

THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

DESIGN, DEVELOPMENT, AND IMPLEMENTATION OF A WIRELESS MESH
COMMUNICATION SYSTEM USING THE LoRa MODULATION SCHEME

ALEXANDER ROBERT STEPKO
FALL 2023

A thesis
submitted in partial fulfillment
of the requirements
for the baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

David Cubanski
Associate Teaching Professor of Electrical Engineering
Thesis Supervisor

Julio Urbina
Professor of Electrical Engineering
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

This work performs extensive study and exploration into the design, implementation, and testing of a wireless mesh communication system. The communication system employs the LoRa modulation scheme, a chirp spread spectrum (CSS) derivative that provides an extremely high link budget, in addition to built-in redundancy inherent in the symbol space. We begin the work via an introduction of network topologies observed in the wireless space and conduct a discussion on the critical factors and layers that must be considered when designing a mesh communication system. A system design is proposed, which includes application, network, and physical layers. Dynamic mesh topology establishment and maintenance is achieved without user intervention via implementation and abstraction of the Distance Vector Routing (DVR) algorithm. We implement the proposed architecture on a dual-core Espressif ESP32-S3 microprocessor and a Semtech SX1262 sub-GHZ LoRa wireless transceiver module contained on a Heltec WiFi LoRa 32 (v3) development platform. Software is written in C with a multi-threaded architecture using FreeRTOS. Following system development, we present results that show the development of a mesh communication system using LoRa is not only feasible but fruitful in terms of spectral efficiency and immunity to noise in high-interference environments. Finally, we conclude with a discussion about the system's performance and explore further areas for system development.

Table of Contents

List of Figures	vii
List of Tables	ix
Listing	x
List of Acronyms	xiii
Acknowledgements	xv
1 Considerations in Mesh Networking	1
1.1 Existing challenges	1
1.1.1 A handheld solution?	4
1.2 The Benefits of a Mesh Approach	6
2 Mesh Topologies: An Overview	9
2.1 Topology of a Mesh Network	10
2.2 Scalability	12
2.2.1 Scalability and Performance	12
2.3 Characteristics and Metrics of Wireless Propagation	15
2.3.1 Received Signal Strength Index	17
2.3.2 Signal-to-noise ratio	18
2.4 The LoRa Modulation Scheme	21
2.5 Routing algorithms	25
2.5.1 Distance Vector Routing (DVR)	25
3 Design and Implementation	31
3.1 Design Requirements	32
3.1.1 End-User Design Criteria	32
3.1.2 Legal Requirements	34
3.2 Hardware Selection	37
3.2.1 System Controller	39
3.2.2 Transceiver System	40
3.2.3 The Platform	43
3.3 Data Structures	45

3.3.1	The Data Packet	45
3.3.2	Queues and semaphores	49
3.4	Process Architectures	53
3.4.1	User I/O	55
3.4.2	Packet Engine Tasks	58
3.4.3	RF Tasks	66
4	Test and Validation	77
4.1	User I/O Test And Validation	78
4.1.1	Send message	79
4.1.2	Set device params	79
4.1.3	View device status	80
4.1.4	View received messages	82
4.1.5	Test send mode	82
4.1.6	Test receiver mode	83
4.1.7	Test continuous tone generation mode	84
4.1.8	Test infinite preamble generation mode	84
4.1.9	Generate network ping	85
4.2	Packet engine validation	87
4.3	Physical layer validation	89
5	Concluding Remarks	95
	Bibliography	99
	Appendices	103
	Hardware Schematics and Diagrams	105
.1	Espressif ESP32 (System Controller)	105
.2	Semtech SX1262 (RF Transceiver)	106
.3	Heltec WiFi LoRa 32 (v3) (Deployment Board)	107
	A Failed Effort	109
	Complete Codebase	111
	Academic Vita	113

List of Figures

1.1	An example of a star network topology	2
1.2	LTE Data Coverage in Pennsylvania by major carriers [1]	3
1.3	FCC Frequency Allocation from DC to 300GHz [2]	4
1.4	An Ailunce HD1 VHF/UHF handheld radio	5
1.5	A typical five-layer network model [3]	6
2.1	Duplex mesh topology	10
2.2	Simplex mesh topology	10
2.3	Routed network topology	11
2.4	Scalability of unmanaged mesh networks	13
2.5	Modulation schemes versus signal noise floors for common modulation schemes . .	19
2.6	QPSK constellation with low SNR	20
2.7	QPSK constellation with high SNR	20
2.8	Example of a LoRa symbol	22
2.9	Waterfall chart of a LoRa signal	24
2.10	Initial DVR stage	26
2.11	Node D begins to join the network	26
2.12	Node C updates with data from D	27
2.13	Node C broadcasts its forwarding table to local nodes	27
2.14	Nodes B and D receive broadcast from C and update their tables	28
2.15	Node B broadcasts its updated forwarding table to local nodes	28
2.16	Node A updates its forwarding table with data from node B	28
2.17	Node A broadcasts its updated forwarding table to local nodes	29
3.1	High-level system architecture	33
3.2	Occupational bandwidth of different radio systems	36
3.3	General architecture for FreeRTOS on the ESP32 microcontroller	40
3.4	SX1262 block diagram [4]	42
3.5	Hardware block diagram of the system as implemented	43
3.6	Heltec Automation WiFi LoRa 32 (v3) development platform [5]	44
3.7	Data structure for a packet in the communication	47
3.8	LoRa preamble/synchronization symbols	49
3.9	Utilization of queues in the system	52
3.10	Finite state machine of a binary semaphore	52
3.11	Graph of the major tasks spawned	54

3.12	Graph of the User I/O tasks and structures spawned	56
3.13	Relative statistical distribution of letters in common English [6]	58
3.14	Global view of the network packet engines	59
3.15	Global view of the network packet engines	60
3.16	Processing system within the packet engine for received messages	61
3.17	Construction of the system controller's random number generator [7]	62
3.18	Data structure for an originator message (OGM)	63
3.19	Forwarding table and localized topology seen from node Q	63
3.20	Logical compression of the forwarding table for Figure 3.19	64
3.21	Physically compressed forwarding table for Figure 3.19	64
3.22	Packet structure for an OGM-ACK	65
3.23	Graph of the RF tasks and structures spawned	67
3.24	SPI timing diagram for the transceiver [4]	69
3.25	State machine to send packets	74
3.26	State machine to receive packets	75
3.27	Channel activity detection mode of the communication system	76
4.1	Test message in transmission	83
4.2	Continuous tone generated at $f_c = 915\text{MHz}$	84
4.3	Infinite preamble signal at $f_c = 915\text{MHz}$, $SF5$, and $BW = 62\text{KHz}$	85
4.4	Topology of the packet engine validation	88
4.5	Observed occupied (99%) channel power of the device	89
4.6	Observed occupied (99%) channel power of the device at 250kHz	90
4.7	Spectral waterfall chart of LoRa symbols at 300kHz	91
4.8	Spectral waterfall chart of LoRa symbols at 125kHz	91
4.9	Spectral waterfall chart of LoRa symbols at 62kHz	92
4.10	Point-to-point range test setup	93
4.11	RSSI and SNR of a point-to-point link at $SF5$ and 125kHz bandwidth	94
1	Minimum required componentry to run the ESP32 [7]	105
2	Processor architecture of the ESP32 [7]	106
3	Typical application schematic for the SX1262 transceiver [4]	106
4	SX1262 block diagram [4]	107
5	Schematic of the deployment board [5]	108
6	Trace layout of the attempted development board	110
7	Top-view render of the attempted development board	110

List of Tables

2.1	γ parameters for common environments and operational frequencies [8]	17
3.1	End-User Design Criteria for the Communication System	34
3.2	Message types of a data packet	47
3.3	Spawned tasks from the main function	54
3.4	Spawned tasks from the main function	56
3.5	Signal names for the transceiver interface	69
3.6	Timing values for the SPI timing diagram shown in Figure 3.24 [4]	70
4.1	Default parameter configuration	80

List of Provided Code

4.1	Landing page presented to the user	78
4.2	Send message dialogue page	79
4.3	Modification of device parameters via dialogue	80
4.4	System status display output	80
4.5	Initial output shown to the user when looking at received messages	82
4.6	Output of test receiver mode	83
4.7	Recipient node ping statistics dump	86
4.8	Forwarding table at node startup for node 36	87
4.9	Forwarding table after topology stabilization at node 36	88

List of Acronyms

AM amplitude modulation	LO local oscillator
CAD channel activity detection	MCU microcontroller
CSS chirp spread spectrum	MIMO multi-input and multiple-output
DSDV Destination-Sequenced Distance Vector Routing	OGM originator message
DSR Dynamic Source Routing	PAM phase amplitude modulation
DSSS direct sequence spread spectrum	PCB printed circuit board
DVR Distance Vector Routing	PSK phase shift-keying
EIRP effective isotropic radiated power	RFID radio frequency identification
FCC Federal Communications Commission	RSSI recieved signal strength index
FHSS frequency hopping spread spectrum	RTOS real-time operating system
FM frequency modulation	SAR search and rescue
HF high frequency	SEND satelite emergency notification device
IC integrated circuit	SNR signal to noise ratio
IF intermediate frequency	SPI Serial Peripheral Interface
IoT Internet of Things	SWaP size weight and power
IP internet protocol	TCP transmission control protocol
ISM Industrial Scientific and Medical	THSS time hopping spread spectrum
LBT listen before talk	UDP user datagram protocol
LNA low-noise amplifier	UHF ultra high frequency
	VHF very high frequency

Acknowledgements

This undergraduate thesis marks one of the proudest moments in my education at Penn State and the beginning of a very, very bright future. I firmly believe that the space this work exists within is one of the most impactful disciplines within electrical engineering. The knowledge I gained as a result of this work demonstrates that there is much to do in this space, and I fully intend to advance mankind in this realm in graduate studies.

To acknowledge the impact that so many people in my life have had on my academic triumph, seething desire to learn, and unfettered determination to push boundaries in this magical space commonly referred to as electrical engineering is *entirely* impossible to conceptualize within the written word of a single page. The knowledge, techniques, and motivation I have gained in this short undergraduate education would not be possible without the dedication of so many individuals to provide me with the tools to learn. I hope that these few paragraphs convey at least some of my gratitude.

The experience I have been afforded during my undergraduate career at Penn State is one I would have never predicted. I entered the University intending to pursue a degree in Aerospace and then build aircraft. As I write this, I fully intend to pursue my doctorate in on-chip RF processing systems – a completely different realm than aircraft design. I can attribute this to the fleeting months of grade school, where I was introduced to the endless possibilities that electrical engineering holds. I consider this to be the turning point resulting in one of the best decisions I have ever made. Had I not made this fateful leap there is no doubt my future would not be as bright.

I would like to extend my love and most sincere thanks to my parents Mark and Amy Stepko,

and my sister Caroline. There has never been a moment in the past four years where their support has waivered, even when I begin to ramble about complex topics that they do not know about. My friends, Colin, Nate, and Reese have shown the same levels of support and I am extremely fortunate to call them some of my closest colleagues. Fate has provided me with a circle of friends that continually elevates my passion for what I believe to be a form of art, one of which has had a profound impact on my emotional well-being as my last semester in undergraduate school comes to an end.

My thesis advisor, Dr. David Cubanski, has played an extremely important role in shaping the path I will take. His dedication to constructing passion within students to achieve something previously thought to be unattainable is one of the most remarkable acts I have seen in my education, and I am infinitely grateful for the opportunity to work with him. The act of creation in engineering is complicated, nuanced, and tough. Dr. Cubanski fully understands these challenges.

I would also like to thank all of the other instructors and professors who dedicate their lives to teaching us undergraduate students the complex minutia of electrical engineering, and then how to leverage such complexity to create something new. To teach someone is one thing. To teach someone the tools necessary to change the world through electrical engineering is an infinitely more complex problem. I have an immense amount of respect for the work that all of the professors at this university do, and I hope that my time in their courses has provided some level of gratitude.

As a final remark, I must comment on the significance of the work I have performed. While an examination of my undergraduate transcript shows I have taken many of the same classes as my peers, the relationships I have built with so many incredible individuals at this university have provided me with limitless possibilities. This thesis is only one of the possibilities I have explored, and cannot thank everyone who has stood by my side enough for the ability to perform this work.

A handwritten signature in black ink, consisting of stylized, overlapping letters that appear to be 'A' and 'S'.

Chapter 1

Considerations in Mesh Networking

1.1 Existing challenges

In the modern age, the gap between system performance and system cost continues to become more disparate. One area where there is a substantial need for a high performance-to-cost ratio is the field of wireless data transmission. The ability to quickly and efficiently transmit data between devices using a wireless channel is – and has been – a centrally important field of design, study, and development across consumer, industrial, and military sectors. While the final application in these sectors differs, each has the common denominator to operate under increasingly more stringent size weight and power (SWaP) design requirements.

To reduce edge device cost, many solutions to wireless data transfer leverage large-scale deployments of “many-to-one” architecture, more commonly called a star topology. Devices that use this architecture are found in nearly every corner of the average individual’s life; for example, cellular phones, satellite communicators, two-way radios, and non-mesh WiFi all function using

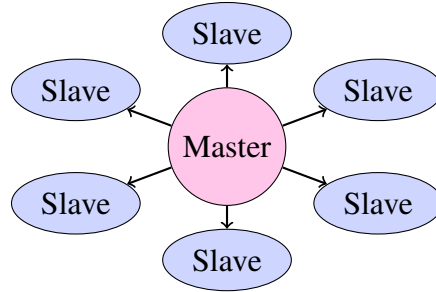


Figure 1.1: An example of a star network topology

the principles of a star topology. While it is evident that this deployment architecture works incredibly well (cell phones are in fact *extremely* capable), its major downfall is that points of failure are concentrated at major network hubs. For example, 5G infrastructure deployment has encountered major challenges in providing full area coverage in dense urban areas due to the propagation characteristics of the mmWave carriers [9].

For a low-cost and comparatively high-performance data transmission system (a SWaP-centric design), mesh network architectures can circumvent many of the challenges around centralized points of failure. The military has recognized these benefits for decades and, for decades, continues to request systems based heavily on distributed architectures where reliability and robustness are major design requirements [10]. Similar systems exist in the civilian market to support multiple failure points and maintain normal operation, even in a star topology. IEEE 802.11s, The WLAN Mesh Standard, defines the operational method to employ mesh networking for WiFi [11]. IEEE 802.15.4, the standard for low-rate wireless personal area network, is the backbone of the ZigBee network that powers much of the Internet of Things (IoT) devices flooding the consumer space [12]. The “Age of the AI Toaster” is upon us, and here to stay.

Perhaps the world’s greatest example of the application of distributed protocol theory is the internet. Core protocols such as internet protocol (IP) follow a “best effort” methodology to transfer data using the most ideal paths. While this does not explicitly guarantee *reliable* data transfer, it provides a framework for additional protocols such as transmission control protocol (TCP) and user datagram protocol (UDP) to navigate the complexities of reliability. Further enhancements to network stability are provided in the physical layer by multi-band multi-input and multiple-output

(MIMO) wireless standards since IEEE 802.11n [13]. These physical links can be further enhanced using redundancy as a figment of MIMO, and also increase net throughput with pseudo-parallel data transfer between clients.

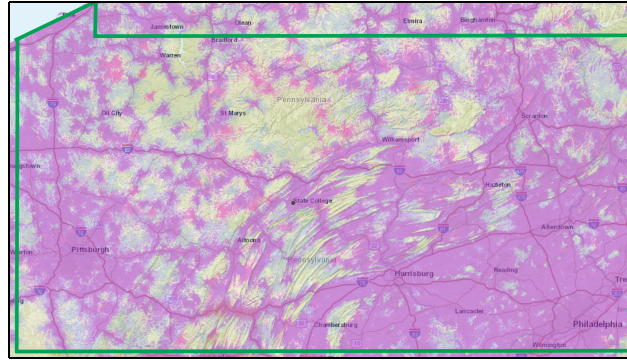


Figure 1.2: LTE Data Coverage in Pennsylvania by major carriers [1]

Systems that are unable to provide reliable communications are crippling for users and, in many applications, dangerous. Military, search and rescue (SAR), hazardous industrial environments, and areas where centralized infrastructure is not always present commonly encounter the pitfalls of unreliable communication. Outdoor enthusiasts, for example, often find themselves in environments without cellular connections. As an example, a 2021 map of LTE data coverage for the state of Pennsylvania is provided in Figure 1.2. We see in the figure that major population centers have good coverage; however, more sparsely populated areas have very little coverage available. High population concentrations like mass gatherings (a Penn State football game, perhaps) also provide extremely challenging environments for cellular infrastructure to connect two customers. Even with connections, these groups of individuals deliberately turn off their devices to conserve power when access to grid power is infrequent. Satellite communicators (SENDs) can supplement reliability in some of these situations, but at a high price and with limited functionality. Maximum effective throughput is heavily dependent on the signal strength, and the sheer distance of satellite communication is such that a received signal (at either end) is often very weak [14].

Individuals who deem handheld radios as a viable supplement for their communication needs



Figure 1.4: An Ailunce HD1 VHF/UHF handheld radio

often operate in the very high frequency (VHF) or ultra high frequency (UHF) frequency range. Non-handheld civilian radios also exist in the high frequency (HF) frequency range, all of which are dictated by the FCC. Operators are restricted to non-encrypted, non-commerce communications, and certain effective isotropic radiated power (EIRP) limits are dependent on the licensure of the operator. Despite the apparent limits, amateur radio is extremely active and frequently demonstrates the overwhelming reliability of the community's decentralized communication systems in times of emergency, and crisis, and at great distances between devices. Because of this reliability, handheld radios are often found in the hands of outdoorsmen, first responders, industrial workers, and the military.

It would appear then that a handheld radio is the perfect platform to use to circumvent challenges surrounding non-reliable wireless communications. However, it is important to note that while handheld radios provide a large degree of functionality, they are inherently limited in what they can do by design and through regulation. These devices cannot transmit encrypted data, rely on repeater networks, and are not easily translated into the digital domain of modern communication systems.

1.2 The Benefits of a Mesh Approach

Section 1.1.1 provides a compelling set of arguments for using handheld radios as the deployment method for the communication system. However, these radios operate in what would be considered Layer 2 on a traditional five-layer network model. A mesh network exists across layers 2 and 3 using the model defined in Figure 1.5. An effective mesh network must understand the physical limitations of its operation, while also maintaining networking to provide connectivity between these physical links. An ideal mesh network has many advantages. The first major advantage is the ability to transmit data across a topology that dynamically adapts to changing network conditions. For example, if a certain link is suddenly broken due to a nodal outage or via high interference, the topology dynamically readjusts and makes a “best effort” to incorporate the node back into the network via new routing topologies [15]. Mesh networks are also incredibly easy to scale. This is due in part to their ability to dynamically modify their topology to adjust to changing link conditions and available network pathways. The establishment of a mesh network requires little to no user interaction, aside from providing power to nodes and the assurance that nodes are within range of reliable signal propagation.

Because nodes are dynamically self-organizing, the network often scales rather well. A densely populated network will attempt to take the lowest-cost path through the network whether that cost is the number of hops or probability of data corruption along the route. In either case, the network is oriented towards efficiency and is inherently motivated towards a topology with a high degree of scalability. Such advantages provide great motivation to use a mesh network in decentralized

5	Application Layer
4	Transport Layer
3	Network Layer
2	Physical Layer
1	Link Layer

Figure 1.5: A typical five-layer network model [3]

communication environments where users dynamically join and leave the network as required. Given a high enough amount of network saturation, it is possible to allow communications between all nodes without interruption, even if a large number of users decide to leave the network. This permits high reliability, and high redundancy communications, which is possible through the usage of physical layers that yield greater global performance than a traditional star topology due to the distributed nature of channel utilization.

Chapter 2

Mesh Topologies: An Overview

This chapter's purpose is to introduce the concept of a mesh network topology. In doing so, we examine link directionality, scalability, and the basic theory behind over-the-air signal propagation. All of these concepts are necessary to consider when attempting to implement a network establishment and routing algorithm, which is the core of an effective mesh topology.

The inherent nature of a mesh communication network suggests that a group of nodes can transmit and receive data from all the other devices accessible in the network graph. There are many benefits to such a topology, mainly the ability to transmit data much farther than previously possible in a point-to-point system. As such, the network often achieves data transmission at higher efficiencies and requires lower-cost hardware [16].

The benefits of mesh networking are not without costs. When looking to design viable topologies, poorly analyzed rules of “mesh establishment” result in a failed or incomplete mesh. Loops also arise in the routing through such a network, and controls must be put in place to avoid such circumstances. An incomplete mesh results in unreliable data transfer, or the accidental removal of nodes from the network, among other challenges [17].

2.1 Topology of a Mesh Network

The study of mesh topology analysis is a very active research community. Countless tools exist to develop different topologies using different rules, and there is a considerable amount of effort invested to reduce network latency as commercially viable applications of a mesh network continue to expand [18].

Perhaps the most basic concept to consider when forming a network is the directionality of a given network link. Consider Figures 2.1 and 2.2. The purpose of these figures is to demonstrate the different topologies present when considering the directionality of links. A duplex link infers that communication is possible in both directions, whereas a simplex link infers that communication is possible in only one direction. For wireless communication in SWaP-centric designs, it is commonplace that devices exhibit symmetrical uplink/downlink capabilities. Therefore, it is reasonable to assume that duplex topologies arise in mesh algorithms.

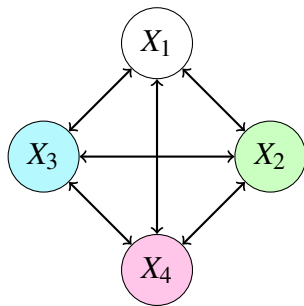


Figure 2.1: Duplex mesh topology

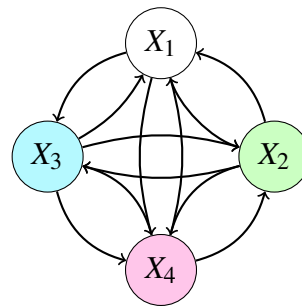


Figure 2.2: Simplex mesh topology

While this assumption is generally true, wireless channels are often subject to phenomena not limited to irregular random attenuation and multipath interference [8]. These phenomena result in unreliable communication channels and subject a wireless link to the possibility of data corruption.

subject a duplex network to unreliable data transfer, packet loss, and a decrease in net throughput [15]. To circumvent these challenges, a mesh topology is developed that is centered on the idea that simplex links may arise, similar to the graph shown in Figure 2.2.

A phrase commonly heard in discussions about mesh topologies is the notion of the “nearest neighbor.” The nearest neighbor is defined as the set of nodes accessible within 1 hop of the parent

node. Furthermore, a “hop” is defined as the act of sending data from one device to the next. In Figure 2.1, for parent node X_1 , its nearest neighbors are X_2, X_3, X_4 . This means that it is possible to contact all nodes in the graph within 0 hops of the parent node X_1 . Packets destined for the nearest neighbors of X_1 would be sent (or addressed) directly to that node. As this work emphasizes the application of mesh topologies in the wireless space, no message is truly sent directly. Moreover, it is sent with identifying information that labels a bundle of data for a certain device. The concept of a wireless link is only apparent in point-to-point microwave links used for backhaul between cell towers or businesses. Note that the graph of a mesh topology becomes increasingly complex as the size of the network grows (see Fig. 2.3).

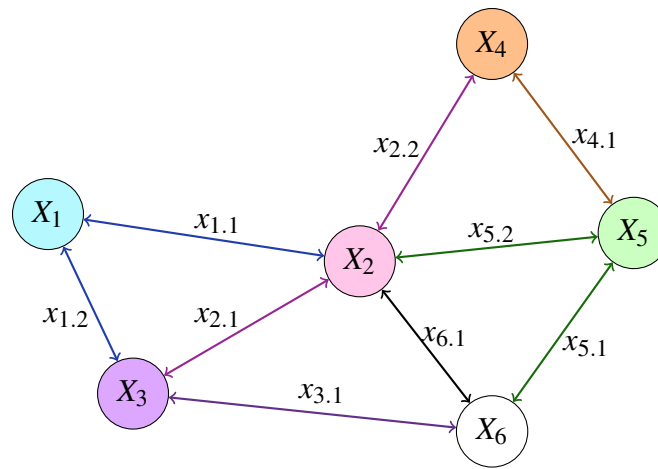


Figure 2.3: Routed network topology

2.2 Scalability

The scalability of a mesh network is entirely dependent on the algorithm used to generate the topology, the routing engine to handle data traffic, the number of devices or aggregate load on the network, and the levels of pseudo-randomness to avoid undue transmission conflicts in the network. The major motivation when trying to build a mesh topology is to have an algorithm such that the network exhibits a behavior of low-effort scalability. This low-effort scalability provides a larger theoretical graph size for a set of nodes and reduces the computational power/transmission energy necessary to update the network's arrangement. A truly effective mesh network is deployed using identical devices running the same software, and arranged without worry of network congestion. Additionally, reduced computational load and uniqueness in the network reduces SWaP requirements – a critical factor in many applications of the technology.

The nuance between scalability, efficiency, and performance exists mainly in the edge cases. An edge case can be thought of as an outlier in the graph, for example, where one cluster of nodes connects to another. It is assumed then the nodes used to connect two clusters will exhibit worse performance than those farthest away from other nodes in the topology. It is also necessary to consider the fact that these edge cases are often mission-critical as they provide critical network pathways, and are heavily dependent on the scalability tolerance of the network.

2.2.1 Scalability and Performance

To demonstrate the immediate need for a scalable topology in a mesh network, it is possible to conduct a small study of path choice. Assume that a routing algorithm has developed a topology that links to all nodes within the graph. Therefore, each node is connected to every other node directly, and through all other nodes. This circumstance is often the case in areas where a graph is dense due to the desire for propagation characteristics of a given modulation scheme to permit long-distance communications. Regardless of any optimization performed to the network to route data through it, there is a finite set of paths that a message could take. This number of paths is of

superpolynomial form and is represented for a bidirectional link as

$$P_d = \frac{n^{n-1}}{2} \quad (2.1)$$

where P_d is the number of paths for a bidirectionally connected graph and n is the number of nodes present in the graph. Similarly, the equation arises for the simplex case, where instead we define P_s being the number of paths in a unidirectionally connected graph to be $P_s = n^{n-1}$. Both of these equations exhibit a limiting behavior of $O(n^n)$, which is one of the worst metrics achievable in terms of computational efficiency. Exactly how bad the results for P_d and P_s are shown in Figure 2.4, where we examine the scalability of the graph as node count increases.

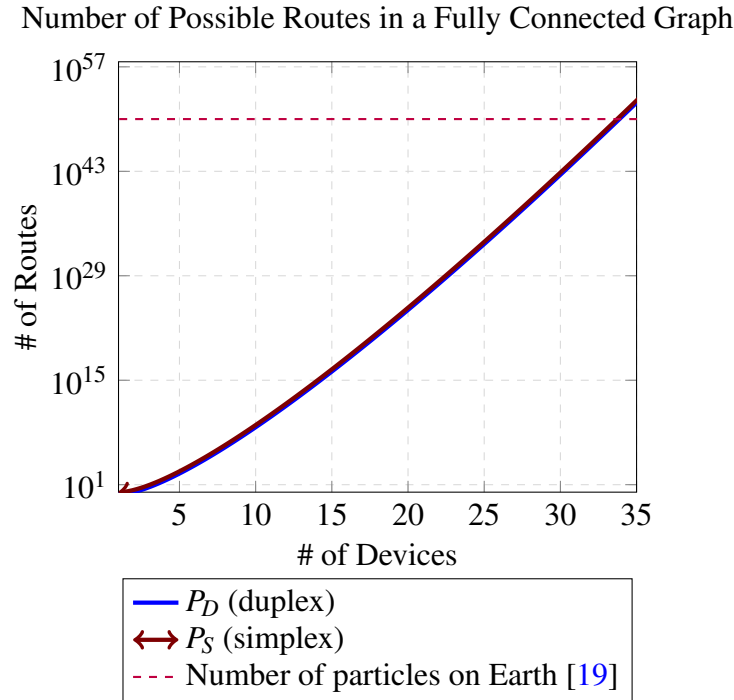


Figure 2.4: Scalability of unmanaged mesh networks

The $O(n^n)$ efficiency of this topology presents an enormous disparity between the number of nodes and the number of potential routes for data to take within the network. A relatively sparse network comprised of approximately 30 devices has over 10^{50} possible routes to deliver a message with an efficiency of $O(n^n)$. This result is not acceptable, and a more clever (efficient) solution is

necessary. Fortunately, this problem has been identified by mesh topology researchers as a major goal to optimize against, and many solutions exist to dramatically cut down on the number of utilized links in a mesh topology.

2.3 Characteristics and Metrics of Wireless Propagation

The wireless space has countless benefits that make it a highly capable medium to transmit data long ranges at comparatively low frequencies (sub-GHz) without major difficulty [8]. This feat has been accomplished by a mixture of material science, analog, and digital processing systems which many will consider (myself included) to be a modern-day miracle. The propagation of a wireless signal also has a significant impact on network performance, and the strength of a received signal is frequently used as a metric of signal quality. Analysis of wireless propagation is an incredibly complex subject [8]. However, the core of most analysis simplifies down to a measure of signal strength at a given distance.

The Friis Transmission Formula

Many methods exist to calculate the attenuation of a signal in different mediums; however, most stem from the Friis transmission formula, a basic extension of the inverse square law of a radiator's field strength at a distance. The Friis transmission formula is defined as

$$\frac{P_r}{P_t} = D_t D_r \left(\frac{\lambda}{4\pi d} \right)^2 \quad (2.2)$$

where P_r is the received power, P_t is the transmitted power, D_t and D_r are the directionality constants of the radiators utilized, d is the distance between radiators, and λ is the wavelength of the signal further defined as

$$\lambda = \frac{c}{f} \quad (2.3)$$

Within the scope of a mesh network that contains identical devices throughout, it is reasonable to assume $D_t = D_r = 1$. Additionally, inaccuracy due to changes in the wave velocity as a result of relative environmental permittivity, ϵ_r , of the wave is insignificant as the model fails to account for resistive loss, frequency drift on transceivers, signal interference, etc. Accordingly, we develop a

simplified version of the Friis transmission formula as

$$\text{FSPL} = \frac{P_t}{P_r} = \left(\frac{4\pi df}{c} \right)^2 \quad (2.4)$$

and as decibels are often a useful metric in the analysis of signals in the RF domain, we derive the decibel representation of the model in equation 2.5.

$$\begin{aligned} \text{FSPL (dB)} &= 10 \log_{10} \left(\left(\frac{4\pi df}{c} \right)^2 \right) \\ &= 20 \log_{10} \left(\frac{4\pi df}{c} \right) \\ &= 20 \log_{10}(d) + 20 \log_{10}(f) + 20 \log_{10} \left(\frac{4\pi}{c} \right) \\ \text{FSPL (dB)} &= 20 \log_{10}(d) + 20 \log_{10}(f) - 147.55 \end{aligned} \quad (2.5)$$

This model is the foundation for most analysis techniques for signal propagation. That being said, the quantities involved leave no room for the impact of other interfering signals on the receiver system. Additionally, the model assumes that major sources of interference are broadly outside of the Fresnel zone. Objects within the zone cause the model's accuracy to break down significantly, especially inside buildings.

The Log Distance Path Loss Model

The log-distance path loss model takes results from the Friis transmission formula seen in equation 2.5 and takes into account signal dispersion and/or loss due to physical or electrical interference. The model also incorporates random background noise in different environments. This model is defined as

$$\text{PL (dBm)} = P_{Tx_{\text{dBm}}} - P_{Rx_{\text{dBm}}} = PL_0 + 10\gamma \log_{10} \frac{d}{d_0} + X_g \quad (2.6)$$

Building Type	Frequency of Transmission	γ	σ [dB]
Vacuum, infinite space		2.0	0
Retail store	914 MHz	2.2	8.7
Grocery store	914 MHz	1.8	5.2
Office with hard partition	1.5 GHz	3.0	7
Office with soft partition	900 MHz	2.4	9.6
Office with soft partition	1.9 GHz	2.6	14.1
Textile or chemical	1.3 GHz	2.0	3.0
Textile or chemical	4 GHz	2.1	7.0, 9.7
Office	60 GHz	2.2	3.92
Commercial	60 GHz	1.7	7.9

Table 2.1: γ parameters for common environments and operational frequencies [8]

where $P_{Tx_{dBm}}$, $P_{Rx_{dBm}}$ are, respectively, the transmitted and received power in dBm. PL_0 is the close-in reference distance obtained using equation 2.5, γ is the path loss factor, d is the distance between radiators, d_0 is the reference distance (the spatial “noise floor”) of the transceivers, and X_g is a gaussian random variable to model flat fading of the communication channel.

Through empirical data collection, values of γ and the σ for X_g for common frequencies and environments can be found in Table 2.1. This model, in comparison to the more basic Friis transmission formula, provides significantly better accuracy for single-tone modulation schemes. When using spread-spectrum techniques the model becomes more difficult to define, as the demodulation techniques of a spread-spectrum signal often result in non-linear attenuation when observing the spectral power density of a signal. Values of d_0 vary depending on the modulation scheme and transmitter power utilized, however for a low-power system like the one discussed in chapter 3, values between 1 – 2km are reasonable [8].

2.3.1 Received Signal Strength Index

Derivatives of free-space and log-distance path models eventually result in the quantity known as recieved signal strength index (RSSI). This index is a measure of received power and is typically defined in the intermediate frequency (IF) stage of a receiver system, although modern direct-rf processing chains define this in the DSP section of the signal chain. RSSI is typically

defined within a certain range by the hardware vendor due to proprietary processing systems in the hardware and software. For example, the RSSI of an radio frequency identification (RFID) tag is defined as

$$dBm_e = -43.0 - 10.0\gamma \log_{10} \left(\frac{r}{R} \right) \quad (2.7)$$

where dBm_e is the estimated active received power in the tag, γ is the path loss constant as defined in Table 2.1, r is the distance to the receiving device, and R is the effective radius of the transmitter. This model is simple but provides some insight into how a given signal's strength is determined. The IEEE 802.11 standard does not define exactly what thresholds should be used for RSSI of the signal, although broadly the closer a signal is to 0 dB, the more power is received. A direct application of the RSSI metric is often used in cognitive radio networks to determine channel activity and/or avoid crosstalk with other clients on the same band. MIMO systems, which also have some form of RSSI, and spread spectrum systems, are much more complicated to estimate the method the indices are generated with, as the techniques used in the spread spectrum modulation schemes often permit higher performance once a received signal is de-spread from the carrier symbols utilized.

2.3.2 Signal-to-noise ratio

Another metric commonly found when characterizing the quality of a link is the signal to noise ratio (SNR). The SNR is a basic measurement of a received signal's power in comparison to the operational noise floor of the chip or transceiver in use. This simple relationship is defined as

$$\text{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} \quad (2.8)$$

where P_{signal} is the received signal power and P_{noise} is the operational noise floor of the signal. The noise floor of various modulation schemes varies greatly due to the spacing between symbols in an IQ constellation, or otherwise for spread spectrum techniques. Transceiver systems that use higher-order modulation are generally more susceptible to interference when SNR is high. This is

due to sideband information being lost below the noise floor. A crude plot illustrating this behavior is shown in Figure 2.5.

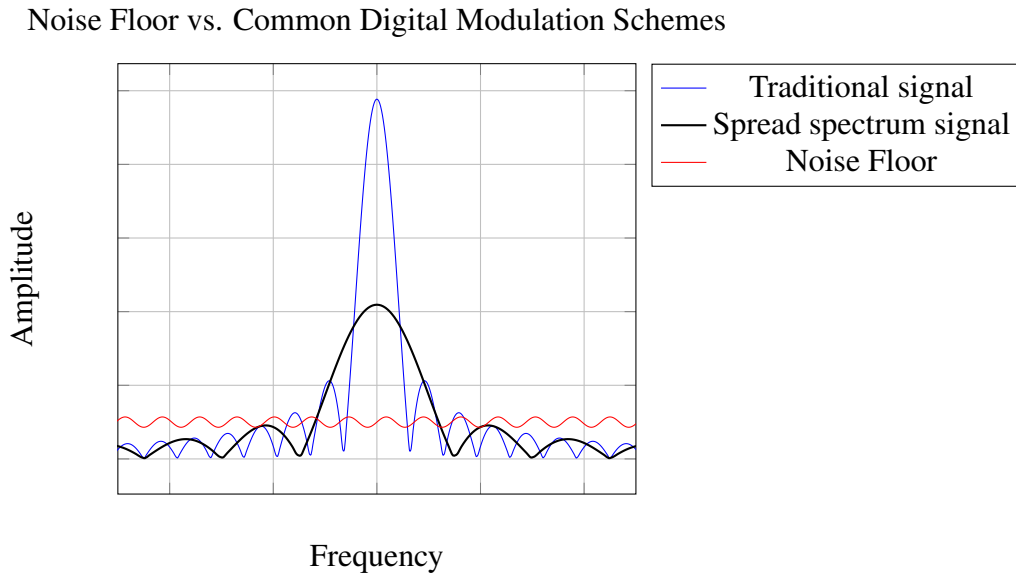


Figure 2.5: Modulation schemes versus signal noise floors for common modulation schemes

Further illustrating the effect of SNR on a signal, we implement a basic QPSK transceiver in GNURadio using a GreatScottGadgets HackRF SDR front-end. The results from the test are shown in Figures 2.6 and 2.7. The poor SNR with the initial signal is indicated by the symbol clusters having a large deviation from the expected positions of $(-1, -1)$, $(-1, 1)$, $(1, -1)$, and $(1, 1)$ which correspond to the signal being in-phase (I) and in quadrature (Q) on the IQ plot. Techniques to correct this effect, like polyphase clock synchronization and Costas loops, are frequently employed to permit operation in such environments. Figure 2.7 shows the same constellation seen in Figure 2.6 after it is digitally processed to correct for local oscillator (LO) mismatch, timing offsets, and mismatch in sampling rates between systems.

For traditional high-order modulation techniques (phase shift-keying (PSK)) or phase amplitude modulation (PAM)) it is easy to understand the relationship that SNR has on the system's performance. A higher noise floor will remove symbols from the receiver, as the receiver will filter out what it presumes to be noise. In the spread spectrum space, it is not uncommon for systems to operate reliably below the noise floor. Direct sequence spread spectrum (DSSS) and chirp

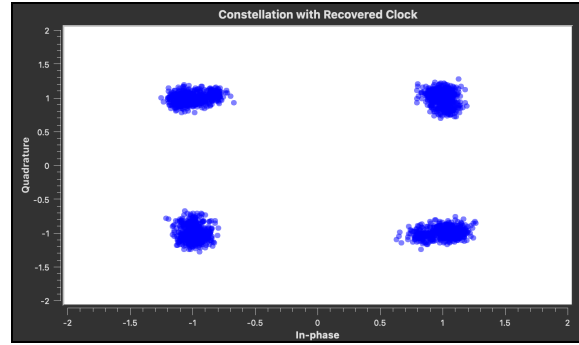
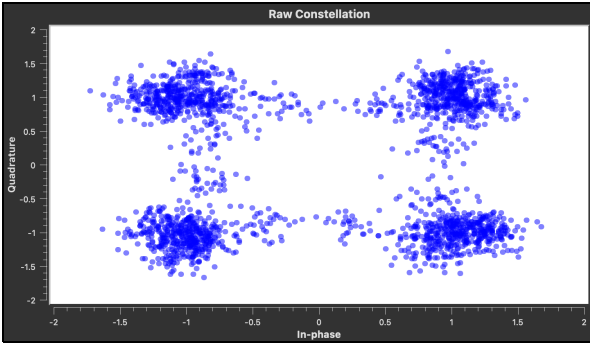


Figure 2.6: QPSK constellation with low SNR Figure 2.7: QPSK constellation with high SNR

spread spectrum (CSS) techniques both mathematically spread the signal over such a large area that it is entirely probable that the signal appears as noise to receivers who do not have properly synchronized de-spreading units.

Some spread spectrum modulation schemes, like LoRa, can operate below the noise floor [20]. Additionally, the CSS signal of a LoRa symbol demonstrates that it is resilient to frequency and timing offsets due to multipath interference [21]. Another advantage of a system that operates below the noise floor is that the detection of such a signal is difficult using traditional band power detection methods. To detect a signal below the noise floor, it is commonplace to first apply a bandpass filter, and then low-noise amplifier (LNA) on the frequency range in question.

2.4 The LoRa Modulation Scheme

In section 2.3.2 we allude to the existence of a modulation scheme known as LoRa. LoRa is a proprietary modulation technology developed by Cycleo (now Semtech) and patented in 2014 under US patent 9,647,718 B2 [22]. LoRa uses a spread spectrum modulation that is similar to and a derivative of traditional CSS modulation techniques. Each symbol is composed of a frequency sweep with defined start and stop sweep limits. This can be thought of as a chirp. The start and stop frequencies are defined by the center frequency f_c and specified bandwidth B . The span of a chip is then defined as

$$(f_c - B/2, f_c + B/2) \quad (2.9)$$

Within this span, LoRa specifies a quantity known as the spreading factor (SF) which determines the number of bits per symbol and indirectly determines the time-on-air for a given piece of data. The SF of the system ranges from [5, 12] bits per symbol. The SF chosen determines the number of symbols M required in the symbol space by $M = 2^{SF}$ [23]. When a symbol approaches the limit of its occupied bandwidth, it simply wraps around to the other end of the spectrum and continues its sweep. A LoRa chirp can be characterized by the time domain equation

$$x(t) = \exp \left(\theta_0 + 2\pi \left(\frac{FR}{2} t^2 + f_0 t \right) \right) \quad (\text{Hz}) \quad (2.10)$$

where

- $FR = \frac{f_1 - f_0}{T}$ is the frequency rate of change
- f_0 is the initial chirp starting frequency, which is bounded by equation 2.9
- f_1 is the final sweep ending frequency, also defined by system bandwidth B
- $BW = 125 \text{ KHz}, 250\text{KHz}, \text{ or } 500\text{KHz}$ (dependent on the transceiver used)
- $T = M/BW$ is the chirp period and number of SF cycles required to fit within the constrained bandwidth
- $SF = 7, 8, 9, 10, 11, \text{ or } 12$ and determines the number of bits per symbol

There are obvious tradeoffs that are made in equation 2.10 concerning energy consumption, data rate, and signal integrity. A lower SF corresponds to higher data rates but reduces the signal integrity and effective range. Conversely, a higher SF implies greater signal integrity and effective range. Coupled with a higher SF is a longer time-on-air and higher energy content provided in the signal [23]. This can reduce the efficiency of the network, but increase its range between nodes. Depending on the directionality of the antennas used and a properly designed amplification circuit, LoRa systems can function successfully up to 3 miles in an urban setting and substantially longer in rural areas [24].

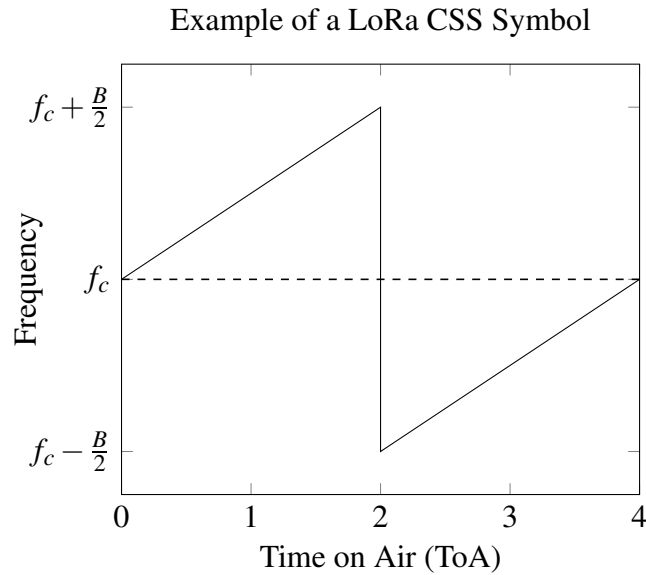


Figure 2.8: Example of a LoRa symbol

The operational frequency of the system favors medium-range propagation, and the method of modulation used permits transmission in environments where traditional PSK or PAM schemes would fail. LoRa transceivers commonly find themselves in IoT applications such as smart homes, agriculture, infrastructure, and industry. Additionally, LoRa successfully operates below the noise floor with a link budget upwards of 150 dBm [24]. The noise floor of a system's signal is generically provided by

$$NoiseFloor = 10\log_{10}(kTB_{1Hz}) + 10\log_{10}(BW_{Rx}) + NF \quad (2.11)$$

where k is the Boltzmann constant, T is the temperature in Kelvin of the system, B_{1Hz} is 1-Hz normalized bandwidth, BW_{Rx} is the bandwidth of the receiver circuitry, and NF is the receiver noise figure in decibels. Using this equation, it is possible to approximate the noise floor of the LoRa system employed [8]. For example, the Semtech SX1262 transceiver cites receiver sensitivity up to -148 dBm – an impressive result and one that permits operating in adverse electrical environments with extremely low power [4]. When paired with an effective front-end, LoRa transceiver systems can reliably send data where many other systems fail.

LoRa systems operate at sub-GHz frequencies. In the United States, LoRa systems are present in the US902-928 frequency band (902 – 928 MHz) [25]. This spectral portion is an unlicensed, power-restricted Industrial Scientific and Medical (ISM) frequency band commonly used by consumer devices [26]. Despite the power restrictions of approximately 100 mW, effective LoRa installations commonly transmit signals up to 10km [24]. In Figure 2.8, we let f_c be the center carrier frequency, and B be the occupied bandwidth of the LoRa symbol. Note that the start and end of a symbol occur at the same value. Exactly which set of bits this symbol corresponds to depends on other factors like the spreading factor of the symbol. Figure 2.9 shows a real-world LoRa signal transmitting random bytes of data. In both figures, the spread-spectrum nature of the signal is blatantly apparent.

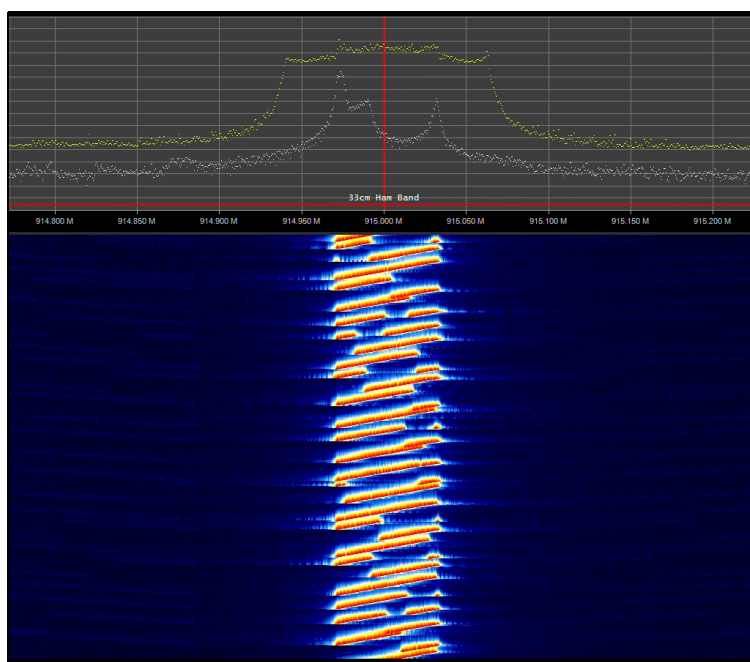


Figure 2.9: Waterfall chart of a LoRa signal

2.5 Routing algorithms

The most complex and nuanced element of a well-planned mesh communication system is the routing algorithm employed. The ability of a system to route data through the network is a centrally important metric of overall performance [27]. Moreover, a clever algorithm achieves the twofold impact to reduce network load while also increasing overall efficiency [28]. Recall that section 2.2 provides that an unmanaged network routes data with efficiency $O(n^n)$. To combat this issue, algorithms such as DVR, Destination-Sequenced Distance Vector Routing (DSDV), and Dynamic Source Routing (DSR) (among many) have been developed and exhibit good performance. When paired with a LoRa transceiver system, DVR demonstrates favorable qualities in terms of functionality and ease of implementation.

2.5.1 Distance Vector Routing (DVR)

Distance Vector Routing (DVR) is a commonly used mesh routing algorithm based on the Bellman-Ford algorithm. Its first widespread use was to route packets in ARPANET and has later evolved into widespread use in ad-hoc wireless networks. The concept of operation for DVR is simple to implement on low-cost devices, and the quantity of data required to generate the network is low enough to support low-bitrate communication links [27]. This simplicity brings to light issues such as the count-to-infinity problem, where a path from one node to another may use the parent node as part of the path, resulting in system loops. A method to combat this problem is loop detection or sequencing of the packet's travel history; however, a sparse network generally does not suffer from this problem.

A Four-node Topology Study

Individuals familiar with the Bellman-Ford algorithm will quickly see that DVR is simply an application-level abstraction from the mathematical model. To illustrate this fact, consider the sparse graph of three nodes as shown in Figure 2.10. For simplicity, we consider each link to be

duplex. Below each node is the forwarding table for data that finds itself at that node. If a node (denoted 'D') wishes to join the network, it does not know where it should send data, as illustrated in Figure 2.11.

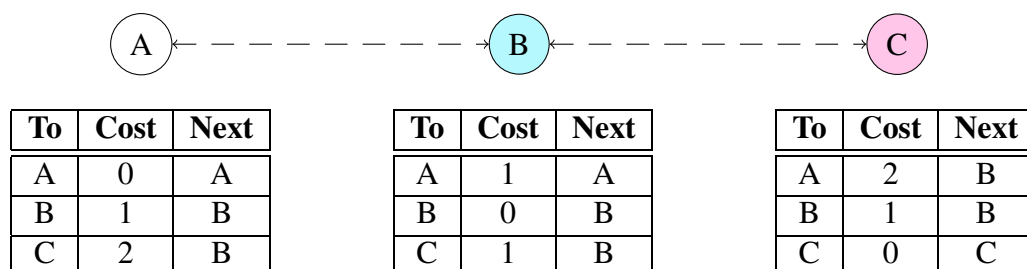


Figure 2.10: Initial DVR stage

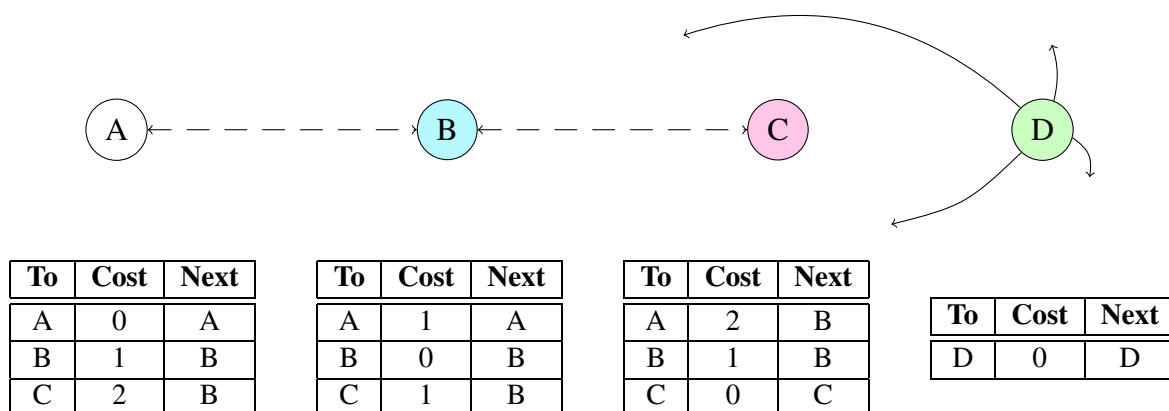


Figure 2.11: Node D begins to join the network

In Figure 2.11, node D begins to join the network by broadcasting its “existence” to the network under the assumption that devices are available to communicate. The node broadcasts its forwarding table (under node ‘D’), which contains a list of reachable nodes, their cost, and the cost to reach all of those nodes. Nodes that receive D’s message calculate the cost of communication using metrics such as RSSI and SNR to determine the link viability. After this information exchange, node C updates its forwarding table (Fig. 2.12) to indicate that node D is a destination it can reach.

The modification of C’s forwarding table triggers node C to start its process of broadcasting its connection paths. It should be noted that all of these processes should occur within a certain time

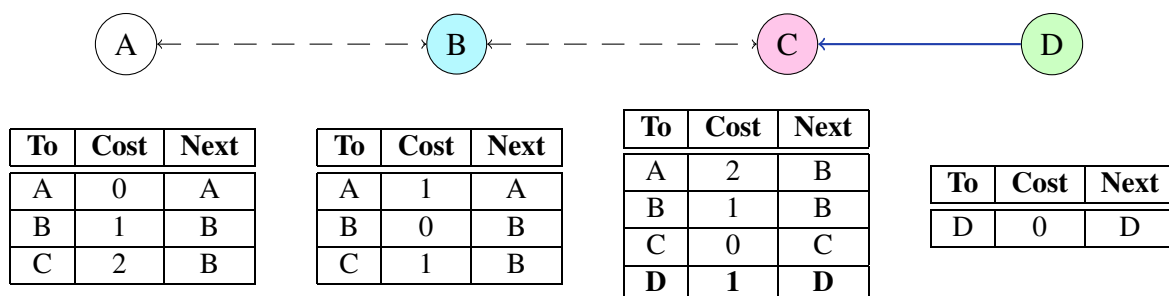


Figure 2.12: Node C updates with data from D

interval and should occur randomly within that interval. This basic method of traffic control helps to avoid network collisions.

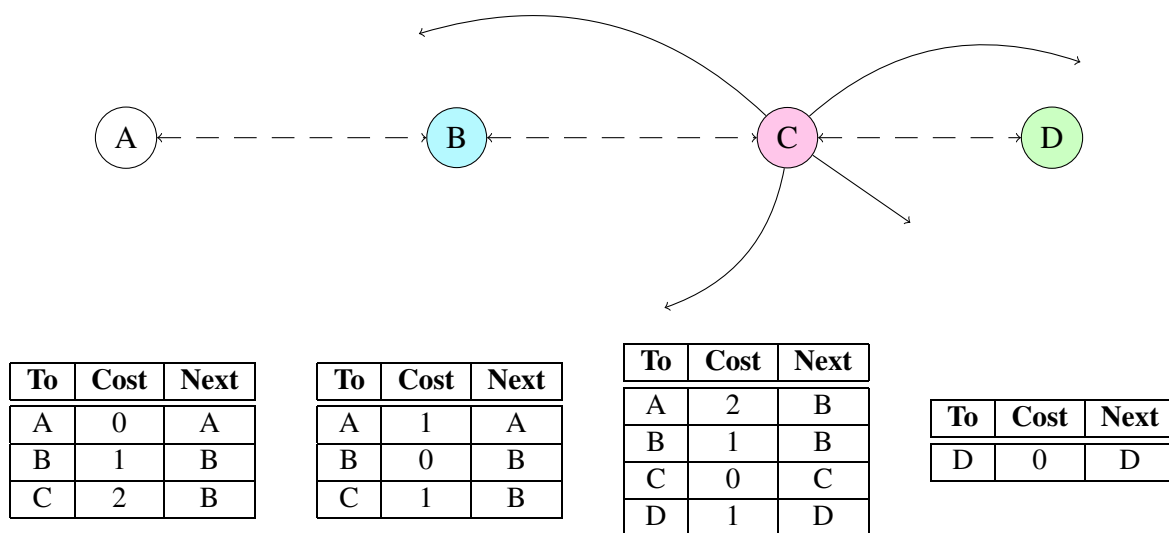


Figure 2.13: Node C broadcasts its forwarding table to local nodes

Nodes that can receive the data from C update their forwarding tables to reflect the addition of node D into the network. About Fig. 2.13, nodes B and D both receive data from C and can update their tables. A forwarding table is only updated if (a) the link quality is higher than previously received and (b) if the cost to reach a given node in that table is lower than previously received. This lowest-cost/shortest-path mindset permits the lowest-effort data transmission in the network. The topology of this network does not essentially illustrate this fact, but more complex topologies benefit greatly from updating their tables in an interval with pseudo-random time between updates. The algorithm then repeats and propagates the data change through the network. A node will only broadcast its forwarding table if it has changed data, or if the random update interval has passed.

This ensures convergence of the algorithm and stops the messages from flooding constantly in the network. This effect is shown in Figure 2.17, where the graph has completely adjusted to the topology change with the addition of node D.

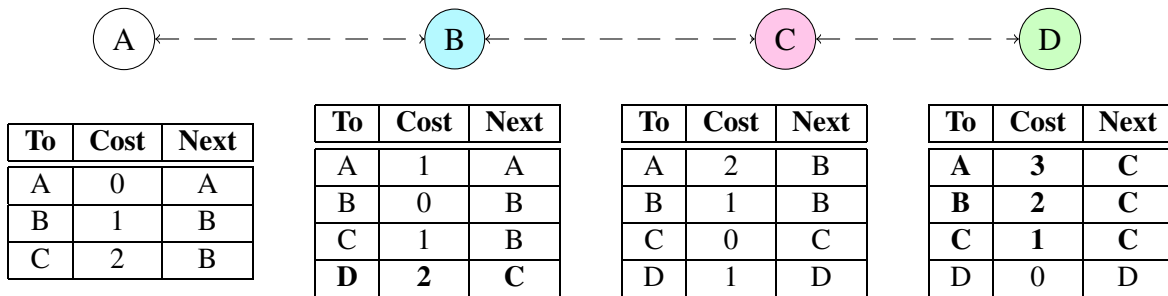


Figure 2.14: Nodes B and D receive broadcast from C and update their tables

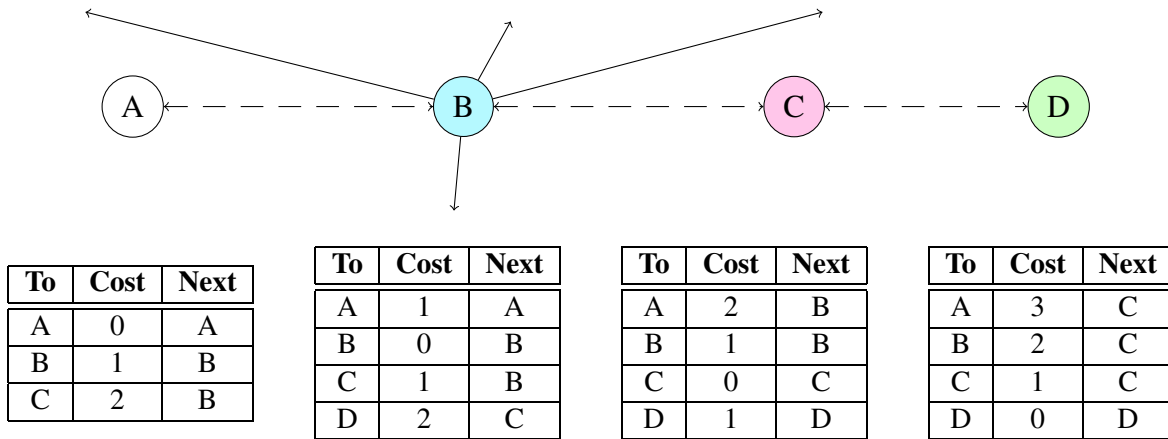


Figure 2.15: Node B broadcasts its updated forwarding table to local nodes

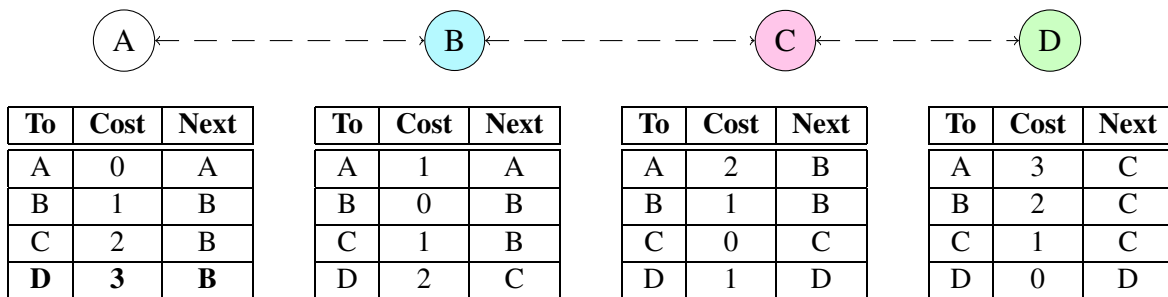


Figure 2.16: Node A updates its forwarding table with data from node B

The forwarding tables present in each node when using DVR are simple and easy to understand. The physical implementation of DVR is similar to the topology study discussed in this section,

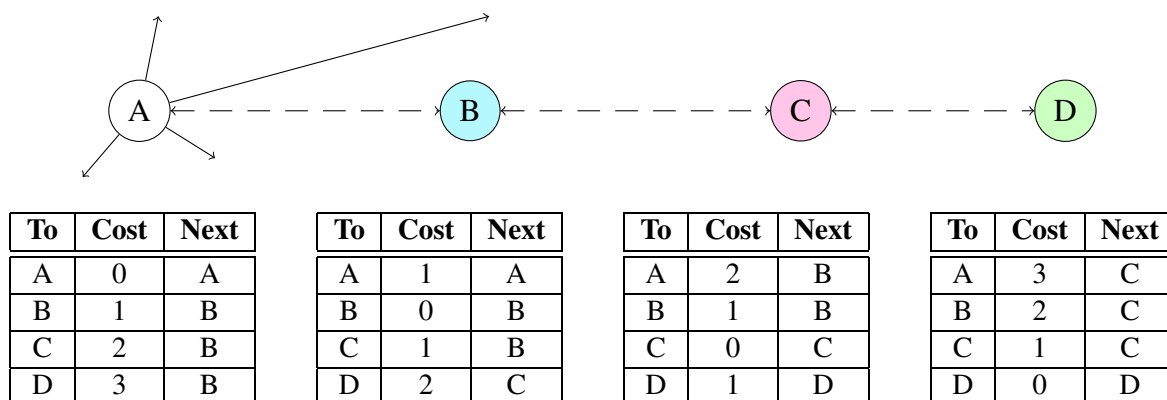


Figure 2.17: Node A broadcasts its updated forwarding table to local nodes

although the notion of global versus local understanding of the network structure must be carefully considered. Much of the algorithm can be implemented via simple finite state machine logic, given that efforts to eliminate deadlock or non-convergence are present. Randomization in the system is used throughout to eliminate communication channel conflicts. Other techniques, like listening before transmitting, or waiting until the channel is open before attempting transmission. This further reduces the notion of a collision within the network.

Chapter 3

Design and Implementation

This chapter discusses the physical implementation of the mesh communication system using design principles discussed in chapters [1](#) and [2](#). While the theoretical situations and challenges presented in these chapters do have a significant influence on the design of the system, much of the nuance surrounding the true implementation of the communication system arises when working to develop the hardware.

Within this chapter, we first discuss the design requirements and targets we wish to meet. Then, we develop process architecture and data structures necessary to satisfy those requirements. With the architecture defined, we then construct the system and discuss the limitations of the implementation. The system is then tested in multiple environments and its performance characteristics are analyzed. Finally, we conclude with a discussion about the system's limitations, lessons learned, and where future improvements should be made.

3.1 Design Requirements

In any design, the requirements or metrics to which something is designed are a crucial set of parameters that ensure the system’s safety, stability, and overall functionality. In this section, we quantify the design criteria into synthesizable design points which are then used throughout the rest of the development and implementation. Within the context of the mesh communication system, the design requirements are essential for two primary reasons: to ensure the system’s alignment with end-user needs and expectations, and to guarantee compliance with legal regulations, particularly those relating to spectral emissions. While the end-user functionality can largely be determined via qualitative criterion, the legal requirements are much more well-defined by quantitative regulation. Accordingly, it is much more difficult to satisfy the end-user requirements than it is the legality of the design. The FCC has well-defined rules and guidelines to ensure compliance within the operational frequency of the system. This is not the case for the end user, which presents a unique set of challenges to satisfy via more qualitative metrics.

3.1.1 End-User Design Criteria

The end-user of the system is a complicated subject whose design criteria are rather subjective. Naturally, the system should provide “maximum performance in any environment”. While we agree that this is an ideal design criterion to meet, it is unreasonable to use this as the main criterion as it is a mainly qualitative metric – one of which is very difficult to obtain, and the ultimate goal of this field in general. A more reasonable, yet substantial, set of criteria should produce acceptable performance within the scope of the project’s goal to develop a wireless mesh communication system. To achieve this goal, we develop the high-level architecture illustrated in Figure 3.1. This architecture is loosely based on the five-layer network model illustrated in Figure 1.5, which demonstrates proven performance in internet architecture [3]. Furthermore, we understand that using such a system architecture permits encapsulation of design elements and as a result, the system’s design requirements are implemented as functional units.

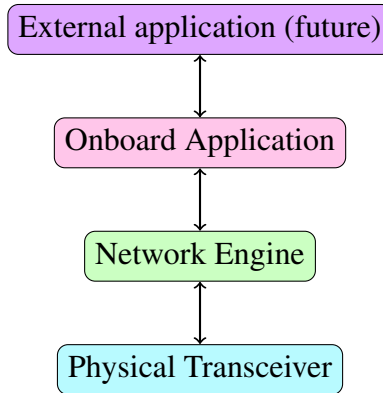


Figure 3.1: High-level system architecture

Given the system architecture and the general motivation to develop such a system as described in Chapter 1, it is logical to use low-cost hardware to develop the solution. When discussing “low-cost”, cost is, like many things in life, a relative term. Additionally, cost per unit functionality is also necessary to consider, as are the expected utilization and implementation hurdles when choosing a technology. The mesh communication system *could* exist entirely in software on a classical computer and uses a simple interface with the transceiver chipset. Such a system would have great binary compatibility and a low hurdle to widespread deployment but require a comparatively large and inefficient piece of hardware. A fully custom solution could use an FPGA with a softcore processor for user interaction, which would yield perhaps the best performance but at a high cost of development and deployment. Somewhere in between a fully configurable logic device and a classical computer exists the microcontroller. Microcontrollers offer low-level access to logic when necessary and also provide a fantastic platform for building rich user interfaces and complicated process control. Many modern microcontrollers perform exceedingly well in these low-resource systems, and when paired with an effective transceiver system there is a large amount of functionality possible with proper firmware. Of course, this is just one of the design criteria necessary to consider.

In Table 3.1, we outlay a set of major design criteria that span the communication system’s layers, its physical constraints, and the end-user requirements. Broadly, we can synthesize this set of requirements into what should be a self-organizing, self-managing set of nodes that can share

Table 3.1: End-User Design Criteria for the Communication System

Design Area	Criteria
Network Engine	The system shall support mesh network topologies, and react dynamically to network changes without user intervention.
Application	The system shall be able to send and receive data to nodes reachable in the topology.
Physical Transceiver	The transceiver shall employ some level of error detection and correction to validate data integrity.
Application	The system shall provide a suite of automatic tests to obtain metrics of link quality and network performance.
Overall	The system shall remain available to both the physical and application layers at all times. At no point should either layer have to wait for another to finish an operation to operate under normal conditions.
Network Engine	The system shall make a best effort to avoid network collisions
Overall	The system shall be low-power and host some level of message storage such that immediate user intervention is not necessary to permit normal operation.

network data as needed to maintain the network, *and* provide end-users of the system with a reliable network to share data. For testing and validation, it is also necessary to have test systems or sub-routines that validate functional units in hardware and software. As an added benefit, Table 3.1 acts as quantifiable metrics of qualification to use when performing tests and validation of the communication system.

3.1.2 Legal Requirements

As discussed in the introduction, the communication system must comply with the legal regulations provided by the FCC. The broad application of RF in the United States has led to the classification of devices into certain categories. The most common type of device encountered in daily life are unlicensed devices. Unlicensed devices are further classified as *Intentional* and *Unintentional* radiators. The rules for unlicensed devices are contained in 47 CFR §15.247, more commonly known as FCC Part 15 [29]. Intentional radiators are devices that intentionally generate and emit RF energy and are pervasive in everyday life. These types of devices include cordless

phones, remote control toys, internet of thing devices, and other low-power transmitters in more embedded applications. The mesh communication system falls into the same category and is subject to the same regulations. It is possible to pursue a license to operate at a given frequency above power limits, although this massively reduces the portability and ease of deployment for a set of nodes. We do not find it necessary to obtain licensure to operate. As such, we are restricted in frequency space to a portion of the spectrum in the unlicensed bands. These unlicensed ISM bands exist at 902-928MHz, 2400-2483.5MHz, and 5725-5850MHz. Within this subset, the 902-928MHz band exhibits the best propagation per unit power, at the expense of lower data rates. Frequencies above ~ 1 GHz begin to exhibit severe attenuation and require more power to transmit, hence the need for licensure for cell phones, radar, and other more powerful transmitters. To avoid individuals using immensely high power transmitters in this space, 47 CFR §15.247 subjects these unlicensed, intentional radiators to certain power restrictions, with the following verbiage present in the text:

For systems using digital modulation in the 902-928 MHz, 2400-2483.5 MHz, and 5725-5850 MHz bands: 1 Watt. As an alternative to a peak power measurement, compliance with the one Watt (sic) limit can be based on a measurement of the maximum conducted output power. Maximum Conducted Output Power is defined as the total transmit power delivered to all antennas and antenna elements averaged across all symbols in the signaling alphabet when the transmitter is operating at its maximum power control level. Power must be summed across all antennas and antenna elements. The average must not include any time intervals during which the transmitter is off or is transmitting at a reduced power level. If multiple modes of operation are possible (e.g., alternative modulation methods), the maximum conducted output power is the highest total transmit power occurring in any mode [29].

In summary, the policy provides a limit of +30dBm total EIRP across the full bandwidth of the system. Because it is determined across the full bandwidth, it is important to understand the impact that reduced (or increased) bandwidth may have on the legality of this unlicensed intentional

radiator. Figure 3.2 illustrates the difference between a traditional and spread-spectrum signal. A traditional signal occupies a smaller segment of bandwidth, and can therefore transmit with higher absolute power at a given frequency. The spread spectrum signal, on the other hand, is limited in absolute power due to the signal's spread over a larger segment of bandwidth. To ensure that the system is compliant with these regulations, it is possible to generate sample transmission data and measure the power output at the antenna port using calibrated measurement equipment. In doing this, we ensure that the system complies with bandwidth utilization and energy emissions. Additionally, validation that the system does operate at maximum spectral power ensures that we emit the maximum legal amount of power, which has the benefit of added range and/or reliability in high-interference or long-range environments.

Amplitude vs. Frequency Occupation for Traditional and Spread Spectrum Radio

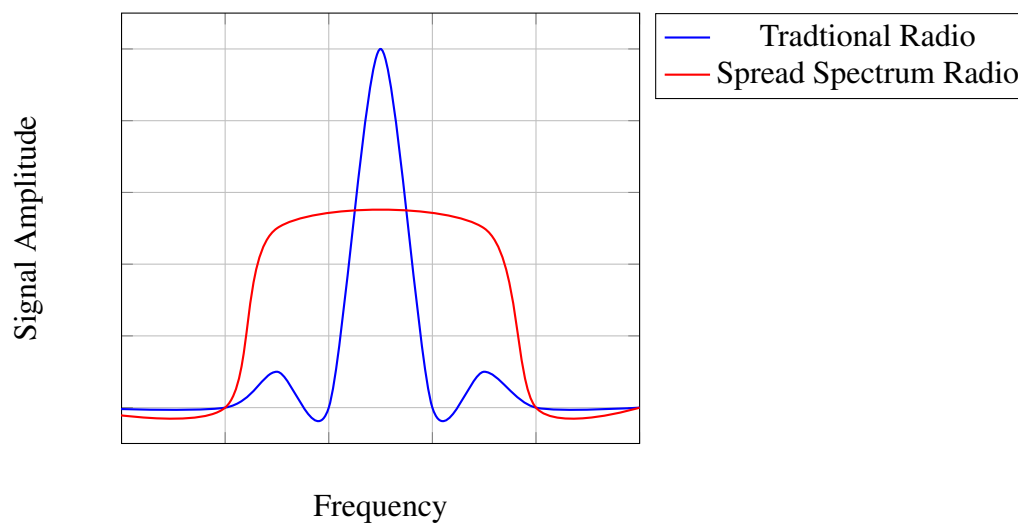


Figure 3.2: Occupational bandwidth of different radio systems

3.2 Hardware Selection

A centrally important topic of discussion in the system’s design and implementation is the selection of hardware and its associated toolchain used to implement the communication system. As eluded to in the design requirements, there are a variety of methods that exist ranging from strictly software to low-level logic. While a software solution would entertain the task, the assessment of system performance is much more interesting and substantial when a physical implementation is used. Theory defined in Chapters 1 and 2 yields a somewhat generalized set of tasks and functions that the system must perform, from handling low-level hardware to generating the mesh network itself. Moreover, the physical implementation of a solution always provides interesting nuance and challenges that tend to yield more impactful results – those that illustrate shortcomings experienced only in the natural world. When selecting hardware for the communication system, we find a set of criteria that the hardware should look to satisfy:

Low energy consumption hardware will yield longer system runtime and more efficiently leverage the onboard processing power. It is woefully unintelligent to use a desktop gaming computer for the system, as is the choice to implement the entire project on an 8-bit microcontroller.

Low cost for the complete set of hardware in a node lowers the barriers to scalability for the system. In a mesh network, we expect many nodes to be present in the given network. Accordingly, the deployment cost is a function of the number of nodes n , so any reduction in cost benefits greatly from the change.

High reliability hardware enables functionality in adverse conditions. These adverse conditions could come in the form of environmental factors like weather, shock, and chassis intrusion; or could come in the form of heavy interference like that found in basements. Reliable hardware will permit operation in these conditions without removing a user from the network.

Portability permits end-users of the system to deploy the nodes where they deem it most convenient, or where they can. Because the network is a mesh, the system will generate a dynamic topology that does not require regular or static arrangement.

Ease of use reduces the barrier for individuals to use the communication system, especially those who are not familiar with how the system should work. Well-explained user menus and thorough input validation should be included, as should physical indicators on the hardware about what the device is currently doing.

Reliable RF operation ensures that data makes it from point “A” to point “B”, through “C”, without undue corruption. Additionally, methods exist inside the transceiver to dynamically compensate for non-ideal transmission or receipt environments to make a “best effort” to generate data for processing.

Rich debugging and logging capabilities greatly reduces headaches when developing the system, and gives insight to future system developers about how the system operates. Without this capability, it is incredibly difficult to pinpoint hardware bugs and failures.

With these hardware selection criteria in mind, we are presented with a complex albeit open-ended set of hardware targets to meet. There is a fine balance between ultimate performance and portability that must be obeyed – one of which plays a direct part in the effectiveness of the network. High-performance networking devices like those found in wireless routers or cellular base stations heavily leverage FPGAs and ASICs to meet their system demands at the cost of size, weight, power, and cost. While both of these options would be incredibly interesting methods of implementation to explore, the development hurdles, timespan, and cost to complete such a task are unattainable. Instead, we turn to the domain of the embedded system to meet our design requirements.

3.2.1 System Controller

The system controller in the mesh communication system must handle user interaction, network maintenance, and communication with the wireless transceiver chipset selected. To achieve this goal, we require multiple tasks running in parallel on the system to avoid massive gaps in system response. A deadband is defined as the inability of the system to respond to either the wireless network or the user at the same time, due to a single-threaded linear architecture. Any time a budding electrical engineer hears “parallel”, they immediately look to use an FPGA as the target hardware. However, FPGAs are expensive and difficult to develop, and while they provide great performance, are often slower than their ASIC counterparts at doing the same task.

Research into device capability heavily suggests moving in the direction of the embedded system and using a microcontroller (MCU) as the system controller. MCUs are a class of devices that find themselves in every corner of our lives, from lightbulbs and toasters to spacecraft and military munitions. They provide a low-cost component that has a comparatively high amount of processing power. In the IoT space, MCUs frequently operates wirelessly in the ISM bands [26]. This has led to the advent of technologies like ZigBee, Bluetooth, and LoRaWAN (to name a few) work rather well alongside their MCU they operate on. One of the major devices used in embedded designs because of its capability is the Espressif ESP32. The ESP32 series (which tries rather hard to *be* an ST Microelectronics STM32, but is not) features a dual-core 32-bit Xtense LX6 processor suited for a wide range of applications, all of which are multi-threaded [7]. The dual-core architecture allows for this multitasking and concurrent execution through the FreeRTOS kernel which makes it rather suitable for real-time control, data processing, and wireless communication systems [30].

FreeRTOS is a powerful and versatile open-source operating system designed to meet the stringent timing and performance demands of embedded and/or real-time systems. The OS focuses on compact, efficient binaries and has gained substantial popularity in the world of microcontrollers [30]. Unlike a traditional operating system, FreeRTOS operates much closer to the base processor which permits low-level control of core device functions. The kernel architecture is

illustrated in Figure 3.3.

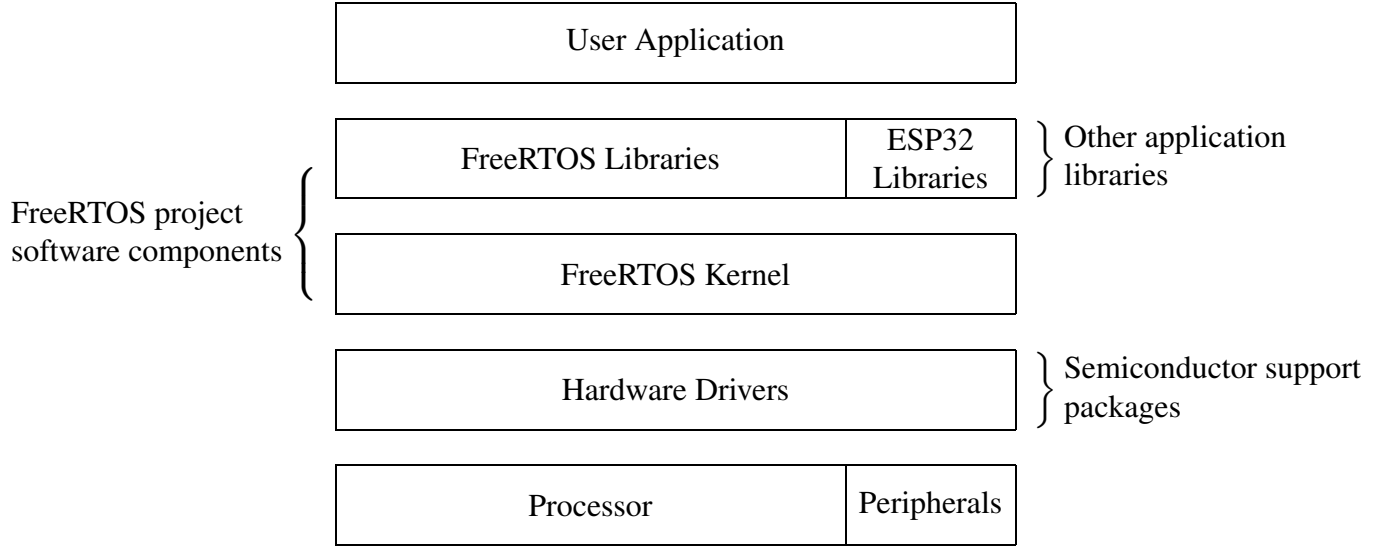


Figure 3.3: General architecture for FreeRTOS on the ESP32 microcontroller

An RTOS differs from a general-purpose operating system (Windows, Linux, macOS, etc.) in the sense that while it tries to maintain a deterministic execution pattern in thread execution, it does not have the wide gamut of functionality provided by its general-purpose counterpart. FreeRTOS provides the framework and functionality for advanced abstract data structures like tasks, lists, and queues. The advantage of using a MCU that runs a real-time operating system (RTOS) is particularly apparent when trying to satisfy multiple control planes within a given system [30]. Moreover, an RTOS provides a scheduled set of instructions to provide time-critical results typically necessary in embedded systems. This is achieved, to some extent, via pre-scheduling of instructions during compilation and is therefore deterministic in its behavior. Pre-scheduling towards a deterministic execution pattern is accomplished by thread prioritization set using scalar values in pre-compilation, or dynamically set in runtime with the invocation of new threads.

3.2.2 Transceiver System

Careful selection of a system controller does not matter if the wireless transceiver employed for a node is unsatisfactory in its performance. The transceiver acts as the bridge between nodes and is the only method to transmit and receive data from other nodes. We could then argue that it is *more*

important than the system controller – both sides of the argument hold. Traditional modulation techniques like amplitude modulation (AM) and frequency modulation (FM) offer reliable performance. However, the world of spread spectrum modulation schemes has the potential to provide much greater performance at the same (center) frequency. For example, the 802.11ax WiFi standard employs 1024-QAM modulation enabling a theoretical data throughput rate of 9000Mbit/s in an ideal operational environment [11]. Unfortunately, it is almost certain that normal usage of the device designed around this standard achieves even fractional levels of its maximum performance. The sensitivity of this high-order modulation scheme is highly susceptible to noise. Attenuation and interference are very real problems in a wireless system.

Aside from WiFi, which is a form of high-order modulation, is a set of modulation schemes in the spread spectrum domain. Primitive methods include frequency hopping spread spectrum (FHSS), direct sequence spread spectrum (DSSS), time hopping spread spectrum (THSS), and chirp spread spectrum (CSS). Designers have exploited these concepts and hybridized them with other schemes to develop extremely high-speed, high-security data links [8]. Spread spectrum systems are extremely versatile and often offer superior performance to their non-spread counterparts. This is, of course, not without additional complexities added to the transmitter and receiver designs.

As discussed in section 2.4, the LoRa CSS modulation scheme demonstrates fantastic propagation characteristics and reliability in high interference or long-range conditions. The link budget and receiver sensitivity are unmatched at similar levels of EIRP in other modulation schemes. LoRa, being a modulation scheme, can be implemented on a variety of different platforms that range from a dedicated RFIC to RF blocks on a microcontroller module. The need for an all-in-one synthesizer, DSP block, and power amplifier in conjunction with a receiver topology suggests a standalone on-chip solution. A popular choice is the Semtech SX1262 sub-GHz radio transceiver [4]. We could discuss at length the great degree of capability this chipset has, but that is not the focus and instead, we greatly appreciate the capability it provides. Notably, it features +22dBm of output power and permits a link budget of +170dBm. These results alone are rather

impressive and yet much of the processing nuance is captured within the embedded DSP blocks on the RFIC. A block diagram of the major system components is provided in Figure 3.4.

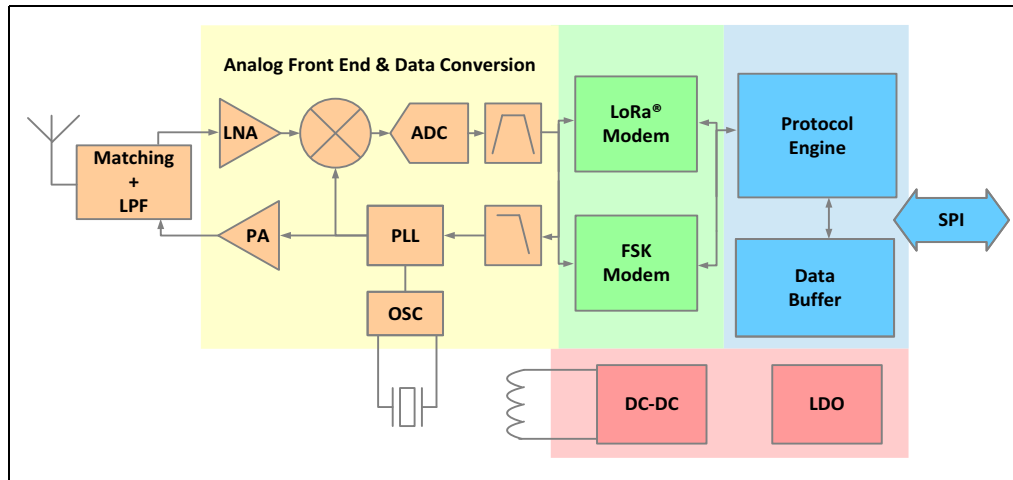


Figure 3.4: SX1262 block diagram [4]

In Figure 3.4 we observe that the RFIC contains other critical blocks found in a transceiver architecture. What is not often present in the traditional architecture are the LoRa and FSK modems available, in addition to the protocol engine. It is via these blocks that LoRa symbols are synthesized and processed, and through these blocks, such capability of the system is achieved [23].

Other miscellaneous componentry

The system controller and wireless transceiver comprise the bulk of the node's functionality. However, none of this is possible without some other supporting hardware (seen in Figure 3.5). To ease software development, signal taps, UART/JTAG interfaces, onboard displays, GPIO, and LEDs can be used to view the status of the machine at any given time. While a "commercial" deployment (no debug methods) needs very few of these things, without them it would be incredibly difficult to produce stable software in a timely fashion.

3.2.3 The Platform

Design criteria and hardware requirements discussed in sections 3.1 and 3.2 yield a capable set of hardware that, when combined with clever software architecture, can achieve the goals specified in the requirements. As the system controller, transceiver, and other miscellaneous componentry all exist as separate units, it is necessary to package the component set on a printed circuit board (PCB). Hardware requires certain voltages, oscillators, passives, and supporting integrated circuits (ICs) to properly function. The ability to contain all of these things on the same PCB also enables portability and ease of use. The hardware diagram of the complete system is illustrated in Figure 3.5.

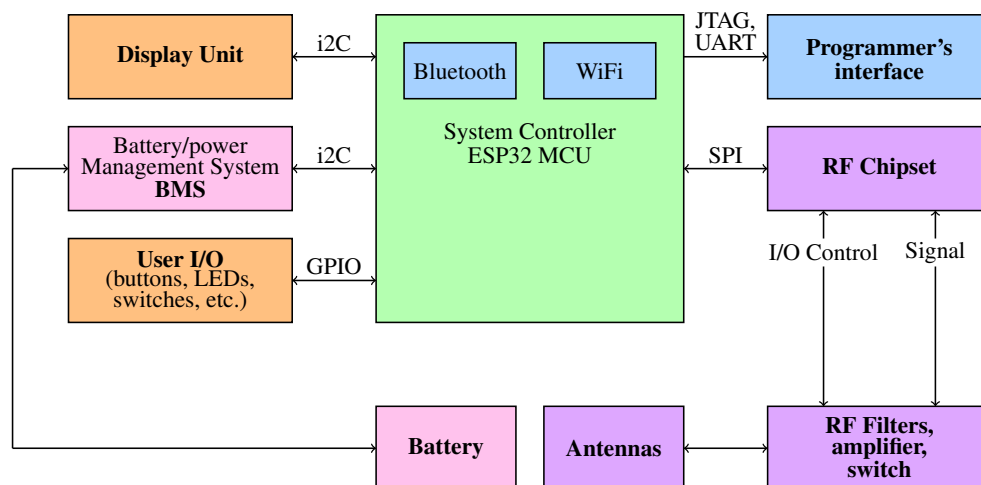


Figure 3.5: Hardware block diagram of the system as implemented

Hardware selected for the design aligns well with IoT developer needs. Many companies like Adafruit, Sparkfun, and resellers on Amazon and other e-commerce platforms produce low-cost PCBs that contain the required functionality. The cost of a device ranges from \$15 to \$30 depending on the available peripherals on the device. To physically produce the device is substantially lower, \$5, and while this option was considered an “off the shelf” option reduced development hurdles. A popular choice when searching “ESP32 SX1262” online is the Heltec WiFi LoRa 32 (v3). This low-cost development platform contains both the ESP32 and SX1262 (naturally) and also gives access to an OLED display, GPIO, WiFi, and Bluetooth.

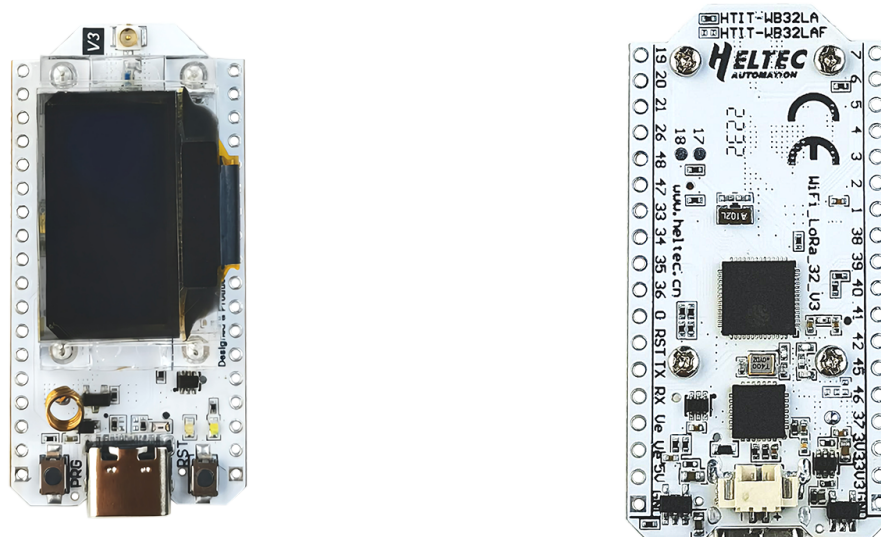


Figure 3.6: Heltec Automation WiFi LoRa 32 (v3) development platform [5]

3.3 Data Structures

The decision to use an RTOS as the underlying development kernel allows us, without a great degree of difficulty, to leverage more complex data structures such as queues, semaphores, and tasks. We further encapsulate data using object-oriented programming primarily through the usage of structures and enumerated types. This allows the generation and management of complex abstract datatypes not commonly available in the standard C language. When using the FreeRTOS kernel, we have the added benefit of task threading to permit a multi-layer control stack shown in Figure 3.1. Each layer in the stack functions broadly as independent functional units interlocked by abstract data structures. The combination of the threading and structures then permits nearly parallel execution of tasks on the processor.

The structures developed for the communication system encapsulate basic data types as objects that greatly decrease the developmental complexities of using purely primitive types. This can be thought of as object-oriented programming in C. With these abstract data types, FreeRTOS further permits process abstraction that is otherwise not possible in the base language like task scheduling, suspension, and deletion. There is a core need for the communication system to encapsulate network data transferred between nodes as packets. This concept is nothing new - the internet has used a packetized architecture since its inception [15]. Within this packet are several fields necessary to satisfy the network architecture and the end-user's needs. A data packet in the communication system is defined as a `message_t`. These messages are sent between FIFO queues in the system between process threads.

3.3.1 The Data Packet

Messages in the communication system, encapsulated as packets, incorporate elements necessary for reliable communication. The communication system is not stream-based, nor is it one that continually broadcasts between nodes like backhaul networks used in the cellular space. Fields that must be present in a packet are synchronization fields, terminators, routing information, and

checksums among the data itself [15]. The data packet is organized in such a manner to allow low-latency processing of the pieces of data. This low-latency processing is permitted by presenting critical fields within the message to the processor first (see Figure 3.7). In a stream or pipelined design, if the timing is set up properly, the node knows how it needs to route (or discard) the packet before it completely arrives at the node resulting in lower processing latency [15]. While we do not employ such an architecture, an effort is still made to use this methodology and avoid long search trees for information. A byte-aligned view of a packet is provided in Figure 3.7. The physical layer of the packet contains the synchronization and terminator fields and the data checksum. This is due in part to the abstraction between the system controller and transceiver and does not affect the overall organization of the system. These fields are added by the transceiver to whatever data it intends to send.

The `originID` and `recipID` are the node identifiers obtained from the last byte of the unique MAC address on the system controller. This permits a theoretical network size of 255 devices using the formula $B = \lceil \log_2(n) \rceil$ where n is the number of devices, and B is the number of bits required in a base-2 (binary) numeral system. As we specify in Figure 3.7 1 byte for either field, we find that we can support 255 unique devices in this packet form. While that is the numerical upper limit, as discussed in section 2.2, it is important to ensure a routing algorithm with such an upper limit is scalable before the design is implemented. For a configuration with less than 2^8 nodes, these fields are still 1-byte long, but the actual ID is a hashed offset by some random amount within the 1-byte range to support a smaller number of devices. Conceptually, we turn a direct-ID system into a n -way associative system.

Another crucial field within the packet is the type of message sent from a node. Similar to the node IDs, we provide 1 byte of space. This amount of space is excessive but satisfies the 8-byte alignment requirements of the transceiver utilized and greatly simplifies pre/post-processing of the message. More complex implementations would want to minimize the required number of bits allocated to this field to permit better packet compaction. There are a variety of message types that exist within a mesh network, some of which are necessary to permit the establishment of the

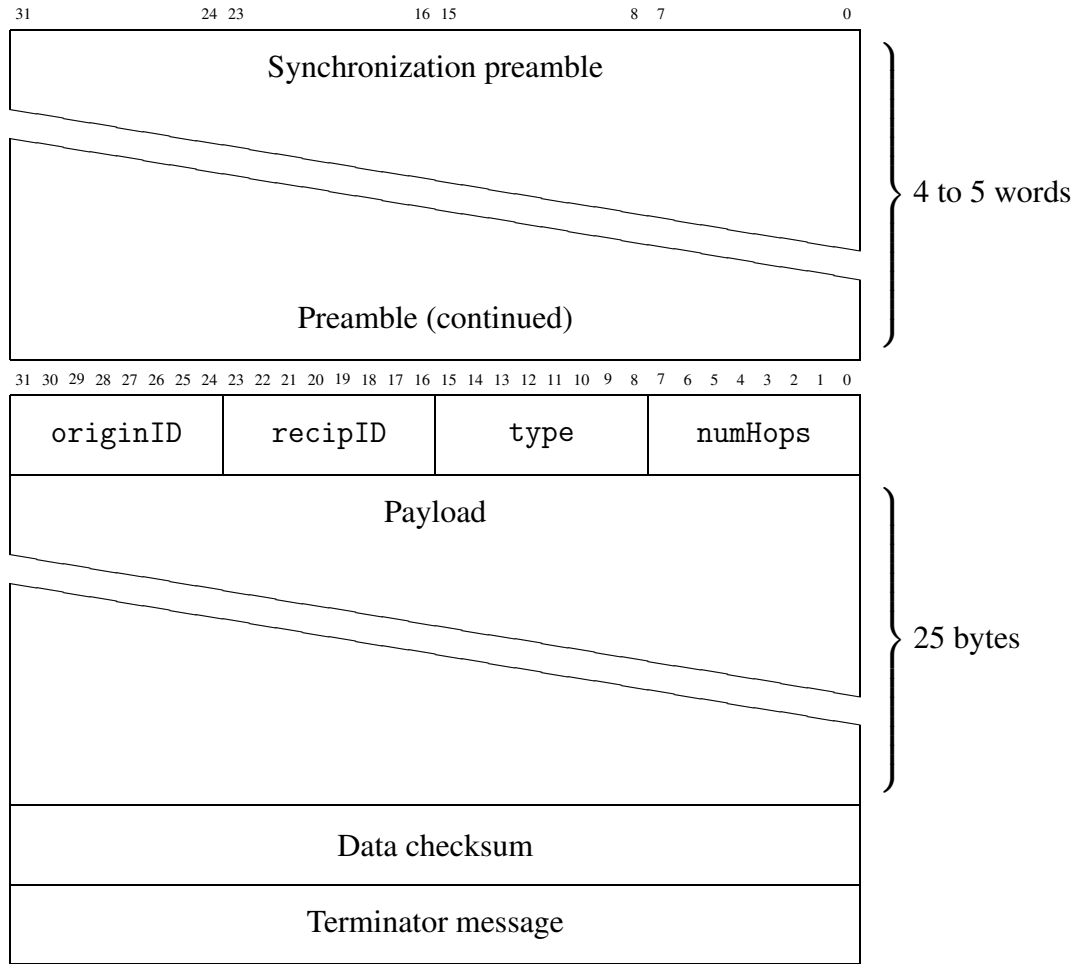


Figure 3.7: Data structure for a packet in the communication

network topology, and others are necessary for basic non-mesh network utilization. The message types encountered in the communication system are provided in Table 3.2.

Table 3.2: Message types of a data packet

Data Type	Value	Purpose
<i>enum</i>	$(1)_{10}$	Normal Message (NORMAL)
<i>enum</i>	$(2)_{10}$	Originator message (OGM)
<i>enum</i>	$(3)_{10}$	OGM Acknowledgement (OGM_ACK)
<i>enum</i>	$(100)_{10}$	Debug (test) packet (DEBUG)
<i>enum</i>	$(101)_{10}$	Ping packet (PING)

All other fields within the data packet are useless without a payload; however, the payload of

the packet depends on the message type. Normal packets that carry network user data will contain human parsable text synthesized as a transmitted and then received message after moving through the network. Network management packets contain nodal connectivity data about the originator node's view of the network topology to ensure the maintenance of the mesh network. Other messages used for network tests contain random data to provide a "whitened" sample for receivers to analyze. The reason for whitening is discussed later in depth. One such reason is if constant values were fed into the transmitter, it is possible that a receiver couples to certain frequency harmonics, which reduces the spectral efficiency of the system and limits the total aggregate power output by the transmitter. Normal user data is generally random, and therefore a random distribution that matches the true data being sent should be used when generating test payloads.

When constructing the mesh topology, the message payload contains a compressed version of the forwarding table found in the origin node as discussed in section 2.5. This compressed table is passed between nodes as 1-byte fields that contain necessary data to form the localized topology view at the node. Each node requires 4 bytes of data as a "line entry" in the forwarding table and as such, we determine that 25 nodes are supported on the network at a given time.

As the packet is eventually sent to a wireless transceiver, the transceiver must also inject some amount of synchronization and termination data into the channel. To properly receive a message which contains digital binary data it is necessary that the receiver corrects for phase and frequency offsets before quantization of the data. Improper alignment with the incoming signal immediately corrupts the data and renders it useless. To prevent this from happening, the transmitter prepends a series of preamble symbols ahead of the message identifying data or payload itself. The length of this preamble is generally 8-16 symbols [31]. A greater number of symbols allows for more opportunity to synchronize and lock at the cost of increased transmission time. LoRa systems generally use 12 symbols to synchronize [23]. A LoRa preamble symbol sweeps across the whole frequency range of the system, as shown in Figure 3.8.

In this implementation, we do not use the transceiver as a streaming data object. Therefore, it is useful for a receiver to know when the received message finishes transmission so that it may notify

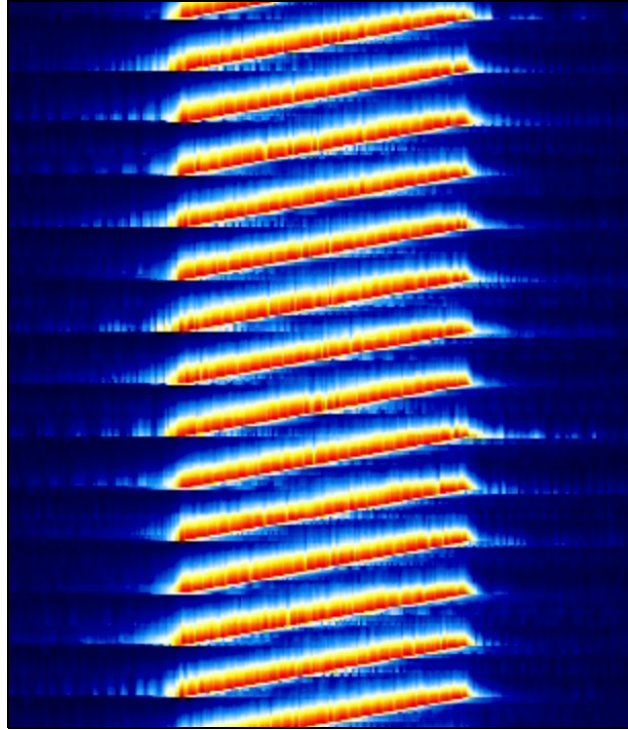


Figure 3.8: LoRa preamble/synchronization symbols

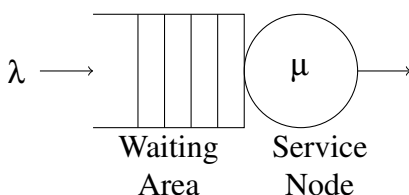
the awaiting location that data is ready to be input into the controller. This is only possible with a known termination sequence that the receiver expects at the end of the message. The transceiver's termination sequence is rather simple being just the absence of activity in the channel at that given spreading factor. When channel activity is no longer detected for that sequence, the transceiver notifies the system controller that data is ready to be pulled from the device.

3.3.2 Queues and semaphores

The system architecture laid out in section 3.4 is only possible through the usage of object-oriented organizational structures and task interlocks such that system resources are accessible in a responsible manner. This so-called responsibility mainly refers to the assurance that data is not lost between tasks, and also that resources on the processor are allocated in a non-blocking fashion. Additionally, data should be processed in the order it was received to ensure that all data through the node is addressed in a first-in-first-out manner to reduce latency for an object. It is

commonplace in basic microcontroller architectures to use some variant of a `delay()` function or to employ constant interval resource polling for system I/O. While both of these techniques work in a limited sense, the usage of timing and polling-based process control is woefully inefficient and impossible to implement in a multi-threaded architecture. When a command to `delay()` is issued, the CPU is stalled. Constant-interval polling, which uses these timers, also often blocks other elements of the program from running resulting in response dead bands. In either case, the processing efficiency of a program is greatly reduced.

To avoid these problems, FreeRTOS provides a set of hardware and software interrupts. These interrupts and their associated interrupt service routines (ISRs) act as a set of scheduled tasks with priority greater than all others on the processor. When the condition for the interrupt is met, it provides minimum latency to distribute the notification that the interrupt has happened. This is useful when a task needs to wait for an input, say, waiting for the transceiver to notify that a packet has been received. Aside from interrupts, tasks can use semaphores and queues, and simply wait to receive data to start processing in a non-blocking fashion. Receipt of a data object from a queue, or taking the semaphore (change of state) is the program interrupt itself. Other more advanced functionality is provided in FreeRTOS when using tasks, like the ability to pass values between tasks when certain calls are made to a task handle, although we do not find the usage of such structures necessary in the system's architecture.



A queue is an RTOS kernel object capable of passing information between tasks without incurring race conditions or overwrites from previous tasks. This structure can be thought of as a simple FIFO data structure that takes data objects λ and stores them in a waiting area. This waiting area contains all stored objects in the order by which they arrived until a service node μ removes the object from the queue. Processes that generate queue entries can feed the same waiting area

simultaneously, although the order by which they are truly organized has no order prioritized by task as a queue is a FIFO structure. Priority queuing could be implemented with multi-level queues and a queue handler using a prioritized scheduler like deficit round robin (DRR) [32]. There can be multiple service nodes, although at that point other control mechanisms must be used to avoid access conflicts. A queue is finite in size, and an efficient implementation in memory requires a careful understanding of heap memory organization. FreeRTOS provides a complete framework for handling queues including queueing, enqueueing, peeking, flushing, etc. which lets us preserve the FIFO structure and yet also gain insight into the behavior of the queue itself. These methods are used throughout the design to maintain the queue's health and avoid overflow of the data structure.

Within the communication system, we identify four queues that act as inbound/outbound buffers between data processing stages. The utilization of the queues is illustrated in Figure 3.9. All of the queues contain FIFO lists of `message_t` packets, which simplifies translation between different queues and ensures that bundled telemetry for a given packet stays with that packet. It is expected that the `rfTXqueue` and `rfRXqueue` will contain more data than the `hostNodeTX` and `hostNodeRX` queues, as the host node queues only handle data inbound/outbound to that node itself. This helps to abstract the amount of data processed by the user interface and avoids corruption of data intended for other nodes by undue processing. Data is delivered to the queues as soon as possible, and it is retrieved from the queues as soon as possible, or after a pseudorandom time interval to avoid undue synchronization of node processes.

Another critical task control method used in the architecture is the usage of semaphores. Semaphores are a class of abstract data types commonly found in RTOS systems to control common resource access and avoid data conflicts, particularly in multithreaded programs. In the scope of the communication system's architecture, we leverage binary semaphores heavily in the RF and User I/O tasks necessary to run the system. The finite state machine for the binary semaphores used in the system is shown in Figure 3.10.

These semaphores are found to control the transmission/receipt of messages, as the transmitter is half-duplex, and also to maintain control over the user interface and operational modes of the

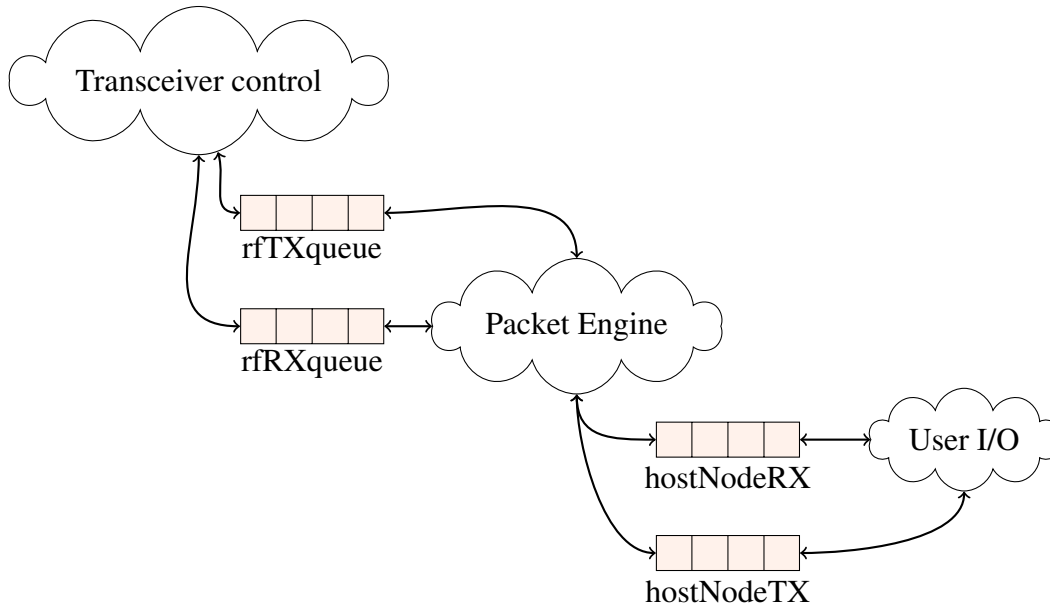


Figure 3.9: Utilization of queues in the system

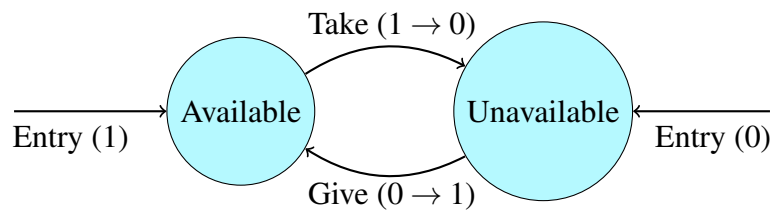


Figure 3.10: Finite state machine of a binary semaphore

system's application. Within the packet engine, we find no semaphores, as all data within that node must be processed immediately and without interlock from other tasks. Any interlocking that would have been required in the packet engine itself is accomplished in the preceding or posting stages of the processing chain.

3.4 Process Architectures

The abstract data types developed or described in section 3.3 were generated to support the overall needs of processes running on the system controller. Notably, Figure 3.1 requires multi-threading on the processor to maintain command and control of the system layers and avoid inaccessibility on some of these layers when trying to run in parallel. The FreeRTOS kernel provides a framework to generate, schedule, prioritize, and manage tasks created on the processor. Using this fact, we determine two major architectural groups in the communication system. The first group is centered around serving the end user of the node and its associated network. The second group acts as the backend processing associated with network establishment and maintenance and maintains command of the transceiver chipset which provides LoRa modulation to data.

The identified groups are further divided into classifications, denoted “User I/O”, “RF”, and “Packet Engine”. A brief summarization of these classifications is provided in Table 3.3. There are additional ancillary tasks that handle the backend system software, although they constitute an insignificant amount of code in comparison to the other groups. Semaphores and queues are used substantially between many of these classifications to permit simultaneous thread execution without worry of data hazards or improper utilization of a function. The target language for the communication system is C on the RTOS. Unlike the traditional `int main()` seen in C programs, on the ESP32, this function is called `app_main()`. The high-level functionality between these functions remains the same. In this threaded environment, it is a poor choice to place lots of logic within this main function. Instead, it is suggested (and truly necessary) to use the main function as a method to spawn tasks identified in architecture design.

In the main function `app_main()`, functions grouped by classification are called to schedule their respective tasks on the processor. Upon startup of a node, these scheduled tasks initialize and make entry into their normal operational modes. A graph of the tasks invoked by the main function is illustrated in Figure 3.11. Some tasks in the architecture are interlocked by semaphores and are in deadlock upon initial startup. As such, we provide those tasks the entry state necessary to begin

normal operation. Threads that are not currently in use are then suspended at their current state to reduce processor overhead and/or undue memory consumption. When required, these tasks are resumed from their suspended state promptly.

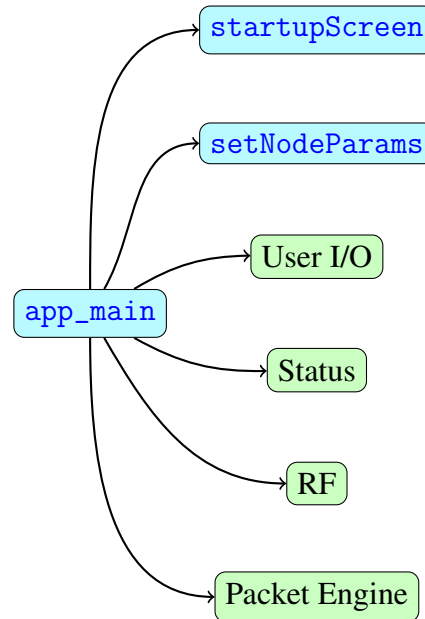


Figure 3.11: Graph of the major tasks spawned

Table 3.3: Spawned tasks from the main function

Spawn name	Purpose
startupScreen	Displays information on the OLED to show node startup.
setNodeParams	Configures the initial node state for network development.
User I/O	classification of tasks to provide interaction with the node.
RF	Classification of tasks to interface system controller with the wireless transceiver.
Packet Engine	Classification of tasks to control the mesh functionality of the network.

The classifications of tasks shown in Table 3.3 quantify the processes necessary to run the system which, to some extent, obey the network model shown in Figure 1.5. Layer 1 & 2, the link and physical layers, are handled by “RF Tasks”. Layer 3 is handled by the Packet Engine, which maintains command and control of the mesh network. Layers 4 and 5 are handled by User I/O, as at that point data is presented to the end-user in a human-parsable form. Within the task classifications, there is also logic to control the node itself. Such logic is not captured within

the network model used to develop the main architecture, but it is prevalent throughout tasks and necessary for nodal function.

The complexity required to develop scalable tasks within the task classifications differs greatly between categories. User I/O has nuance in the methods used to obtain, process, or display user data that interacts with the network. RF requires careful command and control of a peripheral device (the transceiver itself) which itself is physically half-duplex but via a state machine on the system controller provides the illusion of a full-duplex communication link given some latency. The Packet Engine, which exists between the end user and the transceiver, is responsible to establish and maintain the mesh network topology. This task implements the routing algorithm identified in section 2.5 with added functionality to augment the theoretical model for a real-world application.

3.4.1 User I/O

The development of a rich user interface for any system is an incredibly complex topic that requires careful consideration of an almost endless number of factors. In the commercial sector there exists entire teams of people strictly focused on the development of the best user interface possible. The University's own School of Engineering Design and Innovation references The Field Guide to Human-Centered Design as a great reference to use when creating good, interactive designs [33]. For the communication system, the user interface can be rather basic provided it has enough substance to permit easy use. Usage involves not only sending/receiving messages but also controlling parameters about the node and running tests to view the nodal and network performance. Abstraction of these system needs is achieved through the tasks spawned on the processor, as illustrated in the graph in Figure 3.12.

The primary task which maintains control over all other sub-tasks is `dispMenu`. This task presents a numerical menu to the user and provides the full suite of options available to the user to control the node. All other tasks are interlocked to this primary task with binary semaphores

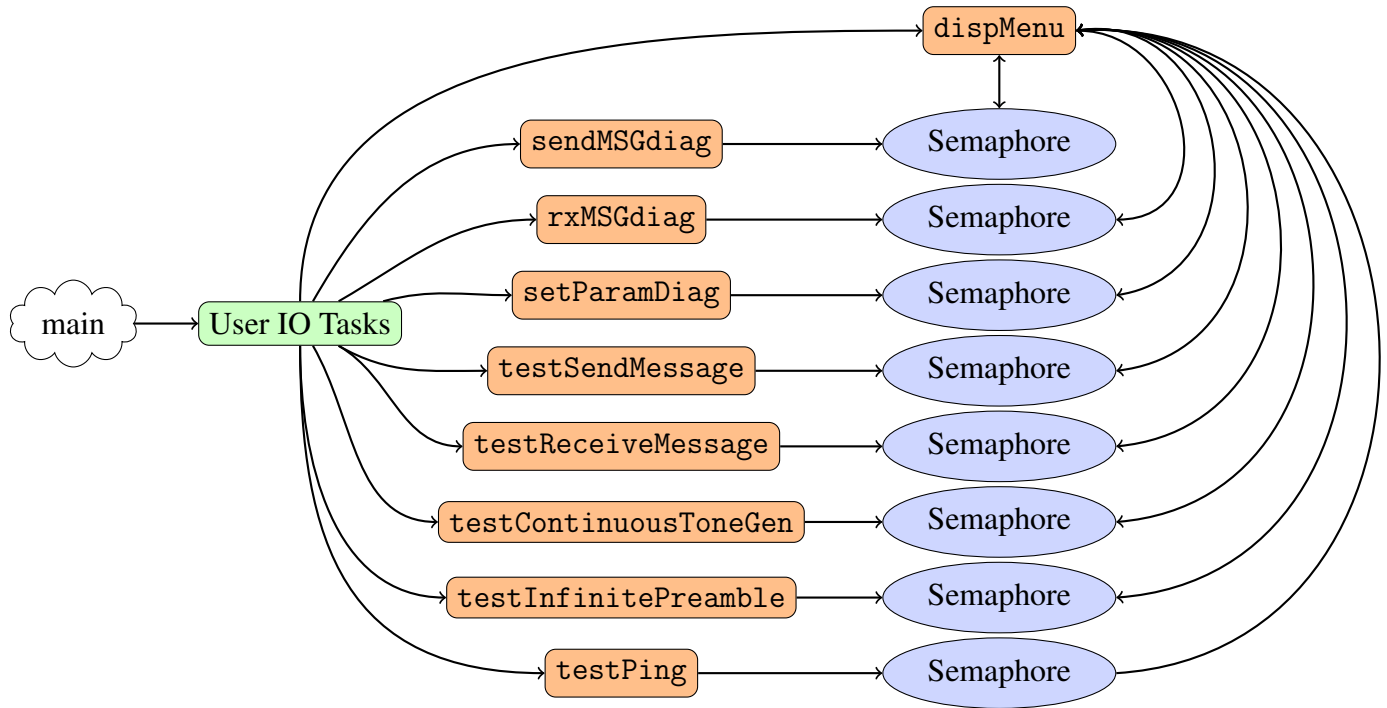


Figure 3.12: Graph of the User I/O tasks and structures spawned

Table 3.4: Spawned tasks from the main function

Spawn name	Purpose
dispMenu	Displays a system menu for users to interact with the node.
sendMSGdiag	Allows users to send messages in the network.
rxMSGdiag	Allows users to receive all messages intended for that node.
setParamDiag	Set basic parameters of the node.
testSendMessage	Send a broadcast test message.
testReceiveMessage	Immediately dump all received messages to console.
testContinuousToneGen	Generate a continuous tone at the configured center frequency f_c .
testInfinitePreamble	Generate infinite LoRa preambles at the center frequency f_c .
testPing	Sends network pings to all nodes in its immediate vicinity.

which use the finite state machine shown in Figure 3.10. In doing so, we can abstract and distribute execution scheduling amongst other sub-tasks, which provides additional memory management and thread prioritization. When a sub-task is called, a semaphore is dispatched to that task. All other threads not used are suspended. The internal logic of the called task then presents dialogue to the user, notifying them of the status or prompting for further interaction.

The primary user interface that end-users of the system will use is the dialogue menus to send

and receive messages. In these tasks, no data is physically sent or received over the air. Instead, data is sent to an outbound queue or received from an inbound queue which is controlled by the Packet Engine. It is not the responsibility of User I/O to ensure the arrival of the packet. The packet engine later discussed at length, is responsible for handling inbound/outbound traffic for the node. As far as the end user is concerned, as soon as a message has been scheduled for transmission, it is en route to the desired destination, given that this node is present in the network. A similar scenario exists to receive a message, only in the opposite order. To send a message, users simply input the destination and their desired payload. When a node receives the message, it places it into an awaiting queue, a sort of “voicemail” which shows in reverse chronological order all messages received and destined for that node.

While manual injection of messages into the network is ultimately the purpose of the communication system, it is also useful to test the system’s ability to send and receive messages. It is commonplace to automate the test procedure of systems as much as possible, and this is a very, very simple abstraction of this concept. An automated message sent as part of the test process should emulate the actual English language, to some degree. A statistical distribution of characters present in the English language is shown in Figure 3.13. Inspection of this figure shows that the character distribution is non-uniform [6]. When sorting the characters in order of frequency, an interesting observation arises that it is a linear, not exponential distribution. Accordingly, we find it necessary to develop pseudorandom test data prepared for a test message that matches this same distribution.

There is a twofold importance to generating non-uniform or single-point sets of test data. A concept commonly discussed in wireless data transmission is whitening. Whitening refers to the transformation of a data packet, which may contain constant values, into pseudorandom white noise. In hardware, this is accomplished via a Linear Feedback Shift Register (LFSR); however, software approaches employ higher-dimension encoding at the cost of processing time [34]. Data that has not been whitened has non-uniform variance across the range, and as a result, will show higher concentrations at certain frequencies. Other phenomena, like oscillator coupling due to

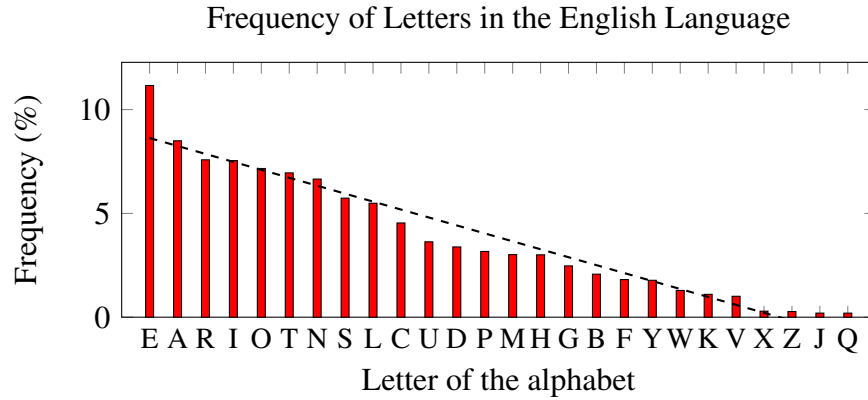


Figure 3.13: Relative statistical distribution of letters in common English [6]

constant bitstreams, also cause detrimental performance in wireless transmission systems [31]. The generation of a pseudorandom, whitened set of data that matches the English language then acts as a very good substitute for normal language for testing network stability.

Aside from automated network tests, the transceiver chip itself has two test modes to characterize the physical performance of the chipset and the required reactive components to properly match the wideband system to the antenna port (see Figure 4). The transceiver can generate a tone at a constant frequency, or an infinite series of LoRa synchronization symbols. These symbols are present at the start of a packet to allow channel activity detection, frequency compensation, and frame synchronization. In initial system development, these modes are useful to check for “signs of life” when communicating between the system controller and transceiver via the hardware abstraction layer. It also enabled validation of hardware driver code. The ability to generate tones at constant frequency permits bandwidth characterization, and the infinite preambles permit other devices to detect channel activity. The ability of the receiver to observe channel activity is used in the RF Tasks classification to see if a node can transmit data without interference.

3.4.2 Packet Engine Tasks

In section 2.5, we discuss the application of DVR as the routing algorithm utilized. This is the physical manifestation of that algorithm, interleaved with added nuance as a figment of im-

plementation. The packet engine is slotted between the User I/O and the physical layer of the communication system. This layer functions as the “networking” segment of the five-layer communications system model (Figure 1.5). While there is not an explicitly large amount of threads spawned in this classification, the tasks that *are* spawned are centrally important for the widespread applicability of the communication system. This classification has the unique task of acting as a part of the distributed, dynamic state machine that makes up a small portion of the network itself. The understanding and abstraction from a global view of the network to a localized form is complicated to not only theorize but also develop.

Each node in the system runs identical software and can be assumed to be completely identical, except for the node ID which must be unique. From a development perspective, we approach exactly how those nodes interact from a globally localized perspective, similar to that shown in Figure 3.14. The definition of global localization means that, while as developers we have a global perspective of how the system functions, there is an extremely limited set of knowledge that the node itself has to function. A node itself, upon startup, knows absolutely nothing about the domain by which it operates. Even when the node does know something about the network, this knowledge is rather limited and prone to inaccuracy by network changes not yet realized.

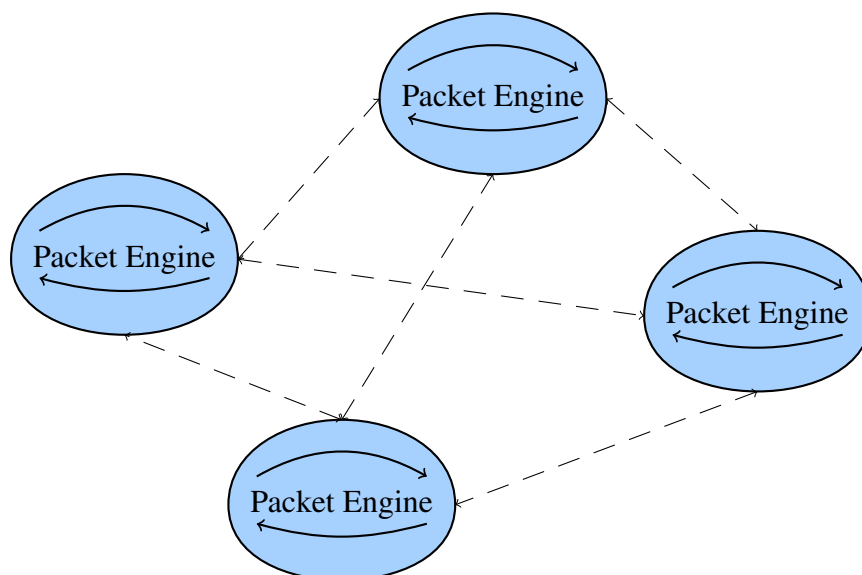


Figure 3.14: Global view of the network packet engines

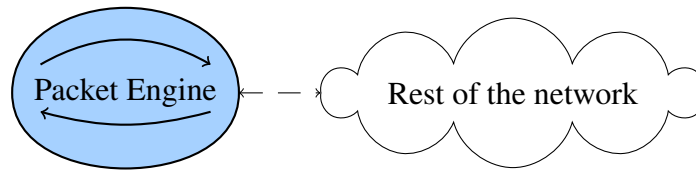


Figure 3.15: Global view of the network packet engines

The packet engine operates in two different modes concerning the network. The first mode is during the startup of the network, where the node attempts to join the network. The second mode, which is more common, is the runtime mode where the node participates as part of the greater mesh to maintain an updated topology and localized view of the network. To initiate the first mode, the node is turned on and allowed to go through the initialization procedure. Recall in section 2.5 we find that for a node to join the network it must inject its intent into the network. In the packet engine, we spawn a task called `remesh()`. This task, whose name represents its runtime behavior, is responsible for the generation and maintenance of the mesh topology via message injection.

When not working to establish or maintain the network, the packet engine acts as a filtration and forwarding service for packets. The queuing structure illustrated in Figure 3.9 shows that the packet engine interfaces with four queues. Two queues are for inbound messages (`rfRXqueue` and `hostNodeRXqueue`), and two are for outbound/through messages (`rfTXqueue` and `hostNodeTXqueue`). These are effective buffers between the control planes of the system. In the packet engine, the type of a message is examined. Messages are then routed through the system using the flowchart as illustrated in Figure 3.16. Outbound messages prepared from the node are packaged in User I/O. The packet engine validates the packet fields and that the destination node is reachable given the forwarding table of the node. When valid, that packet is then sent to the `rfTXqueue` for outbound transmission into the network.

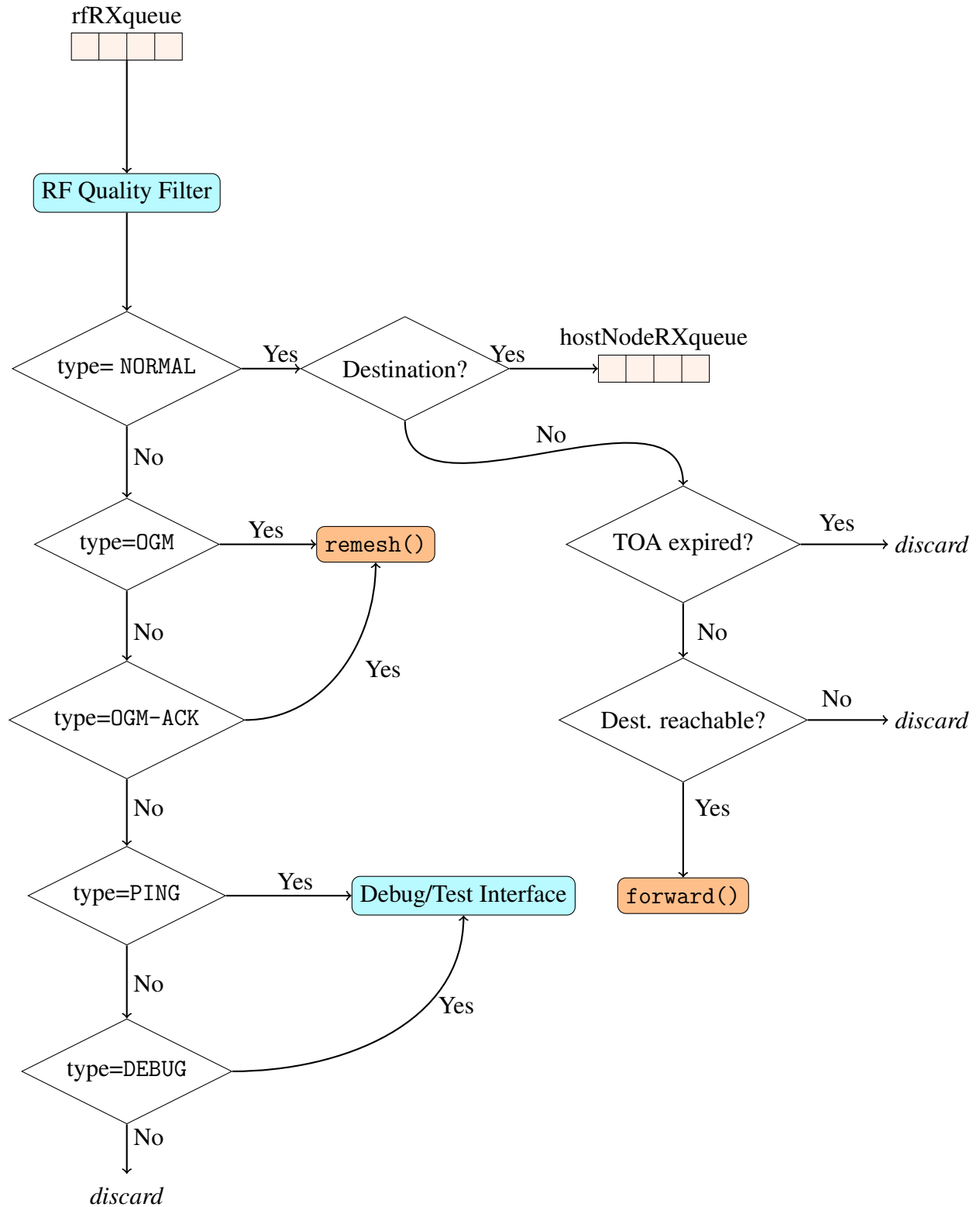


Figure 3.16: Processing system within the packet engine for received messages

Establishment of the topology

To initiate topology establishment, `remesh()` prepares and broadcast transmits an OGM to all reachable nodes in the area. This message is sent after a bounded, random amount of time to avoid network conflict during the simultaneous startup of multiple nodes. The randomness is achieved through a hardware-accelerated random number generator on the system controller, which uses the noise floor of the RF clock sources as its noise source.

The OGM broadcasts to all nodes the packet illustrated in Figure 3.18. This packet broadcasts to all nodes (indicated by `recipID= 255`) reachable given the RSSI and SNR threshold determined in the node. For the base configuration, all packets received with an $\text{RSSI} > -110 \text{ dBm}$ are considered valid links in the network. In the payload of the packet, we provide a pseudorandom sequence of data that matches the distribution of characters used in the English language shown in Figure 3.13. This ensures that data received by potential links appropriately describes the performance of the link for that distribution. This message, once prepared, is sent to the outbound queue and scheduled for transmission on the next state machine transition in the RF tasks.

Nodes that receive this message analyze the RSSI and SNR of the packet. If these levels are above the threshold amount, the node then considers this new node as a viable link and updates its localized forwarding table with the new information. The forwarding table of a node is modeled as a static array of line entries `fwdTable_t`. An entry of the table is indexed by the cost to that specific

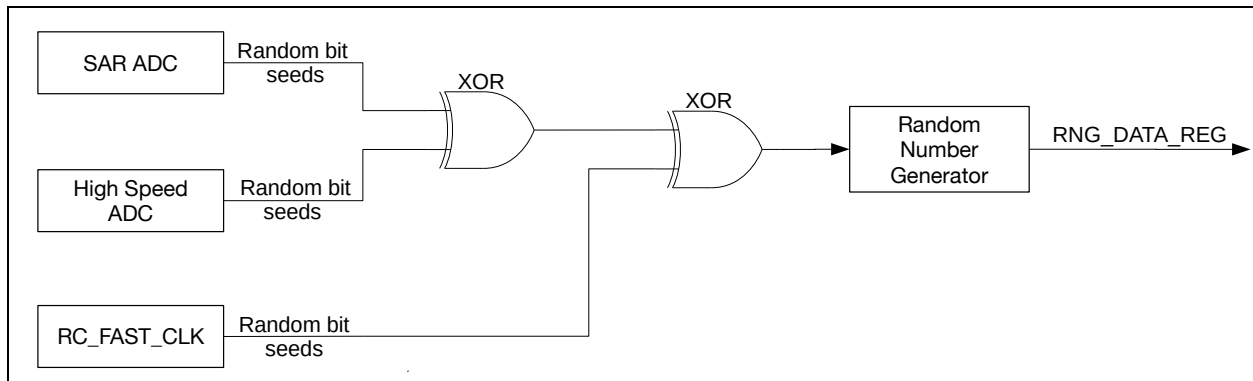


Figure 3.17: Construction of the system controller's random number generator [7]

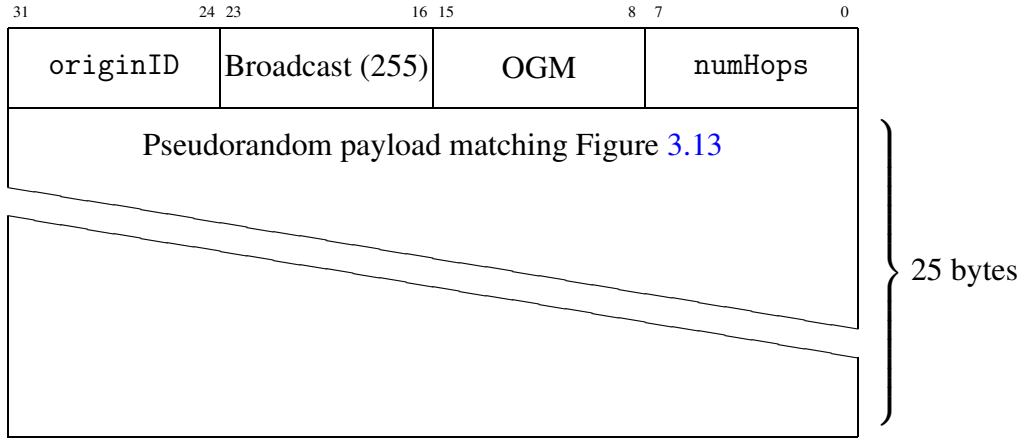


Figure 3.18: Data structure for an OGM

recipID	Cost	Next Hop
0	4	3
1	∞	null
2	20	5
3	∞	null
$\vdots$	$\vdots$	$\vdots$
Q	0	Q
$Q+1$	12	13
$\vdots$	$\vdots$	$\vdots$
$N-1$	∞	null

Figure 3.19: Forwarding table and localized topology seen from node Q

node, and the next hop from that node to reach it. Each table for a node contains the localized topology information for that node which, as shown in section 2.5, is sufficient to construct a mesh topology. An illustration of a forwarding table for an arbitrary node Q is provided in Figure 3.19. Nodal entries with cost listed as ∞ are unreachable. In the physical implementation, we denote infinity as `#define INFINITY 255` as a macro definition pre-compilation. To easily index the data, the array index of the forwarding table acts as the `originID` for a packet. This avoids an additional field in the table and allows easier element access to the data.

When an OGM is received at a node, if any changes to the node's localized forwarding table are made (i.e. changing costs or finding new routes), the node begins the process of preparation and transmission of an OGM acknowledgment message. This message, of type `OGM-ACK`, formally

Index	Line entry	recipID	Cost	Next Hop
0	!	0	4	3
1	!	2	20	5
2	!	Q	0	Q
3	!	$Q+1$	12	13

Figure 3.20: Logical compression of the forwarding table for Figure 3.19

establishes the link between those two nodes, given the originator node wishes to also update its forwarding table. Inspection of Figure 3.19 shows that we require, at the minimum, 3 bytes of information per nodal connection to transmit to the other device its viability as a potential routing path for a packet. It is rather unlikely that the full number of devices will be active in the network under normal operating conditions. Because the forwarding table seen at each node is static in size, there are many entries in the table left unpopulated in the common case. To reduce the amount of data sent by the node, we compress the forwarding table, breaking the bond between the array index of the original table and its compressed counterpart. For Figure 3.19, we compress the data contained in that table to the structure seen in Figure 3.20. Each block (excluding the table header and array index) in the compressed table is a 1-byte `uint8_t` piece of data, logically arranged as a sequential array but shown as line entries for clarity.

	0	1	2	3
Entry 0 {	!	0	4	3
Entry 1 {	!	2	20	5
Entry 2 {	!	Q	0	Q
Entry 3 {	!	$Q+1$	12	13

Figure 3.21: Physically compressed forwarding table for Figure 3.19

In doing this rather basic method of compression, which simply removes unreachable nodes from the data to be transmitted, we drastically reduce the amount of data transmitted between nodes. This reduces the time-on-air of a packet, which can be lengthy given the configurable spreading factor (effective bitrate) of a LoRa packet. Shorter time on air reduces energy expendi-

ture and decreases latency to build the network. The payload shown in Figure 3.20 is then placed into the packet illustrated in Figure 3.22. This packet has a header specific to the OGM-ACK type.

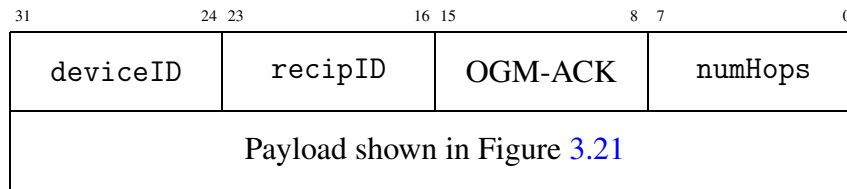


Figure 3.22: Packet structure for an OGM-ACK

With the transmission of this OGM-ACK packet, nodes that receive the data uncompress the field, and determine if a change must be made to their own localized forwarding table. Given that the inherent nature of sending an OGM causes a change in the network, a change to the topology will likely be made. The node goes through the same procedure of sending an OGM to its local nodes, which reply with an OGM-ACK if necessary. The algorithm naturally terminates itself when the topology is synchronized across the network, ensuring that each node is provided with an up-to-date view of the network topology.

Maintenance of the network

As stated in the introduction of section 3.4.2, `remesh()` operates under two main conditions. The first condition, startup, implements the Distance Vector Routing (DVR) algorithm to establish the network. After startup, during normal operation, it is likely that the topology of the network remains rather static. The runtime mode of the packet engine is to maintain an up-to-date topology with the rest of the network. This is achieved via bounded, pseudorandom processes of OGM injection into the network. This then causes all other nodes to respond and start their topology updates, which triggers further transmission from those nodes, and so on.

The communication system does not include any method to acknowledge whether or not a message was received given the current complexity of the system. This means that it is possible for a message to be sent to an invalid node, or one that has been pushed from the network and its

absence not yet realized. Improvements to the communication system would leverage an abstraction of a network “ping” to maintain cognizant awareness of its local single-hop topology. At the very least, this would ensure that the node has not been entirely removed from the network. The system latency to establish and perform updated copies of the network can be determined statistically, using principles developed from a pure ALOHA-like network [15]. A network comprised of N active nodes will have a maximum efficiency of

$$Np(1-p)^{N-1} \quad (3.1)$$

given that it attempts to transmit p percent of the time. This result is approximately $1/2e \approx 18\%$ as $N \rightarrow \infty$. While this result is not great, we can improve the system’s performance with basic, elegant optimization discussed in section 3.4.3.

3.4.3 RF Tasks

The heavy amount of abstraction present in the communication, which greatly simplifies overall development efforts and scalability, at some point must be translated from easily parsed data objects into transmission symbols. Within this plane, we satisfy that need. The RF tasks for the communication system comprise an enormous amount of the actual low-level source code written for the system. Reference to the control planes illustrated in Figure 3.1 shows that the remaining layer to be implemented is the interaction between the system controller and the physical transceiver itself (“Physical Layer” on the graphic). Beyond this point in the software, compatibility between hardware platforms is not possible, as we convert from easily understood abstract data types to byte-level interaction with the physical transceiver itself. As discussed in section 3.2.2, the target hardware for wireless transmission of data generated and/or packaged in the other control planes is the Semtech SX1262 sub-GHz radio transceiver. We then tune, to a great degree, all software written within this task classification to control the transceiver in an efficient and timely

manner.

The transceiver is the only method that sends data to/from nodes, making it a crucially important aspect of the communication system. Optimizations for availability, throughput, and latency must all be considered. Validation of proper interaction with the transceiver must occur at every reasonable opportunity. Compatibility with the system controller and transceiver must also be optimized for simplicity, as should the source code used to control the device itself. To complete this goal, we spawn the tasks shown in Figure 3.23.

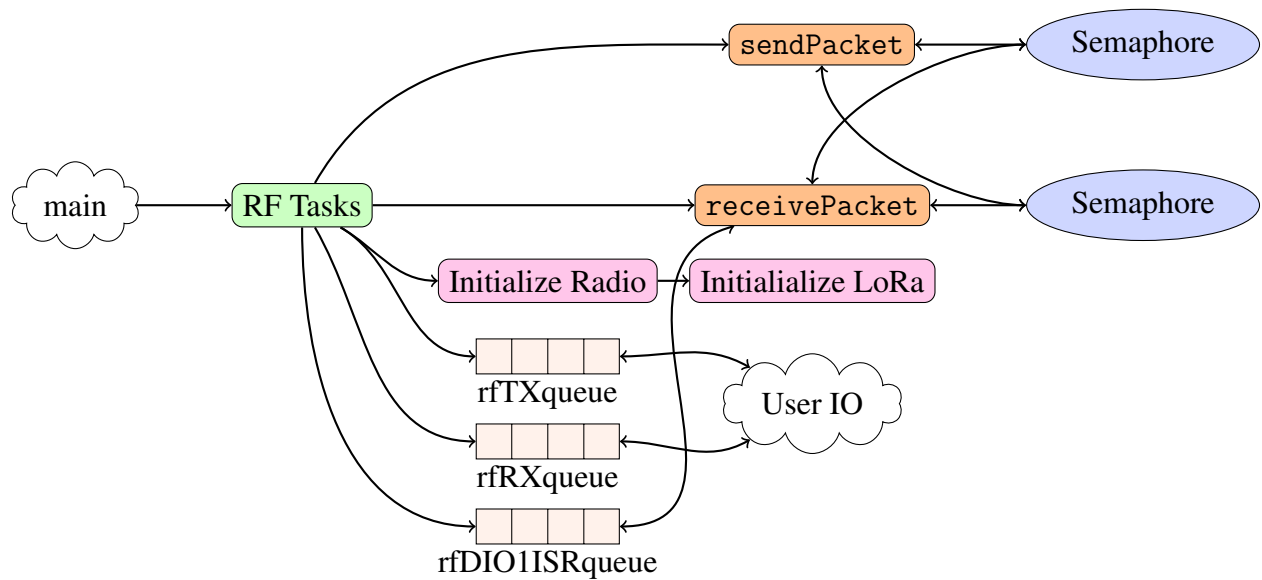


Figure 3.23: Graph of the RF tasks and structures spawned

The tasks and structures shown in Figure 3.23 further contain an extensive low-level hardware driver used to control the transceiver itself. Without this driver it is impossible to send any data from the node and any theory developed cannot be tested. During system startup, we initialize the radio and configure it for its LoRa operational mode. This eliminates some configuration of the device when we need to send or receive data from the node which reduces latency. Then, we make entry into the send/receive packet state machines. These state machines utilize the hardware driver developed for the transceiver to permit the node to send and receive data.

Communication Interface Development

The communication interface between the transceiver and its controlling periphery is via the Serial Peripheral Interface (SPI). SPI is a de-facto communication standard used as the interface between a processing device and peripheral ICs, such as the SX1262. The interface is a synchronous, full duplex master-slave-based interface. The data on the master node is synchronized on subnodes via rising or falling clock edges, dependent on the device. The peripheral and controller can transmit data simultaneously, typically on a 4-wire bus [35]. Timing specifications of an SPI peripheral must be provided by the manufacturer of the device, as they determine the maximum effective speed of the device. Unlike USB (and *especially* Thunderbolt 4), SPI signals are easily decoded by visual inspection. The ability to easily decode the signal accelerates development, as it is possible to validate signal integrity as the system itself is developed.

The SPI interface used on the transceiver requires 4 wires, with additional signals from the transceiver to the system controller being specific to its functionality. The interface wires for the transceiver are listed in Table 3.5. Each of these signals has specific timing and voltage level requirements, provided in the datasheet for the component. In Figure 3.24 we show a typical timing diagram for a 4-wire interface. Table 3.6 provides the required timing ranges to properly interact with the transceiver. Analysis of this table yields the theoretical maximum operational speed of the device. To calculate this maximum we view the interaction between a few different intervals. First, it is necessary to employ a delay, t_1 , when NSS is pulled low to allow the transceiver to start up. Once the device is active, we then transmit command data across the MOSI line and read data on MISO. MOSI must be stabilized by the time the SCK is on its rising edge, as indicated in Figure 3.24. The transceiver captures the value on the MOSI line at this point. It then takes some amount of time, t_2 , between the next bit read from the line. These requirements eventually yield that the effective maximum speed achievable at the interface is approximately 16MHz. To transfer a full payload of 255 bytes, it would take approximately 130 μ s.

All instructions and functionality reachable through the interface are documented extensively in the transceiver's datasheet. While all of the instructions are provided, the actual implementation

Table 3.5: Signal names for the transceiver interface

Wire Name	Purpose
SCK	Interface clock signal
NSS	Active-low “Slave” Select
MOSI	“Master” out → “Slave” in data wire
MISO	“Master” in → “Slave” out data wire
LoRa BUSY	Interface busy signal
LoRa RST	Transceiver reset signal
DIO1	Digital I/O, dynamically configured

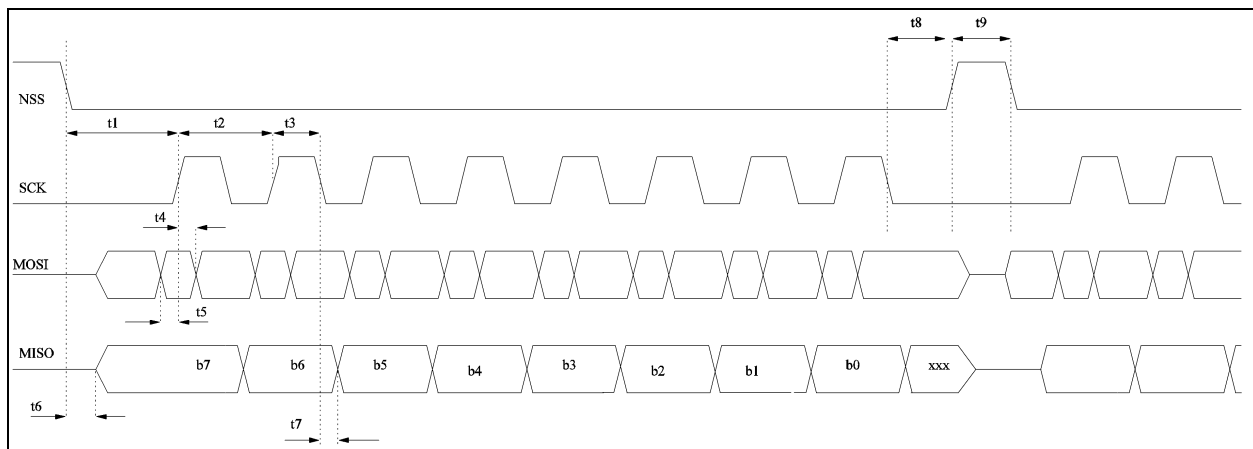


Figure 3.24: SPI timing diagram for the transceiver [4]

of the system controller contains a great degree of nuance. For example, we were unable to run the bus at its maximum speed of 16MHz due to signal interaction and corruption on the printed circuit board when using the Heltec development platform. Attempts to debug the interface by tapping board signals were also attempted, although this was unsuccessful and ultimately sacrificed a few test nodes in the process. Much of the development for the interface was done “blind” and external attempts to verify functionality were made after core communication methods were established. Reaching this point took hundreds of lines of code, and countless hours of frustration simply because the development board lacks signal transparency. Two critical functions, `sx1262read(...)` and `sx1262write(...)`, are used to (unsurprisingly) read and write data to the chipset. These functions are called within other get/set commands developed for the interface. Because SPI is a full-duplex interface it is possible to read and write data from the system simultaneously. To han-

Table 3.6: Timing values for the SPI timing diagram shown in Figure 3.24 [4]

Symbol	Description	Minimum	Typical	Maximum	Unit
t1	NSS falling edge to SCK setup time	32	-	-	ns
t2	SCK period (f_{clk}^{-1})	62.5	-	-	ns
t3	SCK high hold time	31.25	-	-	ns
t4	MOSI to SCK hold time	5	-	-	ns
t5	MOSI to SCK setup time	5	-	-	ns
t6	NSS falling to MISO delay	0	-	15	ns
t7	SCK falling to MISO delay	0	-	15	ns
t8	SCK to NSS rising edge hold time	31.25	-	-	ns
t9	NSS high hold time	125	-	-	ns
t10	NSS falling edge to SCK setup time when switching from SLEEP to STDBY_RC mode	100	-	-	μs
t11	NSS falling to MISO delay when switching from SLEEP to STDBY_RC mode	0	-	150	μs

dle these streams of data the system controller allocates memory space to act as a buffer in startup. Additionally, we utilize a high-speed bus on the controller for the interface bus. While this interface is capable of 40 MHz, we run it at a much lower speed of 2 MHz to avoid signal corruption observed in initial system development. The complete set of instructions was developed on the system controller to operate the transceiver in its LoRa mode.

The transceiver operates in three modes under its LoRa configuration: transmit, receive, and standby/sleep. In transmission, the device is configured to send data placed into the onboard buffer. In receive, the device is configured to detect channel activity and if able, receive symbols and place them into the onboard buffer. In standby/sleep, the device is put into a low power consumption mode, retaining configuration parameters but not essentially ready to transmit or receive a message. The device manufacturer specifies a particular order of operations to set up the device and configure it for any of the aforementioned operational modes. For example, to configure the chipset to transmit, it is necessary to perform the following set of operations [4]:

1. If not in STDBY_RC mode, then go to this mode with the command `SetStandby(...)`
2. Ensure the defined protocol is LoRa via `SetPacketType(...)`
3. Define the RF center frequency f_c with `SetRfFrequency(...)`
4. Define the Power Amplifier (PA) configuration with the command `SetPAConfig(...)`

5. Define the output power and PA ramping time with the command `SetTxParams(...)`
6. Define where the data payload will be stored with the command `SetBufferBaseAddress(...)`
7. Send the payload to the data buffer with the command `WriteBuffer(...)`
8. Define the modulation parameter according to the chosen protocol with the command `SetModulationParams(...)`
9. Define the frame format to be used with the command `SetPacketParams(...)`
10. Configure DIO and IRQ: use the command `SetDioIrqParams(...)` to select TxDone IRQ and map this IRQ to a DIO (DIO1, DIO2 or DIO3)
11. Define Sync Word value: use the command `WriteReg(...)` to write the value of the register via direct register access
12. Set the circuit in transmitter mode to start transmission with the command `SetTx(...)`. Use the parameter to enable Timeout
13. Wait for the IRQ TxDone or Timeout: once the packet has been sent the chip goes automatically to STDBY_RC mode
14. Clear the IRQ TxDone flag

Admittedly, this is not a simple thing to implement. An enormous amount of time was spent to validate this hardware driver, as it is a critical element to ensure that the system functions reliably and without software-induced error. We find in the hardware driver it necessary to set and then get commands sent to the transceiver, to ensure that the command was successfully received. This induces latency in the system as we frequently check that the data sent is valid, although it avoids unexpected behavior not quantified in the statically-programmed state machines running on top of this communication interface. Without this validation, the system would likely, at some point in its operation, enter a deadlock condition and cause network instability.

Send and Receive Packet Logic

While the communication driver permits low-level access and control of the transceiver chipset, it is necessary to design some control logic that uses the commands implemented to properly send and receive packets from the node. In this section, we develop and optimize architecture that permits the transmission or receipt of data from or to the node. There are two major design criteria

to consider when developing this logic. Firstly, the node must make a “best effort” to send packets as soon as they are in the outbound queues from the node. This reduces initial latency into the network and reduces the forwarding latency present at each node in the mesh topology. Secondly, a node must be available to receive data as much as possible. This then suggests that the node should primarily act as a receiver in its main operational mode and a transmitter second. However, it should also react dynamically to channel activity and the node’s need to send data to reduce outbound data latency and increase the effective throughput of the system.

To design this “receiver-first” logic, consider the steps necessary to receive a message. First, it is necessary to configure the node to receive a message. This involves sending or validating that commands have been issued to the transceiver to support receiver operation. Once this is valid, the transceiver can then be placed into a low-power channel activity detection (CAD) mode. The system controller simultaneously the task into a suspended state where it waits for an interrupt-triggered response from the transceiver that data is waiting in the buffer for extraction. When in CAD mode, the transceiver consumes minimal power and looks for LoRa preamble synchronization symbols present at the beginning of a LoRa packet, as described in [section 2.4](#). When the transceiver detects channel activity, it synchronizes system phase-locked loops and LOs to this incoming preamble system, in the hopes that it has locked on and corrected for interference before the encapsulated packet starts to receive. Then, the transceiver processes incoming symbols and places them into the on-chip data buffer. When the entire packet has been read, and no channel activity is detected, the transceiver notifies the system controller via interrupt that it has data ready for input into the controller memory. The task then reads data from the buffer and packages it for further processing in higher levels of the system architecture.

Likely, the system will not detect channel activity for long periods. In this event, we program into the receiver a timeout condition which allows the system controller to send data, if data is available to send. If outbound data has been issued for transmission, the transceiver is appropriately configured and the packet contents are transferred to the transceiver for transmission. The transmission mode verifies that there is no channel activity present (which would contribute to

interference), and then sends the data to its desired location. When complete, the transceiver is handed over to the receiver logic to continue its normal mode of operation.

The somewhat complex relation described is necessary to satisfy the notion of high availability for receiver operation, and also to minimize latency on the transmission side of the control logic. There are additional checks made in the design of the architecture to ensure that the CRC of a received packet is valid, in addition to prioritization made based upon outbound queue thresholds to enable large batches of data to be sent in the network if required at the node. This scenario may occur if the node acts as the only viable connection between two clusters of nodes. While not entirely power efficient, it does satisfy the goal of providing reasonably low-latency mesh communications. There is no such thing as a free lunch. To formally categorize these intertwined processes, we develop the state machines found in Figures 3.25 and 3.26. It is important to note that while the transitional conditions appear to be process-blocking in nature, because we consider each machine as a task interlocked by semaphores and queues (refer to Figure 3.23), the only blocking performed is concerning that task itself. The rest of the system functions as normal with no interruption to its response.

In section 3.4.2, we show in equation 3.1 that the network approaches 18% efficiency. Nodes that wish to transmit do so blindly, and if there is interference, it simply waits a random amount of time before re-sending the message. While this model is simple to implement, it results in low efficiency and reliability concerning data integrity. As discussed, the transceiver is capable of a CAD mode. It is then possible to use this mode to check for channel activity before transmission of the data, commonly referred to as a listen before talk (LBT) mode. This avoids conflicts by random transmission from a large amount of nodes. Observe the channel activity timing diagram illustrated in Figure 3.27. In this illustration, we have two nodes, *A* and *B*, which operate under LBT protocol. At time range $[t, t + 3.5]$, node *A* occupies the channel. All other nodes observe this fact and refrain from transmission. When *A* finishes its transmission, node *B* detects that there is no more channel activity. It takes some amount of time for the transceiver and system controller to realize this result, so we consider a small amount of delay $t + t'$ is incurred. This delay is expected

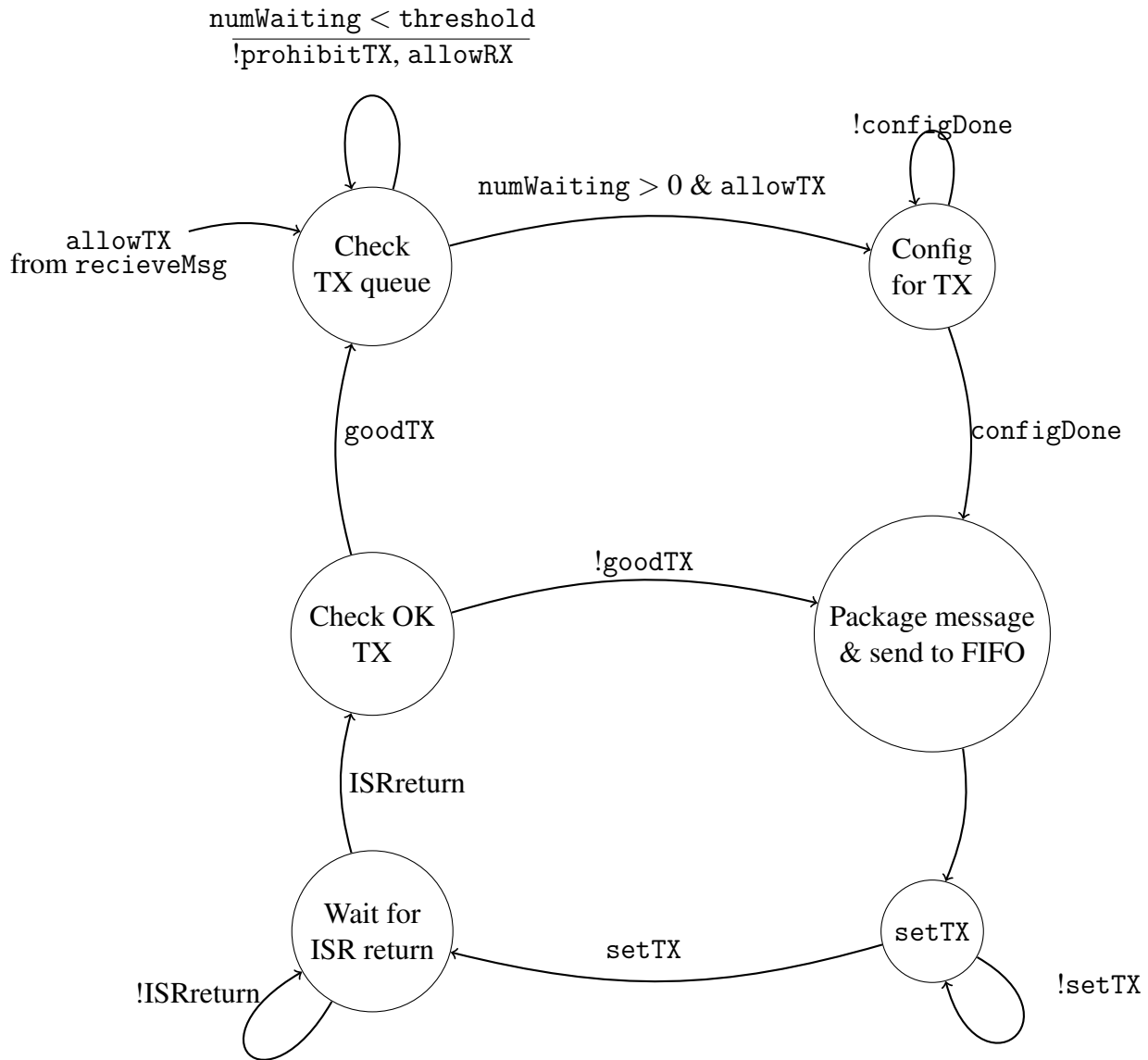


Figure 3.25: State machine to send packets

to be nearly identical between systems, so further delay is required. This amount is pseudorandom and shown in Figure 3.27 as $t + \Delta$. Then, the channel is once more verified to be empty and the data is sent from node B . It is important when operating under this method to randomly delay exactly when a message is sent, to avoid further conflict from a group of nodes trying to send data at once when the channel is open. This rather basic method of collision avoidance is effective enough with the number of nodes present in the network such that data interference is minimized.

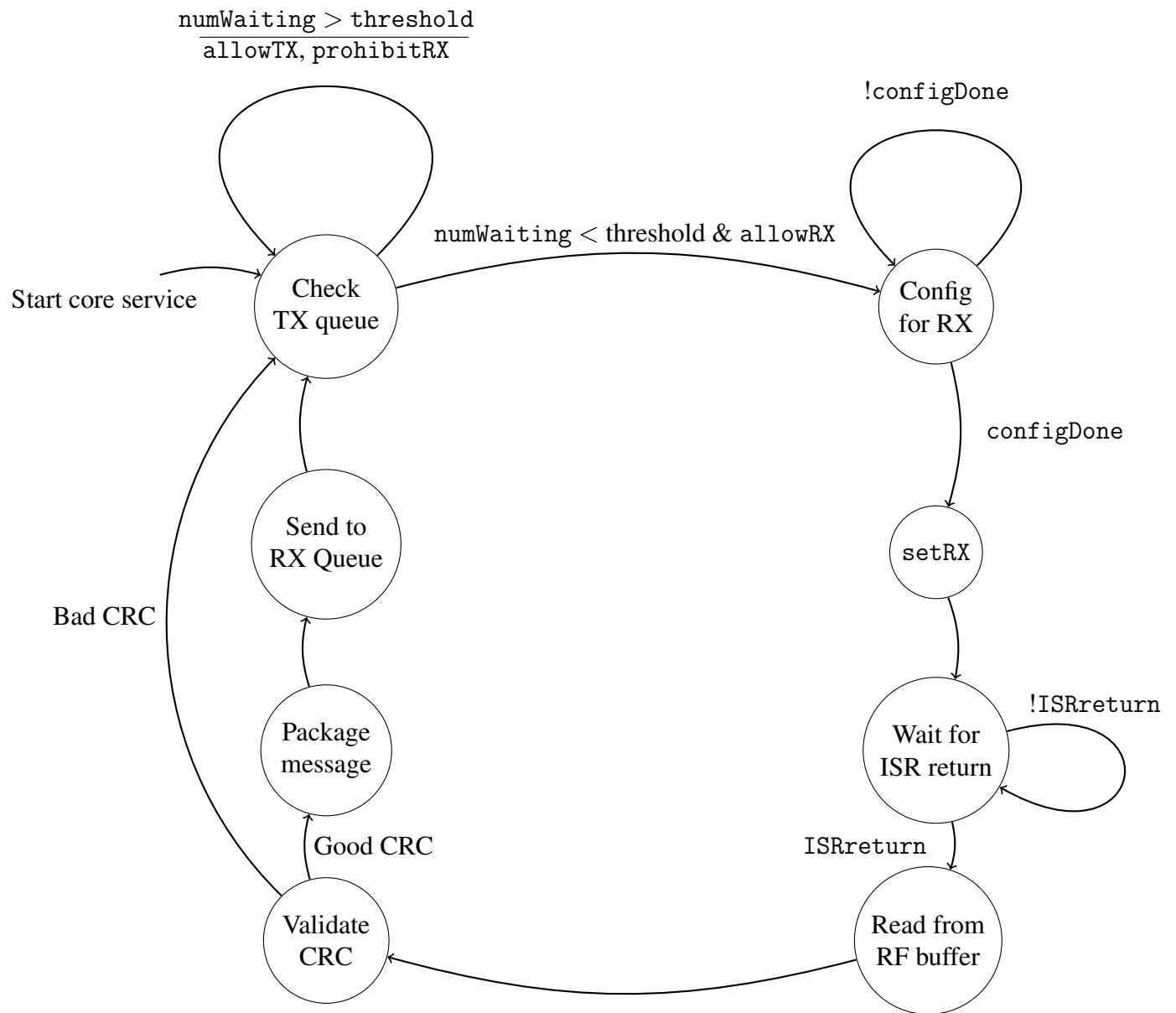


Figure 3.26: State machine to receive packets

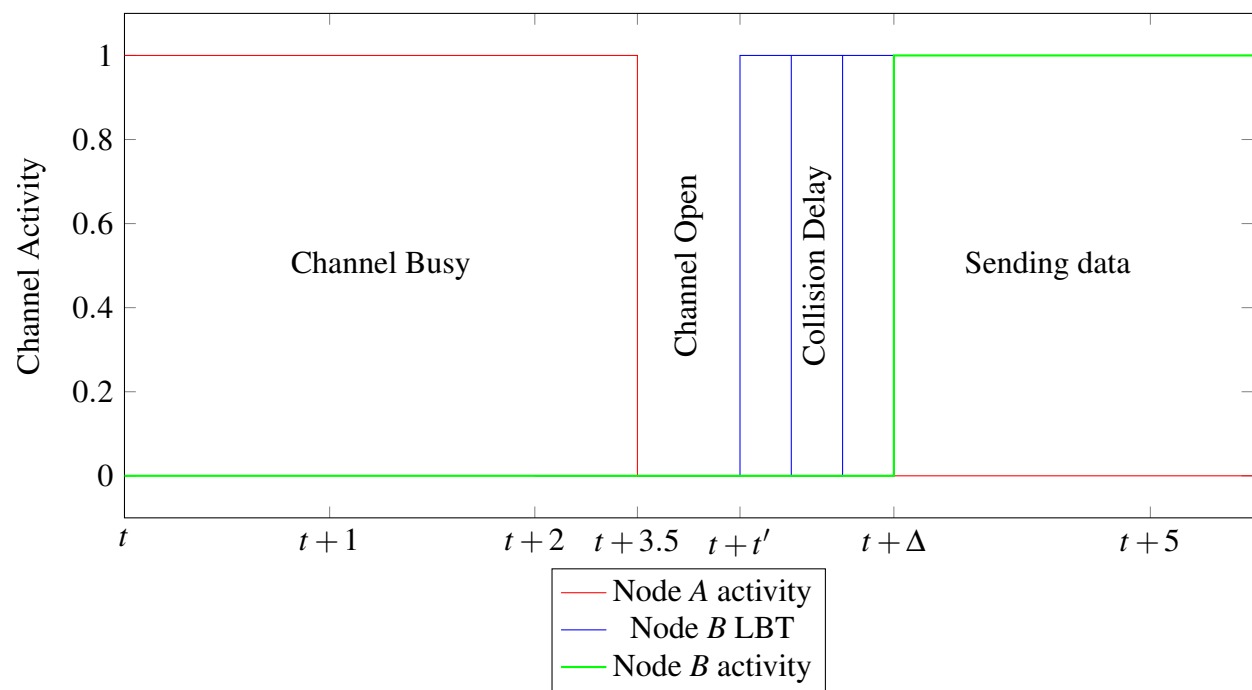


Figure 3.27: Channel activity detection mode of the communication system

Chapter 4

Test and Validation

In this chapter, we perform test and validation of the mesh communication system. The test and validation of a system are crucial to ensure that the theory used for design, design choices before implementation, implementation methods, and more commonly a complex mixture of all of the listed areas combines into a functional, reliable system that meets the design requirements. The nature of the system's behavior is highly application-oriented. As such, we find that many tests can be assessed based on functionality rather than discrete, quantifiable mathematical formulas. Of course, formulae are still leveraged to ensure design criteria meet their goal when possible.

The test and validation methods have been separated into three major categories, similar to how tasks were classified in section 3.11. Accordingly, we will test and/or validate the User I/O, packet engine (routing), and physical layer of the mesh communication system. We use design criteria and results from theory as the metrics to define “functional” and consider that there is likely to be deviation from the design theory proposed and the real-world results.

4.1 User I/O Test And Validation

The User I/O is the application-level abstraction of the system architecture illustrated in Figure 3.1. Table 3.1 provides two major design criteria to satisfy system functionality. The first criterion states that the system should be able to send and receive data to nodes reachable in the topology. The second criterion states that the system shall provide a suite of automatic tests to obtain metrics of link quality and network performance. To satisfy either category, it is necessary to generate an easy-to-use user interface that permits interaction with the node. Upon startup of the device, the user is presented with an array of menu options to begin interaction as shown in Listing 4.1.

```

-----
-- Welcome to the peer2peer comm system! --
-----

System Status:
  Device ID: 112
  System uptime: 1436ms
  Ram remaining 283928
Select the following menu options:
  1 - Send message
  2 - Set device params
  3 - View device status

  4 - View received messages

----- TEST USE ONLY -----
  5 - TEST SEND MODE
  6 - TEST RECEIVER MODE
  7 - TEST CONTINUOUS TONE GENERATION MODE
  8 - TEST INFINITE PREAMBLE GENERATION MODE
  9 - Generate network pings
Menu option:

```

Listing 4.1: Landing page presented to the user

In this menu, it is possible to send/receive messages, change device parameters, view device status, and work with a suite of test programs used in the development and characterization of network performance. We explore the functionality of each design space by entering the desired

numerical choice. Subsections in this section are aligned with this array of menu options.

4.1.1 Send message

In the send message dialogue menu, users are presented with a simple set of instructions to prepare and package a message for transmission. In alignment with Figure 3.1, this dialogue menu flows data into the packet engine for further processing.

```
Entering sendMSG dialogue...
This window allows you to send a message.

Enter a recipient ID [1-254]: 43
Enter your message:FROM: 112
TO: 43
Your message: Hello from node 112!!!

Message sent to queue successfully
```

Listing 4.2: Send message dialogue page

When all prompts are answered, the user presses [enter] which sends the message to the packet engine. The packet engine then validates if the node is reachable and, if there is a viable link, sends the message off for transmission. Messages are treated as separate objects in this architecture, not discrete conversations. While this is rather cumbersome to use, it still permits peer-to-peer communication of data across the mesh topology.

4.1.2 Set device params

It is often useful to be able to set the device parameters of the node in runtime. The physical layer can be reconfigured to meet different bitrate and bandwidth requirements. The default configuration of the system sets the system to the parameters shown in Table 4.1. This set of parameters is rather reasonable for adequate performance of the system and is used in all tests done concerning range.

Users can change the parameters of the device by entering a comma-separated list of the desired

Table 4.1: Default parameter configuration

Parameter Name	Value	Purpose
DEVICE_ID	MAC addr.	Device ID of the node
freqInHz (f_c)	915MHz	Center frequency of the transceiver
pwrInDBM	+22dBm	Transmit power (dBm)
spreadingFactor	<i>SF5</i>	Bits per symbol of the system
bandwidth	LORA_BW_125	Bandwidth of the system

parameter value or string, which evaluates to an enumerated type. For example, to change the device ID to 40 and modify bandwidth to 250KHz, we enter the following data into the console:

```

Entering setParam dialogue...
CURRENT CONFIGURATION:
DEVICE ID=112
OPERATING FREQUENCY=915000000
TRANSMIT POWER=22
SPREADING FACTOR=SF5
BANDWIDTH=LORA_BW_125

Enter the comma-separated list of updated parameters.
Press [enter] to exit the menu without changes.

Configuration: 40, 915000000, 22, SF5, LORA_BW_250

```

Listing 4.3: Modification of device parameters via dialogue

The input seen in Listing 4.3 makes an immediate change to the device's configuration. The network is updated at the next topology maintenance assessment performed by another control plane of the system.

4.1.3 View device status

It is incredibly useful to view the device status to ensure memory leaks do not occur and to verify the processes running on the node. FreeRTOS provides an array of functions to nicely print out the device processes, and it is trivial to generate the forwarding table similar to that seen in Figure 3.19 in an easily read format.

```

===== RUNTIME TASK STATISTICS: =====
System status d 15798 <1%

```

```

main 361380 5%
IDLE 4314532 70%
IDLE 4385910 72%
Remesh engine 413255 6%
Packet engine 156489 2%
ipc1 1223224 20%
Main menu displ 16 <1%
send message di 19 <1%
View recieved m 29 <1%
Set parameter d 18 <1%
Test send messa 20 <1%
Test recieve me 18 <1%
Test continuous 18 <1%
Test inifinte p 19 <1%
PING TX 405 <1%
ipc0 1194525 19%
Send packet log 99 <1%
Recieve packet 8578 <1%
esp_timer 15 <1%

===== TASK LIST STATUS =====
System status d X 2 2644 17
main R 1 6004 4
IDLE R 0 692 6
IDLE R 0 692 5
Packet engine B 10 13312 21
Remesh engine B 6 7680 22
Main menu displ S 9 1280 8
send message di B 4 7104 9
View recieved m B 3 7140 10
Set parameter d B 2 1268 11
Test send messa B 1 3944 12
Test recieve me B 1 4224 13
Test continuous B 4 4216 14
Test inifinte p B 4 4228 15
PING TX S 6 6268 16
ipc0 S 1 2028 1
Send packet log B 13 8940 19
Recieve packet B 15 8732 20
esp_timer S 22 3356 3
ipc1 S 24 2436 2

=====

```

Listing 4.4: System status display output

4.1.4 View received messages

Messages processed through the packet engine that is determined to be for the host node are placed into an inbound queue, `hostNodeRX`, as illustrated in Figure 3.9. The queue is served, in turn, by this received message dialogue in a first-in-first-out manner to avoid missing messages sent to the node. When a user enters the dialogue, they are presented with the console output in Listing 4.5.,

```
This window allows you to view received addressed messages for this node.

Currently, this node has 1 unread messages.
Please enter '1' to view the oldest message: 1

FROM NODE # 43
BODY: Hello from node 112!!!
```

Listing 4.5: Initial output shown to the user when looking at received messages

A message is removed from the queue and effectively marked as “read” when the message is viewed. If a node does not have any messages, the dialogue says there are zero messages unread and exits the dialogue as there are no other options aside from reading messages.

4.1.5 Test send mode

As eluded to in the introduction to the chapter and in chapter 3, an automated test of the system is highly desirable to reduce the manual intervention necessary to check functionality. This menu option is generally only needed in system development and debugging, but it is not harmful to include it in the final iteration of the system’s design. The purpose of the test send mode is to package and schedule a message that is easy to understand, and processed regardless of its sourcing and destination contents to see if the physical layer of the node is functional. The test mode sends five identical messages every 4000ms as a broadcast message to all nodes within range of the transceiver. The message payload is “This is a test message sent from a transmitter..” A spectral plot of the message in the transmission is provided in Figure 4.1.

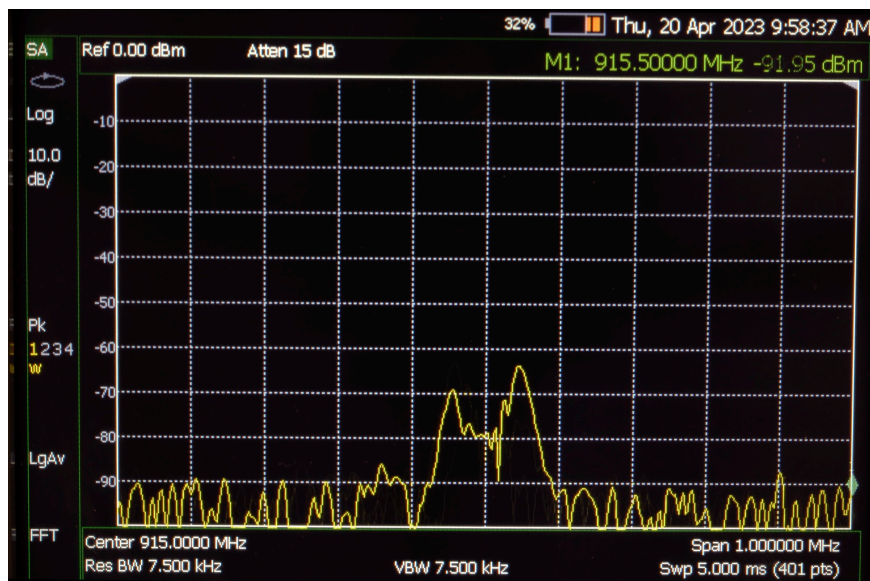


Figure 4.1: Test message in transmission

Nodes that receive this message see it as a broadcast address, but normal type, and route the traffic to the host node inbound message queue.

4.1.6 Test receiver mode

Test receive mode is a complementary operational mode to the test send mode described in section 4.1.5. The purpose of the mode is to permit easy debugging of the physical layer of the communication system. In this mode, as soon as a message is received, its raw contents are dumped to the console as shown in Listing 4.6.

```
Dumping 1 messages in the queue...
ORIGIN: 112, RECIP: 255:
CONTENTS: 11225511This is a test message sent from a transmitter.
Finalized dump. Exiting.
```

Listing 4.6: Output of test receiver mode

It is most useful to run this command after messages have been injected into the network as it will dump all messages waiting at the node's onboard inbound queue. The format presented by the message is not glamorous, although it permits debugging of the byte-level alignment required

between the system controller and LoRa transceiver.

4.1.7 Test continuous tone generation mode

A mode available from the transceiver is the ability to generate a narrowband tone at the configured operational center frequency of the device. The purpose of this mode is likely to characterize the efficiency of downstream circuitry used for matching to the RF switch and antenna port, shown on the schematic in Figure 5. The signal carries no useful data for any modulation scheme. In development, this mode was used to correlate commands sent to the transceiver with the actual behavior of the device concerning operational frequency and approximate output power.

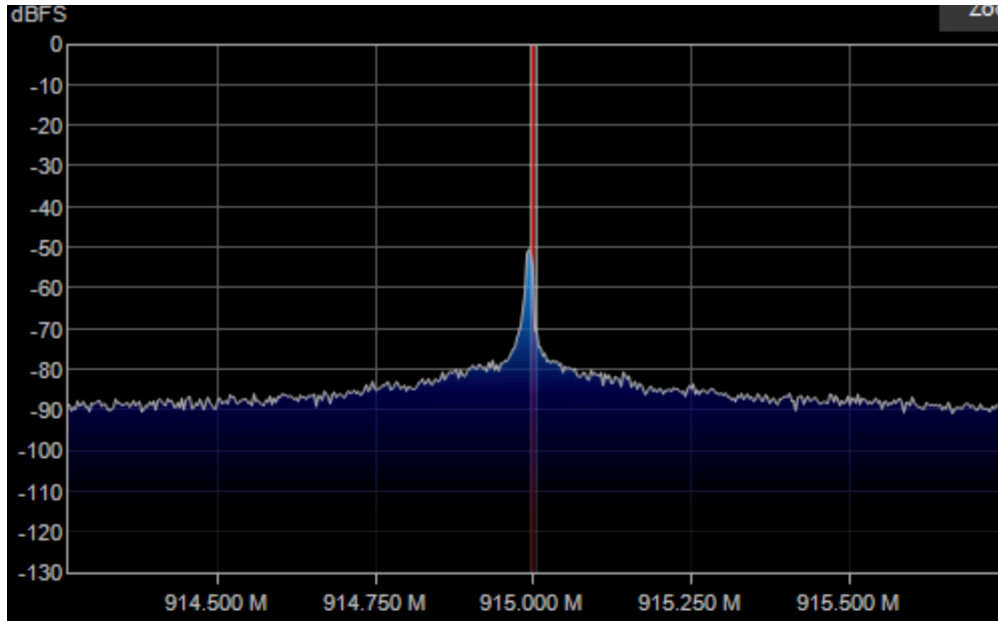


Figure 4.2: Continuous tone generated at $f_c = 915\text{MHz}$

4.1.8 Test infinite preamble generation mode

Another mode available from the transceiver is the ability to generate an infinite sequence of LoRa preamble symbols. Similar to the continuous tone generation mode, the preamble sequence allows the characterization of the transmission hardware. Additionally, because the preamble symbols are detected by a receiver as a legitimate LoRa symbol, the receiver will register the channel

as having activity. While the decoded data placed into the buffer will be nonsense, the preamble generation mode permits validation of both the receiver hardware under transmission and the channel activity LBT function shown in Figure 3.27 operating on the system controller.

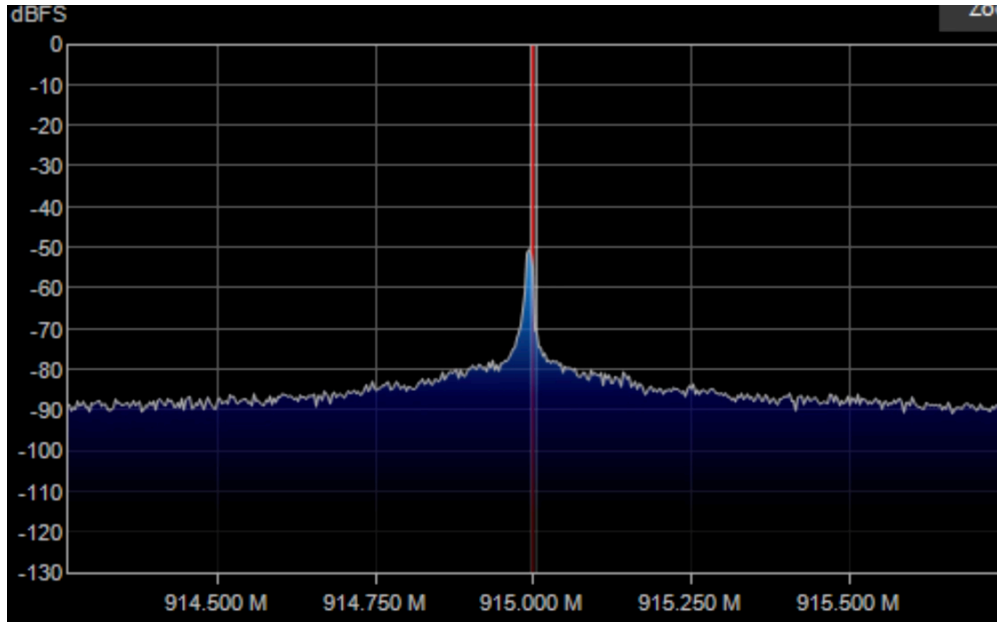


Figure 4.3: Infinite preamble signal at $f_c = 915\text{MHz}$, $SF5$, and $BW = 62\text{KHz}$

4.1.9 Generate network ping

In traditional computer networking, a *ping* in its most basic sense is a network utility used to test the reachability of a host on a network. Within the scope of the mesh communication system, we use the concept of a ping as a unidirectional broadcast message to all nodes in the area with a large amount of pseudorandom data which matches the pseudorandom distribution of letters in the English language. This yields the worst-case scenario for link quality, as a long sequence must be entirely correct. Therefore, RSSI and SNR obtained from this result are more representative of the true viability of the link.

The process to generate a network ping is initiated on a host and observed on other nodes. We enter this mode by selecting it on the main dialogue menu in Listing 4.1. The frequency of message broadcasts is pre-defined to every 5000ms for 50 pings, which provides a sufficient level of link

quality assessment. Nodes that receive the message see it as a network ping and immediately dump to console the results of the ping. A sequence of results is observed in Listing 4.7. The physical formatting of the data seen in Listing 4.7 is:

1. Received sequence (not the ping ID)
2. pingID
3. originID
4. recipID
5. type
6. rssiPkt
7. snrPkt
8. signalRSSIpkt

If the node user does not interact with the node while pings are received, the data is prepared in a comma-separated form ripe for processing in downstream tools like Excel or MATLAB, if desired.

```
0, 32, 112, 255, PING, ab, -67.000, -47.000, -47.000
1, 33, 112, 255, PING, ab, -63.000, -45.000, -47.000
2, 34, 112, 255, PING, ab, -68.000, -45.000, -47.000
3, 36, 112, 255, PING, ab, -96.000, -10.000, -13.000
4, 37, 112, 255, PING, ab, -103.000, -8.000, -10.000
```

Listing 4.7: Recipient node ping statistics dump

This test mode is extremely useful to see which nodes are reachable. We use this mode when characterizing the physical transceiver range to generate data that provides an accurate assessment of link viability at the upper limit of signal strength.

4.2 Packet engine validation

The validation process for the packet engine is achieved via the inspection of forwarding tables at nodes once they have initialized and synchronized their globalized topology to form viable paths between nodes. We observe on a spectrum analyzer in realtime different nodes communicating their data between each other upon startup and see in a debug mode placed into the system software that they are forming the network using the process described in Chapter 3. The forwarding table available at each node is slightly different upon startup but broadly follows Listing 4.8

```
===== NETWORK FORWARDING TABLE =====
Forwarding Table for Node 36:
+-----+-----+-----+
| Destination| Next Hop | Cost |
+-----+-----+-----+
| 1 | 0 | 255 |
| 2 | 0 | 255 |
| 3 | 0 | 255 |
| 4 | 0 | 255 |
| 5 | 0 | 255 |
$$$$$$$$$$$$$ OMITTED FOR CLARITY $$$$$$$$$$$$$$
| 32 | 0 | 255 |
| 33 | 0 | 255 |
| 34 | 0 | 255 |
| 35 | 0 | 255 |
| 36 | 36 | 0 |
| 37 | 0 | 255 |

| 77 | 0 | 255 |
| 78 | 0 | 255 |
| 79 | 0 | 255 |
| 80 | 0 | 255 |
| 81 | 0 | 255 |
| 82 | 0 | 255 |
+-----+-----+-----+
```

Listing 4.8: Forwarding table at node startup for node 36

We add node IDs 112 and 43 into the network. In alignment with the topology shown in Figure 4.4. After a short period, we observe at node 36 its forwarding table has been updated to reflect this new topology. Similar results are observed at nodes 112 and 43. We try a variety of

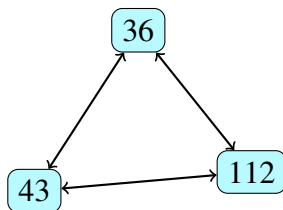


Figure 4.4: Topology of the packet engine validation

topologies aside from a ring and see that the algorithm establishes viable links between nodes and permits the transmission of data along that link. We consider this result successful.

```

===== NETWORK FORWARDING TABLE =====
Forwarding Table for Node 36:
+-----+-----+-----+
| Destination | Next Hop | Cost |
+-----+-----+-----+
| 1 | 0 | 255 |
| 2 | 0 | 255 |
| 3 | 0 | 255 |
$ OMITTED FOR CLARITY $
| 35 | 0 | 255 |
| 36 | 36 | 0 |
| 37 | 0 | 255 |
$ OMITTED FOR CLARITY $
| 42 | 0 | 255 |
| 43 | 43 | 1 |
| 44 | 0 | 255 |
$ OMITTED FOR CLARITY $
| 111 | 0 | 255 |
| 112 | 112 | 1 |
| 113 | 0 | 255 |
+-----+-----+-----+

```

Listing 4.9: Forwarding table after topology stabilization at node 36

4.3 Physical layer validation

Validation of the physical layer shown in Figure 3.1 is divided into two major sections. In the first section, we consider the localized performance of the system with respect to occupied bandwidth and the overall output power from the development board. As stated in Chapter 3, the design is subject to certain regulations as per 47 CFR §15.247. It was determined in section 3.1.2 that it is advisable to operate near the limit for power and bandwidth occupation, as that permits the maximum performance of the node in an adverse environment.

To validate the nodal performance we utilize a Keysight N9914B FieldFox Handheld RF Analyzer and a calibrated cable set to eliminate undue loss or harmonics. The N9914B supports input power levels up to +27dBm, above the stated maximum output power of +22dBm provided by the transceiver chipset employed in the design. We connect the development board directly to the analyzer and configure the node to default parameters seen in Table 4.1. Bandwidth is then swept from [125,300]kHz and the effective radiated power of the system is observed.

Haromics appear heavily in the 300kHz result. This is believed to be a figment of sample rate interaction between the sweep rate of the LoRa symbol and the sampling rate of the spectrum analysis program on the device. The test was re-run at 250kHz with a 15dB attenuator in series with the DUT and spectrum analyzer, presenting the expected result of the system shown in Figure 4.6.

Frequency vs. time domain analysis was also performed on the device. This was to observe the

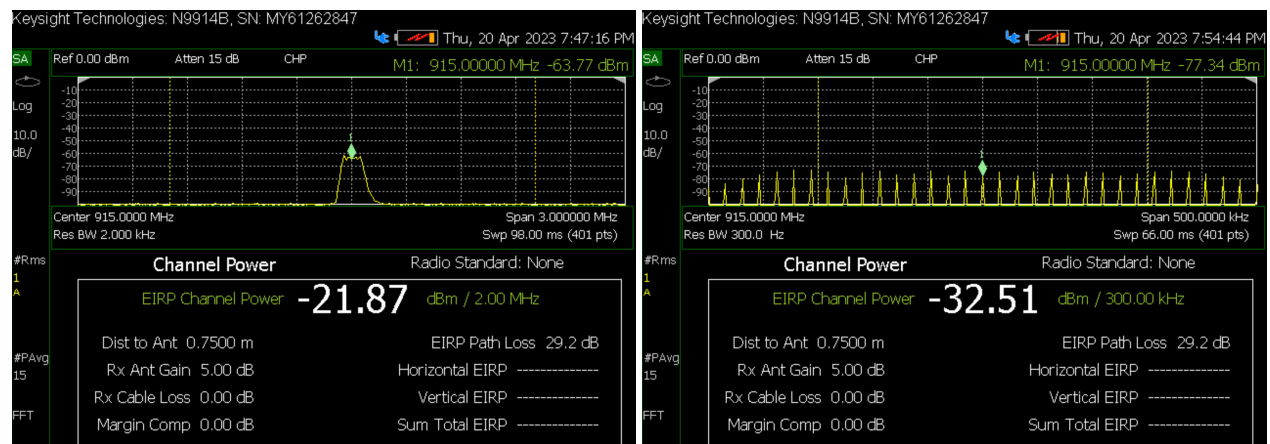


Figure 4.5: Observed occupied (99%) channel power of the device

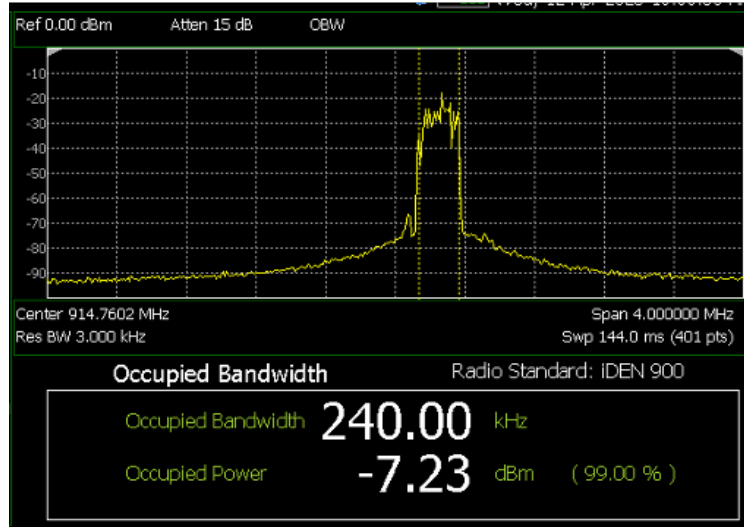


Figure 4.6: Observed occupied (99%) channel power of the device at 250kHz

quality of a chirp sweep across its bandwidth span (specified in equation 2.9) and also to simply see the difference that spreading factors and bandwidth have on the symbol mapping. Measurements were made at a variety of bandwidths, provided in Figures 4.7, 4.8, and 4.9. To obtain the waterfall charts, an RTL-SDR (which uses a Realtek RTL2832U demodulator IC) and SDR++ were used. (Aside from LoRa, the ability to observe a wide variety of frequencies with great ease is fascinating.) At a bandwidth of 300kHz, we again observe strange patterns that can only be surmounted by harmonic interaction in the system circuitry, or sampling rate mismatches given the bandwidth relative to operational frequency. Results improved almost immediately below 300kHz, although data transmission was still successful with high RSSI and SNR metrics obtained for the data packets.

The effective range between the two nodes was also examined. Given that we validate the result of effective network generation, it is only necessary to determine the distance between two nodes to determine the qualitative spread per unit area between nodes. An urban setting is one of the most challenging environments for a wireless device to reliably send data [8]. These environments are densely populated with other wireless devices under competition for the same occupied areas of the spectrum and are often built of large metallic structures that obscure line-of-sight links to some degree [8]. We find the university campus as an ideal “test range” to assess the performance

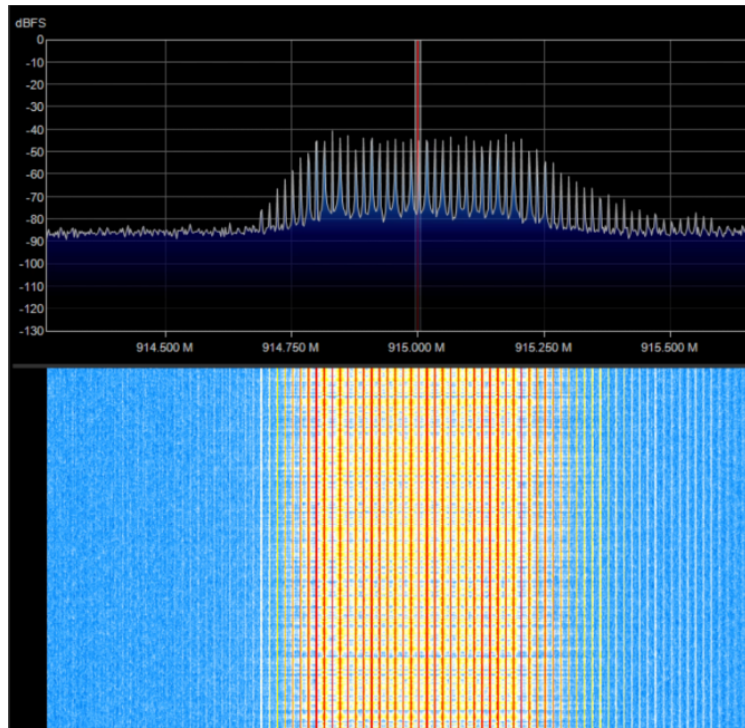


Figure 4.7: Spectral waterfall chart of LoRa symbols at 300kHz

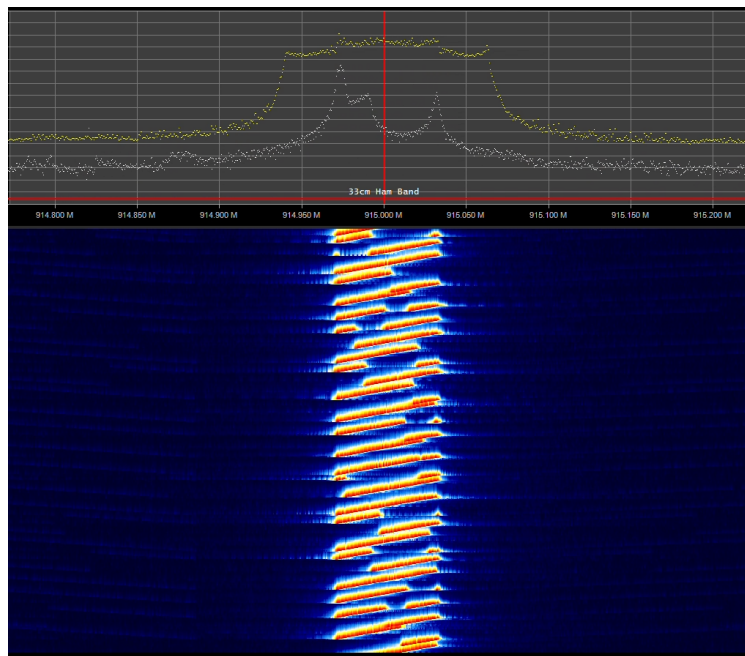


Figure 4.8: Spectral waterfall chart of LoRa symbols at 125kHz

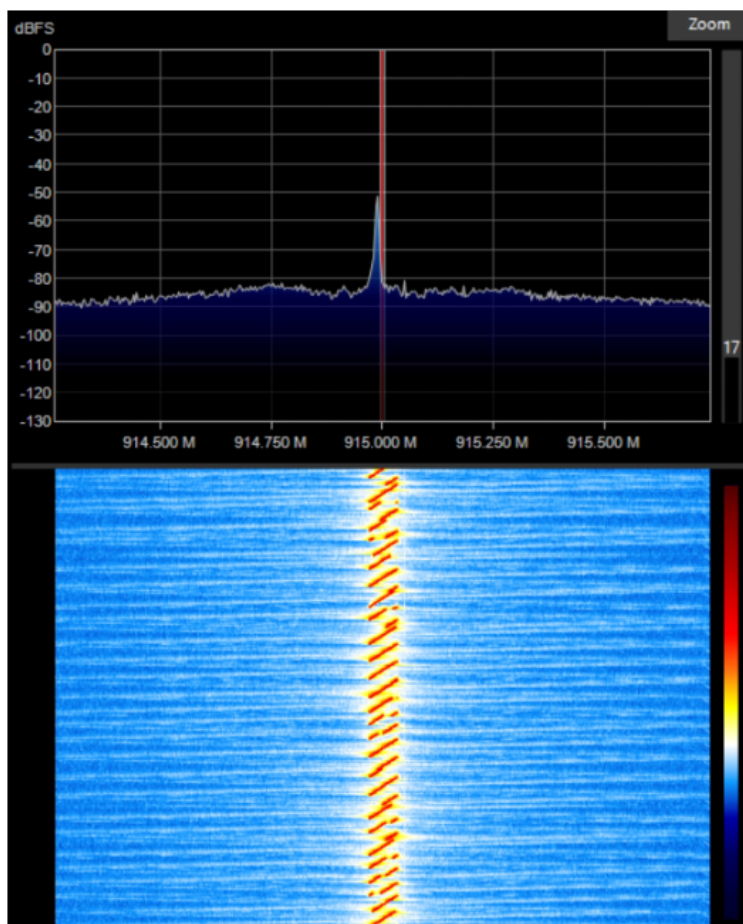


Figure 4.9: Spectral waterfall chart of LoRa symbols at 62kHz



Figure 4.10: Point-to-point range test setup

of node-to-node communications. While not an exact representation of an urban megacity, there is a high concentration of devices that operate in the ISM bands present. During spectral analysis of the system, observations of other commercial activity at approximately 920MHz were seen. It can then only be assumed that many other devices operate below the noise floor at our target design frequency of 915MHz.

With the assistance of some peers, nodes were positioned at locations marked in Figure 4.10. The RSSI and SNR of the device were recorded at close range to be approximately -40 dBm. Decay from this point was exponential, which was expected due to the transmission and path loss models developed in equation 2.2. The link remained very stable up until approximately 1000ft, where signal integrity declined significantly. The results obtained from the test are shown in Figure 4.11. Inspection of Figure 4.11 shows that the system decays rather linearly (in dBm). This result aligns well with expected transmission models and demonstrates that a LoRa symbol is, in fact, rather resilient to interference presented in a dense wireless space. Beyond RSSI values of -118 dBm, the CRC provided in the LoRa packet began to fail considerably. Therefore, packets received beyond this range, while many were still valid, are discarded in the overall system architecture to ensure data integrity.

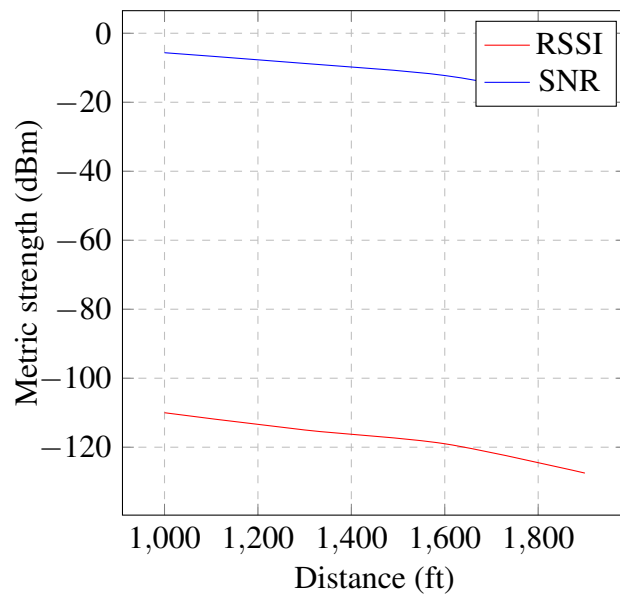


Figure 4.11: RSSI and SNR of a point-to-point link at $SF5$ and 125kHz bandwidth

Chapter 5

Concluding Remarks

The design and development of a mesh communication system is extremely complicated. The amount of effort placed into each layer of the communication system is insignificant if other layers of the system architecture fail to meet their design requirements. There exists lots of nuance in each layer of the architecture, and as such it is important to conduct a thorough study of the implementation techniques to ensure that edge cases of system operation are considered. The software developed for this work spans a massive design space addressing all layers specified in the architecture diagram illustrated in Figure 3.1. Extensive design study into each layer of the architecture was conducted to determine reasonably optimal functional units which, when combined, form a mesh communication system. Before implementation, a great degree of thought goes into the selection and theory behind the techniques and high-level algorithms. These thoughts, which materialized as theoretical models, were then translated into a multi-threaded system architecture. This architecture would not have been possible without multi-threading provided by the processor architecture and intermediate firmware provided by the RTOS. Through the usage of FreeRTOS on the Espressif ESP32 Microprocessor, we can explore the design space in a scalable manner that

permits advanced functionality and encapsulation of the control layers. These layers, cleverly interlocked, communicate the necessary system data to control the application, networking, routing, and physical control of the transceiver employed. Moreover, we demonstrate that the system is capable of running multiple, parallel tasks which satisfies the goal of eliminating system response headbands to the end-user or the greater network by which it is integrated in.

While all of the processing architecture developed for the system controller is interesting and entirely necessary, perhaps the most crucial element of the design is the ability to physically send the carefully packaged pieces of data wirelessly with extremely low amounts of power. As discussed extensively in the work, this would not be possible without LoRa. We show that a LoRa symbol is reliably received up until -118dBm , an unimaginably small amount of power that is not only detectable but correct at the bit level. The techniques used to generate a LoRa symbol are wildly clever, and they contribute significantly to the resilient, reliable production of accurate symbols that encapsulate the data prepared in other layers. With such a reliable link, the distance between nodes in the mesh topology is considerable given the extremely low power consumption required to transmit symbols. More energy is used to prepare the package than send it wirelessly and yet the data is sent in a timely and accurate manner. This result allows the network to span a large amount of area with a small number of nodes. The mesh topology leverages this fact heavily, and we have shown in the work that the system dynamically generates a network topology that is synchronized throughout the network as needed to maintain the optimal route for packets through the network. The algorithm employed, DVR, is perhaps not the most efficient but is easily scaled and optimized to the update intervals and amount of data necessary for transmission across the wireless channel provided by the physical layer. We demonstrate the system's ability to establish the network and maintain viable links between nodes without user intervention, a main goal of the communication system.

In this work, we explore a large design space. The system was designed from the ground up, which spans hardware abstraction and interaction layers, networking, routing, application, and debug modes built-in to each layer to ease system development. As is expected, there are

obvious limitations that have been made to the system to permit fruitful development efforts that meet the overall goals of the mesh communication system. The system lacks thorough design effort in the user application and interaction area, in addition to the network algorithm developed. Interaction with a node is, at best, crude. Individuals familiar with a console-based communication system will find it easy to understand and operate; however, those who are not entirely familiar will find it unidimensional and unlike the graceful user interfaces found in modern mobile and desktop computing platforms. Despite these limitations, the data has been organized in the system architecture to be efficiently repackaged as multi-dimensional message threads without a great deal of effort. Each message contains enough data to form a link between devices, and further development of an application should use this fact to form something users find more friendly to use. The ESP32 microprocessor hosts on-chip WiFi and Bluetooth transceivers, so it is not unreasonable to attempt to design a mobile application for this purpose.

Another design limitation that was undertaken to permit overall system functionality was the complexity behind the development of the mesh topology establishment system. This topic in itself encompasses an entire field of networking and distributed systems, of which there are a multitude of different algorithms and methodologies to achieve the goal of establishing and the maintenance of a mesh topology. The major pitfall behind the DVR algorithm is the amount of data transmitted between nodes when a topology change is detected, and the delay inherent in the algorithm to reach a synchronized topology between nodes. While this limitation was observed in test and validation, the small number of nodes allowed to operate on the network did not dramatically inhibit the usage of the system once a topology was established.

Despite the limitations observed in the complexity of the design, we still prepare a system that accomplishes dynamic mesh establishment, routing through the topology, and usage of a modulation scheme that yields a great degree of overall functionality. The topic of decentralized, low-power, high-resiliency communications is a design space full of nuance, and in this work, we present a multitude of architecture and processing techniques used to overcome limitations inherent in such a design space. The system architecture has been developed in a scalable manner such

that future optimizations to any layer of the system can be made with ease. The software written for the communication system employs a buffered architecture to allow future work to continue without degradation of data integrity. The major goals of the system have been achieved, and we show that the usage of the LoRa modulation scheme as a method to implement a mesh communication system yields interesting, impactful results that must be explored further. We wish to continue work on this concept of mesh communications from a physical perspective in future studies.

Bibliography

- [1] Federal Communications Commission. (2021) 4g LTE coverage as of may 15, 2021 (AT&T mobility, T-Mobile, UScellular, Verizon). Accessed on October 1, 2023. [Online]. Available: <https://fcc.maps.arcgis.com/apps/webappviewer/index.html?id=6c1b2e73d9d749cdb7bc88a0d1bdd25b>
- [2] “United States Frequency Allocation Chart,” National Telecommunications and Information Administration, January 2016. [Online]. Available: <https://www.ntia.gov/page/united-states-frequency-allocation-chart>
- [3] J. Day and H. Zimmermann, “The OSI reference model,” *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.
- [4] Semtech, Inc., *SX1261/2 Datasheet*, December 2021. [Online]. Available: <https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000Un7F/yT.fKdAr9ZAo3cJLc4F2cBdUsMftpT2vsOICP7NmVMo>
- [5] “WiFi LoRa 32 (v3),” Heltec Automation, October 2023. [Online]. Available: <https://heltec.org/project/wifi-lora-32-v3/>
- [6] B. Keating, “The frequency of the letters of the alphabet in english,” University of Notre Dame, 2023. [Online]. Available: <https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html>
- [7] *ESP32 Series Datasheet*, 4th ed., Espressif Systems, 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- [8] T. Rappaport, *Wireless communications: Principles and practice*, 2nd ed., ser. Prentice Hall communications engineering and emerging technologies series. Prentice Hall, 2002, includes bibliographical references and index.
- [9] J.-L. Romero-Gázquez, F.-J. Moreno-Muro, M. Garrich, M.-V. Bueno-Delgado, P. S. Khodashenas, and P. Pavón-Mariño, “A Use Case of Shared 5G Backhaul Segment Planning in an Urban Area,” in *2019 21st International Conference on Transparent Optical Networks (ICTON)*, 2019, pp. 1–4.
- [10] Naval Sea Systems Command, *Distributed Tactical Communication System (DTCS) Radio Only Alpha (ROA) Operator Training*, Department of the Navy, 2008. [Online]. Available: https://info.publicintelligence.net/DTCS-ROA_Operator.pdf

- [11] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, "IEEE 802.11s: The WLAN Mesh Standard," *IEEE Wireless Communications*, vol. 17, no. 1, pp. 104–111, 2010.
- [12] "IEEE Standard for Low-Rate Wireless Networks Corrigendum 1: Correction of Errors Preventing Backward Compatibility," *IEEE Std 802.15.4-2020/Cor 1-2022 (Corrigendum to IEEE Std 802.15.4-2020 as amended by IEEE Std 802.15.4z-2020, IEEE Std 802.15.4w-2020, IEEE Std 802.15.4y-2021, and IEEE Std 802.15.4aa-2022)*, pp. 1–22, 2022.
- [13] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput," *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pp. 1–565, 2009.
- [14] S. Arnon, "Maximum information rate for satellite down link communication with mobile users," in *IEEE VTS 53rd Vehicular Technology Conference, Spring 2001. Proceedings (Cat. No. 01CH37202)*, vol. 4, 2001, pp. 2925–2929 vol.4.
- [15] J. F. Kurose and K. W. Ross, *Computer networking: A top-down approach featuring the Internet*. Boston: Addison-Wesley, 2001.
- [16] Y. Zhang, J. Luo, and H. Hu, *Wireless Mesh Networking: Architectures, Protocols and Standards*. Boca Raton, FL: Auerbach Publications, 2006.
- [17] S. Shahdad, A. Sabahath, and R. Parveez, "Architecture, issues and challenges of wireless mesh network," in *2016 International Conference on Communication and Signal Processing (ICCSP)*, 04 2016, pp. 0557–0560.
- [18] R. Bruno, M. Conti, and E. Gregori, "Mesh networks: commodity multihop ad hoc networks," *IEEE Communications Magazine*, vol. 43, no. 3, pp. 123–131, 2005.
- [19] Fermilab, "Questions about physics," April 2014. [Online]. Available: <https://www.fnal.gov/pub/science/inquiring/questions/atoms.html>
- [20] "What are LoRa and LoRaWAN," Semtech, Inc. [Online]. Available: <https://loro-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/>
- [21] A. Marquet and N. Montavont, "Carrier and Symbol Synchronisation for LoRa Receivers," in *International Conference on Embedded Wireless Systems and Networks*, ser. EWSN '20: Proceedings of the 2020 International Conference on Embedded Wireless Systems and Networks. Lyon, France: Junction Publishing, Feb. 2020, pp. 277–282. [Online]. Available: <https://hal.science/hal-02860476>
- [22] O. Bernard and A. Seller, "Wireless communication system," Utility Patent US9 647 718B2, May 9, 2017, uRL: <https://www.google.com/patents/US9647718B2>.

- [23] *RSSI and SNR for LoRa modulation on STM32WL Series*, 1st ed., ST Microelectronics, December 2021.
- [24] A. E. Ferreira, F. M. Ortiz, L. H. M. K. Costa, B. Foubert, I. Amadou, and N. Mitton, “A study of the LoRa signal propagation in forest, urban, and suburban environments,” *Annals of Telecommunications*, vol. 75, no. 7, pp. 333–351, 2020. [Online]. Available: <https://doi.org/10.1007/s12243-020-00789-w>
- [25] A. Yegin, J. Catalino, I. Giusto, P. Duffy, Y. Gaudin, M. Gilbert, R. Gilson, D. Hunt, and R. Husson, *RP002-1.0.3 LoRaWAN Region Parameters*, rp002-1.0.3 ed., LoRa Alliance, May 2021. [Online]. Available: <https://lora-alliance.org/wp-content/uploads/2021/05/RP-2-1.0.3.pdf>
- [26] National Information Technology Agency, “902-928 MHz,” March 2014. [Online]. Available: https://www.ntia.doc.gov/files/ntia/publications/compendium/0902.00-0928.00_01MAR14.pdf
- [27] R. Baumann, S. Heimlicher, M. Strasser, and A. Weibel, “A Survey on Routing Metrics TIK Report 262,” 01 2007.
- [28] B. Venkataramana and A. Jadhav, “Performance Evaluation of Routing Protocols under Black Hole Attack in Cognitive Radio Mesh Network,” in *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2020, pp. 98–102.
- [29] “47 CFR 15.247,” Electronic Code of Federal Regulations, 2023, title 47 - Telecommunication, Part 15 - Radio Frequency Devices, Section 15.247 - Operation in the band 902-928 MHz. [Online]. Available: <https://www.ecfr.gov/current/title-47/part-15/section-15.247>
- [30] “The FreeRTOS Kernel,” FreeRTOS, 2023. [Online]. Available: <https://www.freertos.org/RTOS.html>
- [31] C. Pham and M. Ehsan, “Dense Deployment of LoRa Networks: Expectations and Limits of Channel Activity Detection and Capture Effect for Radio Channel Access,” *Sensors*, vol. 21, no. 3, 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/21/3/825>
- [32] G. Kesidis, “Performance Evaluation of Queueing Networks,” The Pennsylvania State University, March 2023.
- [33] *The Field Guide to Human-Centered Design*, 1st ed. IDEO.org, 2015, no. 978-0-9914063-1-9.
- [34] A. Marquet, N. Montavont, and G. Papadopoulos, “Towards an SDR implementation of LoRa: Reverse-engineering, demodulation strategies and assessment over Rayleigh channel,” *Computer Communications*, vol. 153, pp. 595–605, 2020. [Online]. Available: <https://hal.science/hal-02485052>
- [35] P. Dhaker, “Introduction to SPI Interface,” Analog Devices, Inc., September 2018. [Online]. Available: <https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html>

Appendices

Hardware Schematics and Diagrams

.1 Espressif ESP32 (System Controller)

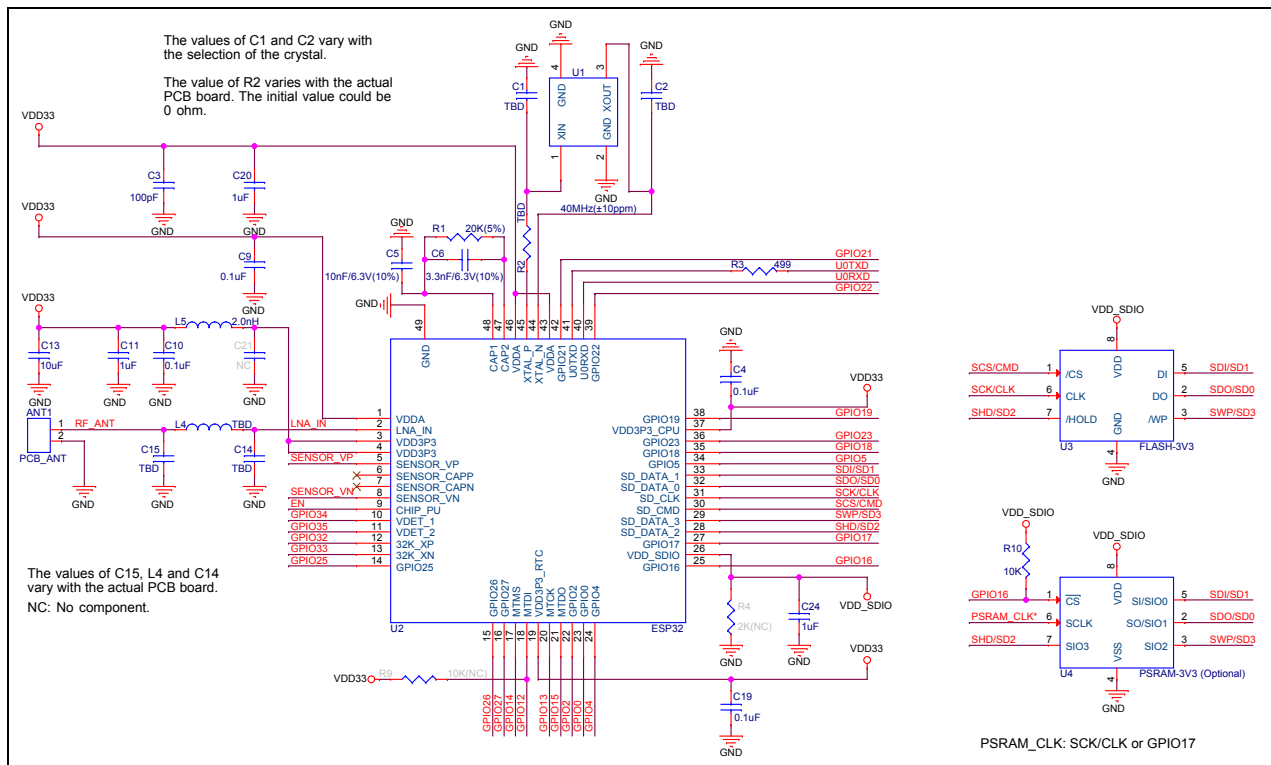


Figure 1: Minimum required componentry to run the ESP32 [7]

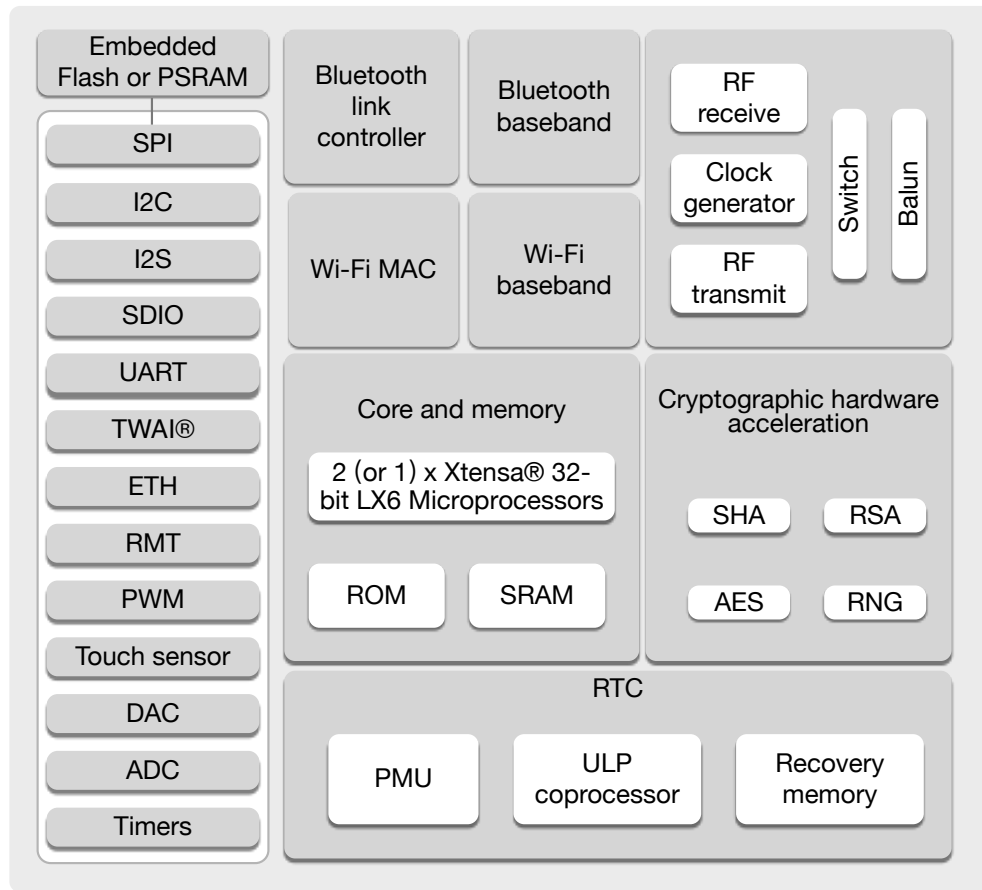


Figure 2: Processor architecture of the ESP32 [7]

.2 Semtech SX1262 (RF Transceiver)

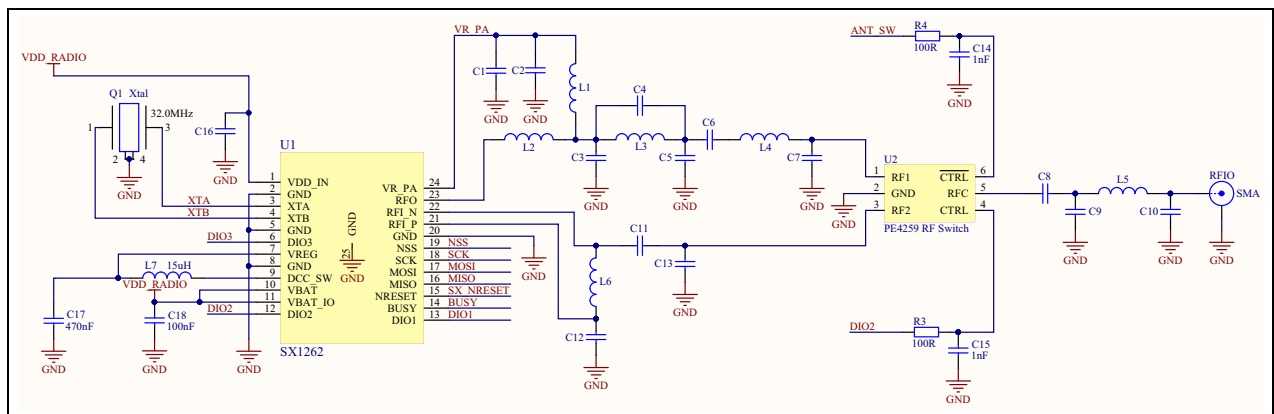


Figure 3: Typical application schematic for the SX1262 transceiver [4]

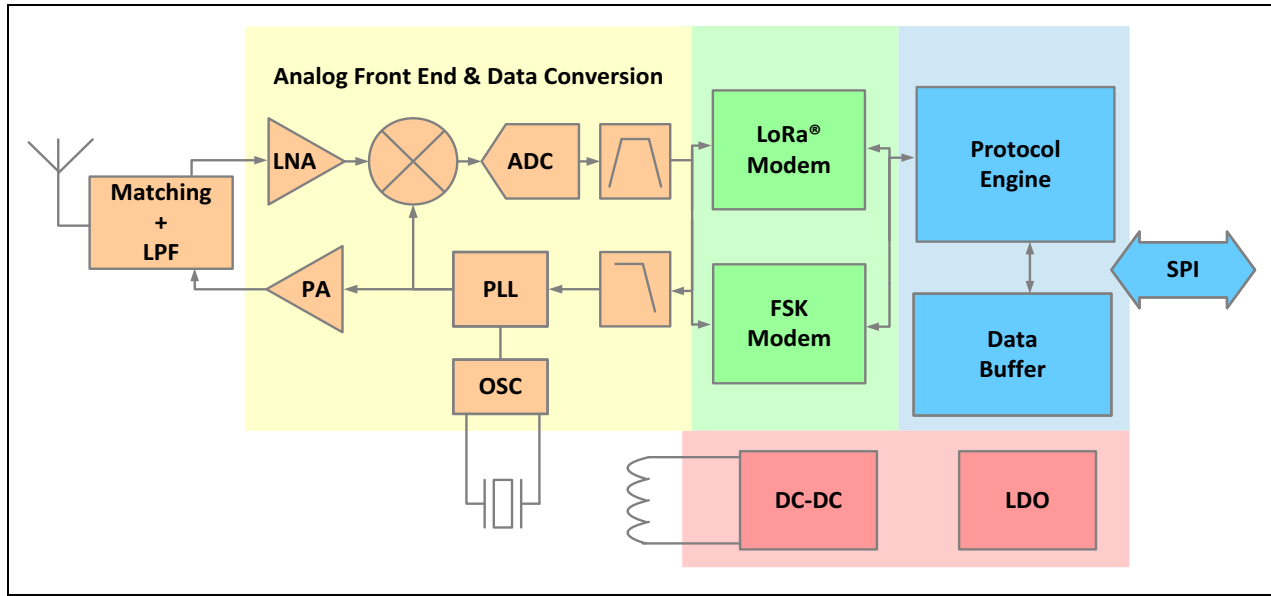


Figure 4: SX1262 block diagram [4]

.3 Heltec WiFi LoRa 32 (v3) (Deployment Board)



Figure 5: Schematic of the deployment board [5]

A Failed Effort

When working with the Heltec development board, it was quickly apparent that writing a hardware driver for the SX1262 transceiver having no prior knowledge in the field was going to be extremely difficult. The board contains no signal tap points for the communication lines between the ESP32 and SX1262, and the available source code for a hardware driver library on the SX1262 abstracts heavily away from the hardware abstraction layer necessary to transmit nicely prepared arrays to/from the transceiver. At this point in development, it was Easter break and I was home for the week. Somehow, I convinced myself that I could design, lay out the components, and get a printed circuit board containing the necessary components finished within the week and in hand in the next.

A valiant effort was made to feverishly design something that closely resembled the provided schematic on the Heltec board, with the addition of lots of test points between nearly every signal line on the microcontroller and its assorted GPIO. The idea of something like this working sounded great. Schematic design and layout were done in KiCad – a program I had also never used before this endeavor but one that I felt was necessary to explore. The schematic and final layout design of the board is provided for posterity, although keep in mind the design failed to implement the UART programming interface and requires the usage of an external programming system.

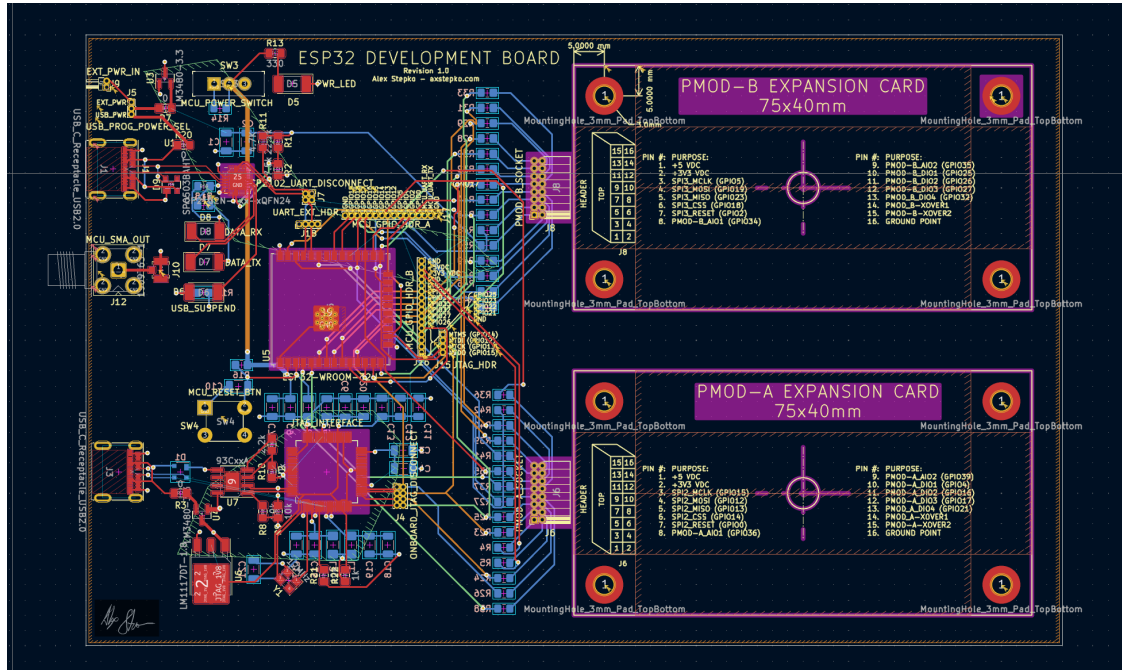


Figure 6: Trace layout of the attempted development board

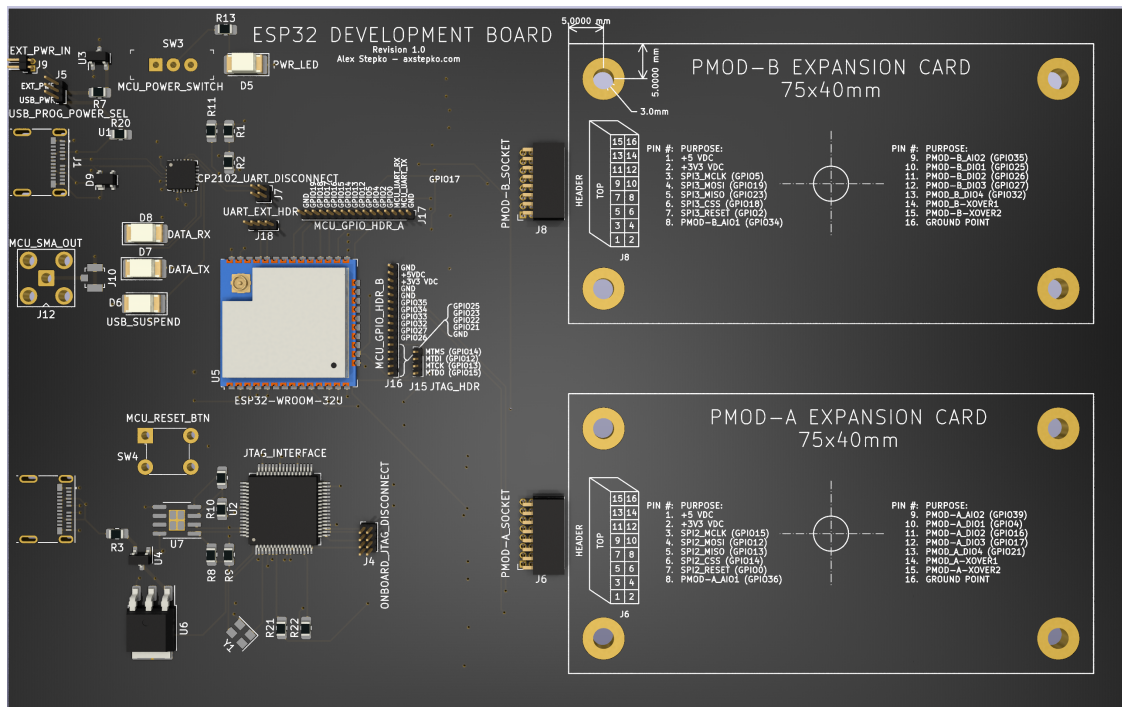


Figure 7: Top-view render of the attempted development board

Complete Codebase

The implementation details discussed in Chapter 3 required an enormous amount of source code. The development of the system was tracked using version control on GitHub, although for archival purposes the codebase as implemented has been moved to my file servers. For the foreseeable future, the source code for the system is available as a directory eligible for view only. As this is a locally hosted service, there is a chance that it will not always be available. Please contact me at alex@axstepko.com if the link provided above does not work. I will do my best to provide the file for reasonable requests.

Accessing the codebase

The codebase is available at:

https://klexnas.quickconnect.to/d/s/vBzsLUzEWxpjuswbc5NwHwPvp2Xx0b1Z/c5hTB-iih-a0zhD7THJM_-fR7PU74BH-fbjAlsdP3Qo

Use the password `meshCommSystem2023` to access the folder.

Use and Redistribution:

While I am happy to share the source code I used to develop this thesis, I also firmly believe that the system I have developed is of my intellect. I did not write this codebase for people to steal my software for their own personal or commercial gain, I wrote this software to demonstrate my capability as an engineer. As such, I have provided my statement on the software's availability for use in a personal, academic, or commercial setting. This statement is also present at the top of every source file I wrote within that codebase. Act accordingly.

Copyright (c) Alex Stepko (axstepko.com) 2023. All rights reserved.

This code was written by Alex Stepko (alex@axstepko.com, the “Author”). Redistribution and use in source and binary forms, with or without modification, are STRICTLY PROHIBITED without prior written permission from the Author.

(a) This software is provided “as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall be liable for any direct, indirect, incidental, special, exemplary, or consequential damages including, but not limited to, procurement of substitute goods or services.

WARNING:

(b) ACADEMIC INTEGRITY: STUDENTS, RESEARCHERS, OR INSTITUTIONS MAY USE THIS SOFTWARE FOR EDUCATIONAL OR RESEARCH PURPOSES ONLY. YOU ARE RESPONSIBLE FOR ENSURING THAT THE USAGE OF THIS SOFTWARE COMPLIES WITH THE COPYRIGHT POLICY STATED ABOVE IN SUBSECTION (a), IN ADDITION TO YOUR INSTITUTION’S POLICIES ON ACADEMIC INTEGRITY AND PLAGARISM. **IT IS PROHIBITED TO REPRESENT THIS WORK AS YOUR OWN, INCLUDING BUT NOT LIMITED TO THE DUPLICATION, MODIFICATION, OR DISTRIBUTION OF THIS SOFTWARE WITHOUT PRIOR WRITTEN PERMISSION FROM THE AUTHOR.** THE AUTHOR DISCLAIMS ALL LIABILITY FOR ANY VIOLATIONS OF ACADEMIC INTEGRITY OR OTHER VIOLATIONS OF AN INSTITUTION’S OR INDIVIDUAL’S ETHICAL STANDARDS COMMITTED BY USERS OR VIEWERS OF THE SOFTWARE.

Academic Vita

The academic vita of the author is provided on the following page. Information contained within the document is considered current as of the time of publication of this work. The author encourages interested readers to visit the author's website axstepko.com to view publications on related topics in Electrical Engineering.

Alex Stepko

alex@axstepko.com • axstepko.com • +1 (412) 719-9699
DoD Secret Clearance (inactive)

Education

The Pennsylvania State University, Schreyer Honors College **University Park, PA**
Bachelor of Science in Electrical Engineering (Senior) August 2019 - December 2023 (Expected)
Minors in Computer Engineering, Engineering Leadership Development
Dean's List for all semesters enrolled

Intend to pursue doctorate in RF Processing Systems *Awaiting application closure, intended start August 2024*

Applicable Coursework

Digital Design: Computer Engineering I, II, III; FPGA Development; Image Processing; Continuous & Discrete Time Linear Systems

Electromagnetics and RF: Electromagnetics I; Circuits & Devices I, II; Nano Electronics I; RF & Microwave Engineering (mmWave)

Relevant Skills

Programming Languages: C, C++, $\LaTeX$, Linux kernel, MATLAB, Rockwell/Siemens PLCs, VHDL, Verilog

Software Packages: Adobe Creativity Suite; Autodesk AutoCAD, Inventor; Cadence Virtuoso; GNURadio; Keysight ADS; KiCAD; NI LabVIEW, LabVIEW CVI, Multisim; Xilinx Vivado, Vitis, Vitis HLS

Other various skills: Circuit validation & test, schematic design, PCB layout; S-parameter measurement; device characterization; additive manufacturing; project management; group leadership; leadership coaching

Engineering Experience

Spread Spectrum Mesh Communication System *Penn State* *November 2022 - October 2023*

- Developing mesh communication system for long-range data transmission using LoRa modulation
- Deployed on an Espressif ESP32 microcontroller, FreeRTOS, and a Semtech SX1262 LoRa transceiver
- Performed device characterization for transmission performance and network stability

RF Test Systems Engineering Intern *Lockheed Martin RMS, Owego NY* *May 2023 - August 2023*

- Provided automated test solutions for RF hardware systems
- Generated preliminary designs, schematics, and prototype of test system module
- Performed hardware checkout and test system delivery to customer
- Supported engineering, production, and qualification of RF hardware systems throughout product lifecycle

Discrete FIR & CORDIC Conversion System *Penn State* *March 2022 - May 2022*

- Led the design, optimization, and physical implementation of a hardware-accelerated signal processing system, integrating a 1-D FIR filter and a CORDIC processor
- Engineered a low-latency accelerator targeted for integration in an AXI compliant design
- Emphasized hardware optimization and compatibility with AXI standards for streamlined integration

Direct Sequence Spread Spectrum Transceiver *Penn State* *August 2022 - December 2022*

- Developed a software-defined digitally encrypted direct sequence spread spectrum (DSSS) transceiver
- Validated spread-spectrum behavior using spectrum analyzers

Automation Engineering Intern *Premier Automation* *May 2022 - August 2022*

- Led development of training documentation for SCADA software products
- Assisted development of project quotations and budgets
- Programmed process control systems and Human-Machine-Interfaces (HMI)

Electrical Engineering Co-Op *Kimberly Clark, Chester Mill* *August 2021 - December 2021*

- Implemented an automatic whitewater consistency instrumentation & control system
- Performed checkout & validation of variable-speed motor control systems
- Developed robust manufacturing network device descriptions for approximately 660 devices
- Designed and implemented human machine interface upgrades for production machines

Other Related Interests and Experience

- Teaching Assistant, ENGR 408 | [Penn State School of Engineering Design and Innovation](#)
- Bicycle Mechanic: Pro Bike + Run North Park | *Pittsburgh, PA* *November 2018 - Current*
- Nature and Portrait photographer | [axstepko.com](#) *August 2017 - Current*
- Mountain Biker