Internet of Things (IoT)

by

Zain Ul Islam Siddiqui

Supervisor: Professor Jeremy Smith

Dissertation submitted to the University of Liverpool in partial fulfilment of the requirements for the degree of Master of Science in *Engineering in Microelectronic Systems and Telecommunications*.

September 2025

Declaration

I hereby declare that this thesis, entitled “Internet of Things IoT,” is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma at any university or equivalent institution.

Abstract

This thesis presents the design, evaluation, and implementation of low low-power, long-range wireless sensor network based on LoRa technology and ARM Cortex-M based microcontrollers, managed by Zephyr RTOS. The objective of this project was to develop a mesh network suitable for infrastructure-limited regions such as conservation areas, farmlands, or disaster-prone regions. A custom mesh protocol was developed with features including packet forwarding, suppression of duplicate packets, and time-to-live (TTL) enforcement to ensure a reliable multi-hop communication. Experiments confirmed that successful point-to-point, one-hop, and multihop communication was established, demonstrating the feasibility of extended coverage. Results highlight the protocol's robustness and scalability. The integration of Zephyr provided scheduling, modularity, and power management, confirming its viability for constrained IoT platforms. The work concludes with recommendations for improving the security, large-scale deployment, and contributing towards a resilient IoT network for remote applications.

Contents

Nomenclature

CRC	Cyclic Redundancy Check
CSS	Chirp Spread Spectrum
dBm	Decibel-milliwatts
EN	End Node
FEC	Forward Error Correction
GPIO	General-Purpose Input/Output
I2C	Inter-Integrated Circuit
IoT	Internet of Things
ISM	Industrial, Scientific, and Medical (frequency band)
LoRa	Long Range (radio modulation technology)
LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network
MAC	Media Access Control
mW	Milliwatt
mWh	Milliwatt-hour
PDR	Power Delivery Ratio
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
RTOS	Real Time Operating System
SPI	Serial Peripheral Interface
STM32	STMicroelectronics 32-bit Microcontroller
TTL	Time-to-Live
UART	Universal Asynchronous Receiver Transmitter

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Prof. Jeremy Smith, for his continuous guidance, encouragement, and valuable feedback throughout the course of this project. I am also thankful to the staff and academic team at the University of Liverpool for providing access to the necessary resources and technical support. Finally, I appreciate the support of my peers and family, whose encouragement has helped me stay focused and motivated.

List of Figures

List of Tables

1. INTRODUCTION

1.1 Background

The Internet of Things IoT has rapidly evolved into a transformative technology, enabling automation and real-time monitoring of environmental sensors. At the core of IoT are sensor networks, which connect distributed nodes to a central unit via wireless communication. With the growing demand for energy-efficient and timely data transmission, designing robust and low-power networks remains a significant engineering challenge.

Environmental and conservation applications explicitly highlight these challenges. Use cases include water quality monitoring in reservoirs, soil conditions across a large inaccessible terrain, wildfires, detecting logging, and providing early warnings in flood-prone areas. Such applications demand long-range, low-power, scalable, and cost-efficient communication infrastructures.

1.2 LoRa and Mesh Networking

LoRa is widely adopted as the physical layer technology for low-power Wide Area Networks (LPWANs). It employs chirp spread spectrum (CSS) modulation with forward error correction (FEC), enabling reliable, long-range communication over unlicensed ISM frequency bands [1]. Building on this, the LoRa Alliance developed LoRaWAN [2], an open protocol at the MAC and network layers that facilitates secure, scalable communication among the LoRa-enabled devices [3].

However, LoRaWAN uses a star of stars topology, which is highly dependent on internet-connected gateways. Which may or may not be available in remote or infrastructure-poor environments. The limitation of cloud connectivity encourages the use of LoRa mesh networks. Where nodes forward a packet over reconfigured paths dependent on active nodes (Figure 1.1). Mesh topologies extend coverage and reliability by relaying data across multiple nodes, overcoming obstacles such as terrain or foliage, and rerouting around failed nodes.

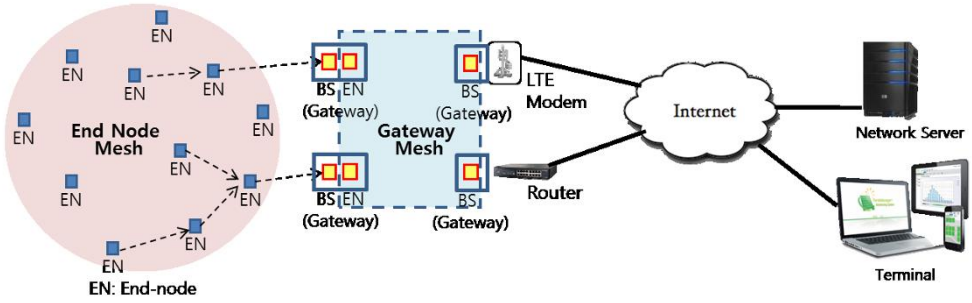

Figure 1.1 LoRa Mesh Networking [4]

Remote regions such as agricultural lands, conservation areas, national parks, and disaster-stricken zones suffer from a lack of reliable means of communication. Which is why there is a significant demand for communication networks that are both efficient, reliable, and long-range. These areas typically lack conventional cellular coverage or internet connectivity, making the deployment of conventional IoT technologies particularly challenging [5].

Accurate and timely monitoring of parameters concerning the safety of the people, animals, and the environment is critical for effective decision-making and early warning mechanisms. However, existing wireless technologies, including Wi-Fi, Sigfox, and GSM, have fallen short due to their limited communication range and high energy requirements. Figure 1.2 provides a comparison of communication technologies in terms of range and data transfer rate.

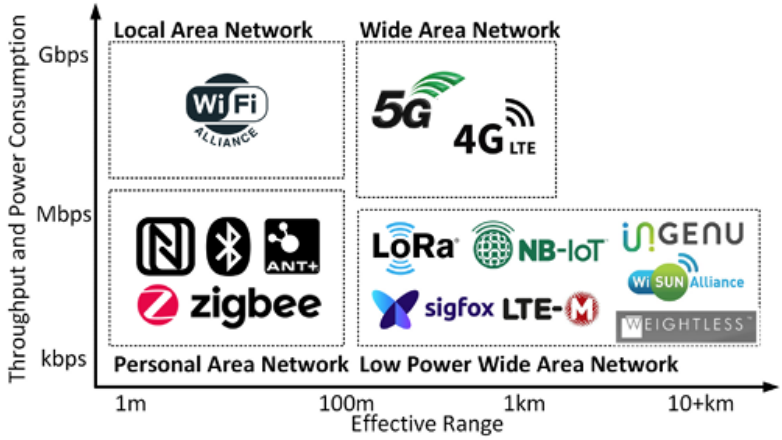

Figure 1.2 IoT Technologies in Relation to Range and Throughput [6]

A comparison of candidate low-power long-range technologies is presented in Table 1.1 [6]. This highlights the trade-offs among available LPWAN options and further motivates the selection of LoRa mesh for this project.

Table 1.1 Comparison of Existing Low Power Long Range Technologies [6]

Technology	Ownership	PHY	MAC	Spectrum License	Frequency Band	Modulation Used	Bandwidth (per	Estimated Coverage	Standards	Security
LoRa [77]	Proprietary	Open	Proprietary	Unlicensed	Sub-1 GHz ISM, 2.4 GHz ISM	CSS	125, 250, 500 kHz	Up to 15 km	LoRa WAN	AES 128
Sigfox [78]	Proprietary	Open	Proprietary	Unlicensed	Sub-1 GHz ISM	BPSK	100 Hz (EU), 600 Hz (US)	10–40 km	Proprietary	AES 128
LTE-M [64]	Open (3GPP)	Open	Open	Licensed	700–2600 MHz	BPSK, QPSK, 16QAM, 64QAM	1.4 MHz	~11 km	3GPP Release 12	LTE Security
NB-IoT [64]	Open (3GPP)	Open	Open	Licensed	700–2100 MHz	BPSK, QPSK	200 kHz	~15 km	3GPP Release 13	LTE Security
Weightless N/P/W [8]	Open	Open	Open	N/P: Unlicensed ISM W: Licensed TVWS	ISM: 868, 915 MHz TVWS: 400–800 MHz	N: BPSK P: FDMA, TDMA W: TDD	12.5 kHz, 100 kHz	Up to 2 km	Weightless Alliance	AES 128/256
Ingenu [39]	Proprietary	Proprietary	Proprietary	Unlicensed	2.4 GHz, Sub-1 GHz	RPMA (O-QPSK at 2.4 GHz, BPSK at Sub-1 GHz)	1 MHz	Up to 48 km	Proprietary	AES 128
Wi-SUN [61]	Open	Open	Open	Unlicensed	Sub-1 GHz ISM, 2.4 GHz ISM	O-QPSK, BPSK	200 – 1200 kHz	Up to 4 km	Wi-SUN Alliance	AES 128

While LoRa mesh topology offers significant potential for large-scale energy-efficient communication systems, existing research rarely touches on the use of the Zephyr Real-Time Operating System within LoRa mesh networks. Zephyr offers well-structured multitasking,

scheduling, and fine-grained power management, making it well-suited for low-power IoT applications. Zephyr offers greater flexibility across multiple hardware platforms and is backed by the Linux Foundation, ensuring long-term support. However, performance trade-offs in combining Zephyr with LoRa mesh network remain unexplored.

1.3 Objectives and Contributions

This project focuses on developing a LoRa-based mesh sensor network using Zephyr RTOS, targeting ARM Cortex-M microcontrollers. The main objectives are:

1. To select suitable embedded hardware and design appropriate software architecture.
2. To implement a wireless communication protocol for sensor-to-gateway interaction.
3. To evaluate system performance in terms of latency, reliability, and energy consumption.

The project will conclude with a functional prototype supported by performance metrics and documented design methodologies.

2. LITERATURE REVIEW

The Internet of Things (IoT) continues to push the development in the wireless communication domain. Central to this rapid development are low-power, wide-area technologies that enable data transfer across geographically dispersed sensor nodes. Among these are LoRa (Long Range) stands out for its ability to transmit data over vast distances (several kilometres) and energy efficiency. Making it one of the most widely used physical layer protocols for IoT. When paired with the LoRaWAN protocol, LoRa supports encrypted, addressable, and scalable communication for thousands of nodes per gateway, albeit with the trade-off of relatively low data rates [7]. This limitation confines its use to applications such as environmental monitoring and telemetry, where small, infrequent data packets are sufficient. Figure 1.1 illustrates a LoRa mesh network architecture.

Early work on extending LoRa's capabilities beyond traditional star topologies has explored multi-hop and mesh networking. One of the first such studies is LoRa Blink [8], which enabled multi-hop communication by introducing forwarding strategies among LoRa-enabled nodes transmitting to a central sink. Subsequent approaches [9,10] demonstrated that mesh-based communication significantly improves packet delivery rates compared to single-hop star topologies. These studies also adopt LoRaWAN terminology, using the term *gateway* instead of *sink*, with gateways coordinating communication using polling mechanisms. Similarly, [11] proposed a time-slotted, event-driven scheme to reduce packet collisions while maintaining reliable transfer across multiple hops.

Practical use cases further highlight the benefits of multi-hop LoRaWAN. For example, in maritime monitoring scenarios, ships outside the gateway's direct communication range (orange and red nodes) can relay data through intermediate nodes (green nodes), while additional opportunistic relays (yellow nodes) improve delivery reliability. This principle generalizes to stationary sensor deployments, where multi-hop routing enhances coverage and ensures reliable data transmission across challenging terrains. Figure 2.1 depicts a representative mesh network with a central gateway [12].

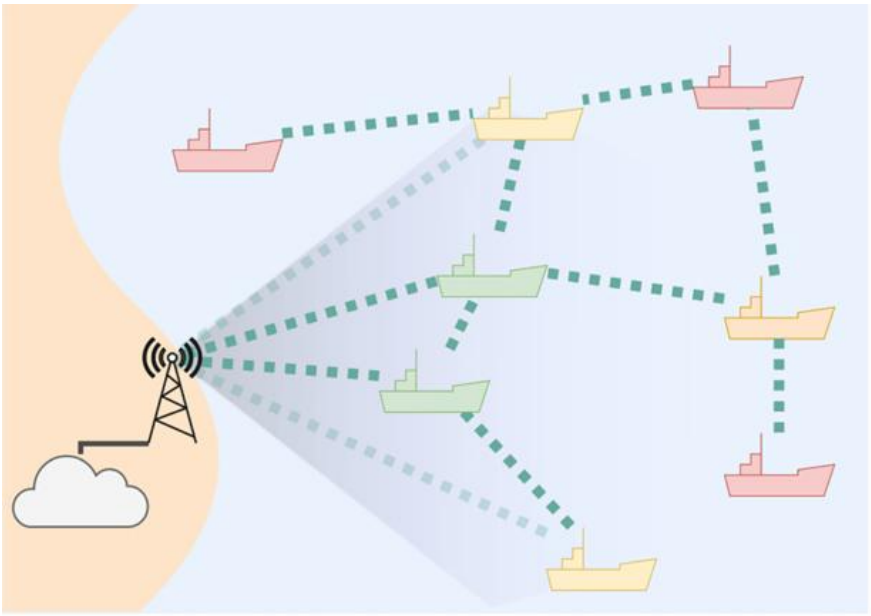

Figure 2.1 Mesh Network with Central Gateway [12]

Despite these advances, routing remains a core research challenge in LoRa mesh networks. Factors such as energy constraints, dynamic link quality, and node failures complicate protocol design. Recent studies classify routing approaches into proactive, reactive, and hybrid categories, each suited to different network contexts [13]. For example, Huang et al. [14] demonstrated that a modified AODV protocol could achieve reliable multi-hop communication up to 13.6 km, while Osorio et al. [15] showed that gossip-based routing improved packet delivery under non-line-of-sight conditions, albeit at the cost of increased latency. Other approaches [16,17,18,19,20] explore trade-offs between throughput, energy consumption, and reliability under varying topologies and workloads.

Routing protocols for stationary wireless networks typically include five core components: route discovery, route selection, route maintenance, data forwarding, and route metrics [21]. Mobile ad-hoc routing protocols are generally classified into three categories: proactive, reactive, and hybrid [22]. Within each category, many protocols exist, with variations and derivatives widely studied in the literature [21], [22], [23].

Proactive protocols such as Optimized Link State Routing (OLSR) [24] always maintain up-to-date routing tables, which is effective for large and dense networks but adds unnecessary overhead in smaller, static deployments such as agricultural LoRa networks. Reactive protocols such as Destination-Sequenced Distance Vector (DSDV) [25] discover routes only when needed, reducing control overhead but still requiring frequent updates in mobile scenarios [26]. For LoRa-based IoT applications with largely static nodes and limited mobility,

the continual updates of proactive or mobile-oriented protocols are often inefficient. Instead, lightweight routing mechanisms that minimize overhead while ensuring reliability are more suitable. Table 2.1 lists the recent works done on routing protocols for LoRa networks.

Table 2.1 Previous Works Related to Multi-hop Routing Protocols in LoRa Networks [6]

Ref	Author	Year	Proposed Name	Classification	Topology	Objective	Validation	Metrics Measured	Outcome	Limitation
[38]	Huang et al.	2018	Modified Ad-hoc On-demand Distance Vector (AODV)	Reactive	Linear	Test AODV on LoRa	Simulation	PDR, RSSI	Multi-hop up to 13.6 km with 99% PDR. Single hop reliable up to around 1.6 km.	LoRa constraints are not considered during route formation and RX/TX.
[53]	Lundell et al.	2018	Hybrid Wireless Mesh Protocol (HWMP) and AODV	Hybrid	Linear	Transmission efficiency and node lifetime.	Simulation and real-world	RSSI, Route construction duration	Able to uplink messages traversing a four-hop network.	Route failure handling, errors, timers, and ACKs have yet to be implemented
[42]	Kawabata et al.	2019	Converge cast	Hybrid	Tree	Assess Converge cast and Adaptation Duty Cycle Control (ADCC)using a combination of synchronous and asynchronous protocols.	Simulation and real-world	RX/TX energy, node count, node lifetime(h), to-sink delay(s)	Near nodes use synchronous, far nodes use asynchronous. Compared to ADCC, TX delay and lifetime increased by around 1.6 times, respectively	Synchronized protocol saves more power but requires accurate time sync, and far nodes have a higher chance of sync errors.
[25]	De Farias Medeiros et al.	2020	Distance Vector Routing (DVR), AODV, and Dynamic Source Routing (DSR)	Reactive and Active	Tree	Comparison of DVR, AODV, and DSR	Simulation	PDR, round-trip delay, throughput, power consumption	Reactive is better for dynamic networks with high throughput, proactive protocol for static networks.	Said protocols are unable to cope with high-speed mobile nodes due to infrequent route updates.
[88]	Yang et al	2021	FerryLink	Reactive	Linear	Enable on-demand packet relaying	Real world	Loss rate, detection accuracy	The loss rate is lowerthan1.6%, while the detection accuracy is above 98.8%.	The experiment only evaluated two hops. Unknown for more than two hops.
[66]	Osorio et al.	2022	MACGSP6	Reactive	Mesh	Apply gossip-based routing and evaluate end-to-end delay	Real world	PDR, Hop Count, Round-trip Delay	PDR improved from0% (single hop) toa53%inthe same network with NLoS conditions.	Everyhopadds3 s delay. Not recommended for time-sensitive data.
[36]	Hong et al.	2022	Hierarchical-based Energy Efficient (HBEE)	Reactive	Mesh	Evaluate PER and node lifetime in comparison to AODV	Simulation and real-world	PDR, PRR	Better power efficiency, lower round-trip delay. 95.7%and89.8% PRR on four-hop test during network formation and normal operation.	The current implementation has all radios constantly powered on.

LoRaWAN is highly efficient in terms of power and best suited for scaling IoT systems, but its reliance on internet-connected gateways limits its applicability in remote regions. This is a major constraint that has motivated research into decentralized mesh topologies, where multiple gateways and self-organizing nodes provide redundancy and extend coverage [27].

In parallel, the role of real-time operating systems (RTOSs) has grown in enabling structured, low-power IoT deployments. Zephyr RTOS offers modular scheduling, energy-aware task management, and broad hardware support on ARM Cortex-M microcontrollers. Compared with alternatives such as FreeRTOS, CMSIS, ThreadX, and RIOT [28], Zephyr [29] provides greater flexibility for integrating custom networking protocols [30]. However, existing research rarely combines LoRa mesh networking with Zephyr RTOS, leaving performance trade-offs and optimization strategies largely unexplored. This gap motivates the present project, which seeks to evaluate and extend the integration of LoRa mesh topologies with Zephyr in energy-constrained IoT sensor networks.

Belleza and Freitas [31] conducted a study on the feasibility and comparison of multiple RTOSs. Several benchmark tests were conducted to determine the best-performing RTOS. These benchmarks include task switching time, the time for getting and releasing a semaphore, time to pass and receive a message, time to pass a message between tasks, time to acquire and release a fixed-sized memory region, and finally, the time to activate a task from within an interrupt service routine. For commercial applications, open-source OS RIOT and Zephyr can be an advantage because, concerning IoT devices, privacy and security are fundamental, and open-source software can address these easily due to the collaborative way the development takes place.

3. DESIGN & ANALYSIS

LoRa serves as the backbone for the proposed mesh network design by offering long-range and low-power wireless connectivity, which is tailored for small payloads typically under 50kbps. Moreover, configurable parameters such as spread factor, bandwidth, and coding rate allow trade-offs between range, reliability, and throughput. Making LoRa suitable for energy-constrained IoT applications. These fundamentals are later discussed in Section 3.5, while the subsequent sections focus on the specific requirements and architecture of the system.

3.1 System Requirements

The project requirements were to design and develop a functional model of LoRa Mesh-Multi hop communication model, which is energy efficient, and provides real-time sensing. Figure 3.1 shows the system block diagram where as Figure 3.2 shows the structure of a single node.

3.1.1 Block Diagram

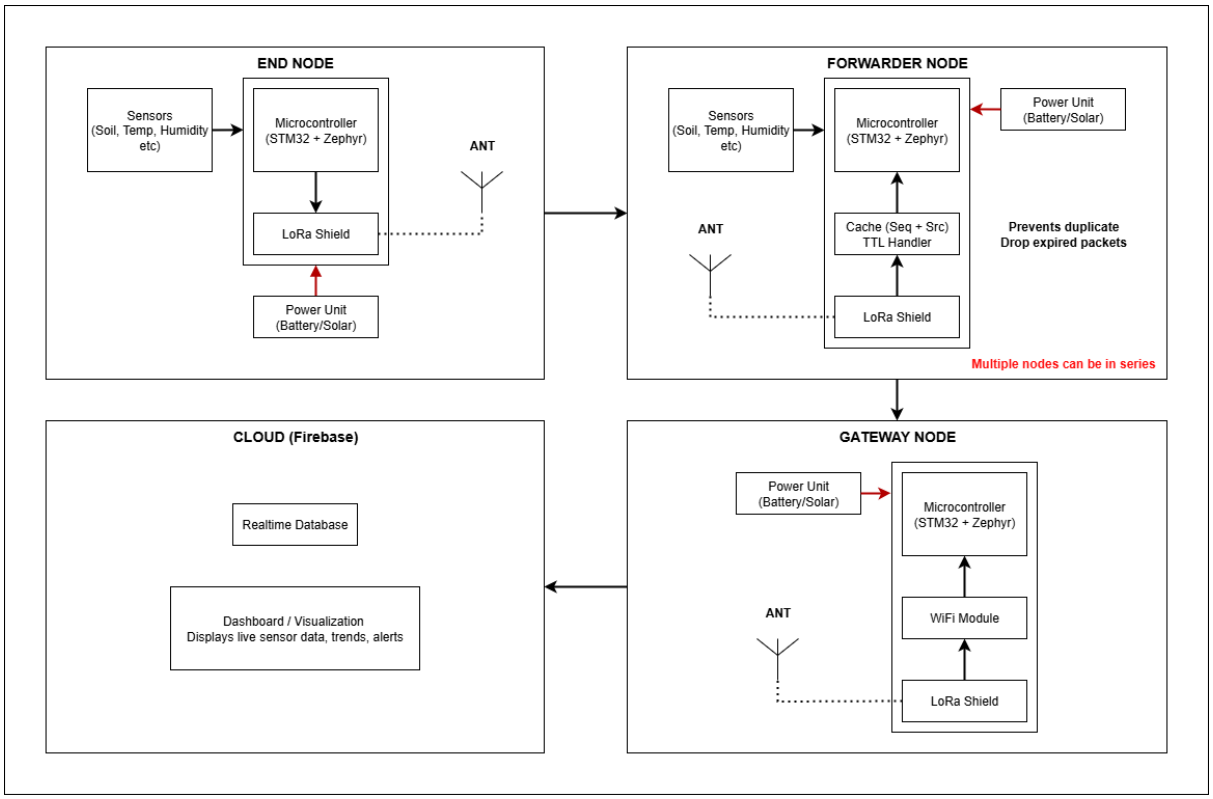

Figure 3.1 System Block Diagram

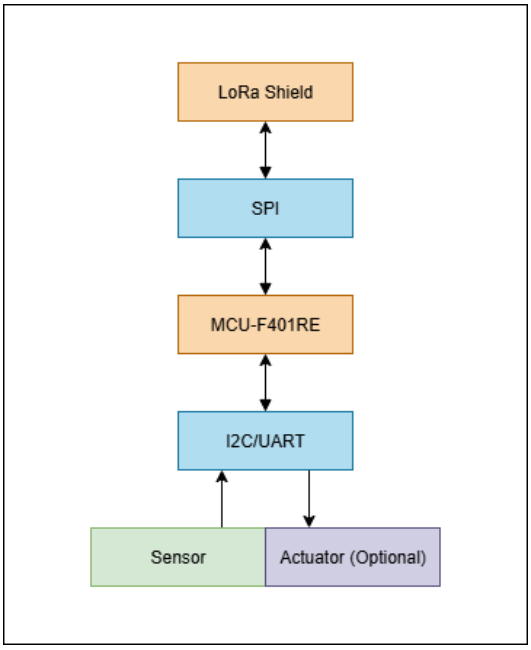

Figure 3.2 End Node Internal Block Diagram

3.1.2 Functional Requirements

The system must:

- Support environmental data reading by integrating sensors such as soil moisture, temperature, humidity, and air quality.
- Enable multi-hop wireless communication between widely distributed sensor nodes using LoRa, allowing for coverage extension in non-line-of-sight and remote areas.
- Facilitate data delivery to a central gateway, where information can be aggregated for further analysis.
- Ensure robustness against node or link failure, with the mesh topology dynamically rerouting packets through alternative paths.

3.1.3 Non-Functional Requirements

In addition to core features, the system should meet the following criteria for non-functional requirements:

- Energy efficiency: LoRa nodes must operate on a battery, with minimum consumption. Allowing the battery to last several years.
- Scalability: The design should be flexible to allow for multiple nodes to connect, typically ranging from 10 to 100 nodes.
- Cost: Both hardware and software must remain affordable to enable large-scale deployment.

- Modularity: Software design should be modular, allowing easy adaptation to new sensors or protocols.

3.1.4 Constraints

The following are the constraints in terms of hardware and software:

- Energy Limitations: Battery-powered nodes should minimize time on transmission.
- Communication Range: LoRa offers a long range but compromises on data rate.
- Regulatory Constraints: Following the ETSI standard, keeping the transmission duty cycle less than 1%. Transmitting 36 times or fewer every hour.
- Unreliable Links: Remote regions might cause excessive packet loss, requiring repeated transmission.
- Hardware Constraints: ARM Cortex-M microcontrollers come with limited memory and processing power.
- Environment: Deployment in outdoor conditions requires development to withstand harsh weather.

3.2 System Architecture

The proposed architecture consists of sensor nodes, a gateway, and a software stack developed over Zephyr. Figure 3.3 provides an overview of the proposed architecture.

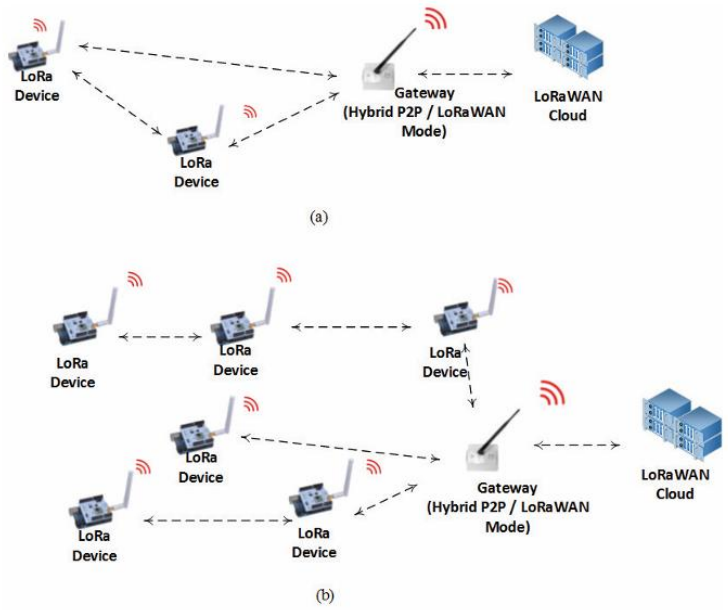

Figure 3.3 Mesh Topology (a) 2 Node Mesh Network (b) 5 Node Mesh Network [32]

Sensor Nodes: Each node integrates environmental sensors, an ARM Cortex-M-based microcontroller running Zephyr RTOS, and a LoRa transceiver. Nodes are designed for low-power operation, with extended sleep cycles and short active transmission windows. In addition to sensing, nodes participate in packet forwarding, enabling multi-hop mesh networking.

Gateway: The gateway functions as the data sink, aggregating packets relayed from multiple nodes. It provides local storage and optional backhaul connectivity via Wi-Fi. In scenarios where internet access is unavailable, the gateway maintains data locally for offline analysis. The gateway is built using a LoRa EN with an ESP32 microcontroller with Wi-Fi capability. The gateway uploads the data to a cloud server hosted by Firebase.

Mesh Network Topology: Nodes collaborate in a self-organizing mesh. Each node can serve as both a source of environmental data and a relay, forwarding packets toward the gateway. The proposed structure ensures a reliable connection against individual node failures and allows coverage across large and obstructed terrains.

Software Stack: A Lightweight mesh routing protocol is implemented over Zephyr, customized for LoRa's low data rate and long-range characteristics. Application layers handle sensor data collection, packet formatting, and transmission scheduling.

3.3 Hardware Architecture

The hardware platform consists of:

- **Microcontroller**: STM32 NUCLEO F401RE (ARM Cortex-M4) development boards for nodes and gateway. This board was selected as it is based on Cortex-M architecture and supported by Zephyr, while being relatively low-cost.

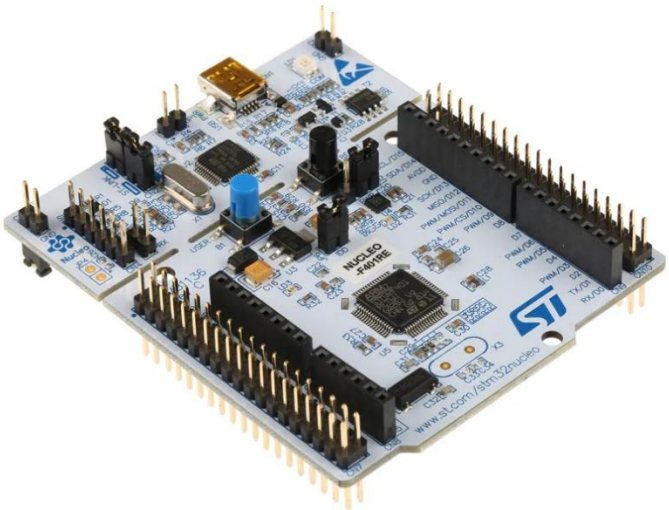

Figure 3.4 NUCLEO-F401RE [33]

- **Transceivers**: Semtech SX127x-series LoRa modules with a custom-built shield for long-range RF communication. The selection of this module was based on the availability of resources, application notes, detailed datasheets, and software development.

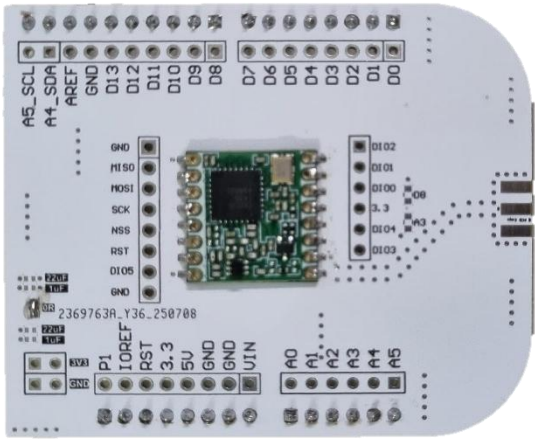

Figure 3.5 (a) Custom LoRa-SX1276 Shield (b) PCB Designed for LoRa Shield

- **Power Source**: Battery is used as a power source for nodes with a regulated 3.3V supply.
- **Sensors**: Environmental sensors such as temperature, humidity, motion, etc., will be connected to the EN.

3.4 Software

Development will be made using the **Zephyr OS**, chosen for its:

- Real-time deterministic behavior
- Energy-aware scheduling
- Built-in drivers and modular subsystems

Zephyr offers greater precision in managing thread timing, interrupts, and sleep cycle features compared to traditional bare-metal or lighter RTOSs.

3.5 Communication Model

LoRa is a proprietary communication protocol that uses chirp spread spectrum modulation technique [34], [35]. To initialize LoRa transmission, several parameters such as spreading factor (SF), bandwidth (BW), and coding rate (CR) are required.

3.5.1 Spreading Factor (SF)

LoRa employs multiple orthogonal SF ranging from SF 7 – 12. SF provides a trade-off between data rates and maximum range. Higher SF means a high data rate and a lower range. Each symbol is spread by a code length of 2^{SF} chips, shown in Table 3.1. At the transmitter node, the spreading code expands each bit of the information into a long chip sequence, which helps increase the robustness of the transmission against noise and interference. The receiver knows the same spreading code and multiplies it with the incoming signal to recover the original message contained within the packet.

Table 3.1 Spread Factor vs Chip Length [36]

Spreading Factor (SF)	Chip Length 2^{SF}
7	128
8	256
9	512
10	1024
11	2048
12	4096

3.5.2 Bandwidth (BW)

The LoRa protocol supports three bandwidth configurations: 125 kHz, 250 kHz, and 500 kHz, as illustrated in Figure 3.6. LoRa BW of 125kHz corresponds to a chip rate of 125 kcps.

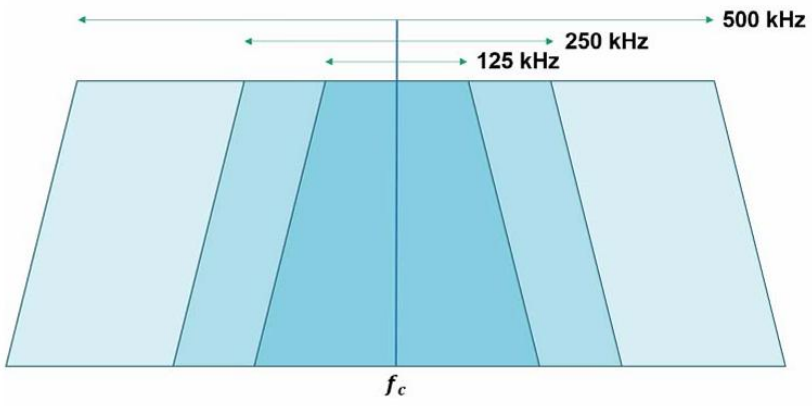

Figure 3.6 LoRa Bandwidth [36]

3.5.3 Coding Rate (CR)

The code rate defines the forward error correction (FEC). FEC is used to further increase the receiver sensitivity. LoRa offers a variable code rate of 0 to 4, where CR = 0 means no FEC. LoRa uses code rates in fractions, 4/5. 2/3, 4/7, and 1/2. Illustrated in Table 3.2 is the coding rate vs data rates. This means that if the code rate is denoted as k = n, where k stands for useful information and the encoder generates n number of output bits, then n-k will be the redundant bits.

Table 3.2 Coding Rate [36]

CR value	1	2	3	4
no. of redundant bits	1	2	3	4
Coding rate	4/5	2/3	4/7	1/2

The redundancy often allows the receiver node to detect and correct errors in the message. This decreases the effective data rate. Data rates corresponding to each CR value in LoRa are shown in Figure 3.7, over three bandwidths. As the CR value increases, the effective data rate decreases in each bandwidth spectrum.

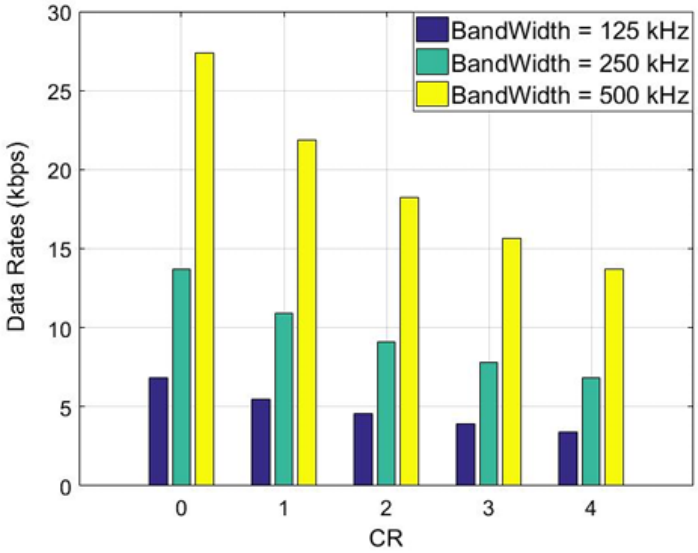

Figure 3.7 Data Rate vs BW vs CR [36]

3.5.4 Packet Structure:

LoRa offers a maximum packet size of 256 bytes. Table 3.3 shows the packet in the source code.

The LoRa packet is composed as follows [37]:

- Preamble field: is used for synchronization purposes. The receiver synchronized with the incoming data flow.
- Header field: depends on the choice of two available operation modes. In the default explicit operational mode, the number of bytes in the header field specifies the forward error correction (FEC) code rate, payload length, and the presence of CRC in the frame. The second implicit operational mode specifies that the coding rate and payload in a frame are fixed. In this mode, the frame doesn't contain this field, which gives a reduction in transmission time. The header field also contains a 2-byte CRC field, which allows the receiver to discard packets with an invalid header. The header field, along with its CRC field, is 4 bytes long and is encoded with a 1/2 coding rate, while the coding rate for the rest of the frame is specified in the PHY header. The first byte of the header field specifies the payload length.
- Payload field: Maximum payload varies from 2 to 255 bytes. This field further contains the following fields:
    - MAC header: defines the frame type (data or acknowledgment), protocol version, and its direction (uplink or downlink).
    - Payload: contains actual data.
    - MIC: is used as the digital signature of the payload.
    - CRC: is optional and comprises cyclic redundancy check (CRC) bytes for error protection for payload (2 bytes).

LoRa packet transmitted via our LoRa EN consists of the following:

Table 3.3 LoRa Packet Structure Breakdown

Field	**Type**	**Size (bytes)**	**Purpose**
src	uint8_t	1	Source node ID (0–255)
dst	uint8_t	1	Destination node ID (0–255)
seq	uint16_t	2	Sequence number for duplicate detection and ordering
ttl	uint8_t	1	Time-to-live (max hop count before packet is dropped)
len	size_t	4 (on ARM Cortex-M, 32-bit)	Length of the payload in bytes
payload	uint8_t []	Variable (0–MAX_MESH_PAYLOAD)	Actual application data (e.g., sensor readings, messages)
crc16	uint16_t	2	CRC-16 (polynomial 0x1021) for integrity verification

Packet size = 1 + 1 + 2 + 1 (hdr) + 4 (len) + N (payload) + 2 (CRC16) = 11 + N bytes.

Where **N** = payload length (*snprintk* call shows a short ASCII string, e.g., "*Hello from node X*") is ~20 bytes.

Max **N** is bounded by MAX_MESH_PAYLOAD. If set to 200, the packet tops out at 211 bytes.

TTL.

- Time To Live: In multi-hop/mesh networks, Time-To-Live (TTL) is a field included in each transmitted packet to specify a maximum number of hops that the packet can travel before being discarded. At every node, the TTL value is decremented by one. If the TTL reaches zero, the packet is dropped. This simple mechanism prevents packets from circulating indefinitely in the network due to routing loops or repeated retransmissions, thereby reducing congestion and conserving energy.

For a packet originating at node *i* with TTL = TTLinit, the effective TTL at the *k-th hop* is:

$$TTL_k = TTLinit - k \quad (1)$$

where TTL_k is the remaining TTL after *k* hops. A packet is forwarded only if:

$$TTL_k > 0 \quad (2)$$

Otherwise, the packet is dropped.

There are some practical constraints in LoRa mesh:

- Each hop consumes transmission time proportional to the Time-on-Air (ToA) of the packet.
- The maximum end-to-end lifetime of a packet in the network can be expressed as:

$$T_{end-to-end} = \sum_{k=1}^{N_{hops}} T_{tx,k} + T_{prop,k} + T_{proc,k} \quad (3)$$

where:

- $T_{tx,k}$: transmission time (ToA) at hop *k*.
- $T_{prop,k}$: propagation delay (generally negligible in LoRa due to low data rates vs. speed of light).
- $T_{proc,k}$: processing delay at the relay node.
- Maximum Reachable Hops

3.6 Zephyr RTOS Integration

The program is divided into three easily configurable files: main.c, mesh_handler.c, and mesh_handler.h which is a header file.

3.6.1 Main.c

The main file acts as the application entry point, responsible for initializing the mesh system. This is where the nodes are defined, whether a transmitter, receiver, or relay node is established. It sets the LoRa device, establishes the payload handler that recognizes incoming messages. If the node is established as a transmitter, it periodically constructs a packet containing the source ID, message, and assigns a header field such as source, destination, sequence number, and TTL, finally enqueuing it for transmission.

3.6.2 mesh_handler.c

The Mesh_handler.c file expands on the main.c file by encapsulating the core mesh networking logic, separating application code from protocol handling. It provides APIs for transmission and reception of packets, implements cyclic redundancy, and sequence checks, ensuring data integrity. Messages received via LoRa are validated, logged, and either delivered to a local node or forwarded if the TTL parameter permits. The file also configures the LoRa modem for both TX and RX operations and manages background threads that handle transmission. Overall, it provides the engine of the mesh protocol, removing the lower-level radio details.

3.6.3 mesh_handler.h

The header file defines the data structures, constants, and function prototypes required for mesh operation. Global variables and structures like message header format (source, destination, sequence number, TTL), maximum payload sizes, queue lengths, and utility macros are specified within this section. Mesh_handler.c serves as an interface between main.c and mesh_handler.c, enabling code modularity and clarity.

3.6.4 Flowchart

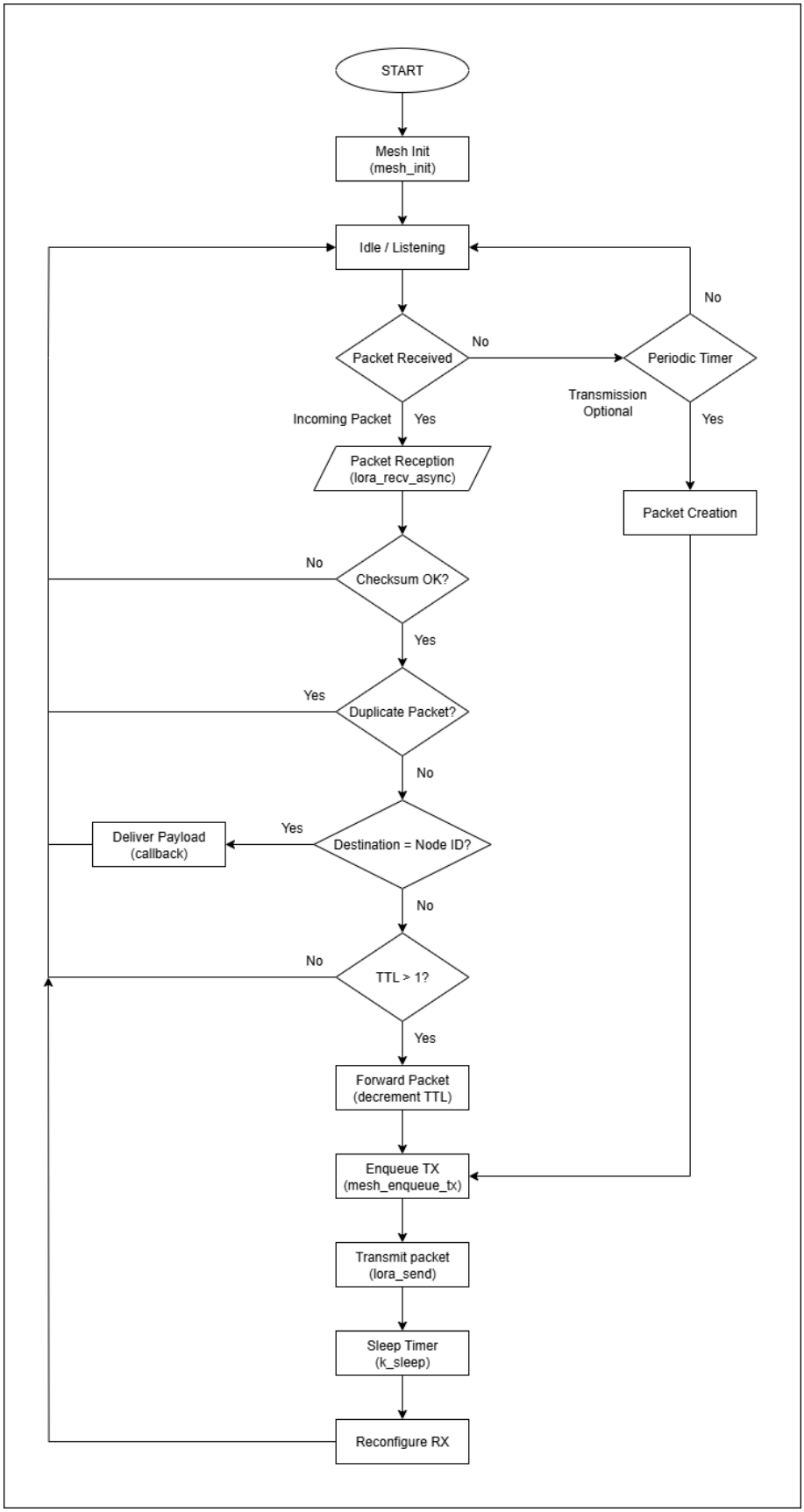

Figure 3.8 System Flowchart

3.6.5 Pseudo Code (Transmitter)

```
function transmitPacket(sensorData, destinationID):
    packet = createNewPacket()
    packet.src = NODE_ID
    packet.dst = destinationID
    packet.seq = getNextSequenceNumber()
    packet.TTL = MAX_TTL
    packet.payload = sensorData

    addToCache(packet)          // remember it locally to avoid duplicates
    enqueueForTransmission(packet)
```

3.6.6 Pseudo Code (Forwarder)

```
function forwardPacket(packet):
    if packet in cache or packet.TTL == 0:
        drop(packet)
    else:
        addToCache(packet)
        if packet.dst == NODE_ID:
            deliverToApplication(packet.payload)
        else:
            packet.TTL -= 1
            enqueueForTransmission(packet)
```

3.6.7 Pseudo Code (Gateway)

```
function handleAtGateway(packet):
    if packet in cache or packet.TTL == 0:
        drop(packet)
    else:
        addToCache(packet)
        if packet.dst == NODE_ID:
            storeInDatabase(packet.payload)    // save locally
            uploadToCloud(packet.payload)     // e.g., Firebase
        else:
```

```
// gateway usually does not forward
log("Packet for different node ignored")
```

3.6.8 System Configuration (prj.conf)

The Zephyr build system utilizes a Kconfig-based configuration file, called prj.conf, to enable or disable specific OS features and device drivers, which are compiled at runtime. This helps reduce memory usage and program overhead by utilizing only the required components built into the firmware, thereby tailoring the RTOS to the intended IoT application.

The configuration parameters used in this project are:

Table 3.4 Kconfig Parameters

Configuration Option	Explanation
CONFIG_LOG=y	Enables Zephyr's logging subsystem, which allows formatted logs to be output via UART/console. Used here for debugging mesh packet transmissions and receptions.
CONFIG_LORA=y	Enables support for the LoRa driver in Zephyr. Without this, the LoRa modem APIs (lora_send, lora_recv_async, etc.) would not be available.
CONFIG_PRINTK=y	Enables the lightweight printk() logging function for kernel-level debug messages, useful during development and low-level debugging.
CONFIG_ENTROPY_GENERATOR=y	Enables the entropy driver, which provides a source of hardware-generated randomness. Essential for operations like generating random IDs, testing, or simulating stochastic network behaviors.
CONFIG_TEST_RANDOM_GENERATOR=y	Provides a software-based pseudo-random number generator (PRNG). Useful for testing when no hardware entropy device is available.
CONFIG_HEAP_MEM_POOL_SIZE=4096	Defines a heap memory pool of 4096 bytes. This heap is used by dynamic memory allocation functions like k_malloc() and k_free() to handle variable-sized mesh packets.
CONFIG_KERNEL_MEM_POOL=y	Enables the kernel's memory pool subsystem, which manages memory chunks efficiently for dynamic allocation, ensuring real-time safety in constrained embedded environments.

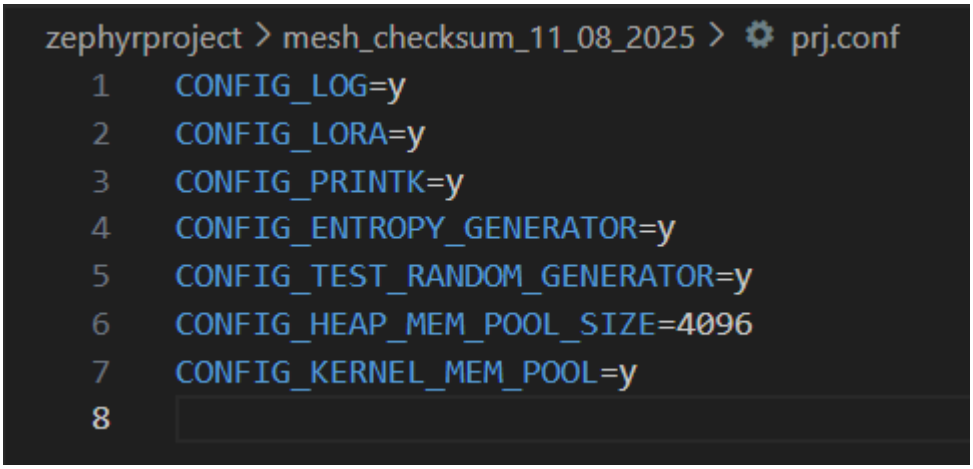

Figure 3.9 Prj.conf Snapshot

3.6.9 Overlay

Zephyr RTOS uses a file called the Device Tree to abstract hardware details such as peripherals, GPIO mappings, and external modules. Instead of hardcoding pin assignments in the application code, these are described in a DeviceTree overlay (`.overlay` file). This approach improves portability, readability, and makes it easier to adapt the firmware to different boards or transceivers.

The overlay used in this project configures the **Semtech SX1276 LoRa transceiver module** connected via SPI on an Arduino-compatible board. The relevant entries are:

Table 3.5 Overlay Parameters

Overlay Element	**Explanation**
aliases {lora0 = &lora_semtech_sx1276mb1mas; };	Defines an alias lora0 that refers to the SX1276 node. This alias is later used in code (e.g., DEVICE_DT_GET(DT_ALIAS(lora0))).
&arduino_spi {status = "okay"; ...}	Activates the Arduino-compatible SPI bus for communication with the LoRa module.
lora@0 {compatible = "semtech,sx1276"; reg = <0x0>;...}	Declares the SX1276 LoRa device at SPI address 0. The compatible property tells Zephyr which driver to use.

```
/ {
    aliases {
        lora0 = &lora_semtech_sx1276mb1mas;
    };
};

&arduino_spi {
    status = "okay";

    cs-gpios = <&arduino_header 16 GPIO_ACTIVE_LOW>;    /* D10 */

    lora_semtech_sx1276mb1mas: lora@0 {
        compatible = "semtech,sx1276";
        reg = <0x0>;
        spi-max-frequency = <DT_FREQ_M(1)>;

        reset-gpios = <&arduino_header 0 GPIO_ACTIVE_LOW>;   /* A0 */

        dio-gpios = <&arduino_header 8 (GPIO_PULL_DOWN | GPIO_ACTIVE_HIGH)>,       /* DIO0 is D2 */
                    <&arduino_header 9 (GPIO_PULL_DOWN | GPIO_ACTIVE_HIGH)>,       /* DIO1 is D3 */
                  <&arduino_header 10 (GPIO_PULL_DOWN | GPIO_ACTIVE_HIGH)>,      /* DIO2 is D4 */
                  <&arduino_header 11 (GPIO_PULL_DOWN | GPIO_ACTIVE_HIGH)>,      /* DIO3 is D5 */
                  <&arduino_header 14 (GPIO_PULL_DOWN | GPIO_ACTIVE_HIGH)>,      /* DIO4 is D8 */
                  <&arduino_header 15 (GPIO_PULL_DOWN | GPIO_ACTIVE_HIGH)>;      /* DIO5 is D9 */

        rfo-enable-gpios = <&arduino_header 4 GPIO_ACTIVE_HIGH>; /* RXTX_EXT is A4 */
    };
};
```

Figure 3.10 Overlay Snapshot

This overlay ensures that the Zephyr LoRa driver is correctly bound to the SX1276 hardware, enabling seamless SPI communication and interrupt handling.

3.7 Mesh Network Protocol Design

There are two major types of mesh networks used in LPWAN applications: hierarchical and flat. Where hierarchical protocols are based on dynamic routing and a level-based identification, flat-network protocols rely on concurrent transmissions. In our application, we have created a flat-based network that relays packets from the EN to the gateway through the available ENs between the gateway and the node.

Consider two ENs with source IDs 1 & 2 and a gateway with source ID 4, spread out with EN 1 covering the range with EN 2, and EN 2 covering the range with the gateway. For EN 1 to be able to pass on the data packet to the gateway, EN 2 should act as a forwarder/relay node, as shown in Figure 3.11.

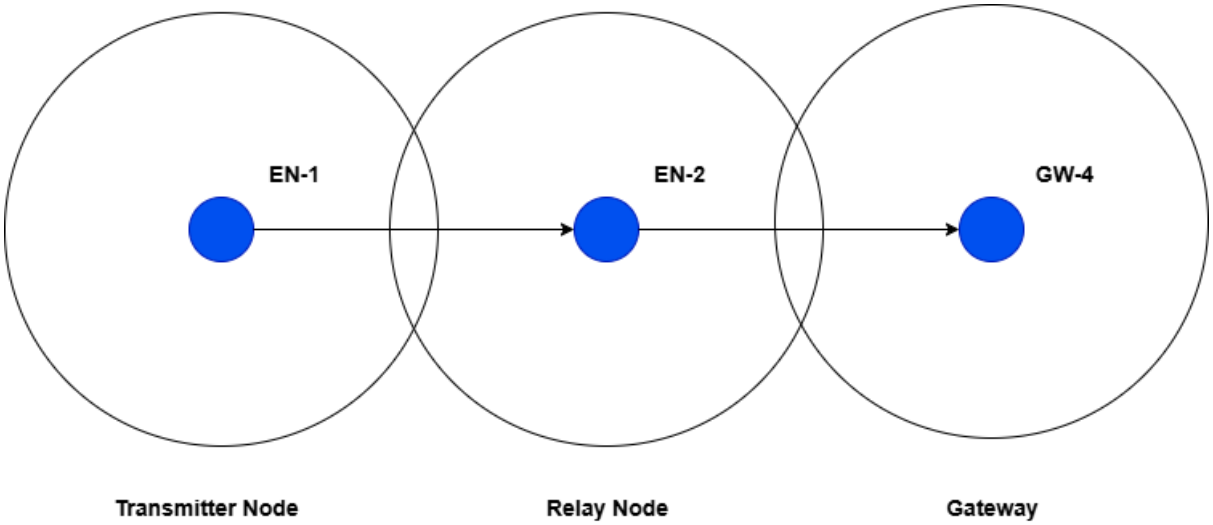

Figure 3.11 Three Node Setup

One of the major challenges in the algorithmic design of mesh networks is the avoidance of redundant traffic caused by multiple nodes re-transmitting the same packet repeatedly. If each edge node (EN) were to forward every received packet without discrimination, this would result in looping transmissions of identical packets, leading to severe network congestion and data loss. To overcome this, the proposed design incorporates a caching mechanism in which each EN stores the sequence number and the transmitter ID of recently received packets. This enables the node to identify and discard duplicates, thereby preventing unnecessary retransmissions.

In addition to the cache, a Time-To-Live (TTL) parameter is implemented within each packet. The TTL value is initialized when a packet is generated and decrements at every hop during its journey through the network, as expressed in Equation 1. See Figure 3.12. Since the maximum number of hops in any route cannot exceed the number of ENs along the longest path between an end node and the gateway, the TTL can be conservatively set to a fixed value. This ensures that undeliverable or looping packets are eventually discarded, further reducing congestion and improving the reliability of the mesh network.

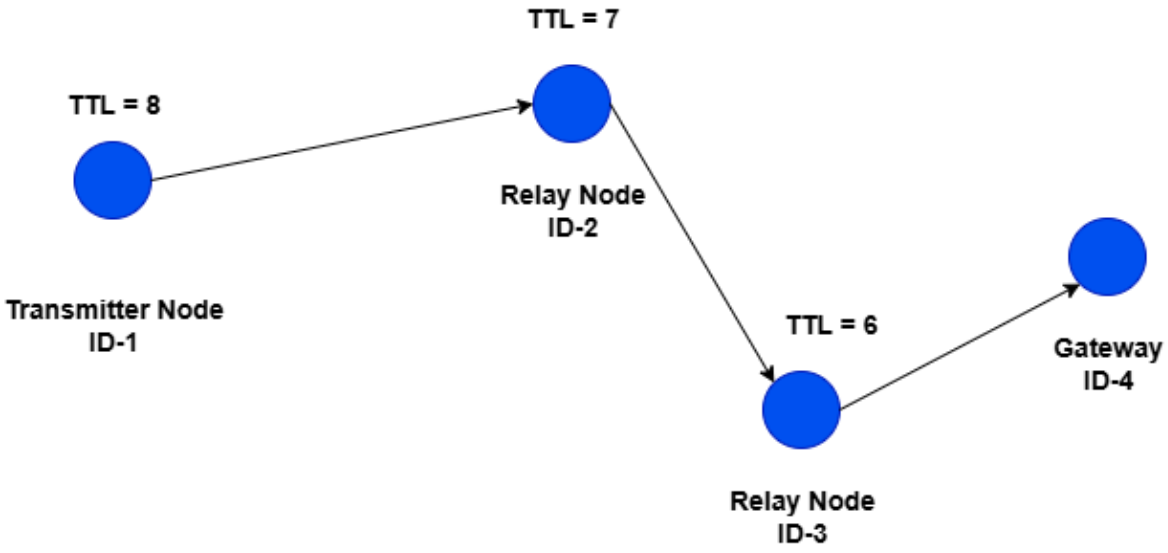

Figure 3.12 Four Node Setup

3.8 Security Considerations

The implemented mesh protocol incorporates basic integrity and reliability measures. Every packet is appended with a 16-bit cyclic redundancy check (CRC16) to verify the received message. Furthermore, a cache mechanism was implemented to make sure retransmission or malicious replay attempts are discarded. The Time-To-Live parameter further adds redundancy to this check by disabling infinite loops, preventing network flooding.

However, the system does not implement encryption or authentication, meaning that data confidentiality is not guaranteed. The node cannot identify the authentication of the sender, be it actual node or an imposter pretending to be an end node of our system. This limitation distinguishes the implementation from protocols such as LoRaWAN, which provide AES-128 encryption.

4. RESULTS

To evaluate the performance of the proposed LoRa-based mesh network, a series of experiments was conducted focusing on two primary aspects: communication range and multi-hop forwarding capability. These parameters directly reflect the system's suitability for deployment in real-world Internet of Things (IoT) applications such as environmental monitoring in remote areas. The experiments were designed to progressively validate different features of the system, beginning with basic point-to-point communication and extending to single-hop and multi-hop mesh functionality. The following three tests were conducted:

1. **Range Test with Two Nodes** to determine the maximum reliable communication distance between a source and a receiver.
2. **Hop Test with Three Nodes** to validate the one-hop forwarding mechanism using an intermediate node.
3. **Multi-Hop Test with Four Nodes** to demonstrate the scalability of the mesh protocol across multiple forwarding hops.

4.1 Range Test

The objective of the range test was to verify the maximum range between the two LoRa nodes. The initial experiment was performed indoors to evaluate the point-to-point range between two nodes. Reliable communication was observed up to approximately **2**00m, beyond which packet loss increased significantly as shown in Figure 4.1. The minimum recorded RSSI was –126 dBm, which approaches the sensitivity threshold for LoRa devices at SF12. While this range is below the kilometre-scale distances reported in outdoor LoRaWAN deployments, it reflects realistic constraints of operation in an urban setting and provides a baseline for evaluating multi-hop extension in subsequent tests. Limited range was observed due to a RF and antenna issues with the custom-built hardware.

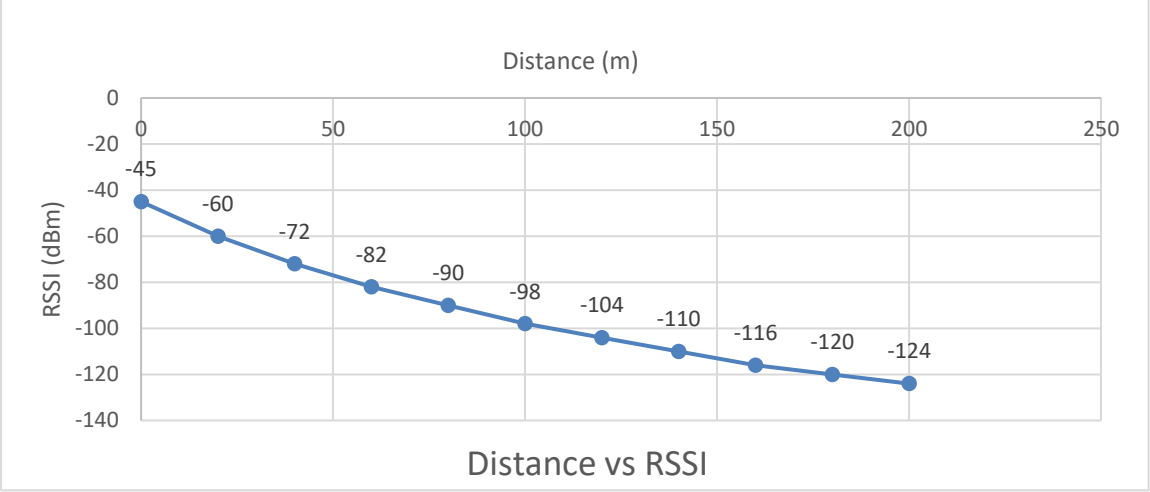

Figure 4.1 RSSI vs Distance

A typical packet transmitted by a node contains the following data:

Table 4.1 Received Packet Breakdown

Raw Packet			
01 10 53 09 08 54 DC A4 14 00 00 00 48 65 6C 6C 6F 20 66 72 6F 6D 20 6E 6F 64 65 20 31 49 53			
Hex Value	**Field**	**Bytes**	**Explanation**
01	Source (SRC)	1	Node ID = 1
10	Destination (DST)	1	Node ID = 16 (0x10)
53 09	Sequence Number	2	Decimal = 1827
08	Time-To-Live (TTL)	1	Initial TTL = 8
54 DC A4 14 00 00 00	Payload (LEN)	4	Decimal = 20 bytes
48 65 6C 6C 6F 20 66 72 6F 6D 20 6E 6F 64 65 20 31	Payload (ASCII/Hex)	20	"Hello from node 1"
49 53	Checksum (CRC16)	2	Ensures packet integrity

Figures 4.2, 4.3, & 4.4 show a decrease in RSSI with an increase in distance between the two nodes.

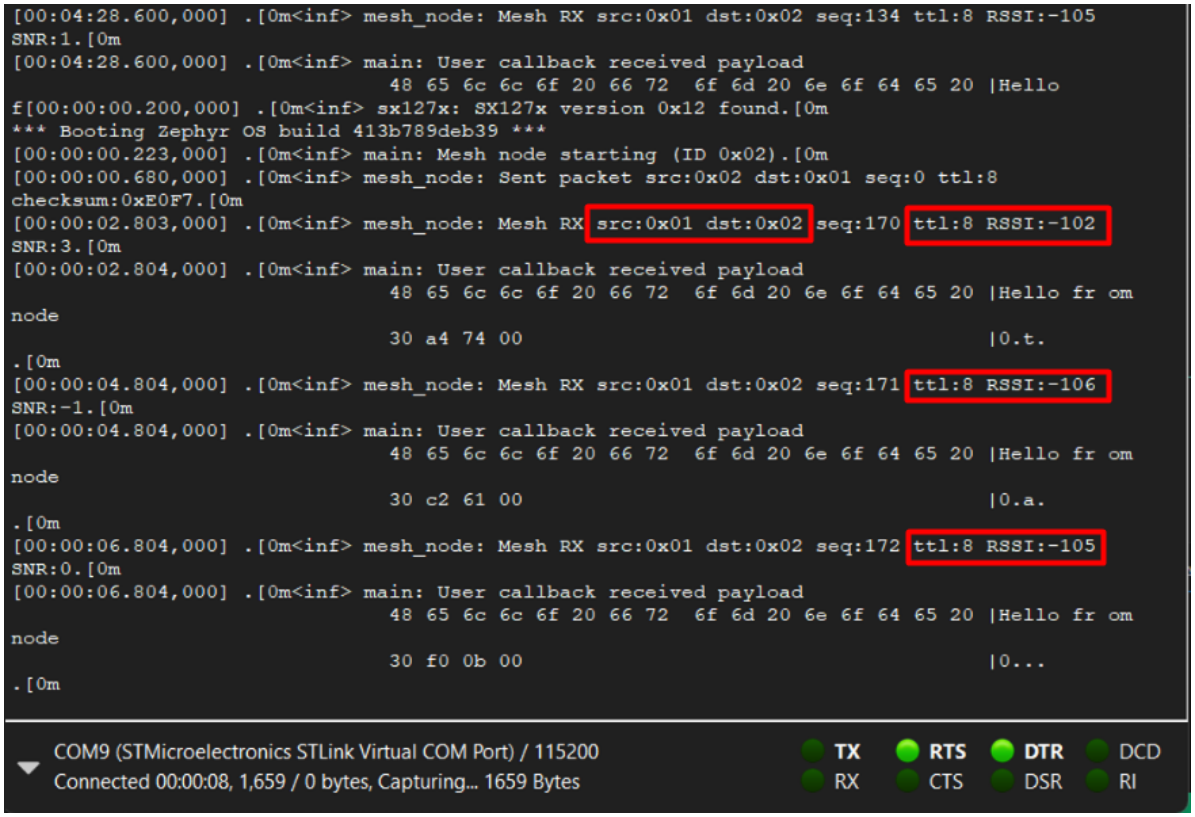

Figure 4.2 Test at 100m

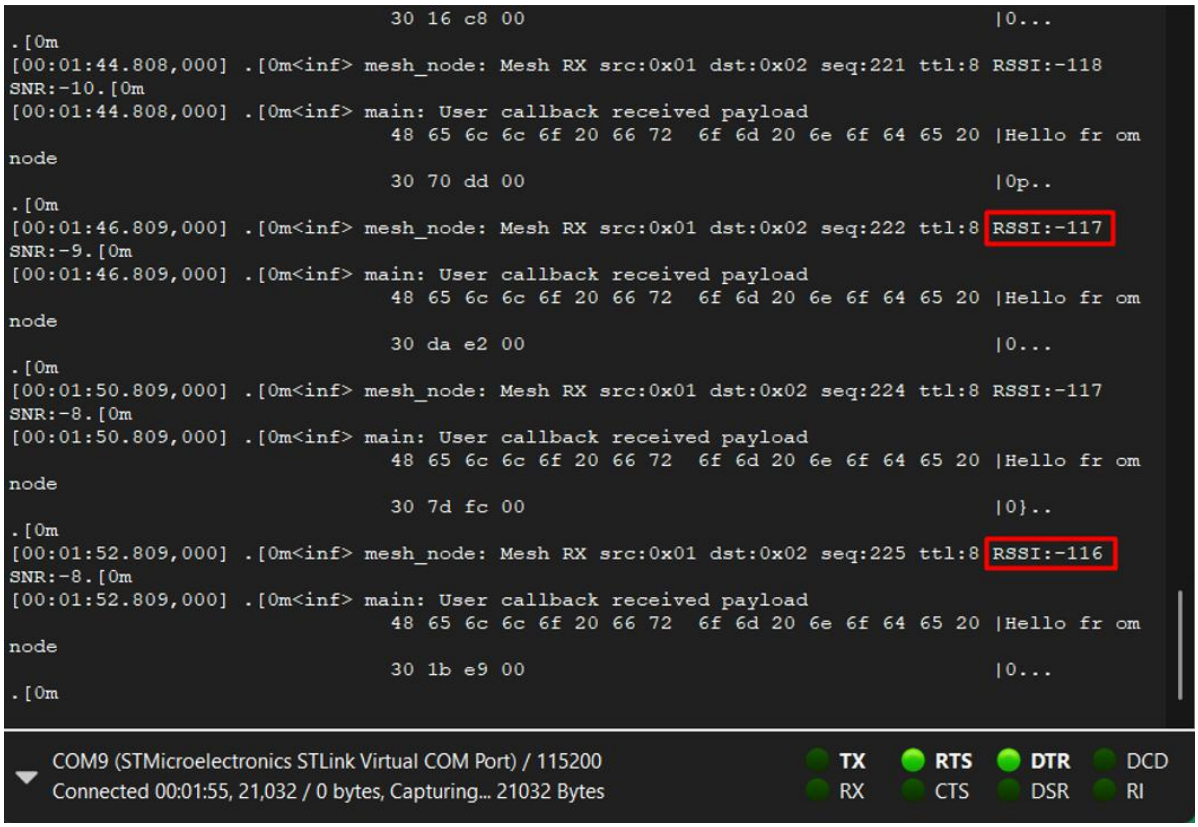

Figure 4.3 Test at 150m

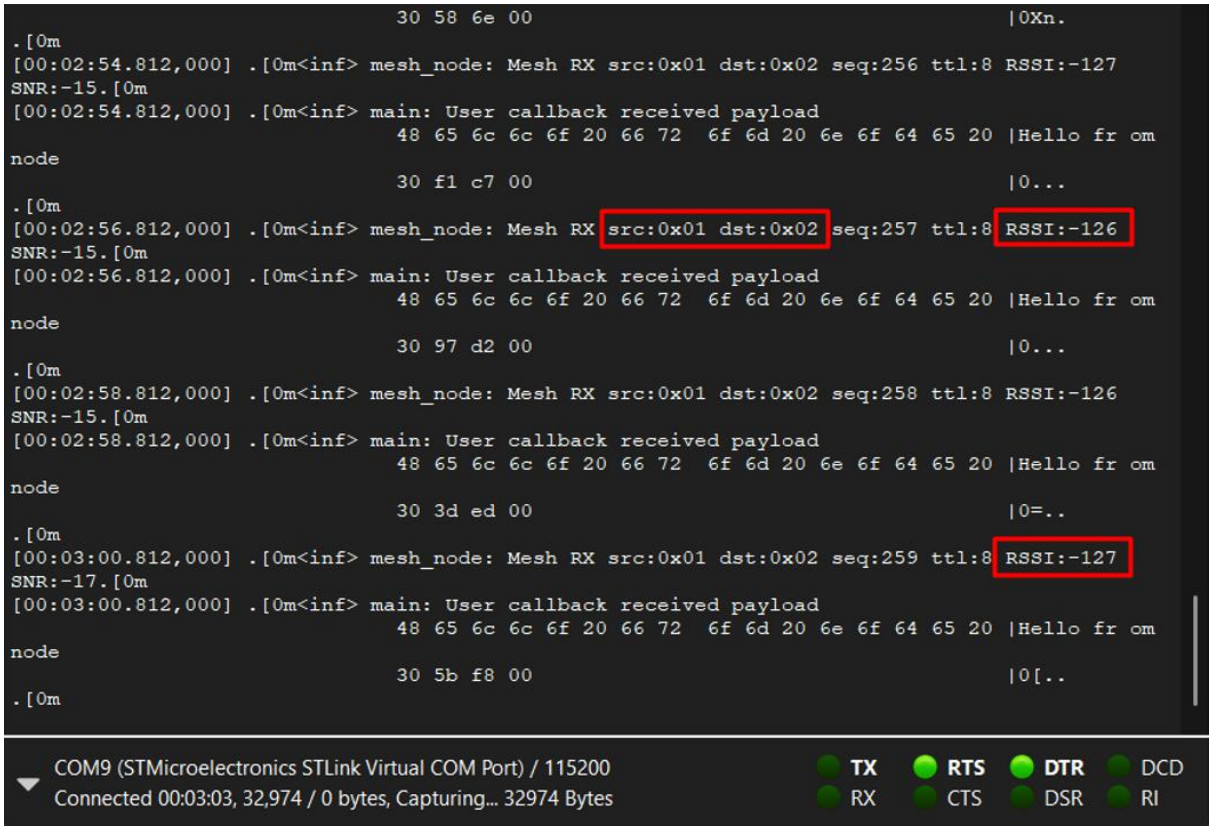

Figure 4.4 Test at 200m

4.2 Three-Node Hop Test

The purpose of this experiment was to validate the one-hop forwarding mechanism of the proposed mesh network. Three nodes were deployed in a linear arrangement: Node A (source) transmitted periodic packets, Node D (gateway/destination) received them, and Node B served as the forwarder. Node A and Node D were placed beyond each other's direct communication range under the chosen indoor conditions, making Node B essential to relay packets successfully. See Figure 3.12 for the topology.

Figure 4.5 shows a log excerpt from Node D (Gateway), demonstrating that it successfully received packets originating from Node A with a decremented TTL value. This confirms that the packets were forwarded through Node B, with the TTL correctly reduced at each hop to reflect the progression through the network. Furthermore, the cache mechanism implemented at Node B prevented the retransmission of duplicate sequence numbers, ensuring that each packet was forwarded only once.

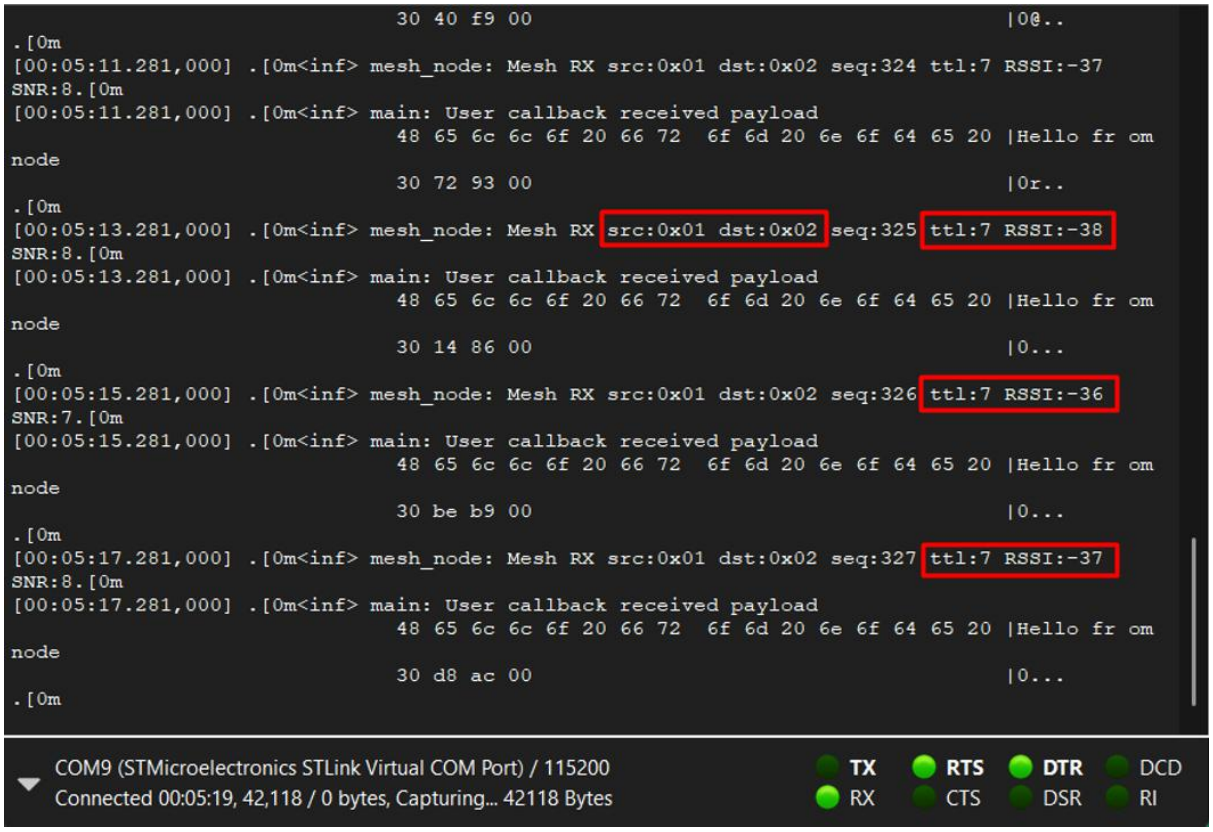

Figure 4.5 Three Hop Test Log

This test demonstrates the successful operation of the one-hop forwarding mechanism. It confirms that the proposed design allows intermediate nodes to relay packets while maintaining loop prevention through TTL enforcement and cache-based duplicate detection. This provides a solid foundation for scaling the network to multi-hop scenarios. A test conducted to determine the PDR for the 3 Node system yielded the results depicted in Table 4.1. PDR turns out to be ~70%.

Table 4.2 PDR

Packet Sequence	TTL at Node A	TTL at Node B	Received at D (Gateway)
1	-	-	No
2	8	7	Yes
3	-	-	No
4	8	7	Yes
5	8	7	Yes
6	8	7	Yes
7	8	7	Yes
8	8	7	Yes
9	8	-	No
10	8	7	Yes

4.3 Multi-Hop Test

This experiment aimed to demonstrate the scalability of the proposed mesh protocol by validating multi-hop packet delivery across a chain of four nodes. The test specifically sought to confirm that packets could traverse multiple forwarding hops, with TTL values correctly decremented at each stage, and that the system maintained reliable communication without introducing loops or excessive redundancy.

Four nodes were deployed in a linear topology as in Figure 3.12. Node A served as the source, Nodes B and C acted as intermediate forwarders, and Node D functioned as the gateway (destination). Each node was positioned so that it could only communicate with its immediate neighbor, ensuring that successful delivery to the gateway required two consecutive forwarding operations. Figure 4.6 shows the log received at the gateway during the test, which indicates a successful reception of the packet after four hops.

```
RSSI : -102
Raw packet: 01 10 E2 04 06 25 99 82 14 00 00 00 48 65 6C 6C 6F 20 66 72 6F 6D 20 6E 6F 64 65 20 31 CE C0
SRC: 1, DST: 16, SEQ: 1250, TTL: 6, LEN: 20
Payload (hex): 48 65 6C 6C 6F 20 66 72 6F 6D 20 6E 6F 64 65 20 31 CE C0 00
Payload (ascii): Hello from node 1ÎÀ
Pushing payload for node 1...
Firestore Success: {
  "name": "projects/lora-mesh-network-51116/databases/(default)/documents/nodes/node_1/history/data_1756995299",
  "fields": {
    "timestamp": {
      "timestampValue": "2025-09-04T14:14:59Z"
    },
    "payload": {
      "stringValue": "Hello from node 1  "
    }
  },
  "createTime": "2025-09-04T14:15:00.196790Z",
  "updateTime": "2025-09-04T14:15:00.196790Z"
}

RSSI : -101
Raw packet: 01 10 E5 04 06 25 99 82 14 00 00 00 48 65 6C 6C 6F 20 66 72 6F 6D 20 6E 6F 64 65 20 31 FC AA
SRC: 1, DST: 16, SEQ: 1253, TTL: 6, LEN: 20
Payload (hex): 48 65 6C 6C 6F 20 66 72 6F 6D 20 6E 6F 64 65 20 31 FC AA 00
Payload (ascii): Hello from node 1üª
Pushing payload for node 1...
```

Figure 4.6 Multi-Hop Test Log

While running a 4-node test, Node E with a stronger range was powered on between Node B and Node C. Topology is shown in Figure 4.7. After powering up Node E, the packet was directed towards it rather than taking the longer path through Node B & C. This shows that the system can self-identify the shortest path and reroute the packet. This is confirmed by the increase in TTL value from 6 to 7.

Figure 4.8 presents log excerpts from Node D (Gateway). The results show Node A transmitting a packet with an initial TTL value of 8. At the gateway, the packet was observed with the expected TTL value of 7, confirming proper hop-by-hop progression via Node E. The cache mechanisms at intermediate nodes prevented the retransmission of duplicate sequence numbers, ensuring that packets were forwarded only once per node.

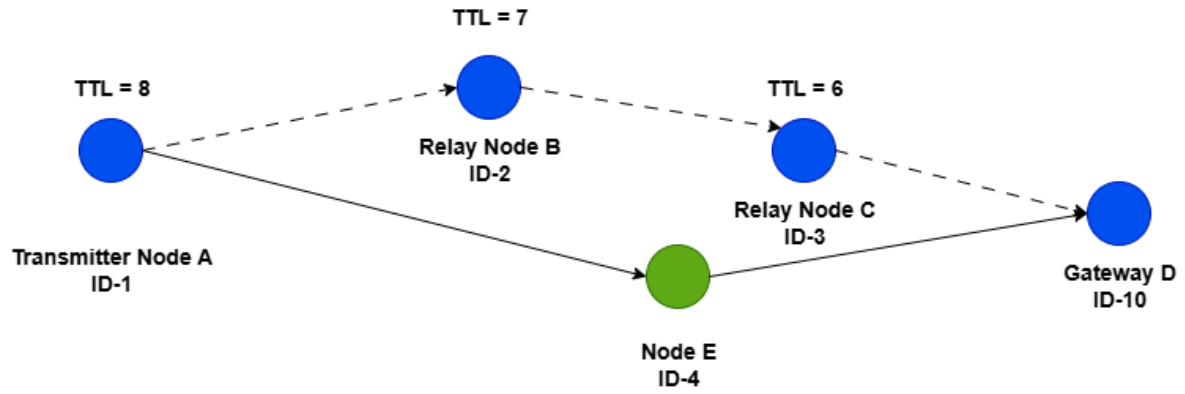

Figure 4.7 Five Node Setup

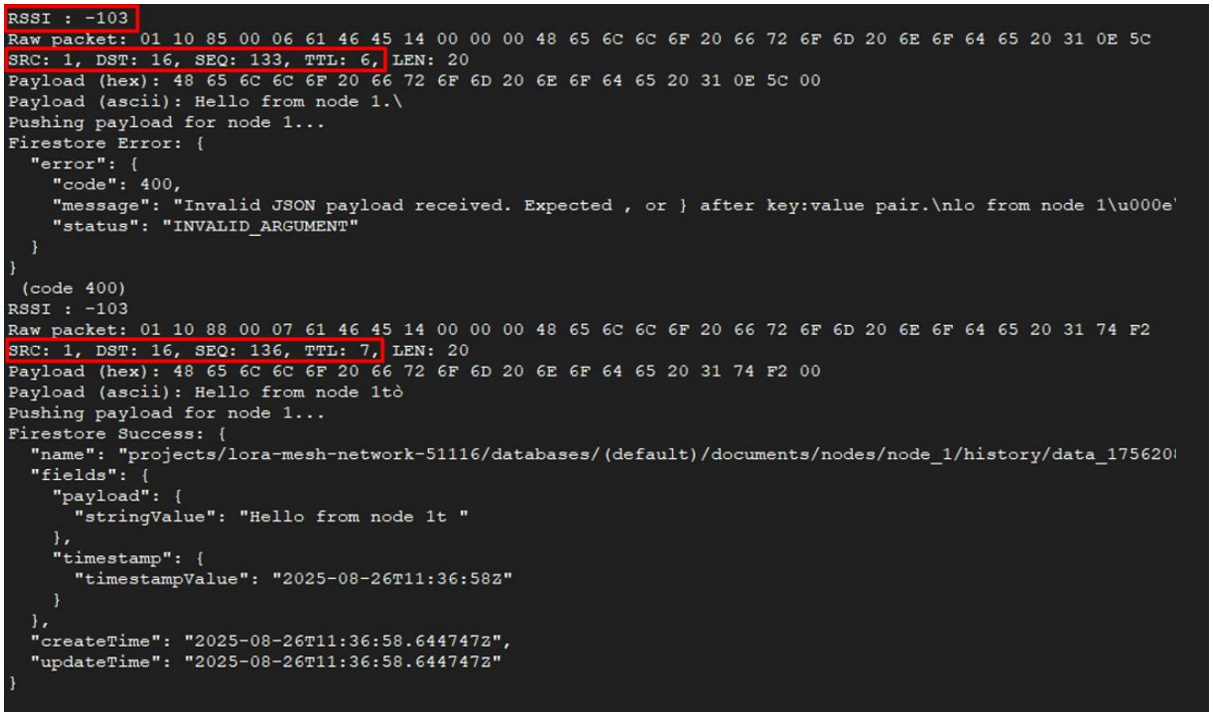

Figure 4.8 Five Node Setup Log

This test confirms that the designed mesh protocol scales effectively to multi-hop scenarios. Packets were successfully relayed across multiple nodes while maintaining integrity, reliability, and loop prevention. These results validate the robustness of the protocol in scenarios where direct communication is not feasible and highlight its suitability for large-scale IoT deployments in range-limited or infrastructure-poor environments.

5. CONCLUSIONS

The project successfully demonstrated the design and implementation of a LoRa Mesh network built using Zephyr OS, designed for resource-constrained IoT applications such as environmental monitoring in remote regions. Through the integration of an STM32 ARM Cortex-M4-based microcontroller, RFM95 an SX1276 LoRa transceiver, and a custom mesh network logic, the system achieved functional point-to-point, one-hop, and multi-hop communication.

The hop test with three nodes validated the forward/relaying capability of the intermediate nodes. Results showed TTL decrease accurately and the suppression of duplicate packets via the cache implementation. This confirmed that the developed protocol effectively prevented redundant traffic and retransmission.

The multi-hop test was an extension of the one-hop test, where one node acts as a transmitter and three nodes act as relay nodes. The logs added above confirmed the TTL progression from 8 down to 6, showing that the packets are travelling through each of the intermediate nodes, finally reaching the gateway.

From a system design perspective, the integration of Zephyr greatly improved the development process as it added modularity, allowed efficient task scheduling, and reliable memory management. The use of CRC16 checksums, TTL, and cache contributed to a robust protocol. Ensuring packet integrity and preventing routing loops. However, the lack of packet encryption or source authentication leaves a major drawback in this implementation that can be tackled in a later study.

In summary, the project has achieved its primary objectives of demonstrating functional mesh communication on a low-power embedded platform. It has provided proof of concept for energy-efficient multi-hop IoT networks suitable for deployment in environments where conventional infrastructure is unavailable.

6. RECOMMENDATIONS

While the system achieved its primary objectives successfully, there are several areas of research that need to be expanded upon for a fully functional and deployable system. First, energy profiling should be conducted to quantify the power usage of each node during transmit, receive, and idle states. Measuring duty cycles of sleep, transmit, and receive modes will provide the data for battery consumption and allow for fine-tuning for years of long battery life.

Another important aspect that needs development is security. The current design ensures packet integrity through CRC checks but does not provide encryption. Introducing lightweight security features such as AES-128 and message authentication codes would strengthen against eavesdropping and spoofing. Furthermore, the mesh protocol design can be extended by incorporating adaptive routing strategies. While the current flat mesh protocol showcases reliable results, adaptive strategies could be integrated to allow for a hierarchical mesh network.

The system would also benefit from testing at larger scales. Expanding the testbed to include tens of nodes would enable thorough testing and a complete evaluation of parameters such as RSSI, SNR, PDR, latency, and congestion under realistic conditions. Such large-scale experiments would help identify bottlenecks that might not occur in indoor or small-scale testing.

Finally, future iterations should integrate actual sensor hardware, such as soil moisture, air quality, or water level sensors, to demonstrate the practical applicability of the system in real-world monitoring scenarios and to validate its effectiveness beyond synthetic message exchange.

7. REFERENCES

1. Augustin. A, Yi. J Clausen. T Townsley. “W.M., A Study of LoRa: Long Range & Low Power Networks for the Internet of Things,” Sensors, vol. 16, pp. 1466, 2016.
2. Haxhibeqiri. J De Poorter. E Moerman. I, Hoebeke. J. “A Survey of LoRaWAN for IoT: From Technology to Application,” Sensors, vol. 18, pp. 3995, 2018.
3. Marais. J. M Malekian. R. Abu-Mahfouz, A.M. “LoRa and LoRaWAN testbeds: A review,” IEEE AFRICON, pp. 18–20, 2017.
4. Heon Huh, and Jeong Yeol Kim. “LoRa-based Mesh Network for IoT Applications,” IEEE 5th World Forum on Internet of Things (WF-IoT), pp. 525, 2019.
5. Welsh Government, “The Internet of Things with LoRaWAN,” *Farming Connect*, 2023.
6. Andrew Wong, M Hasan, and Salman Fattah. “Multi-Hop and Mesh for LoRa Networks: Recent Advancements, Issues, and Recommended Applications,” ACM Computing Surveys, Vol. 56, No. 6, Article 136, 2024.
7. R. Raymundo Belleza and E. de Freitas Pignaton. “Performance study of real-time operating systems for Internet of Things devices," *IET Software*, Vol. 12, No. 3, pp. 176-182, 2018.
8. Bor. M Vidler. J. E Roedig. U. “LoRa for the Internet of Things,” International Conference on Embedded Wireless Systems and Networks (EWSN), 2016.
9. Liang. Q.W., Zeng. G. J Lin. J. H Lee. H. C. “Demo Abstract: A LoRa Wireless Mesh Networking Module for Campus-scale Monitoring,” ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 2017.
10. Lee. H. C Ke, K. H. “Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System Design and Evaluation,” IEEE Trans. Instrum. Meas., Vol 67, pp. 2177–2187, 2018.
11. Tzimotoudis. P., Keranidis. S Kazdaridis. G, Symeonidis. P., Korakis. “T. LoRa Mesh Network Experimentation in a City-Wide Testbed,” International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization, 2019.
12. R. P. Centelles, F. Freitag, R. Meseguer, and L. Navarro. “Beyond the Star of Stars: An Introduction to Multihop and Mesh for LoRa and LoRaWAN,” *IEEE Pervasive Computing*, Vol. 20, No. 2, pp. 63-72, 2021.
13. Edouard Lumet, Antonin Le Floch, Rahim Kacimi, Mathieu Lihoreau, and André-Luc Beylot. “LoRaWAN relaying: Push the cell boundaries,” 24th International ACM

Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'21), 2021.

14. Hsiang-Yu Huang, Kai-Sheng Tseng, Yu-Lun Chiang, Jen-Cheng Wang, Yu-Cheng Yang, Cheng-Ying Chou, and Joe Air Jiang. "A LoRa-based optimal path routing algorithm for smart grid," 12th International Conference on Sensing Technology (ICST'18), IEEE, 2018.
15. Alfonso Osorio, Maria Calle, Jose Soto, and John E. Candelo-Becerra. "Routing in LoRa for smart cities: A gossip study," Fut. Gener. Comput. Syst. Vol. 136, pp. 84–92, 2022.
16. Shunroku Kawabata, Raito Matsuzaki, and Hiroyuki Ebara, "Mixed synchronous and asynchronous duty-cycling protocol in sensor networks," 48th International Conference on Parallel Processing, ACM, 2018.
17. Daniel Lundell, Anders Hedberg, Christian Nyberg, and Emma Fitzgerald. "A routing protocol for LoRA mesh networks," A World of Wireless, Mobile and Multimedia Networks, (WoWMoM'18), 2018.
18. Douglas de Farias Medeiros, Mariana Rodrigues Villarim, Fabricio Braga Soares de Carvalho, and Cleonilson Protasio de Souza. "Implementation and analysis of routing protocols for LoRa wireless mesh networks," Electronics and Mobile Communication Conference (IEMCON'20). IEEE, 2020.
19. Jing Yang, Zhenqiang Xu, and Jiliang Wang. "FerryLink: Combating link degradation for practical LPWAN deployments," IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS'21). 2021.
20. Shengguang Hong, Fang Yao, Yulong Ding, and Shuang-Hua Yang, "A hierarchy-based energy-efficient routing protocol for LoRa-mesh network," IEEE IoT J. 9, pp. 22836–22849, 2022.
21. M. Lee, J. Zheng, X. Hu, H. Juan, C. Zhu, Y. Liu, J. Yoon, and T. Saadawi, "A new taxonomy of routing algorithms for wireless mobile ad hoc networks: The component approach," IEEE Communications Magazine, vol. 44, no. 11, pp. 116–123, 2006.
22. G. Walikar and R. Biradar, "A survey on hybrid routing mechanisms in mobile ad hoc networks," Journal of Network and Computer Applications, Vol. 77, pp. 48–63, 2017.
23. E. Alotaibi and B. Mukherjee, "Survey paper: A survey on routing algorithms for wireless ad-hoc and mesh networks," Computer Networks, Vol. 56, pp. 940–965, 2012.
24. T. Clausen and P. Jacquet. "Rfc 3626: Optimized link state routing protocol (OLSR)," 2003.

25. C. E. Perkins and P. Bhagwat, “Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers,” ACM SIG COMM computer communication review, Vol. 24, No. 4, ACM, pp. 234–244, 1994.
26. S. Mohapatra and P. Kanungo, “Performance analysis of AODV, DSR, OLSR, and DSDV routing protocols using NS2 simulator,” Procedia Engineering, Vol. 30, pp. 69–76, 2012.
27. Cotrim, J. R., & Kleinschmidt, J. H., “LoRaWAN Mesh Networks: A Review and Classification of Multihop Communication,” *Sensors*, *20*(15), pp. 4273, 2020.
28. Baccelli, E., Hahm, O., Gunes, M., et al. “RIOT OS: towards an OS for the internet of things,” IEEE Conf. on Computer Communications Workshops (INFOCOM WKSHPS), pp. 79–80, 2013.
29. L. Foundation: ‘Zephyr project’, available at https://www.zephyrproject.org/, accessed 1 October 2016.
30. Hahm, O., Baccelli, E., Petersen, H., et al. “Operating systems for low-end devices in the internet of things: a survey,” IEEE Internet Things J., 3(5), pp. 720–734, 2016.
31. R. Belleza and E. Fretias, “Performance study of real-time operating systems for internet of things devices,” IET Softw., Vol. 12, Issue. 3, pp. 176-182, 2018.
32. M. Anedda, D. Giusto, and M. Murroni, “An Energy-efficient Solution for Multi-Hop Communications in Low Power Wide Area Networks,” IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), 2018.
33. Digikey. “Nucleo-F401RE.” *DigiKey Electronics*, www.digikey.gr/en/products/detail/stmicroelectronics/NUCLEO-F401RE/4695525.
34. “SX1276/77/78/79—137 MHz to 1020 MHz Low Power Long Range Transceiver.” Semtech. [Online]. Available: https://www.semtech.com/ products/wireless-rf/lora-core/sx1276#downloads.
35. “LoRa Modulation Basics.” Semtech. May 2021. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-core/sx1276# downloads.
36. U. Noreen, A. Bounceur, and L. Clavier, “A Study of LoRa Low Power and Wide Area Network Technology,” 3rd International Conference on Advanced Technologies for Signal and Image Processing, 2017.
37. Semtech, http://www.semtech.com/wireless-rf/lora/LoRa-FAQs.pdf:

Appendix A (Project Specification)

Objective:

The aim of this project is to design and implement an IoT-based sensor network using Zephyr RTOS, establishing low-power wireless communication over BLE or LoRa. The system will transmit environmental data between nodes and a central gateway in real-time. Which will be evaluated for performance and power usage with subsequent optimizations. The aims of the project are as follows:

- To design and implement an IoT system using Zephyr RTOS on ARM Cortex-M microcontroller.
- To explore wireless protocols (BLE or LoRa) for sensor-to-gateway communication.
- To analyze and optimize the system performance and power consumption.
- To understand the trade-offs between data reliability, communication range, and power efficiency.

Methodology:

The project will be developed using Zephyr RTOS for embedded development, enabling efficient resource management for low power architecture. The system will consist of multiple sensor nodes (wireless and wired) collecting and transmitting data to a central gateway over LoRa or BLE, which aggregates and possibly forwards data to a server. Development will involve programming and configuring sensors and microcontrollers, implementing data acquisition routines, and ensuring reliable wireless transmission. Following successful implementation, performance and power profiling tools (e.g., Zephyr's built-in metrics or external logging tools) will be used to collect data.

A similar approach to low-power network design was researched by *Modieginyane et al. (Modieginyane M., M. Matome, et al., "Wireless sensor networks application: A survey," Int. J. Comput. Sci. Inf. Technol., 2014)*. Zephyr's own community has also documented case studies showing effective use in constrained, battery-powered systems (*Zephyr RTOS Project, https://www.zephyrproject.org/*).

1. **Literature Review:** Study existing research on IoT focusing on LoRa & BLE. While integrating Zephyr RTOS into the design.
2. **Design:** Develop hardware/software architecture using Zephyr compatible boards. Select sensors, design communication flow and implement drivers for data acquisition and processing.
3. **Performance Evaluation:** Measure packet reliability, latency, and power consumption under various scenarios.
4. **Optimization:** Improve communication and transmission parameters to increase efficiency and responsiveness.
5. **Risk Assessment:** Identify potential risks involved.
6. **Documentation:** Maintain documentation throughout the design process, including evaluation and recommendations for future work.

2. OUTLINE OF THE WORKPLAN

	Phase	Activity	Dependencies/Resources
1.	**Research & Feasibility**	Literature review on BLE, LoRa, Zephyr RTOS, and similar IoT systems	Online academic journals, Zephyr documentation
2.	**Hardware Selection**	Choose ARM Cortex-M board, sensors, and communication modules	Development kits (e.g., nRF52, STM32, RFM95)
3.	**System Design**	Define hardware/software architecture, data flow, and system topology	Schematic diagrams, flowcharts
4.	**Firmware Development**	Set up Zephyr RTOS, driver development, and sensor integration	Zephyr SDK, VSCode, hardware kits
5.	**Wireless Communication**	BLE or LoRa implementation and node-to-gateway communication	BLE/LoRa libraries, antennas/modules
6.	**System integration**	Bring all components together into a functional system	Debug tools, multimeters, test scripts
7.	**Performance Testing**	Evaluate transmission latency, range, and power consumption	Profilers (e.g., Nordic Power Profiler), timers
8.	**Optimization**	Implement and test power-saving and timing optimizations	Firmware tuning, adaptive algorithms
9.	**Final Documentation**	Compile report, codebase, results, and conclusions	MS Word, GitHub, diagrams, graphs

Gantt Chart:

	Timeline	June				July				August				September	
Task	Description	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14
1	Research & Feasibility		M 1												
2	Hardware Selection														
3	System Design														
4	Firmware Development														
5	Wireless Communication							M 2							
6	System integration								M 3						
7	Performance Testing										M 4				
8	Optimization											M 5			
9	Final Documentation													M 6	
Deliverables				A			B	C				D			E/ F

Milestones & Progress indicators:

Milestones	
M 1	Completion of system architecture and hardware selection.
M 2	First successful wireless transmission between nodes.
M 3	Stable sensor readings and data transfer using Zephyr OS.
M 4	Performance and power data collected.
M 5	Optimized version shows measurable improvement.
M 6	Project is fully documented and ready for submission.

	Deliverables
A	Hardware Sourcing
B	Interim Report
C	Custom Hardware Procurement
D	Working Prototype
E	Thesis Report
F	Presentation

Risk Assessment:

Risk Factor	Likelihood	Severity	Risk Rating	Mitigation Strategy
Difficulty configuring the BLE/LoRa stack on Zephyr	3	4	12	Use official Zephyr samples and community-supported drivers; start with simple working examples.
Delays in sourcing or delivery of hardware components	4	4	16	Begin sourcing early; have backup vendors or use development kits initially.
Poor wireless communication performance (range/data loss)	3	4	12	Adjust antenna placement and transmission power; test in different environments early.
Firmware bugs leading to unstable system behaviour	3	5	15	Modular code with unit testing; use Zephyr's logging and debugging facilities.
Lack of time for proper optimization and final testing	4	3	12	Plan optimization early after basic implementation; reserve final weeks for refinement.
Inadequate documentation/source material during development	2	3	6	Maintain project logs and version control (Git); set documentation checkpoints.

0 – 5 = Low Risk 6 – 10 = Moderate Risk 11 – 15 = High Risk 16 – 25 = extremely high unacceptable risk		Severity of the potential injury/damage				
		Insignificant damage to Property, Equipment or Minor Injury 1	Non-Reportable Injury, minor loss of Process or slight damage to Property 2	Reportable Injury moderate loss of Process or limited damage to Property 3	Major Injury, Single Fatality critical loss of Process/damage to Property 4	Multiple Fatalities Catastrophic Loss of Business 5
Likelihood of the hazard happening	Almost Certain 5	5	10	15	20	25
	Will probably occur 4	4	8	12	16	20
	Possible occur 3	3	6	9	12	15
	Remote possibility 2	2	4	6	8	10
	Extremely Unlikely 1	1	2	3	4	5

Revised Gantt Chart:

	Timeline	June				July				August				September	
Task	Description	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14
1	Research & Feasibility		M 1												
2	Hardware Selection														
3	System Design														
4	Firmware Development														
5	Wireless Communication							M 2							
6	System integration								M 3						
7	Performance Testing										M 4				
8	Optimization											M 5			
9	Final Documentation													M 6	
Deliverables				A			B	C				D			E/ F

Appendix B (main.c)

```
#include <zephyr/kernel.h>
#include <zephyr/logging/log.h>
#include "mesh_handler.h"

LOG_MODULE_REGISTER(main, CONFIG_LOG_DEFAULT_LEVEL);

void payload_handler(uint8_t *payload, size_t len)
{
    LOG_HEXDUMP_INF(payload, len, "User callback received payload");    //
Print the received data in hex format for debugging/logging
}

void main(void)
{
    const struct device *lora_dev = DEVICE_DT_GET(DT_ALIAS(lora0));

    if (mesh_init(lora_dev, payload_handler) < 0) // Initialize the mesh
stack (configures RX, starts TX task, registers callback).
    {
        return;
    }

    LOG_INF("Mesh node starting (ID 0x%02x)", NODE_ID); // Log message
showing that this node has started, printing its NODE_ID.

    uint16_t seq_counter = 0; // Sequence counter to uniquely identify
packets from this node.
    /*
    // If outside while (Receiver- transmits once)
    struct tx_msg_dyn *tx = k_malloc(sizeof(struct tx_msg_dyn) +
MAX_MESH_PAYLOAD);
    if (!tx)
        return;

    tx->hdr.src = NODE_ID;          // Source node ID
    tx->hdr.dst = RECEIVER_ID;      // Destination node ID
    tx->hdr.seq = seq_counter++;    // Packet sequence number
    tx->hdr.ttl = MAX_TTL;          // Time-To-Live (max hops allowed)


    tx->len = snprintk((char *)tx->payload, MAX_MESH_PAYLOAD,   // Fill
payload buffer with a message string
                       "Hello from node %d   ", NODE_ID);

    mesh_enqueue_tx(tx);    // Enqueue the message for transmission
    */
    /*
```

```
    // Only  Receiver (Forwarder)
    int ret = configure_rx(lora_dev);
    if (ret < 0)
        LOG_ERR("Failed to reconfigure RX");
    */
    while (1)
    {
        // Transmit periodically
        struct tx_msg_dyn *tx = k_malloc(sizeof(struct tx_msg_dyn) +
MAX_MESH_PAYLOAD);
        if (!tx)
            return;

        tx->hdr.src = NODE_ID;          // Current node's ID as source
        tx->hdr.dst = RECEIVER_ID;      // Target node's ID
        tx->hdr.seq = seq_counter++;    // Increment sequence number for
uniqueness
        tx->hdr.ttl = MAX_TTL;          // Set maximum allowed hops


        tx->len = snprintk((char *)tx->payload, MAX_MESH_PAYLOAD, // Prepare
payload (print a message into buffer)
                           "Hello from node %d   ", NODE_ID);

        mesh_enqueue_tx(tx);    // Enqueue prepared message into TX queue for
transmission
        k_sleep(K_SECONDS(BROADCAST_INTERVAL_S));   // Sleep for
BROADCAST_INTERVAL_S seconds before sending the next packet
    }
}
```

Appendix C (mesh_handler.c)

```
#include "mesh_handler.h"               // Custom header
#include <zephyr/kernel.h>
#include <zephyr/drivers/lora.h>
#include <zephyr/logging/log.h>
#include <zephyr/sys/printk.h>
#include <string.h>
#include <stdlib.h>

// FUNCTION DEF

LOG_MODULE_REGISTER(mesh_node, CONFIG_LOG_DEFAULT_LEVEL);   // Register this
source file ("mesh_node") as a logging module

// Make tx_msgq private
K_MSGQ_DEFINE(tx_msgq, sizeof(struct tx_msg_dyn *), TX_QUEUE_LEN, 4);   //
Message queue for outgoing packets (private to this file).
K_THREAD_STACK_DEFINE(stack_area, STACK_SIZE);                          //
Stack memory and thread object for the transmit task.
static struct k_thread transmit_thread;                                 //
Stack memory and thread object for the transmit task.

static struct recent_entry  // Cache for recently seen packets to prevent
duplicate forwarding.
{
    uint8_t src;
    uint16_t seq;
} recent_cache[CACHE_SIZE];

static int cache_idx;   // Tracks cache insertion index
static mesh_receive_callback_t receive_callback = NULL; // User callback for
application layer

// ----------------------- Utility Functions -----------------------//

// Compute 16-bit CRC checksum (polynomial 0x1021) over given data.
//Used for error detection in transmitted packets.
static uint16_t calc_checksum_crc16(const uint8_t *data, size_t len){
    uint16_t crc = 0x0000;
    for (size_t i = 0; i < len; i++)
    {
        crc ^= (uint16_t)data[i] << 8;
        for (uint8_t j = 0; j < 8; j++)
        {
            if (crc & 0x8000)
                crc = (crc << 1) ^ 0x1021;  // Polynomial division
            else
                crc <<= 1;
```

```
        }
    }
    return crc;
}

// Check if a (src, seq) pair was already processed (duplicate suppression).
static bool seen_before(uint8_t src, uint16_t seq)
{
    for (int i = 0; i < CACHE_SIZE; i++)
    {
        if (recent_cache[i].src == src && recent_cache[i].seq == seq)
            return true;
    }
    return false;
}

// Add a new (src, seq) pair to the recent packet cache.
static void add_to_cache(uint8_t src, uint16_t seq)
{
    recent_cache[cache_idx].src = src;
    recent_cache[cache_idx].seq = seq;
    cache_idx = (cache_idx + 1) % CACHE_SIZE;   // Circular buffer
}


// ----------------------- LoRa Config -----------------------//

// Configure LoRa modem in RX mode.
static int configure_rx(const struct device *dev)
{
    struct lora_modem_config cfg = {
        .frequency = 865100000,         // Operating frequency
        .bandwidth = BW_125_KHZ,        // Channel bandwidth
        .datarate = SF_12,              // Spreading factor
        .preamble_len = 8,              // Preamble length
        .coding_rate = CR_4_5,          // Error correction coding rate
        .iq_inverted = false,
        .public_network = false,
        .tx_power = 14,                 // Transmission power in dBm
        .tx = false,                    // RX mode
    };

    int ret = lora_config(dev, &cfg);
    if (ret < 0)
    {
        LOG_ERR("Failed to configure RX: %d", ret);
        return ret;
    }
```

```
    // Register asynchronous receive callback
    lora_recv_async(dev, mesh_receive_cb, NULL);
    return 0;
}

// Configure LoRa modem in TX mode.
static int configure_tx(const struct device *dev)
{
    // Disable RX callback while transmitting
    lora_recv_async(dev, NULL, NULL);

    struct lora_modem_config cfg = {
        .frequency = 865100000,
        .bandwidth = BW_125_KHZ,
        .datarate = SF_12,
        .preamble_len = 8,
        .coding_rate = CR_4_5,
        .iq_inverted = false,
        .public_network = false,
        .tx_power = 14,
        .tx = true,                 // TX mode
    };

    int ret = lora_config(dev, &cfg);
    if (ret < 0)
    {
        LOG_ERR("Failed to configure TX: %d", ret);
    }
    return ret;
}

// ------------------------ Send Message ------------------------//

// High-level API to send a mesh message.
int mesh_send_msg(const struct device *dev, struct tx_msg_dyn *buffer)
{
    // Check payload length constraint
    if (buffer->len > MAX_MESH_PAYLOAD)
    {
        LOG_ERR("Payload too large (%zu > %d)", buffer->len,
MAX_MESH_PAYLOAD);
        return -EINVAL;
    }

    // Remember this packet in duplicate cache
    add_to_cache(buffer->hdr.src, buffer->hdr.seq);

    // Compute CRC16 over header + length + payload
```

```
    uint16_t checksum = calc_checksum_crc16((uint8_t *)buffer,
                                            sizeof(struct mesh_hdr) +
sizeof(size_t) + buffer->len);

    // Switch radio to TX mode
    int ret = configure_tx(dev);
    if (ret < 0)
        return ret;

    // Build transmit buffer: header + length + payload + checksum
    size_t total_size = sizeof(struct mesh_hdr) + sizeof(size_t) + buffer-
>len + CHECKSUM_SIZE;
    uint8_t *send_buf = k_malloc(total_size);
    if (!send_buf)
        return -ENOMEM;

    memcpy(send_buf, buffer, sizeof(struct mesh_hdr) + sizeof(size_t) +
buffer->len);
    memcpy(send_buf + sizeof(struct mesh_hdr) + sizeof(size_t) + buffer->len,
&checksum, CHECKSUM_SIZE);

    // Perform LoRa transmission
    ret = lora_send(dev, send_buf, total_size);
    k_free(send_buf);

    if (ret < 0)
        LOG_ERR("lora_send failed: %d", ret);
    else
        LOG_INF("Sent packet src:0x%02x dst:0x%02x seq:%u ttl:%u
checksum:0x%04X",
                buffer->hdr.src, buffer->hdr.dst, buffer->hdr.seq, buffer-
>hdr.ttl, checksum);

    // Switch back to RX mode after TX
    ret = configure_rx(dev);
    if (ret < 0)
        LOG_ERR("Failed to reconfigure RX after send");

    return ret;
}

// ------------------------ Receive Callback ------------------------//
// Callback called whenever a packet is received over LoRa
void mesh_receive_cb(const struct device *dev, uint8_t *data, uint16_t size,
                     int16_t rssi, int8_t snr, void *user_data)
{
    ARG_UNUSED(user_data);
```

```
    // Validate minimum packet size (header + len + checksum)
    if (size < sizeof(struct mesh_hdr) + sizeof(size_t) + CHECKSUM_SIZE)
        return;

    // Extract and verify checksum
    uint16_t received_checksum;
    memcpy(&received_checksum, data + size - CHECKSUM_SIZE, CHECKSUM_SIZE);

    if (calc_checksum_crc16(data, size - CHECKSUM_SIZE) != received_checksum)
        return;

    // Cast received data into mesh message format
    struct tx_msg_dyn *rx_msg = (struct tx_msg_dyn *)data;
    size_t payload_len = rx_msg->len;

    // Drop duplicates
    if (seen_before(rx_msg->hdr.src, rx_msg->hdr.seq))
        return;
    add_to_cache(rx_msg->hdr.src, rx_msg->hdr.seq);

    // Log packet details
    LOG_INF("Mesh RX src:0x%02x dst:0x%02x seq:%u ttl:%u RSSI:%d SNR:%d",
            rx_msg->hdr.src, rx_msg->hdr.dst, rx_msg->hdr.seq, rx_msg-
>hdr.ttl, rssi, snr);

     // If this node is the destination, deliver to application
    if (rx_msg->hdr.dst == NODE_ID)
    {
        if (receive_callback != NULL)
        {
            receive_callback(rx_msg->payload, payload_len);
        }
    }
    // Otherwise, forward if TTL > 1
    else if (rx_msg->hdr.ttl > 1)
    {
        struct tx_msg_dyn *fwd = k_malloc(sizeof(struct mesh_hdr) +
sizeof(size_t) + payload_len);
        if (!fwd)
            return;
        fwd->hdr = rx_msg->hdr;
        fwd->hdr.ttl -= 1;                      // Decrement TTL
        fwd->len = payload_len;
        memcpy(fwd->payload, rx_msg->payload, payload_len);
        k_msgq_put(&tx_msgq, &fwd, K_NO_WAIT);  // Enqueue for transmission
    }
}
```

```c
// ------------------------ TX Task ------------------------//
// Background thread that continuously sends messages from the TX queue
void tx_task(void *p1, void *p2, void *p3)
{
    const struct device *dev = DEVICE_DT_GET(DT_ALIAS(lora0));
    if (!device_is_ready(dev))
        return;

    struct tx_msg_dyn *msg;
    while (1)
    {
        // Block until a message is available in queue
        if (k_msgq_get(&tx_msgq, &msg, K_FOREVER) == 0)
        {
            mesh_send_msg(dev, msg);    // Send via LoRa
            k_free(msg);                // Free memory after sending
            k_sleep(K_MSEC(100));       // Small delay to prevent congestion
        }
    }
}

// ------------------------ Initialization ------------------------//
// Initialize the mesh subsystem
int mesh_init(const struct device *dev, mesh_receive_callback_t cb)
{
    if (!device_is_ready(dev))
        return -ENODEV;

    // Register application callback for received payloads
    receive_callback = cb;
    // Start background TX thread
    k_tid_t tid = k_thread_create(&transmit_thread, stack_area, STACK_SIZE,
                                  tx_task, NULL, NULL, NULL,
                                  PRIORITY, 0, K_NO_WAIT);
    k_thread_name_set(tid, "transmit_thread");
    return configure_rx(dev);
}

// ------------------------ Helper API ------------------------//
// Enqueue a message for transmission by TX task
int mesh_enqueue_tx(struct tx_msg_dyn *msg)
{
    return k_msgq_put(&tx_msgq, &msg, K_NO_WAIT);
}
```

Appendix D (mesh-handler.h)

```c
#ifndef MESH_HANDLER_H
#define MESH_HANDLER_H

#include <zephyr/device.h>
#include <zephyr/kernel.h>
#include <stdint.h>
#include <stddef.h>

// MACROS DEFINED IN HEADER FILE

#define NODE_ID 0x02
#define RECEIVER_ID 0x10

#define MAX_MESH_PAYLOAD 240
#define MAX_TTL 8
#define CACHE_SIZE 64
#define BROADCAST_INTERVAL_S 2
#define TX_QUEUE_LEN 4
#define CHECKSUM_SIZE sizeof(uint16_t)

#define STACK_SIZE 1024
#define PRIORITY 5

struct mesh_hdr
{
    uint8_t src;
    uint8_t dst;
    uint16_t seq;
    uint8_t ttl;
} __packed;

struct tx_msg_dyn
{
    struct mesh_hdr hdr;
    size_t len;
    uint8_t payload[]; // Flexible array member
};

typedef void (*mesh_receive_callback_t)(uint8_t *payload, size_t len);

// Initialize mesh module and LoRa device
int mesh_init(const struct device *dev, mesh_receive_callback_t cb);

// Send a message through mesh
int mesh_send_msg(const struct device *dev, struct tx_msg_dyn *msg);

// Enqueue a message for transmission (helper API)
```

```
int mesh_enqueue_tx(struct tx_msg_dyn *msg);

// Thread function for handling transmission
void tx_task(void *p1, void *p2, void *p3);

// LoRa receive callback
void mesh_receive_cb(const struct device *dev, uint8_t *data, uint16_t size,
                     int16_t rssi, int8_t snr, void *user_data);

#endif // MESH_HANDLER_H
```

Appendix E (Logs)

The images below are the screenshots from the live testing of the Mesh Network, showing the serial print from the Gateway.