 934, 2019. DOI: 10.1109/ACCESS.2019.2957073.

[16] W. Kim, H.-G. Seok, G. Lee, *et al.*, “A fully integrated ieee 802.15.4/4z-compliant uwb system-on-chip rf transceiver supporting precision positioning in a cmos 28-nm process,” *IEEE Journal of Solid-State Circuits*, vol. 58, no. 12, pp. 3408–3420, 2023. DOI: 10.1109/JSSC.2023.3317433.

[17] S. Liu, Z. Shen, and W. Meng, “Cluster-based Wireless Sensor Network Deployment for Lunar Exploration,” in *2020 12th International Conference on Communication Software and Networks (ICCSN)*, IEEE, Jun. 2020, pp. 138–143, ISBN: 978-1-7281-9815-6. DOI: 10.1109/ICCSN49894.2020.9139098.

[18] S. Maneewongvatana and D. M. Mount, “Analysis of approximate nearest neighbor searching with clustered point sets,” *CoRR*, vol. cs.CG/9901013, 1999. [Online]. Available: https://arxiv.org/abs/cs/9901013.

[19] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.

[20] *Freeze (package) Python programs into stand-alone executables*, 2025. [Online]. Available: https://github.com/pyinstaller/pyinstaller.

[21] *Rasterio reads and writes geospatial raster datasets*, 2025. [Online]. Available: https://github.com/pyinstaller/pyinstaller.

[22] Duracell, *CR2032 Lithium Coin Battery Technical Datasheet*, https://www.duracell.com, Accessed: 2025-06-10, 2023.

[23] Python Software Foundation. “Extending python with c or c++.” Accessed: 2025-06-12, Python Software Foundation. (2024), [Online]. Available: https://docs.python.org/3/extending/extending.html.

[24] PyInstaller Development Team. “What pyinstaller does and how it does it.” Accessed: 2025-06-12. (2025), [Online]. Available: https://pyinstaller.org/en/stable/operating-mode.html.

[25] *MyPy: Optional static typing for Python*, 2025. [Online]. Available: https://github.com/python/mypy.

[26] *Ruff: An extremely fast Python linter and code formatter, written in Rust.* 2025. [Online]. Available: https://github.com/astral-sh/ruff.