A Stateless Broadcast Network Protocol for LoRa

A Thesis

Presented in Partial Fulfilment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Douglas M. Park

Major Professor: John C. Shovic, Ph.D.

Committee Members: Robert Rinker, Ph.D.; Alan Kolok, Ph.D.;

Department Administrator: Terence Soule, Ph.D.

December 2020

AUTHORIZATION TO SUBMIT THESIS

This thesis of Douglas M. Park, submitted for the degree of Master of Science with a major in Computer Science and titled "A Stateless Broadcast Network Protocol for LoRa," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor _____Date_____
John C. Shovic, Ph.D.

Committee Members _____Date_____
Robert Rinker, Ph.D.

_____Date_____
Alan Kolok, Ph.D.

Department Administrator _____Date_____
Terence Soule, Ph.D.

Abstract

In this thesis a Stateless Broadcast Network Protocol for LoRa (SBNPL) is proposed. Simulations of the protocol were done for validation of the algorithms using the ns-3 simulation software and ns-3 Lora Module (NLM). The result is a viable broadcast protocol that can be used on low cost Long Range (LoRa) devices to create a reliable, secure, and easily deployed sensor network. For busy locations with many LoRa devices the amount of data, while limited, can fall within the regional duty cycle and dwell time for LoRa devices.

Table of Contents

List of Tables

List of Figures

List of Acronyms

SBNPL	Stateless Broadcast Network Protocol for LoRa
IoT	Internet of Things
GPS	Global Positioning System
LoRa	Long Range
Wi-Fi	wireless local area network
WAN	Wide Area Network
LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power, Wide Area Network
RF	Radio Frequency
ISM	Industrial, Scientific and Medical Band
MAC	Media Access Control
CSS	Chirp Spread Spectrum
TQ	Transmit Queue
BQ	Backup Queue
VL	Visitor List
BPS	Broadcast Protocol for Sensor Networks
MANET	Mobile ad hoc network
CIA	Confidentiality, Integrity, and Availability
NLM	ns-3 Lora Module
LBT	Listen Before Talk
FSK	Frequency Shift Keying

OOK	On Off Keying
SHA	Secure Hash Algorithm
SARB	Stateless Adaptive Reliable Broadcast Protocol
RTOS	Real Time Operating System
DOS	Denial of Service
EMP	Electromagnetic Pulse

Chapter 1

Introduction

With commonly available, inexpensive, off the shelf Internet of Things (IoT) hardware it is deceptively easy to create a network of interconnected IoT hardware that perform various tasks, such as video surveillance, environmental measurements (e.g. Temperature, humidity etc.), and many other tasks. The possible uses are limited only by imagination. Eventually, the data collected will need to be communicated from the device to a central repository, such as a database residing on a cloud computing device. There are many devices that collect data that require a direct connection to another device to extract, possibly by storing on an SD card or non volatile memory. This requires physical access to the hardware and limits timeliness of data collection. Some devices can collect data for years without needing service.

A more convenient solution is for data to be transmitted wirelessly through a network of devices, eventually accessing a device that is connected to the Internet. Doing so presents several obvious problems including power requirements, transmission distance limitations, security, and data throughput.

In this thesis we will examine a new method Stateless Broadcast Network Protocol for LoRa (SBNPL) that uses broadcast transmission with and algorithms to mitigate the problems inherent in broadcast mesh networking. The result is a protocol that is straight forward to implement, reliable, and secure-able. While designed to run on low cost Long Range (LoRa) IoT devices specifically it is designed with all IoT communication hardware in mind. A mathematical basis for throughput, number of packets in the system and storage requirements is derived. A simulation is done and results presented that show SBNPL works well under the conditions of a real world oriented simulation of noise and distance.

It is simpler to think of networking as a linear task. Transmit from one node to the next and then the next until the destination is reached. In a complex system where nodes are connected to multiple other nodes this linear thinking leads to creating an end to end route so that at each node the next node in the route is known. This requires routing tables which contain potentially large amounts of state information allowing each node to pick the best route. It also requires either a manual step of establishing the route, or a discovery process that may need to be done repeatedly when devices are added, removed, moved, or replaced.

Another option is to use a broadcast network which cooperates to pass packets from one device to the next eventually getting to the desired location. A broadcast networking system was chosen for this thesis.

1.1 Broadcast Networking

It is fairly simple in wireless communication to simply broadcast a packet and then any node that is listening can then rebroadcast the packet. Eventually, assuming connections are good the packet will find its way to the destination. This simple method does have drawbacks, such as the *broadcast storm*[17] problem. There is also a need to ensure that nodes within communication distance of each other are not broadcasting at the same time causing interference. The broadcast method is well studied[17][32][21][7][34][28][4][20][27][3][2].

Ideally a mesh network is self organizing. A mesh network that uses a routing table needs to discover the available nodes, best routes etc. and then disseminate that information throughout the network with each node saving the information that it requires. When routing information is used the state of the network topology becomes an integral part of the network routing algorithm[34]. A mesh network that uses broadcasting inherently doesn't require a routing table as it relies on other nodes

rebroadcasting the information that a source node initially broadcast. This makes a broadcast network self organizing, within the limitations of collisions and other factors.

Typically the problem of supporting a large number of potentially moving devices is solved using a grid of receiving devices each of which handles the devices that can communicate with it. Cell phone service is a familiar example of this where the availability of a tower or other collection point is important for good service.

With IoT devices there exist several competing communication protocols including LoRa, Sigfox, zigbee, LTE-M, White-Fi, HaLow, and of course the traditional wireless local area network (Wi-Fi)[33][1]. At the start of development of SBNPL the choice to use LoRa was made, based on a combination of distance and cost considerations.

1.2 LoRa

LoRa[1] is a Radio Frequency (RF) communications technique using Chirp Spread Spectrum (CSS) technology developed by Semtech which is sometimes used for IoT networks. Long Range Wide Area Network (LoRaWAN)[2] a Low Power, Wide Area Network (LPWAN) protocol is frequently used for connecting LoRa devices. LoRa uses unlicensed frequencies in the Industrial, Scientific and Medical Band (ISM). The frequencies allowed vary depending on the region. The LoRa hardware supports frequencies from 137MHz to 1020MHz[25]. Regional regulations further restrict the frequencies that may be used.

The LoRaWAN provides for several Media Access Control (MAC) classes of operation. LoRa can be used in a plain text transmission mode, but that is usually not an acceptable level of security. The MAC classes are designated A-C. Class A provides a baseline bidirectional communication method which uses a type of ALOHA[9][11] pro-

[1]LoRa is a registered trademark of Semtech
[2]LoRaWAN is a registered trademark of Semtech

tocol. Transmission of data is done based on a predefined time with a small variation in the time window allowed. It is followed by a two short time windows for receiving data. This is the lowest power option. The class B method uses a synchronization beacon to establish a time basis for the gateway device to know when the device is listening. More receive slots are also provided. The Class C method provides near continuous listening windows that are only closed when transmitting. This is usually only acceptable at a location not relying on battery power[13].

Class C communication is normally undesirable for battery and solar powered nodes. Class B appears to be the most useful as it allows for control of the frequency of send and receive windows.

LoRaWAN is designed mostly to be used in a star configuration with a gateway device which manages many aspects of connectivity. It doesn't provide a standard method of extending a mesh network beyond one hop. There are methods proposed for doing multi-hop extensions of LoRaWAN[3][35]. These can be very useful in situations where powered gateways can be distributed over an area, but for an area where power and/or Wide Area Network (WAN) access is limited or not available the distance limitations of LoRa transmission preclude this design.

LoRaWAN is allowed to do other radio transmission so long as it stays within regional duty cycle and dwell time limitations[13][30]. LoRaWAN is a fairly heavy weight protocol which is more than needed for sensor applications and is potentially prone to security flaws as a result of code complexity. A simplistic transmit/receive protocol that works on a periodic basis, similar to the class A protocol is designed and modeled in this thesis. It doesn't rely on specialized gateway hardware. Low cost devices with LoRa, Global Positioning System (GPS) and Wi-Fi are available that provide all the hardware required to build this system.

1.2.1 Radio Overview

The details of the Semtech LoRa radio are much larger than can adequately be detailed in this document. Indeed the datasheet is in excess of 100 pages[26]. However, there are several things that need to be understood to use it effectively as a mesh network. The radio, depending on the region is designed to use one of several frequency ranges, in the US, which is the primary focus of this thesis, the frequency is in the 900Mhz range. Several sub channels in that range are used. The LoRa communication scheme relies on a number of things that allow some overlap in bands. This means that multiple devices in close proximity to each other can transmit and receive with limited collisions. Frequency Hopping allows large packets that take a long time to transmit to change frequencies in a way that allow other devices to transmit and receive with minimal conflict. When a radio is receiving it cycles through a predetermined set of frequencies until it finds the frequency of the transmission.

In addition to a LoRa Modem a Frequency Shift Keying (FSK)/On Off Keying (OOK) modem are part of the Semtech package. The different modems provide differing communication capabilities, with LoRa being the least susceptible to interference but the least efficient with bandwidth use. There is also a single packet size limitation of 256 payload bytes, while the FSK modem allows packets of up to 2047 bytes[26].

There are several other features that allow for reliable connections. For the most part the connection just works without much tuning.

One of the options is a CRC calculation that provides for a minimal level of reliability in the transmission of data. The SBNPL can also use an additional cryptographic hash on top of this improving the Confidentiality, Integrity, and Availability (CIA)[29, p 3].

1.2.2 Frequency Hopping

LoRa uses a frequency hopping scheme when the transmission time of a packet may exceed regulatory requirements. Each LoRa packet holds the information in a lookup table provided by the host microcontroller. A hopping period is used to switch the transmit and receive devices to the next channel to continue transmission of the long packet. The management of the hopping is done by the LoRa hardware but the choice of hopping frequencies is provided by the host microcontroller. For a mesh network this means each node needs to have an agreed upon hopping sequence[26].

1.2.3 Ad Hoc Network

Complexity in software often leads to issues that degrade the performance, reliability and security of software. The stateless method proposed uses a simple broadcast model. Typically broadcast models suffer from the *broadcast storm* problem. The proposed method mitigates this problem in several ways which will be discussed.

Mobile ad hoc network (MANET)s are a highly studied network concept. It includes stateful and stateless designs. Stateful designs tend to rely on very detailed knowledge of the network nodes and often employ a routing table that is shared by all devices. This is costly in both memory usage and computational expense to generate and maintain the routing[27]. Routing can be both stateful and stateless depending on the algorithm employed.

Stateless MANETs may or may not incorporate knowledge of the network topology. By their very nature MANETs require dynamic determination of the network topology which will change over time resulting in various issues including the loss of connection of a moving node as well as changes in the best route for nodes that remain in contact[27] [20]. One of the more common methods of eliminating the need for a routing table is to use a broadcast method. The most simplistic method is to broadcast packets blindly with each node that receives them rebroadcasting hoping

that eventually the packet will get to the intended destination. This method is usually referred to as flooding. It works, but can result in packets being heard by the same node multiple times. This is costly in terms of power and potential collisions. For example if a node is close enough to hear two other nodes it is possible that both would be broadcasting at the same time. This collision at best degrades transmission performance. Depending on the topology it is possible that packets continue to be rebroadcast resulting in a what is known as a *broadcast storm*[17].

1.3 Motivation

As with many research projects, this project started out as something quite different than the final result. The initial idea was to create a networked mesh of environmental sensors that would be easily deployed, and inexpensive to build. In particular water quality measurements in various lakes and other bodies of water in Idaho were considered. A buoy design with onboard GPS, LoRa, temperature, air quality, and water quality sensors was envisioned. Ideally these would be configured in a way that would be easy to maintain and reliable enough to deliver a few measurements each day. These measurements would be collected and visualized to make informed decisions about the management of Idaho's water resources.

One of the problems to resolve is how to transmit the data. Traditional LoRa configurations use a star topology and special gateway hardware. This creates a limit to how far nodes can be from the gateway. An obvious solution to this would be to pass data from one device to another in hops until a device that is visible to the gateway could pass the data on.

In thinking this through the idea of using a broadcast networking protocol to do this was born. The result is this thesis.

Chapter 2

Prior Work

This chapter expands on some of the prior work that was touched on in Chapter 1. It more specifically covers some of the prior research in the area of building stateless ad hoc networks.

2.1 Insightful Work

There are various methods for creating a broadcast sensor network and exhaustive elaboration of these is not felt to be useful for the purpose of this design. However, several methods do provide insight.

The Broadcast Protocol for Sensor Networks (BPS) method appears to be a viable method that uses a stateless distance based metric for determining whether a node should forward a packet to other nodes in the network[4]. It makes some assumptions about spatial relationships between nodes that are not well suited for real world conditions where node placement is irregular[20].

A derivative of BPS called Stateless Adaptive Reliable Broadcast Protocol (SARB) is proposed where each node controls its broadcast distance. This method uses a probabilistic model and load information from two-hop neighbors to adjust the broadcast distance. It uses the observed idle time of neighbors and assumes an average node density to compute the broadcast distance. In addition local and global adaptation is proposed. Local adaptation allows each node to control the broadcast distance, while global adaptation uses a single distance computation which is broadcast to all nodes for initialization[20]. It is noted that this method could be employed to reduce interference in the SBNPL.

Several LoRaWAN mesh networking options have been explored. These are referred to in literature as Multi-Hop LoRa extensions. Some, such as LoRaBlink [2] and others use stateful routing table solutions, sometimes unidirectional [3] [35]. These methods while interesting and workable in a narrow set of conditions are not general enough for the often necessary bidirectional transmission of data.

The ENDCAST method provides a fairly well analyzed method inspired by cell growth. It uses a counter based method that keeps track of how often a packet is received from neighboring nodes. A timing based mechanism is also used to decide whether to rebroadcast or to drop the data. When the data reaches the desired destination an inhibitor packet is sent. This stops all attempts to transmit a packet[27]. This method looks very promising however it suffers from an obvious latency flaw in that it assumes that another node will be the first node to rebroadcast a given packet. A discussion about how this method compares to the protocol simulated for this thesis is found in Section 3.6.1.

2.1.1 Differences of SBNPL

The SBNPL provides a method for deploying IoT devices that is efficient, redundant and generally requires very little deployment geometry consideration. It doesn't require routing tables like LoRaBlink and others. It makes minimal assumptions about geometry which makes for significantly simpler deployment. Redundancy at each node is provided making it more effective at packet delivery than ENDCAST.

Chapter 3

Stateless Broadcast Network Protocol

This chapter defines the Stateless Broadcast Network Protocol for LoRa (SBNPL) , various timing considerations and provides insight into the the theoretical limits of the protocol.

3.1 Is This Really Stateless?

A stateless network protocol is commonly understood to be one in which the receiver of a message retains no information about the session. In other words it doesn't store beyond the duration of message processing information about the sender of the message[22].

Strictly speaking the protocol described does have state, in that some information is retained for a limited time. The queues and constructs used to mitigate the *broadcast storm* problem could be considered state. Looking at this from a long term standpoint there is no state information. Once a packet has propagated through the network, the information can safely disappear from each node in the system, this seems to fulfill the requirement of being stateless.

3.2 Requirements

When transmitting environmental measurements data integrity is extremely important. Some checks should be provided to validate that data has not changed from the time of collection until it is stored at the server.

The CIA of the system depends on the data measurements being taken. If one is simply measuring temperature or rainfall and the data is destined to be visible publicly then perhaps less confidentiality is required. Even in that case if public safety decisions are to be made based on something like rainfall data some assurance of data integrity is required. Availability of the data is also very important. Chapter 5 expands on security considerations.

Another requirement is that the system be capable of running on typical IoT devices used to create environmental sensor networks. For purposes of this thesis the primary device used as a design consideration is a Espresif ESP32 [5]. This device represents one of the better low cost leading edge devices which provides a reasonable amount of memory, communications hardware, and processing power. It is expected that over time improved devices will emerge that provide better performance. There certainly are other devices which are commonly used that have less ideal specifications. These are not considered in detail.

Since it is considered poor practice to use a significant amount of dynamically allocated memory in an embedded system much of the design considers the restriction of avoiding dynamic allocation of memory.

The *broadcast storm* problem needs to be addressed. A method for limiting broadcasts of packets to maximize system throughput needs to be developed. There are several schemes extant that could work. However, most of them are complex to implement. Figure 3.1 shows how flooding works, with only 4 nodes within communication distance of each other $n(n-1)$ transmit and receive operations will occur if each node broadcasts what it receives only once. When broadcasts are repeated or more nodes are added the problem compounds.

A more ideal connection is shown in Figure 3.2. If nodes are at or near the limit of the communication distance this should happen naturally. For many obvious reasons this is not recommended. This is not a restriction that is acceptable in the general

case. The goal with a mesh networking scheme is to find the ideal path through the nodes such that, ignoring all the other nodes in the network, the result is as shown in Figure 3.2.

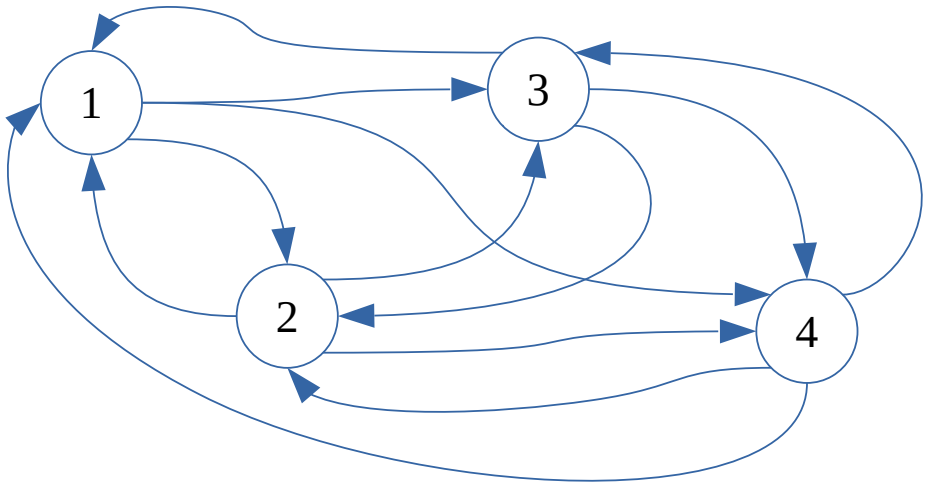

Figure 3.1: Flooding. Assuming nodes 1-4 are all within the communication distance. A broadcast from one with only 1 rebroadcast at each node will result in $n(n-1)$ transmit and receive sessions.

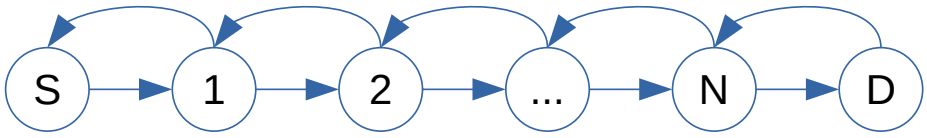

Figure 3.2: A more ideal connection where each node only transmits once in each direction.

3.2.1 Algorithm

The algorithm that will be presented and analyzed is designed to be simple to understand. Complex systems may perform better but are hard to implement and invariably difficult to secure.

While this algorithm is designed specifically for LoRa devices, the concepts can translate to any other hardware that provides as a minimum the ability to transmit and receive during specific time windows.

Statelessness is also embraced for a number of reasons, not the least of which is reducing complexity. Once a node is provisioned it can be deployed to a new location with very little thought to how that location effects the network. About the only thing that is important is that the location be within communication distance of at least 1 other node in the network. This assumes deployment is done from a gateway out. Memory requirements at each node as well as a reduction in information communicated are also side effects of stateless protocols.

It is noted that in this system some state is required in the form of queues and lists of the IDs of packets that have visited previously. The amount of data required for these constructs is examined in the simulations described in Chapter 6.

3.2.2 Concept

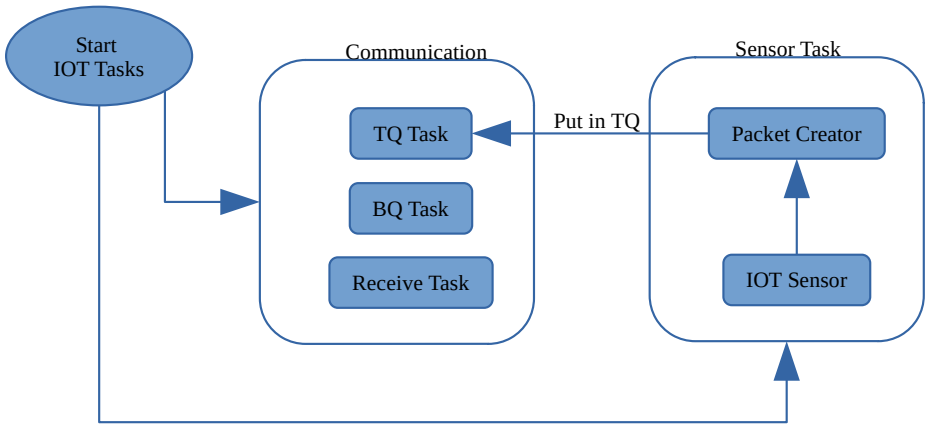

Figure 3.3: Overall Flowchart

The concept of a SBNPL is to broadcast packets of information that will contain node and packet identification along with a payload. Each packet can have a `PacketType` which uniquely identifies what information the packet carries. Packets that need to be broadcast will be placed into a queue to be transmitted which is designated the Transmit Queue (TQ). When the packet is received by the destination node (or server) a receipt called the `EndReceipt` will be broadcast which the sending

node should receive and then remove the item from the TQ. There is a TQ on each device.

In addition to the `EndReceipt` a `NodeReceipt` will be broadcast when any node receives a `DataPacket`.

Packets will be uniquely identified so that if a packet is sent to a node more than once it will not be queued again. In even moderately complex networks packets will likely be received by a given node multiple times. Figure 3.4 illustrates how this can happen. Some protocols such as ENDCAST [27] also use this packet duplication as a flood control mechanism.

Eventually, the sending node should get a `NodeReceipt` and remove the packet from the TQ. However, there is no guarantee that this would occur depending on the reasons for the receipt failure.

To provide redundancy in case of temporary communications failure a Backup Queue (BQ) is provided. When a packet is broadcast that requires redundancy, a copy of the packet will be placed in the BQ. Some packets, such as `NodeReceipt`s, do not require redundancy and will not be placed into the BQ. It is noted that if one wants to have a guarantee of a receipt it could be broadcast as a custom `DataPacket`. However, that would add the additional load of the receipt packets on top of a system that may already be busy delivering other packets.

When an `EndReceipt` is received by any node the packet it references will be removed from both the TQ and BQ.

Periodically the data in the BQ will be placed into the TQ. The assumption is that after some period of time has elapsed it is safe to assume the data did not make it to the server.

The size of the queues will depend on available memory on each device. Devices that need to operate in areas where connectivity is poor may need to provide more memory for the queues to reduce permanent data loss.

In concept all nodes are identical, except that at least one node will be connected to the Internet, probably through Wi-Fi. These nodes will be designated as *gateway nodes* to distinguish their function. It is important to note that the designation as a *gateway node* does not imply the same functionality or hardware requirements as a traditional LoRaWAN gateway. However, there is no conceptual reason why a *gateway node* in SBNPL cannot connect to a LoRaWAN gateway.

The only additional function that a *gateway node* has is to connect to the Internet and send command packets to a server where they will be decoded and stored. Packets coming from the server will be broadcast to each gateway node and so forth until the packets reach their destination. Memory limitations and other hardware limitations may prevent the use of both LoRaWAN and this protocol on some devices.

This design allows for a very broad LPWAN with potentially redundant connections to the Internet making it possible to have pools of nodes that could be viewed as islands. Inter island transmission may occur although ultimately the data needs to make it to a single server. The ability to bridge between islands provides a redundancy in case of power loss or other communication failure at *gateway nodes*.

Figure 3.4 shows the SBNPL system which effectively controls both flooding and the broadcast storm problem. A packet is sent from S to D. S broadcasts the packet. Nodes within communication distance receive the packet and place it in their Transmit Queue (TQ). At the next available broadcast time the packet at the front of the TQ is broadcast. Figure 3.5 shows the overall flow of the receive task. Figure 3.6 shows how different packets are processed. Packets are normally treated the same, except in cases where a receipt isn't needed. In the same or possibly next broadcast time slice a `NodeReceipt` packet is broadcast which holds the packet identification and the identification of the node that last broadcast the packet. It is assumed that the node that last broadcast the packet is still within communications distance. Figure 3.7 shows the logic of broadcasting.

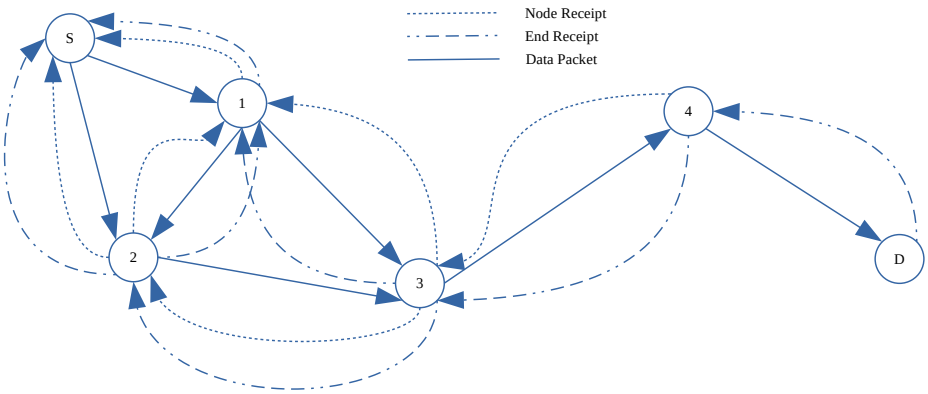

Figure 3.4: Flood and storm control using double receipt method. Solid lines show broadcast of a packet starting at S that needs to get to D. As each node receives the packet it is placed into the TQ and rebroadcast at the next available time slot. When it is broadcast a `NodeReceipt` is broadcast that is directed to the node that broadcast the received packet. This stops rebroadcast of that packet which is then placed into the BQ on that node. As the packet ripples through the system flooding is proactively stopped. When the packet eventually reaches the destination D an `EndReceipt` is sent back through the system which clears out all traces of the packet for each node that it reaches.

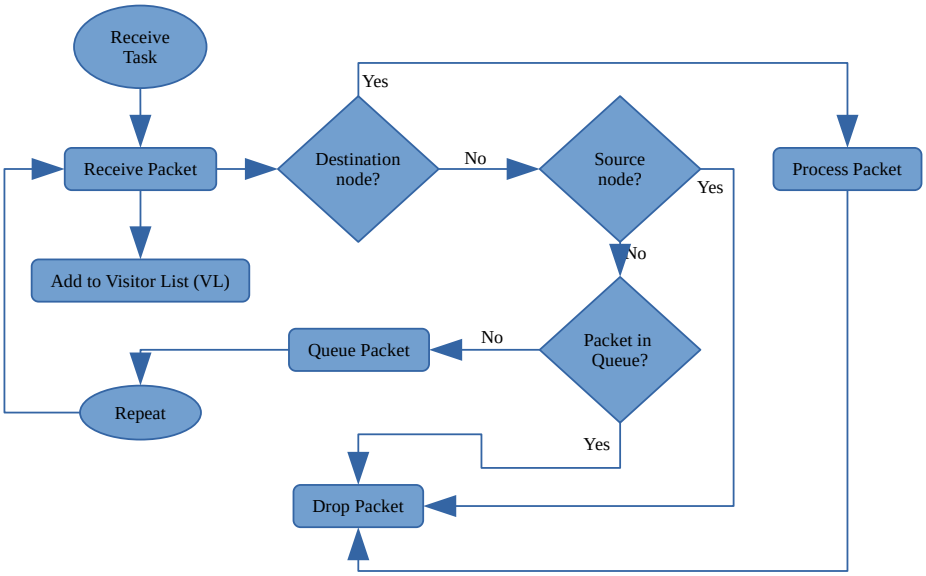

Figure 3.5: Receive Task Flowchart

As packets work through the system nodes that broadcast or rebroadcast a packet are made aware of their success in passing the packet on to other nodes. Since flooding is controlled as the packet ripples through the system there is no need for

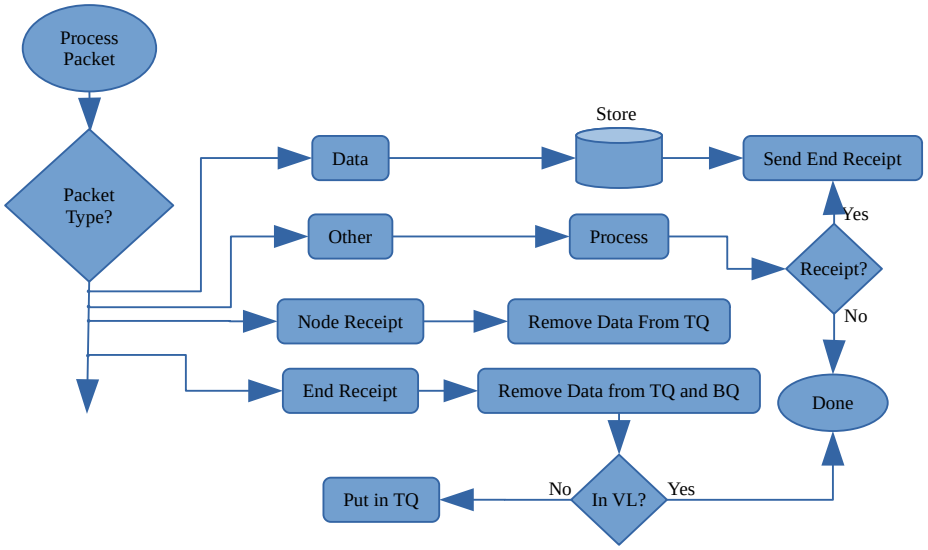

Figure 3.6: Process a Packet Flowchart

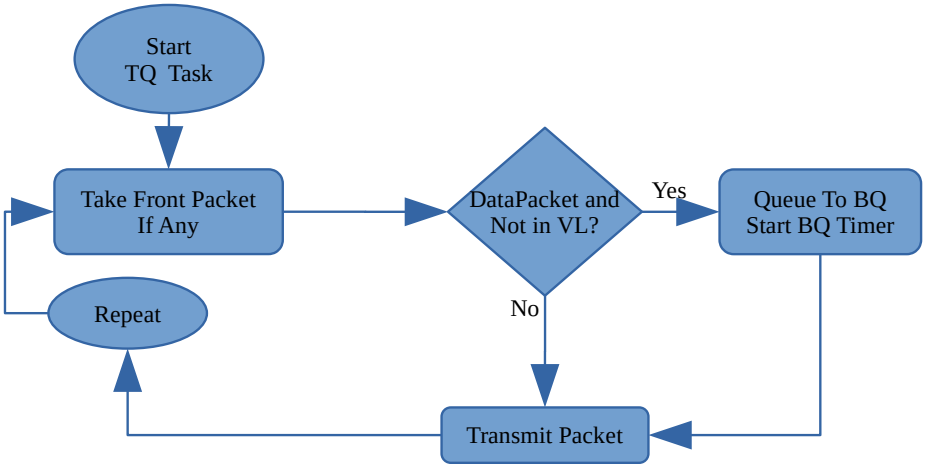

Figure 3.7: TQ Task Flowchart

more elaborate methods that may increase latency.

Eventually when and if the packet reaches the destination D an `EndReceipt` is sent through the system which will clear all traces of the transmitted packet from all the nodes it visits.

If a packet fails to reach D then no `EndReceipt` packet will be generated. Packets are placed into the BQ at the time of broadcast as shown in Figure 3.7.

This redundant broadcast mechanism allows for dynamically changing conditions

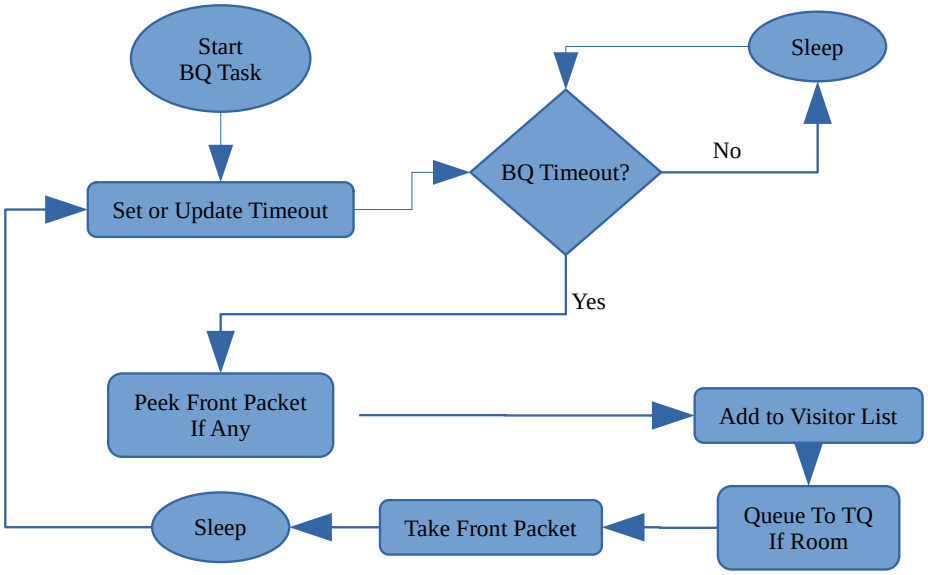

Figure 3.8: BQ Task Flowchart

where a node may from time to time lose the ability to reach its neighbors. For example if a packet were to reach node 4 but the destination node D was unreachable for a time the packet would eventually be broadcast again, after BQ timer expires. This assumes no other packets are ahead of it in the BQ and TQ. Figure 3.8 shows the flowchart of how the BQ task transmits.

The question of how the system responds when a transmission restart is initiated after the BQ timeout is reached needs to be considered. It is possible that more than one node may rebroadcast the same packet from the BQ. The interaction at this point ideally would be such that any node that is in the state of not having received an `EndReceipt` of a new broadcast should be treated as if the first node to rebroadcast is the source.

If nothing is done in this case a new broadcast would fan out toward both the source node S and the destination node D. When a node receives this duplicate packet which was already suppressed and placed into the node's BQ and the retransmit timer has been started there exists the problem of another restart of the broadcast from yet another node. A simple solution would be to move that packet to the end of the

BQ and restart the retransmit timer. This prevents a cascade flooding of packets while not sacrificing the redundancy of the BQ. Eventually a node that has had a packet moved to the back will timeout again and rebroadcast. However, if the first rebroadcast from another node does manage to reach the destination D and an `EndReceipt` is sent and it makes its way through the system all the nodes reached will forget about the packet.

There remains the possibility that one or more nodes never receive the `EndReceipt` and will attempt to broadcast it again. Depending on the type of packet and the how the end node handles the packet, this could result in the duplication of the requested action. For example when a `DataPacket` was sent a receipt will be sent out for that packet. Most of the time data will be sent to a server. The server is expected to keep track of data it has received and ignore duplicates. Storage on most nodes is limited so keeping a long term log is not possible. However, a collection of packet IDs designated the Visitor List (VL) can be kept. The VL can have fixed, presumably small, size. This can be thought of as fixed sized queue where the newest items are at the front. As space is required older items in the VL will be discarded. In a network that is performing well packets will make it around the system fairly quickly. The simulations that were run used a VL with unbounded size allowing the ability to track the memory requirements for a VL. These results are shown in Chapter 7.

Figure 4.1 shows the transmit window. The transmit and receive time T_{on} needs to be large enough at least for a `DataPacket` packet. It is possible to create a multi-part packet in case larger amounts of data need to be transmitted.

3.3 Broadcast

The SBNPL assumes an exclusive transmit and receive capability of the radio. This means that simultaneous transmit and receive on a single device is not necessary.

This is an important property required to use the LoRa radio. Simultaneous receive and transmit are not precluded using this protocol.

To make this work nodes need to agree on when to transmit and when to receive. This requires either an accurate independent time source or a time synchronization method.

In SBNPL both can be used. In systems that have a GPS device the GPS time can be used. Assuming a relatively short transmission distance the GPS time will be very close to the same assuming an unobstructed view of the sky. This is not always possible. Various atmospheric effects and other problems can cause periodic lack of contact with the GPS. Even when using GPS the synchronization should be done via a timer. It can be reset periodically from the GPS.

Another option is to synchronize the time based on when a receive begins. Relying on the GPS reset provides a network-wide synchronization that should be accurate enough to kick off reliable communication. Once established it is probably sufficient to synchronize at the start of every receive. An additional periodic GPS synchronization would provide for a correction if for some reason the device lost track of the timer.

The time for listening and sending doesn't need to be microsecond precise, (refer to Section 4.1 for a detailed discussion of the timing). Inherent in the time sliced nature of this protocol is the minimum period at which a packet can be sent. Flooding and the subsequent *broadcast storm* are already somewhat limited. It is not sufficient to rely on this.

The ability to update timing parameters is possible by either transmitting these values as part of the payload of `DataPacket` or via a custom command. This change needs to propagate to all nodes before it is activated. It also requires that an `End-Receipt` be received from every node. Ideally this is a rare event.

3.4 Queues

Each node has two queues. The first queue, called the Transmit Queue (TQ) is used to actively manage packets that have yet to be forwarded to another node. Packets in this queue will be sent in queued order. When an `EndReceipt` is received the item will be taken out of the TQ and marked for deletion in the Backup Queue (BQ).

When a packet is transmitted it is also placed into the BQ in case it never gets to the destination.

The BQ holds packets that have yet to receive an `EndReceipt`. Packets in this queue are removed when an `EndReceipt` is received for the packet.

Packets in the BQ will be put back into the TQ based on certain actions or timeouts. In a system where RF connections are reliable both queues should remain relatively empty. When data is received from another node the next available item in the BQ that has not been marked for deletion, will be moved to the TQ if there is room. This assumes that communication in each direction is equally reliable. Figure 3.8 shows the operation of the task that monitors the BQ.

It may make sense for some applications to periodically move things into the TQ based on an event (perhaps when the system wakes from a low power state.)

3.4.1 Queue Data Storage

While it is possible to store data within the queue structure, this is an inefficient storage mechanism. Packet sizes vary from large to small. The majority of packets that are in play at any given time will be small. For example nearly half of the packets should be `EndReceipt` packets. Others like `DataPacket` packets can be significantly large.

Use of an efficient storage mechanism for the TQ, BQ and VL on an embedded system is important and should be a core requirement when implementing this algo-

rithm on such devices. The simulation that was done uses comparatively inefficient storage mechanisms.

Table 3.1: Packet Structure as used in Simulation. This is common to all packet types including `DataPacket`, `NodeReceipt`, and `EndReceipt`.

Part	Bytes	Description
`PacketType`	1	Enumeration. At a minimum `DataPacket`, `NodeReceipt`, and `EndReceipt`
`OriginatingNodeAddress`	2	A unique integral value. Could be first 6 bytes of MAC address of the node that originated the packet
`PacketID`	4	4-byte unsigned integer unique to device
`DestinationNodeAddress`	2	A unique integral value. Could be first 6 bytes of MAC address of the node that is to receive the packet
Total	7	Space required

Table 3.2: Data Packet Structure as used in Simulation.

Part	Bytes	Description
`Header`	7	As specified in Table 3.1
`Payload`	N	The data to be transmitted. 20 Bytes in simulation
`Hash`	32	Hash of payload. 32 Byte Secure Hash Algorithm (SHA)-256 in simulation
Total	39 + N	Space required. 59 Bytes in simulation

Table 3.3: Receipt Packet Structure as used in Simulation. It would be possible, perhaps recommended to add the hash to receipt packets.

Part	Bytes	Description
`Header`	7	As specified in Table 3.1
`NodeGettingReceipt`	2	A unique integral value. Could be first 6 bytes of MAC address of the node that originated the packet
`IDOfPacketNeedingReceipt`	4	Packet ID on `NodeGettingReceipt` of the node that requested the receipt. The size of this ID is dictated by how many packets are in the network.
Total	13	Space required.

3.5 Packet Structure

Each packet will contain several parts as enumerated in Tables 3.1, 3.2, and 3.3. Each packet type is enumerated by a byte sized integral value. The entire packet may be encrypted using various encryption algorithms. The Node Address could be the first 6 bytes of the MAC address of the device or any other byte sequence if no MAC address is available. In the simulation it is stored as a 2 byte unsigned integer.

To reduce data transmission size and improve precision, in particular for transmission of floating point values, numbers should be stored in binary. It is recommended that byte ordering be enforced throughout the system (big or little endian).

To be friendly with embedded systems where the memory footprint needs to be constrained a fixed size limit for packets is suggested to be enforced at compile time.

The number of bytes used for a packet is shown in Tables 3.1, 3.2, and 3.3. The sizes and layout of packets as shown is flexible and may be adapted to the needs of the implementation. An effort was made to keep these small for purposes of the simulation but with effort the packet sizes could be reduced further. Clearly only 2 bits would be required to prove 3 unique values in the enumeration and data could be compressed using any of several compression algorithms.

3.5.1 Receipts

Receipts are core to the functioning of this system. There are two kinds of receipts, `NodeReceipt`, and `EndReceipt`.

`NodeReceipt` is sent when any other node receives a `DataPacket` that requires a receipt and the command has been either successfully processed or queued to the TQ.

`EndReceipt` is sent by the server or a node when it has received a command addressed specifically to a the node or server that requires a receipt.

Receipts control the removal of items from the TQ and BQ.

3.5.2 Data

The payload of a `DataPacket` depends on the type of data being transmitted. It follows the form as shown in Table 3.2. This packet is very open to modification subject to the limits of the hardware and software used in implementing the system. The `PacketType` in the `Header` allows for a variety of unique packet types.

It is possible to have packets that are a continuation of another packet, or possibly packets that send GPS position and timing data etc.

3.6 Other Options

Some other options were considered on the way to the SBNPL. This section elaborates these options.

3.6.1 Comparison to ENDCAST

Many methods exist to limit packet transmission frequency. The ENDCAST[27] method provides a model to inspire a less complex storm control method. In SBNPL and in the ENDCAST method a packet is no longer required to be in any of the broadcast queues when it reaches its ultimate destination whether that be a server or the destination node. In the model based on ENDCAST that was considered. A `EndReceipt` packet is broadcast back out through the system. This will continue to propagate through the system systematically removing the packet it identifies from the TQ and BQ. Our model uses a secondary backup queue that sticks around over a relatively long time so that occasional communication loss can recover data. *ENDCAST provides no such redundancy.* Otherwise our method is strikingly similar to ENDCAST.

Storm control in the ENDCAST method is achieved by first delaying rebroadcast of the packet using a delay time, called *chalone wait*, then by keeping a count of the

number of times that a packet has been received by a node from its neighbors. If the number of hits exceeds a threshold the packet is broadcast and removed from the nodes queue. This recognizes the intuition that neighboring nodes are unsuccessful in forwarding the packet to the destination. This intuitively assumes that a neighbor is a better choice for passing the packet along. In SBNPL a `NodeReceipt` is broadcast when a non-receipt packet reaches a node. This has the effect of taking it out of the TQ but leaving it in the BQ allowing for recovery of lost transmissions via multiple nodes.

Flooding control is achieved in ENDCAST by sending a packet called an inhibitor when the original packet reaches the desired destination. The inhibitor then removes the packet from the broadcast queue. In ENDCAST a log is maintained of packets received. If an inhibitor is received and the node saw the packet previously the inhibitor is passed on. The inhibitor has thus stopped flooding, although it can start up again if a node that has not received the inhibitor rebroadcasts the packet. This is nearly identical to the SBNPL `NodeReceipt`.

One of the benefits of the inhibitor concept in ENDCAST is that it can be sent before the packet reaches the destination providing for some flood control ahead of the transmission completion. SBNPL could easily be adapted to incorporate an inhibitor packet if desired. *It is noted that in SBNPL the same effect is achieved assuming an inhibitor packet is sent at each hop.*

The ENDCAST algorithm intuitively appears to work and could be used unmodified as a stateless broadcast mesh network. Inherent in the ENDCAST method is a latency. The *chalone wait* is designed to slow the flooding propagation which is good. In an ideal environment the node that is closest to the next node in the chain that gets the data to the destination should rebroadcast as soon as possible.

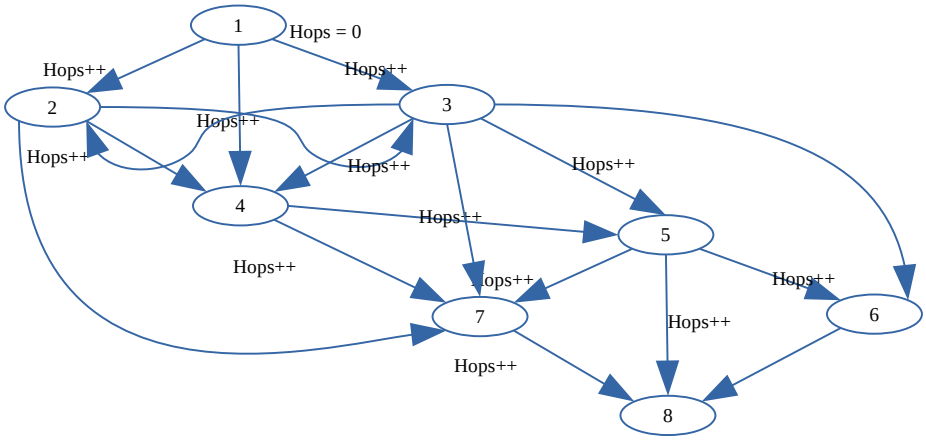

Figure 3.9: Illustrative example of hop counting. Routes are hypothetical. For example starting with node 1 and ending with 8 (1,2,4,7,8) is 5 hops, while (1,2,7,8) is 4

3.6.2 Reputation System

Using ENDCAST as a basis, a reputation based system was proposed for reducing the latency due to the *chalone wait*. Let's call that *chalone latency*. The latency is multiplicative by the number of nodes that impose the additional wait time. To that end a reputation count for a node is used based on how often the node rebroadcasts frames. A counter is kept and incremented every time a packet is rebroadcast and decremented whenever a packet is discarded because it has been dropped from the queue due to the *chalone threshold* check. The reputation will range between 0 (the minimum) and some maximum reputation. If a node has a reputation of 0 that means it frequently drops packets due to the *chalone threshold* check. Higher reputations mean the node rarely drops packets without rebroadcasting them. If the reputation of a node is sufficiently high the *chalone wait* is ignored and the broadcast is done in the next broadcast window.

Figure 3.9 shows some potential paths for a receipt returning from node 1. The number of hops N_h is termed *route cost*. While each transmission may take more or less time in the design proposed the radio send and receive windows are on a fixed time

period t_p as illustrated in Figure 4.1. So the rough time cost is roughly $t_p N_h$. If it can be determined that a node is likely on the best path from source to destination the reputation system can choose to take a packet from the TQ and transmit immediately instead of waiting for other nodes to have a chance to transmit.

Chapter 4

Computational Models

4.1 Timing Model

The time that the LoRa radio is on and off is key to making this protocol work on that hardware. Figure 4.1 shows the general idea of this timing. It is similar to the timing suggested in [13]. Packets queued to the TQ are popped and a transmit is attempted a Listen Before Talk (LBT) is used to determine if a packet viable for receive is in the process of being transmitted. This timing is done in more or less lock step using the timing from some time source such as a GPS. A valid time initialization is necessary to get things started. Some variance is possible as described in [26].

In general this timing model should be flexible enough for other RF communication hardware with possibly some adjustments to the delays in order to optimize throughput.

The radio will cycle between transmit and receive cycles using an intervening idle time T_r or T_p to allow for processing of data on each device before the next transmit receive cycle. At the end of a receive cycle the packet at the front of the TQ is transmitted if possible using a prioritization scheme. If the front of the TQ has a `EndReceipt` then a short delay T_r is inserted after which that packet is transmitted. Otherwise a longer delay T_p is inserted and the next packet on the TQ is transmitted. This gives priority to `EndReceipt` which will clear out waiting packets at nodes where they are received.

Given that a receive has just finished there is a high probability that nodes that can hear the broadcasting node also finished the same receive. Assuming this was a `DataPacket` and the receiving node is the gateway (the probable case when the

gateway is nearby) the next packet to be broadcast will be a `EndReceipt`. This packet will likely make it to the node that just broadcast the measurement. It will also likely hit other nodes that may or may not have the measurement queued for broadcast. By broadcasting `EndReceipt` packets quickly, queued Combined packets that have the same ID as the one that just made it to the destination will get cleaned out quickly.

Another factor is how frequently to pop the BQ. A pop of an item on the BQ results in moving an item to the back of the TQ. The frequency of this has an effect in how long it takes for packets to make it through the system. In the model this is expressed as a multiplier of the T_{on} value. Values explored are enumerated in Chapter 6. Examination of other values is valuable. A value <= 1 doesn't make sense as that would effectively negate the usefulness of the TQ.

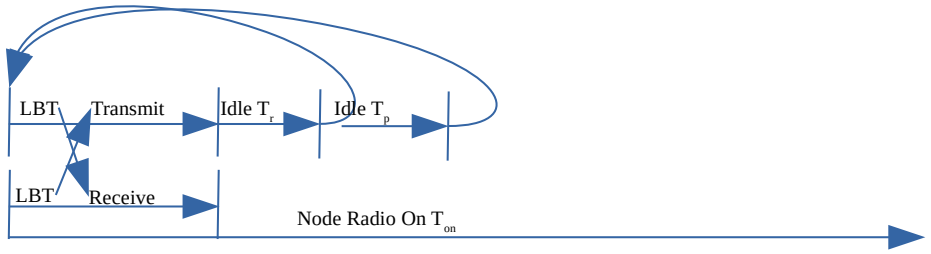

Figure 4.1: A long period where the node is on designated T_{on}. It is off for an equivalent amount of time T_{off}. At the start a LBT is done to decide whether the node should send or receive. It will then send or receive depending on what is heard. After the send or receive is done if a `EndReceipt` or `NodeReceipt` is at the front of the TQ a short idle delay T_r is inserted. Otherwise a longer idle delay T_p is inserted. The process then repeats for as long as the radio is on.

4.1.1 Time On Air

The time on air calculation for the LoRa Modem is documented here and is taken from [26].

$$R_s = \frac{BW}{2^{SF}} \tag{4.1}$$

$$T_{preamble} = (n_{preamble} + 4.25)T_s \tag{4.2}$$

$$n_{payload} = 8 + max\left(ceil\left[\frac{(8PL - 4SF + 28 + 16CRC - 20IH)}{4(SF - 2DE)}\right](CR + 4), 0\right) \tag{4.3}$$

BW is the programmed bandwidth

SF is the spreading factor

$n_{preamble}$ is the programmed preamble length

PL is the number of payload bytes (1 to 255)

SF is the the spreading factor (6 to 12)

$IH = 0$ when the header is enabled, IH=1 when no header is present

$DE = 1$ when *LowDataRateOptimize=1*, DE=0 otherwise

CR is the coding rate (1 corresponds to 4/5, 4 to 4/8)

$$T_s = \frac{1}{R_s} \tag{4.4}$$

$$T_{payload} = n_{payload}T_s \tag{4.5}$$

$$T_{packet} = T_{preamble} + T_{payload} \tag{4.6}$$

Taking the above equations $PL = [1, 255]$ and $n_{preamble} = 12$ (the default) a calculation was done for all combinations of BW, SF, IH, DE, and CR. The

result is shown in Table 4.1. These values cover the range of possible frequency configurations for various locales and offer a good starting point for picking how long a device needs to stay on at a minimum to transmit or receive packets of 255 bytes or less. Note there is a Frequency Shift Keying (FSK) modem on the Semtech devices that allows for packets larger than 255. This may be a useful option in certain cases where larger data packets need to be sent. It would require coordination between devices to use this option. The transmit time for FSK is not considered. Faster data transmission could result because acfsk uses more bandwidth. FSK may make sense if larger packets are required.

Considering the Table 4.1 a value of T_{on} of 500 seconds might seem like a good default assuming no power budget or other constraints. However, for most situations this is likely much longer than is needed. In addition to the T_{on} the delays T_r and T_p to start the next transmission need to be considered. Delays of from 1 to 15 seconds are allowed in LoRaWAN[13]. Using a shorter delay is better because it allows faster overall throughput. Multiple values for T_{on}, T_r and T_p are explored in the simulations as described in Chapter 6 .

Table 4.1: Maximum Transmit Time (seconds)

Size	Max Transmit Time
1	13
32	62
64	112
128	213
255	412

4.2 Theoretical Limits

This will present various mathematical limits of the protocol with respect to LoRa hardware. LoRa hardware has some fairly straightforward limits. For example some-

where around 10km depending on various factors such as the antenna, terrain, elevation of the antenna, interference, etc. the signal loss will result in a transmission failure.

4.2.1 Transmit Time

The time it takes a packet to get from a node to a server and receive a receipt back is one of the more important units of measure. If it can be computed accurately then decisions about the number of nodes in a network the power requirements and other factors can be estimated. In this section a lower bound on the round trip transmit time T_{rt} is derived.

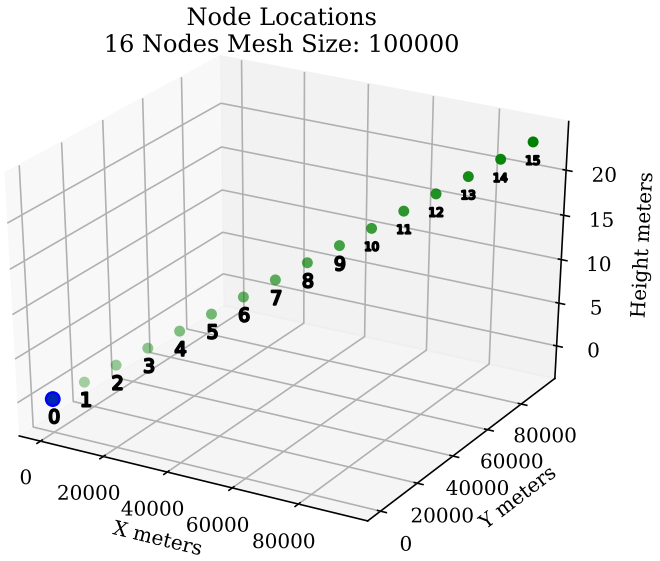

Figure 4.2: Representative Linear Network with 16 nodes. Other node counts are created in a similar fashion. By convention node 0 is the gateway node.

Linear Distribution

The simplest model to understand is that of a linear distribution of nodes, such as that depicted in Figure 4.2. Figure 4.1 shows the timing model. Assuming:

- N nodes in a line.
- Evenly distributed.
- Close enough to each other to communicate
- Far enough apart so that only the previous and next node in the sequence can communicate.
- All nodes use the same values for T_{on}, T_{lbt}.
- `NodeReceipt` and `EndReceipt` are identical in size.

The value of T_{on} gives us a start. There are 3 packet types to consider the `DataPacket`, `NodeReceipt`, and `EndReceipt`.

First consider a 2 node case. The first node in the sequence is designated S, the source, and the last node in the sequence is designated D, the destination. Assuming that each transmission is heard by the other node. Data sent from S to D will get there in one hop. Node D knows the `DataPacket` it just received is destined for D by examination of the packet and sends only a `EndReceipt`. The transmit time can be estimated from Equation 4.6. Starting with one `DataPacket`, recall that in this model LBT is done first. These are simultaneous within some margin of error. This time will be designated T_{lbt}. In Figure 4.1 note that for this model after a packet is received a short wait time T_r is completed before sending the `EndReceipt`. Designating the time on air for the `DataPacket` as T_{doa} and the time on air for the receipt packet as T_{roa} the round trip time using Equation 4.7 can be estimated for the 2 node case.

$$T_{rt}^{(2)} = \begin{cases} 2T_{lbt} + T_{doa} + T_{roa}, & \text{if } T_{lbt} + T_{doa} + T_{lbt} + T_{roa} < T_{on}. \\ 2T_{lbt} + T_{doa} + T_{roa} + T_{on}, & \text{otherwise.} \end{cases} \tag{4.7}$$

The 3 node case requires us to consider the `NodeReceipt` transmit time from the middle node M and S and the time to forward the `DataPacket` from M to D and the time to pass the `EndReceipt` from D to M to S.

The first step is the transmission of `DataPacket` from S to M and the transmission of a `NodeReceipt` from M to S. Assuming that the size of the `NodeReceipt` is the same as `EndReceipt` the transmission time from S to M is identical to $T_{rt}^{(2)}$ as shown in Equation 4.7.

The second part is the transmission of `DataPacket` from M to D and the return of the `EndReceipt` from D to M. Using the size assumptions for step 1, it is clear that the time to do this is identical to the first step.

This leaves only the time to broadcast the `EndReceipt` from M to S. We have previously designated this as T_{roa}.

$$T_{rt}^{(3)} = 2T_{rt}^{(2)} + T_{roa} \tag{4.8}$$

The theoretical best case for the linear model can now be generalized for N hops to Equation 4.9.

$$T_{rt}(N) = (N-1)\begin{cases} T_{lbt} + T_{doa} + 2T_{roa}, & \text{if } T_{lbt} + T_{doa} + T_{lbt} + 2T_{roa} < T_{on}. \\ T_{lbt} + T_{doa} + 2T_{roa} + T_{on}, & \text{otherwise.} \end{cases} \tag{4.9}$$

Mesh Distribution

The linear case is also the theoretical best case for a mesh where the route to destination and back happen to hit the same nodes. The probability of this happening seems fairly likely for nodes distributed in manner that attempts to optimize line of sight and distance while simultaneously minimizing the number of deployed nodes.

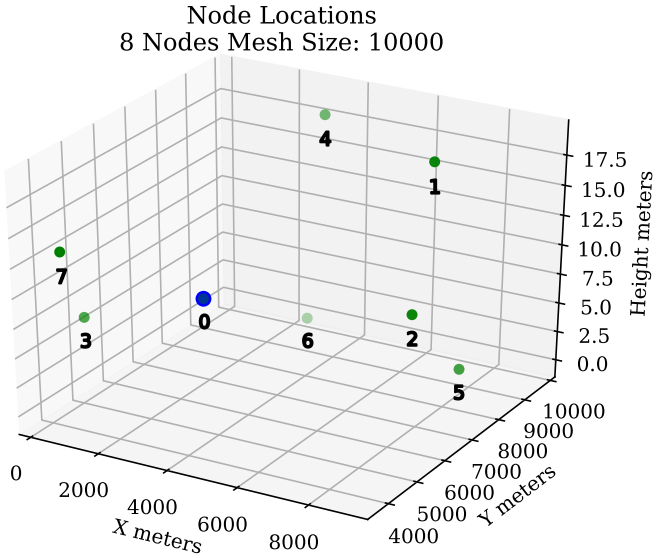

Figure 4.3: Representative Grid Network with 8 nodes. By convention node 0 is the gateway node.

In a real system there is likely going to be interference and cases where the LBT prevent transmission or receipt while the nodes are on. The intuition is that the probability of a node not being able to transmit or receive would be a function of the geometry and number of nodes involved. For the number of nodes designate $N \cdot P_{nf}$ and for the geometry $G \cdot P_{gf}$ where N is number of nodes and G is a factor derived from the geometry. It would seem that in a linear geometry G would be smaller than with more spread out geometries. Assuming that a probability of 1 adds one complete T_{on} cycle results in Equation 4.10.

$$T_{rtf}(N) = T_{rt}(N) + (NP_{nf} + GP_{gf})T_{on} \tag{4.10}$$

4.3 Packets in System

Another measure to consider is how many packets are in the system. More specifically, what is the maximum number of packets a node needs to buffer in the TQ and BQ and also how many packet IDs will need to be remembered. This provides insight into how much memory each device will require to function reliably.

Linear Distribution

Table 4.2: Two Node Queue Sizes

Step	Action	S Size			D Size		
		TQ	BQ	VL	TQ	BQ	VL
0	Start	0	0	0	0	0	0
1	S enqueue 1 `DataPacket`	1	0	0	0	0	0
2	S Add packet ID to the VL	1	0	1	0	0	0
3	S Add packet to BQ	1	1	1	0	0	0
4	S Transmit	0	1	1	0	0	0
5	D Receive	0	1	1	0	0	1
6	D Process	0	0	1	0	0	1
7	D Prepare and Enqueue `EndReceipt` add to BQ	0	0	1	1	1	1
8	D Transmit	0	0	1	0	1	1
9	S Receive `EndReceipt`	0	0	2	0	1	1
10	S Cleanup TQ BQ	0	0	2	0	1	1
Max		1	1	2	1	0	2

Given N nodes in the system, size of the TQ $= l_{tq}$, and size of the BQ $= l_{bq}$ it is trivial to write the theoretical maximum number of packets in the system at any one time as Equation 4.11.

$$n_{pmax} = N l_{tq} l_{bq} \tag{4.11}$$

First consider 2 nodes designated S and D. Breaking this down into discrete steps as shown in Table 4.2. Clearly under ideal conditions the TQ and BQ stay small. The VL is bigger but still small requiring only storage for 2 Packet IDs.

Now consider the 3 node case. Adding a middle node M. The first thing to recognize is that the sizes in S and D don't change. The main differences will be in node M. M will receive a `DataPacket`, generate and enqueue to the TQ a `NodeReceipt` and enqueue to the TQ the `DataPacket` it just received. At this point the size of TQ is +1 larger than it would be for the D node considered earlier. Since the `NodeReceipt` doesn't get placed on the BQ when finally transmitting the `DataPacket` packet to S and D the BQ now has a size of 1. After broadcasting the TQ drops back to 1. When the `EndReceipt` comes back through M the BQ gets cleared out, the `EndReceipt` is put on the TQ and in the VL, transmitted, but not added to the BQ because it is on the VL.

Table 4.3 shows how this plays out. The first thing to note is that in the 2 and 3 node scenarios the VL is never used. The next thing to note is that for the general case with only one `DataPacket` packet in the network the conditions at S and D will be the same under ideal conditions. This leaves us with expanding out the case of M to $M_1, M_2, \cdots, M_n$. Working through in a similar fashion as show in Table 4.3 the result is an obvious repeating pattern where $M_1, M_2, \cdots, M_n$ takes on the values of steps 5-8, D repeats steps 9-12.

This leads to the best case of a linear system with only 1 `DataPacket` generated. TQ max is 2, BQ max is 1, and VL max is 3. Of course there is nothing preventing each node in the linear case from injecting new `DataPacket`s into the system. Assuming $N-1$ nodes are actively injecting new `DataPacket`s into the system it is easy to see that at least for the BQ space for $N-1$ packets is required to avoid packet loss. In reality it is less than that because items in the BQ are removed when the associated `EndReceipt` packet passes through. For the TQ the max value of 2 is very short lived. It is probably gone in the same T_{on} window that received the packet it is associated with. Unless the network is very busy the space required for the TQ is N. Looking at VL it is observed that it does get used once in the case of $M_{i+1} \Rightarrow M_i$ to track

the `DataPacket` and also in the case of $M_i \Rightarrow M_{i+1}$ tracking `EndReceipt`s. Assuming that only one `DataPacket` and `EndReceipt` are in the network in a single out and back cycle, a size of 2 is likely sufficient since only the most recent packet to visit is ever used. Using similar logic as for the TQ and BQ results in a max size of VL of $2N$.

For a very busy network these limits may not be sufficient to avoid all loss. In particular, assuming the node D is a *gateway node* it is possible that it will end up with a backlog of receipts to transmit. In addition many other conditions such as interference may create nodes in the system that end up with more packets in their queues. This may be particularly true of a node that happens to be the only node visible to several other nodes and is also visible to the next node on the way to a *gateway node*.

Mesh Distribution

For a non linear mesh take the results of the linear mesh in the previous section as the best case. The reasoning in Section 4.2.1 applies here as well. It is a likely that the best case is found when nodes are deployed in a way to optimize transmission. Interference and other factors can change this.

Using the logic for deriving Equation 4.10 results in Equations 4.12, 4.13, and 4.14 for the general case.

$$Size_{tq}(N) = N\left(1 + \frac{1}{P_{nf}GP_{gf}}\right) \tag{4.12}$$

$$Size_{bq}(N) = N - 1\left(1 + \frac{1}{P_{nf}GP_{gf}}\right) \tag{4.13}$$

$$Size_{vl}(N) = 2N\left(1 + \frac{1}{P_{nf}GP_{gf}}\right) \tag{4.14}$$

Table 4.3: Three Node Queue Sizes.[1]

Step	Action	S Size			M Size			D Size		
		TQ	BQ	VL	TQ	BQ	VL	TQ	BQ	VL
0	Start	0	0	0	0	0	0	0	0	0
1	S enqueue `DataPacket`	1	0	0	0	0	0	0	0	0
2	S packet ID to the VL	1	0	1	0	0	0	0	0	0
3	S enqueue to BQ	0	1	1	0	0	0	0	0	0
4	S Transmit	0	1	1	0	0	0	0	0	0
5	M Receive	0	1	1	0	0	1	0	0	0
6	M Produce `NodeReceipt`	0	1	1	1	0	1	0	0	0
7	M Enqueue `DataPacket`	0	1	1	2	0	1	0	0	0
8	M Transmit `NodeReceipt`	0	1	1	1	0	1	0	0	0
9	M Transmit `DataPacket` S and D receive	0	1	1	0	1	1	0	0	1
10	D Enqueue `EndReceipt`	0	0	1	0	1	1	1	0	1
11	D Transmit `EndReceipt` M receives enqueues to TQ adds to VL, cleans BQ	0	0	2	1	0	2	0	1	1
12	M Transmit `EndReceipt`, S and D receive, D and S cleans BQ	0	0	3	0	0	2	0	0	1
Max		1	1	3	2	1	1	1	1	1

[1] In the simulation the `NodeReceipt`s were counted and collected in the unbounded VL used for the simulation. This table assumes that `NodeReceipt` packets are not put into the VL as would be the case in an IoT implementation. Packets are only added to the VL once.

BQ

The final thing to consider is the BQ. Assuming the pop of the BQ and placement on the BQ occurs when the network is quiescent and there are not pending TQ packets,

the time for a BQ packet to make it through the system is identical. It simply becomes an additive load to the system until it is cleared out or lost due to queue size limits.

Chapter 5

Securing the SBNPL

The SBNPL is designed to be securable or not depending on requirements of the implementation. It is unfortunate that historically security has been an afterthought in software design and implementation. Often assumptions about the lack of a need for security are based on an optimistic trusting view of those who may have access to a system. Given that the SBNPL is specifically designed to be used in a wireless environment neglecting security would be an egregious oversight. This chapter looks at the risks, problems and means of securing a system using SBNPL.

5.1 Risk Assessment

Considering the motivating case of an environmental sensor network as the impetus for the design of SBNPL the first thing one should do is to assess the risks.

The risks of an environmental sensor network are low. The data, in the described scenario are intended to be public so access to the data itself is not something that needs extremely tight control. However, data integrity is important to protect.

Detecting data corruption at key points in the communication chain, whether intentional or accidental, is key to ensuring the data is correct.

An illustrative example helps to understand the risks. Assume air quality is being monitored, in particular smoke from forest fires. If measurements are made that show dangerous air quality and a community reacts to the problem by helping at risk citizens there is an associated cost. A greater cost would be if measurements suggested that there wasn't a problem and someone died because of relying on measurements that were wrong.

It is clear that in both cases an incorrect measurement has an associated potential cost to the community. Someone who is in a position to profit from either of these cases may be motivated to inject measurements that support their agenda. While it is hoped that in the case illustrated that other factors, including good judgment would mitigate the risk of bad data, there is an implicit ethical requirement to take reasonable care to ensure the data integrity is maintained.

Some things may not be possible to correct, such as someone artificially influencing the measurements. It isn't practical to attempt to control all possible factors.

In an implementation of SBNPL for environmental measurements things that are necessary to provide minimal security for the low level risk are expected to be the first things required to provide higher levels of security. It is likely that one could overshoot the low level of security without much additional effort. However, to provide a medium or high level of security would require significantly more effort. Depending on the requirements more effort may be warranted.

5.2 Known Problems

As was mentioned in Section 3.2 the ESP32 hardware was used as a baseline for understanding the requirements of the SBNPL. It is useful to examine known issues with this hardware to gain insight in securing an implementation. Several known defects were found in the ESP32 and ESP8266 [12]. A fix has been released for the ESP32 hardware. The ESP8266 was not fixed at the time of the cited publication. Another hack of the ESP32 security features using a voltage glitching technique has also been performed [10].

Typically embedded systems require the use of fairly low level programming languages to provide real-time performance. C and assembly language have traditionally been used. Both carry a legitimate stigma as being prone to coding errors that can

be exploited. In more modern systems C++ is available and while it allows all the same problems as C and insertion of assembly instructions, it also provides higher level constructs that when judicially used can make it easy to avoid certain common security coding mistakes, while at the same time providing performance that can be better than traditional C and now out-of-date C++ programming methods.

A cursory examination of various open source code that is available reveals many vulnerabilities. Much code has obvious buffer overrun potential. Even well respected libraries that when used correctly are secure can be easy to use in a way that introduces exploitable vulnerabilities. Experience suggests that this fragility is often neglected, even by experienced programmers.

An expected implementation of SBNPL would use a Real Time Operating System (RTOS). The use of FreeRTOS will be briefly explored. While it has significant exposure that allows for easy to make security mistakes some things are simple to make safer.

A very interesting and in depth document about hacking a Jeep Cherokee using only wireless access shows how a complex system using a wide range of systems created a very big target for attack. It also illustrates the benefits of penetration testing and good ethical reporting that resulted in timely patching of the vulnerabilities [16].

5.3 Likely Problems

The exemplar system comprises many different common software components including network communications, possibly public web based access to the data, a database and devices that are physically located in areas that are not secured. Focusing on the most obvious places where vulnerabilities are likely to exist is considered a best practice.

5.3.1 Common Attacks

Common IoT attacks [23, p. 44]

- Network scanning and mapping
- Protocol
- Eavesdropping
- Cryptographic algorithm or key
- OS and application integrity
- Denial of Service (DOS) and jamming
- Physical security
- Escalation of privilege

While not exhaustive this list provides a starting point to identify a focused set of areas to examine and protect. Depending on the hardware being used some of these may not be worth worrying about. For example some RTOSes don't have the privilege models that are common in operating systems such as Linux.

However, they can be much more susceptible to buffer overruns because of the use of C, C++, and the occasional assembly code. It is typical of programmers who use C or don't know C++ well to use coding practices that lead to vulnerabilities. In addition interrupt handlers, RTOS tasks, and other similar things require correct synchronization constructs or race conditions can occur that could be exploited, or at best lead to a system crash or hang.

5.3.2 Physical Security

Physical security is of particular concern for the sensor network example because the IoT devices will in many cases be easily accessible. A buoy anchored and floating in a lake can be easily accessed by a bad actor in a boat. They can simply pull the device from the water and take it away to be opened up in an unobserved location.

5.3.3 Measurement Validation

Measurements can be wrong, a sensor can fail. It may fail slowly or abruptly. In both cases it is ideal to detect failures. Some failures are easy to detect, for example if we are measuring water temperature and there is another measurement, perhaps from another sensor on the device or another nearby device, or sent from the main server a sanity check can be performed. For example if we know the approximate air temperature is above freezing we can estimate a range of temperatures that are reasonable above and below the water.

Other sensors have sanity checks as well. A definitive bad reading should be marked as such before it is sent back to the server. Suspect readings may be another category. Other analysis may be done at the server or potentially any other node in the system to detect anomalous readings.

Detection of obviously bad data can also help identify a possible intrusion. While intrusion is more than likely not the cause of a bad data measurement, there is a chance that an intrusion has occurred.

In any case the sooner in the communication chain that a reading can be validated the better. The sooner the defective instrument can be replaced the better.

5.3.4 Communication

When environmental measurements are taken they will need to be placed into a communications queue and transmitted to the server. A mechanism, perhaps as simple as a check-sum or perhaps more complicated such as block-chain [36] can be used. The simulation of SBNPL explores use of a SHA-256 cryptographic hash [29, p. 637-640].

Block-chain seems like a fairly heavy weight protocol for data where the security risk is low to medium.

Data should be transmitted as it is collected, especially if the bandwidth of the

radio used to transmit the data is small. One could collect the data on the remote device and send it in an encrypted block as a possible double check.

5.3.5 Software Vulnerabilities

The obvious vulnerabilities such as buffer overrun, SQL injection, race conditions, and scripting attacks need to be considered.

5.3.6 Local Data Storage

Data stored locally on the IoT device is at risk of being compromised. The storage medium may be subject to Electromagnetic Pulse (EMP). Some mitigation strategies such as a Faraday cage may be helpful for that sort of attack.

Additionally, physical security of the device may be a concern. If an SD card or similar storage device is used to store data until it can be transmitted then making the SD card difficult to access is important.

Again, transmitting the data as soon as possible is the most effective way of ensuring there is no loss or corruption.

5.4 Possible Solutions

5.4.1 Common Software Vulnerabilities

Buffer overrun, SQL injection, race conditions, and scripting attacks are easy to prevent using a handful of simple programming techniques.

The most simple conceptual tool is to wrap common and likely vulnerable operations into classes and/or functions that provide a single point to examine and test for problems. The idea of wrapping is identical to system call interposition used for security monitoring.

Avoiding things like the deprecated and notoriously problematic C functions like strcpy is an obvious technique. The use of these or similar functions in open source libraries should be a red flag to look deeper for more errors in a library.

Good coding habits and leveraging warnings and other static code analysis tools should eliminate or identify the most egregious, but not all problems.

Software Libraries and Tools

For many IoT devices there exist a wide range of libraries, many of which are open source. While an in-depth examination of tools such as an RTOS is time consuming a cursory examination of smaller libraries is often fairly straightforward. In many cases one can quickly ascertain the level of attention to detail and quality standards that the author(s) of the code strive to attain.

For more complex systems like an RTOS choosing a library that is very commonly used and that has a good reputation can go a long way toward avoiding issues.

Encapsulating commonly used routines from libraries also provides a level of defense and an opportunity to provide quicker fixes to vulnerabilities that have been discovered.

On the server side languages such as PHP or python that are less susceptible to common problems like buffer overflow may be used. The use of well respected and up-to-date distributions of web servers and databases will help to keep the server side of the system secure.

There is also a possibility of using a language like embedded python on the IoT devices as well. However, in some cases real time requirements may preclude the use of such languages, and often they are based on exactly the same open source C++ code libraries that would be used to support access to sensors. This makes the possible security advantages questionable.

5.4.2 RTOS

It may be possible to use a more secure RTOS. For example SAFERTOS [24] is an RTOS based on the FreeRTOS design that claims to be more secure by design. However, it doesn't support the ESP32 at this time. Another option is the Amazon-FreeRTOS [6]. Which appears to be supported on some ESP32 Espressif hardware.

One of the major needs in FreeRTOS is to access a queue. To that end code has been written to all but eliminate the potential errors inherent in casting to and from a void pointer. This code provides a type safe API to an egregiously unsafe API hiding the gory details that are easy to get wrong. This is shown in Listing 5.2. This also demonstrates the notion of wrapping of a software API from a library. An example of the use of TypedQueue is shown in Listing 5.1 This is most commonly referred to as the adapter pattern.

Regardless of the choice of RTOS, use of the adapter pattern is very useful. In C++ there can be zero overhead of using this when the adapter is an inline function. It allows later modification to possibly fix vulnerabilities and/or place hooks to detect attacks.

This makes it simple to declare a queue, put something on it and later take it off. A properly declared struct or class may also be safely used. The hiding of the details along with sufficient testing of the class to validate that it works makes it possible to use what would otherwise be a security nightmare routine with a high degree of confidence that it is safe. This does of course assume the underlying routines are secure. However, if they are not, changes in this routine could potentially work around issues at the lower level.

Listing 5.1: Sample usage of TypedQueue

```
// A queue of float values with length 20
TypedQueue<float, 20> theQueue;

// put something on the queue
```

```
theQueue.send(12.5);

// Some time later presumably in another task.
float v;
theQueue.recieve(v);
```

While the method of isolation is definitely a best practice it doesn't guarantee safety. Other methods such as compile time warnings, and static code analysis should be used to identify vulnerabilities.

Listing 5.2: Templatized Type Safe queue using non type safe API.

```
template <typename T, UBaseType_t QueueLength = 1>
class TypedQueue {
public:
const static UBaseType_t Length = QueueLength;

QueueLength
TypedQueue()
: mQueue(xQueueCreate(QueueLength, sizeof(T))) {}
~TypedQueue() { vQueueDelete(mQueue); }

UBaseType_t size() const {
return uxQueueMessagesWaiting(mQueue);
}

bool empty() const { return size() == 0; }

bool full() const { return size() == Length; }

BaseType_t receive(T &data, TickType_t timeout) {
return xQueueReceive(mQueue, &data, timeout);
}

BaseType_t send(const T &data, TickType_t timeout) {
return xQueueSendToBack(mQueue, &data, timeout);
}

void reset() { xQueueReset(mQueue); }

private:
QueueHandle_t mQueue;
};
```

5.4.3 Formal Methods

More formal methods to design in security from the start would be important for a system where the risk is higher. However, these typically add very significant development time. For some systems multiple years may be spent in various phases of development[23, p. 80]. For many systems the time allotted for development start to finish is on the order of months. Focus on using the most efficient means of vulnerability prevention should be employed in these cases.

5.4.4 Hardware Security Support

IoT hardware may provide security support. For example the ESP32 provides secure boot and encrypted flash options to help ensure that only your code gets executed. While there is no guarantee that these features don't have vulnerabilities, they present an obvious target for penetration testing so vulnerabilities in these areas are more likely to be discovered. Another option that the ESP32 provides is built-in hardware accelerated encryption [5].

5.4.5 Remote Software Update

In an ideal world a secure method of remotely updating the IoT device software would be available. If a vulnerability has been identified, having a way to remotely update the software on the device makes correcting the flaw much easier. For systems like the one considered here the remote and possibly somewhat inaccessible location of the device makes doing a remote wireless update an attractive proposition. It is noted that this may need to be done over Wi-Fi rather than LoRa because of bandwidth limitations. Still, it would be a lot easier to just get close enough to do a Wi-Fi update instead of pulling the IoT device updating and then redeploying.

5.4.6 Physical Security Measures

Addressing physical security is another important factor. If the devices have communication and geolocation features, one of the possibilities is to track the device location logging it for as long as possible. In the best case the bad actor that has stolen the device might not get out of communication range and a precise location of where the device stopped moving can be found.

This is of course fairly easy to hack by either active jamming or covering the radio antenna(s) with a Faraday cage.

Regardless of the situation a loss of connectivity would be detectable. For example a regular check-in "I'm alive" signal could be a core part of the system for reasons beyond just security.

Another possibility is to put a tamper detection circuit into the system that would disable the system.

Another option on physical security is to provide some sort of active feedback such as a very loud beeping. This too could be triggered by a tamper circuit.

One of the more probable scenarios of tampering would likely be by someone who is overly curious but not intending to bypass security. An audible alarm could then be an effective deterrent.

Making opening the system up require a certain amount of destruction may be sufficient to deter unmotivated intruders. Still the ability to open up and service the device without destruction seems like a necessity, so the tamper resistance needs to be thought out.

5.4.7 Security Alerts

Alerts to security problems for the software and hardware used should be monitored on an ongoing basis for the service lifetime of the system. Every problem that pops up should be evaluated.

Updates to the various libraries used should be periodically evaluated and integrated into the system.

5.4.8 Coding for Maintenance

All software needs to be maintained. Updates to add additional features, fix defects and security flaws create an opportunity to inject a defect into the system. Typically defect fixes have something like a 7% bad fix rate or worse in the case of poorly written code.[8, p 287] A good rule of thumb is that for about every 10 fixes a new, potentially worse, problem is likely injected into the system.

Spending the time to design code that is easy to understand leads to fewer problems when the time comes to patch a security flaw. Even so, updates should not be made without an appropriate need as that can lead to new security issues.

5.4.9 Certified Reusable Code

One of the most significant ways to reduce defects is to use code that has been certified to be reusable. This is code that is well tested and has a history of being used in the field for enough time that there is a suitable level of confidence that it works correctly [8, p 123, 132]. While formal certification requires independent testing, doing informal ad hoc, but careful, code reuse can have significant benefits in quality improvement.

Software engineering processes as basic as finding a review of a library, or a cursory review of the code can provide significant insight into code quality. Code that is hard to read is often an indicator, of poor quality, but not always. Code that is tested and has significant field use, even when poorly written, tends to become immune to defects over time. For that reason finding libraries that have good history of use and fixes can help avoid problems that would come from brand new code. Although new code can be good it is usually best to choose a mature library.

5.4.10 Penetration Testing

In order to have high confidence that security is adequate penetration testing is necessary. For the system that is being considered doing extensive penetration testing would require a lot more time than is available. However, if the system is replicated and used in a larger environment this should be considered.

5.5 Security Conclusions

IoT devices present an attractive target depending on the devices. The proper evaluation of risk at the start of development of a system using IoT devices is a critical first step. While the environmental sensor network system is considered to have a low risk, there are consequences of poor data integrity that motivate the use of enough security to prevent obvious, straight forward attacks.

Integrity of the data in an environmental sensor network is probably the most important aspect of security to be considered. By designing a system that provides an adequate level of data integrity assurance using just the idea of defensive and quality coding standards a system with modest security can be developed even when compressed time schedules are imposed.

Keeping software in the system up-to-date and proactively looking for and implementing fixes to vulnerabilities will make a system much more secure. This minimal level of security provides a baseline for building more secure systems where higher risk is involved. In many cases the needs for minimal security and the needs for maximal security overlap.

With good software development methods, such as using a reasonable development process and preferring reusable code along with cautious timely updates for security, an IoT system will not only be secure, but it will remain secure and potentially become more secure over time.

Activities like penetration testing along with security specific development methods can be used to improve the overall security for cases where higher risk is involved.

Chapter 6

Simulations

This chapter will discuss the various simulations performed and details about the simulation.

6.1 Simulation Details

This section discuss the simulation used, along with modifications that were done in support of the simulations. It will also reference the online location of the modified software along with the simulation code. It would have been possible to build a network of IoT LoRa devices to validate the SBNPL. One could start with 2 nodes and get that working, but that doesn't provide much in terms of validating that data is getting correctly forwarded and interference issues are being properly handled. Additionally, it would require some sort of instrumentation, perhaps an SD card to collect packet information as it flowed through the device. Other possibilities such as hardware failure would not necessarily be obvious. With a simulation the number of nodes can easily be expanded and they will work consistently, assuming the simulation software is correct.

After taking these things into consideration it became obvious that a simulation would provide the means of validating that the SBNPL would work. A review of network simulation software suggested fairly quickly that the ns-3 simulator [19] appeared to provide all the needs for doing a simulation. This simulator appeared to be well supported and has been around for a long time. It is a discrete event network simulator. This means that events are created and processed in time order. It is not a realtime simulator so a very long running simulation can be done in a relatively

short amount of computer time. After some research was done it was found that a LoRa simulation module had been developed to run under ns-3 [31]. This seemed to be a good direction to go as building a LoRa simulator is not trivial to do.

Ns-3 is very broad and deep in functionality and has a long history of use for many research projects. This is both good and bad in that software that has been around for a long time tends to be more stable, but it is often, as in the case of ns-3 burdened with legacy concepts. Written in C++, it uses smart pointer concepts, a fair amount of template meta programming and other advanced C++ concepts that have been replaced in newer revisions of C++. These make certain things easy to implement, but hard to debug. As such it proved to be both worthy of its reputation and something that made understanding the ns-3 Lora Module (NLM) harder.

6.1.1 Simulation Design

Using the NLM [19] developed by To and Duda [31] a simulation was done to test the flood control and *broadcast storm* control aspects of the method proposed. The NLM example code required several modifications in order to run correctly which were done and are available at `https://github.com/dougparkarosa/SBNPL_LoraSimulation` along with a snapshot of the software to run the simulations and produce many graphs, including those in this thesis. While there is sufficient justification to question the validity of the model based simply on observed code quality, it works well enough to validate the broadcast method proposed. Transmission is done using the NLM which is claimed to provide a realistic model of the LoRa ALOHA like protocol. The model generates a specified number of LoRa end nodes at random locations in a 2 dimensional grid of a specified width and depth each with a random elevation and a linear distribution of nodes at uniform intervals. Code to simulate the TQ, BQ, VL and other essential elements of the algorithm was created.

Data collected are the number of packets in each queue at discrete times, the time

at which packets were transmitted and received, and the number of times a node participates in a successful end to end transmission. Tracing of packets through the system along with arrival times of packets. Also recorded are failure events and in the case of artificial corruption the number of corrupted packets and the number of corrupted packets detected.

6.1.2 Initial Simulations

Various ad hoc simulations were performed as a validation test to show that packets travel through the system and that the simulation code works as expected. In these experiments initially a grid of devices was created at locations where each node is visible to every other node. In one case a packet was scheduled to be transmitted between two randomly picked nodes. Each node in the simulation is also scheduled to listen and periodically forward queued packets. An ideal LoRa radio model was used to provide best possible throughput. In this fashion a series of modifications were made with increasing sophistication of the simulations and data collection. At each iteration graphs were produced to provide confidence in the simulation.

Packets are tracked at each node and a report at the end is generated showing which nodes saw packets and how many packets remain in the BQ and TQ. In these scenarios ideally the transmitted packet reaches the destination and an `EndReceipt` is queued and transmitted in the next transmit window for the device.

Various sizes of grid were used. The software emulating the Lora device had a hard-coded limit of 254 devices. It was expected that moving that limit to at least 64K devices would be necessary to make this experiment valid in the limit of real world networks where upwards of a 1000 or more devices may need to be deployed. Ultimately the idea of doing more than 100 nodes was abandoned due to limits of the simulation software which ran into the limits of memory, disk, and CPU performance.

Initial Results

A change to support 64K devices was made and tested with 1000 and 10,000 nodes. This was doable without logging of results. The results were consistent with running a small number of nodes.

6.1.3 Primary Simulations

The primary simulations were done using a node, designated the *gateway node* which was set as the target of periodic transmission of data from every other node. In a real world situation this node would be responsible for forwarding packets to the Internet based server to collect the data. For the simulations done this detail is not implemented as it is conceptually simple and has a large number of easily found examples. The simulations record that the data was received and then discard it after creating a Destination Receipt packet that is queued for transmit in the TQ.

Success in this simulation is receiving all `DataPacket`s, potentially multiple times, that were sent, sending receipts and verifying that at least one receipt makes it back to the node that originally sent the packet.

Table 6.1: Fixed Value Parameters

Parameter	Value	Comments
Simulation Time	36,000s	Long enough to allow most packets to make it through the system based on the other parameters
Period	7200s	Number of seconds between periodic transmits from each node.
Runs	20	Each combination of parameters were run repeatedly to get a reasonable statistical sampling

In the modeling software there are two models an “Ideal” and a “Thorp” model. The Ideal model assumes fairly ideal conditions, while the Thorp model attempts to calculate drop off of signal strength with distance at relatively good accuracy. Both

models also look at interference, which is two or more nodes broadcasting simultaneously thus creating noise that prevents other nodes from being able to receive data.

While LoRa provides a simple checksum for authenticating transmission SBNPL packets carry a SHA-256 hash for ultimate validation. It is arranged to be at the end of the sent packet and is verified upon packet receipt at each node.

Table 6.2: Grid Sizes to Investigate

Size	Comments
100m Grid	This should provide under nearly all conditions nearly 100% packet delivery.
1km Grid	This should provide under nearly all conditions nearly 100% packet delivery.
10km Grid	At the edge of theoretical limits more packet loss is expected
100km Grid	Successful transmission will require finding a multi-hop path
1km Linear	This should provide under nearly all conditions nearly 100% packet delivery.
10km linear	At the edge of theoretical limits more packet loss is expected
25km Linear	155miles, with sufficient number of nodes
500km Linear	310miles, with sufficient number of nodes

Further verification is done by randomly choosing to insert noise in the packet, including potentially the hash. In addition the packet may be truncated to simulate other transmission failures. This is done, for practical purposes at transmission time and checked upon receipt. Verification of the failure is done when the packet is received. Since the simulation is being done in software this failure is very artificial and provides little in the way of real world validation. It did however expose defects in the implementation, so was in a way an interesting experiment in destructive software testing.

Initially it was thought that running a model with 1000 nodes would be feasible. However, there are many places in the code where $\mathcal{O}(n^2)$ algorithms are used by the ns-3 software, and there are possibly some other cases that are worse. Running with

Table 6.3: Number of Nodes to Investigate

# of Nodes	Comments
2,4,8	A small number to make debugging problems easier
16,32	Enough nodes to create significant interference loss
64	More interference and potentially more paths to discover

1000 nodes was attempted but it proved to be very slow on available hardware so the number of nodes was scaled back to 2, 4, 8, 16, 32, and 64. The hardware available for this ultimately ran into memory limitations for logging. Even with 128GB of memory it proved impractical to run anything more than 64 nodes.

To examine bq_{mult} the multiplier of T_{on} to determine the frequency for popping items off of the BQ the values $bq_{mult} = [2, 4, 8, 16]$ was done.

Table 6.4: Packet Time Tuning

Time Parameter	Values	Comments
T_{on}	30, 60, 120, 250, 500	Looks at effect of transmitting more or less often
T_r	1, 5	Looks at effect of larger delays in forwarding `EndReceipt` packets
T_p	10, 20	Looks at effect of larger delays in forwarding other packets

The ability to relay data over a long distance is worth investigating so a simulation where nodes are distributed in a line with equidistant spacing at 250km and 500km is performed. This would allow, measurements along a long drainage. For example, one could have instruments spread along the Coeur d' Alene river, St. Joe River, Lake Coeur d' Alene etc. all relaying to a single *gateway node*. Longer runs should also be theoretically feasible as well. Instrumenting the Nile, Amazon, Columbia, rivers could be possible although perhaps not practical.

Table 6.1 lists some of the modeling parameters where only a single value was used. Table 6.2 shows the various sizes of areas to be covered using a uniform random distribution of nodes. Table 6.3 shows the number of nodes to be generated in each

Table 6.5: Loss Modes

Mode	Comments
Ideal	Simplistic loss model includes some interference loss
Thorp	Distance based loss model, includes some interference loss
Random Noise	Noise inserted to validate SHA checksum

of the various areas. Table 6.1.3 shows the variations in timing parameters that will be modeled in an attempt to find optimal values. Table 6.5 shows the transmission loss models that will be simulated. Finally, Table 6.6 provides an overall summary of the simulations.

Multiple runs were done for each simulation configuration. A set of 20 runs was deemed sufficient to cover various conditions.

Table 6.6: Summary of Simulations. Noise is only tested using 10 nodes.

Size	Nodes				Parameters		Noise
	2,3,4,8	16	32	64	Ideal Thorp	T_m T_r T_p bq_{mult}	$p = 0.5$
100m Grid	✓	✓	✓	✓	✓	✓	
1km Grid	✓	✓	✓	✓	✓	✓	
10km Grid	✓	✓	✓	✓	✓	✓	✓
100km Grid	✓	✓	✓	✓	✓	✓	
1km Linear	✓	✓	[1]	[1]	✓	✓	
10km Linear	✓	✓	[1]	[1]	✓	✓	
100km Linear	✓	✓	[1]	[1]	✓	✓	
250km Linear			✓	✓	✓	✓	
500km Linear			✓	✓	✓	✓	

[1] Not run because of computational time constraints.

Chapter 7

Results

The simulations that were run produced a massive amount of data. The first full run of the simulations as outlined in Chapter 6 took about a week of compute time to produce the data and resulted in over 4000 separate simulations. In this chapter the results are presented. Several cross experiment analyses were done to help understand the results. Graphs of individual simulations were also produced. Example of these are shown in Figures 7.1, 7.2, 7.3, and 7.4 show a selection of these. While useful for tracking down specific issues, it is the overall collection and analysis of the data that these graphs represent, that is most useful.

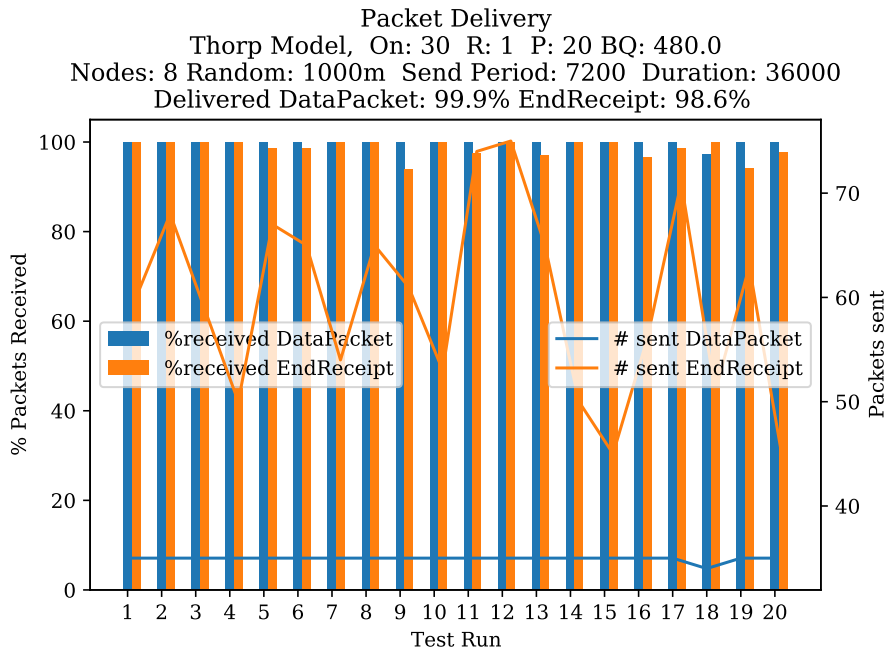

Figure 7.1: Typical simulation run output showing success of packet transmission and number of packets sent.

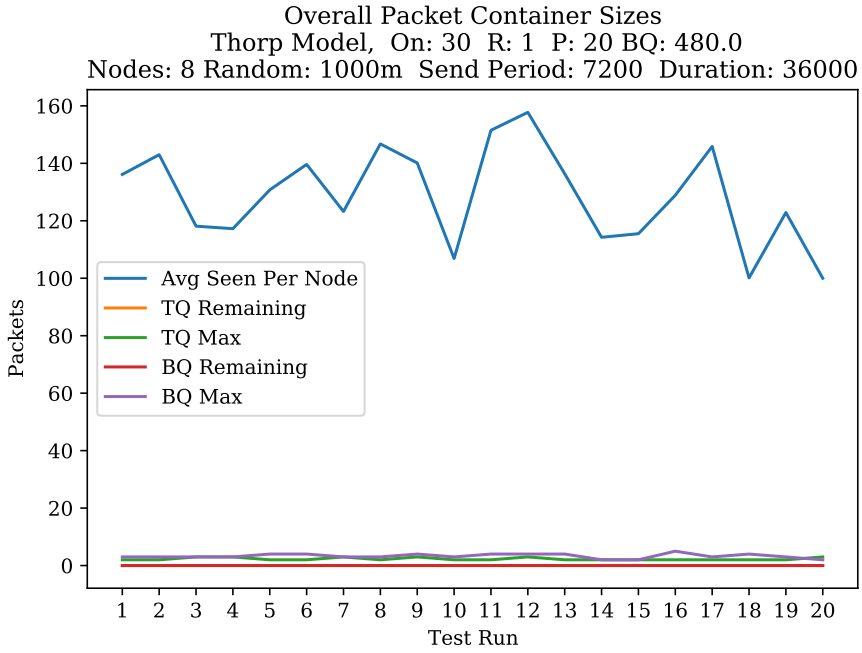

Figure 7.2: Typical simulation run output showing sizes of TQ, BQ, and VL.

7.1 Simulation Challenges

The simulations were run on a desktop server grade machine with a 16 core Xenon processor with 128GB of RAM along with a 1TB disk for storage running Ubuntu 20.04. While this isn't the most powerful of machines available, it does provide significant computational power. The memory requirements for this large of a simulation were significant and even with 128GB it was easy initially to run the system out of memory. Disk space became a concern as well as the amount of data logged for analysis was enormous. It would have been better to send the data directly to a database. The runs as outlined in Chapter 6 eventually took about 1 week of continuous computation to complete. A script was created using Python 3 to manage the execution and attempted to utilize as many cores as possible. An initial attempt to run simulations up to 128 nodes was attempted but memory and time constraints prevented this. A great deal of effort was expended to make the memory footprint smaller and to

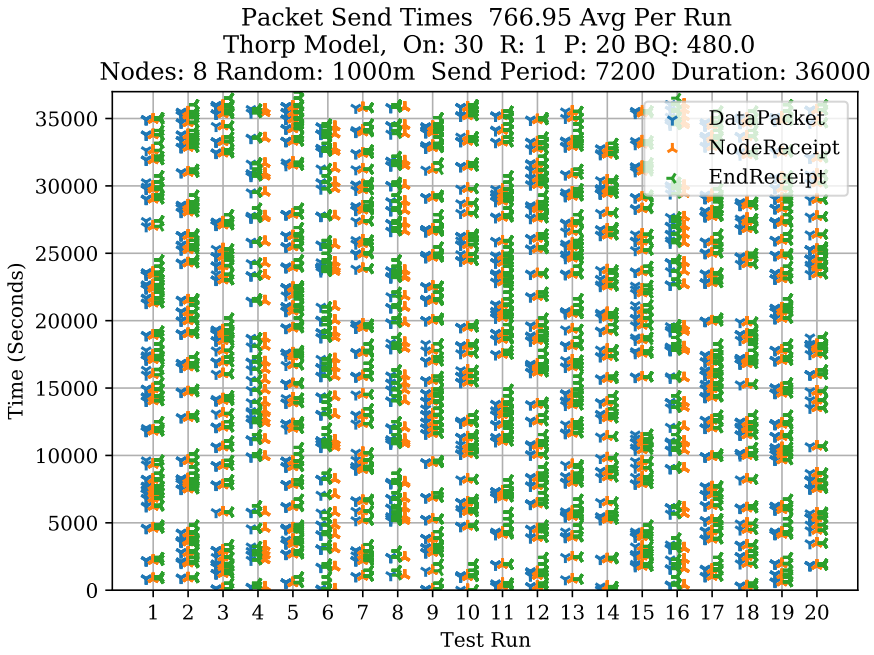

Figure 7.3: Typical simulation run output showing the time that various packets were sent.

optimize the performance of the simulation which is inherently $\mathcal{O}(n^2)$ on both space and time where n is number of nodes. Mining of the resulting data was also time consuming, especially the cross simulation analysis. Certain data was extracted from the text logs and placed into binary files using dill, which is a drop-in replacement for pickle, to work around limitations of the built-in pickle facility[14][15] so that generation of cross simulation graphs was faster. The primary motivation for using dill instead of pickle is that pickle is not capable of serializing moderately complex dictionaries.

7.2 Grid and Linear Models

Two categories of models were picked for simulation. The first was a grid, which is square and ranges of values in the X and Y and Z directions. The Z direction has a range of 25m. The 25m value was chosen to provide some vertical randomness within

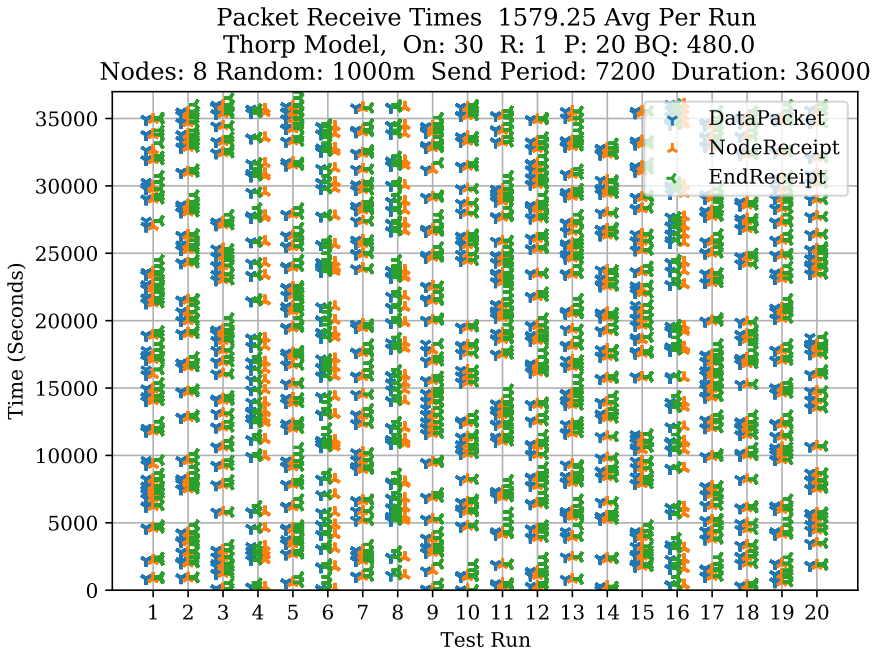

Figure 7.4: Typical simulation run output showing the time that various packets were received.

the range of heights of typical structures. It is an arbitrary value. The size X and Y are always equal. Nodes are picked using a uniform random distribution in X, Y and Z. A representative linear network is shown in Figure 4.2. A representative grid network is shown in Figure 4.3.

7.3 Packets In System

The ability to judge how well the protocol works in controlling the *broadcast storm* problem can be judged by examining the packets in the system. Each simulation run was done 20 times for a given set of input parameters. This section presents the results of the several cross simulation graphs produced.

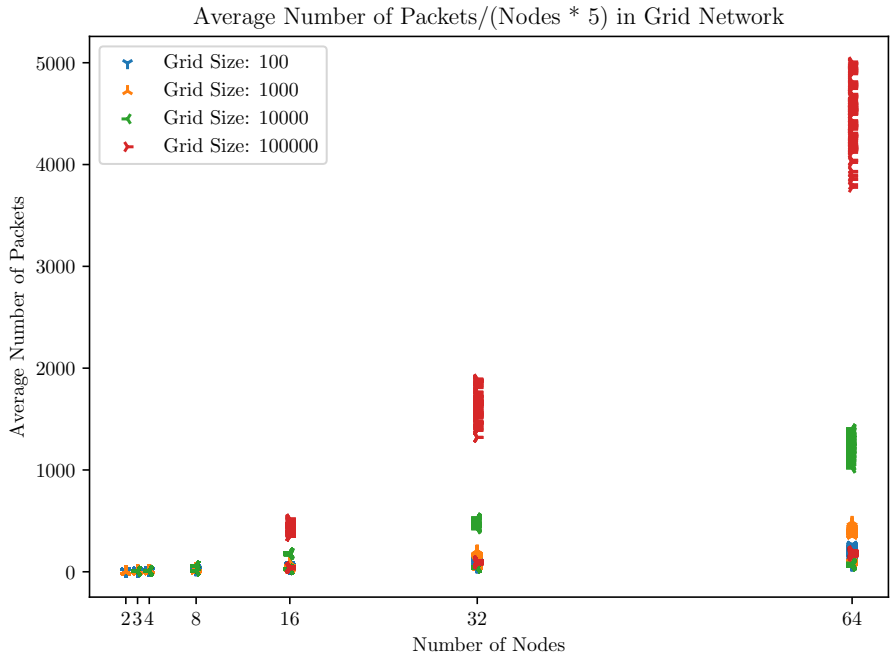

Figure 7.5: Scatter plot of packets in a Grid network. Random number $Seed_1$

7.3.1 Abbreviated Run

After generating the initial set of graphs it was noted that the graphs of several of the secondary parameter simulations did not generate obvious differences in patterns.

After some examination of the simulation software it was noted that for both the Thorp loss model and the Ideal loss model there are distance based effects. The scattering is assumed to be the result of both the modeling and geometry effects. Analysis of these effects in detail is beyond the scope of this thesis. It is important to understand that both the Thorp and Ideal models are an approximation of the actual hardware. They do provide a useful estimation of the hardware and within the limits of the accuracy of the model provide a sound basis for modeling the efficacy of the SBNPL. Considering Equation 4.10 there is an expectation that number of nodes and geometry would increase transit time and thus number of packets in the system.

To further eliminate questions about geometry effects based on the random num-

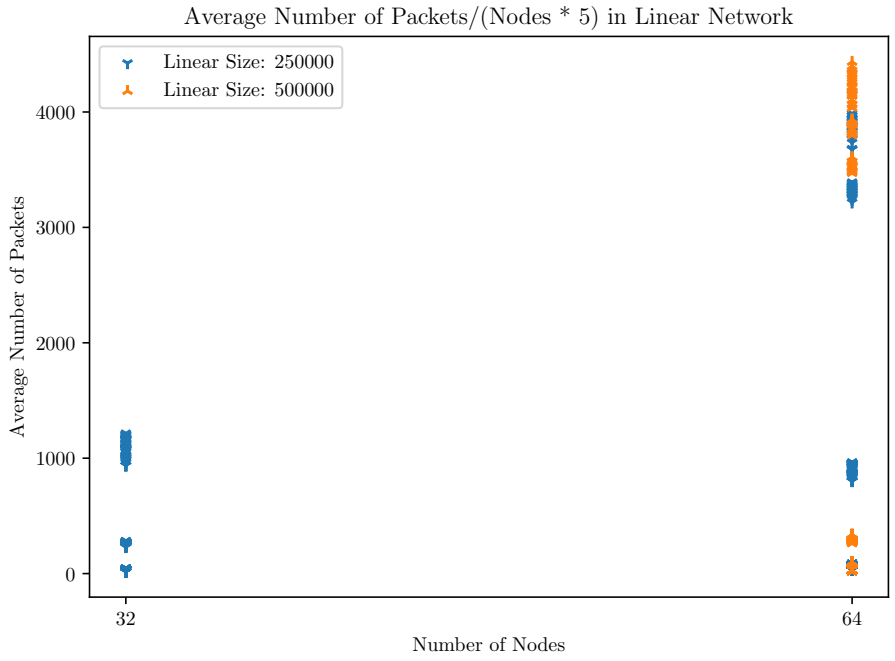

Figure 7.6: Scatter plot of packets in a Grid network. Random number $Seed_1$

ber seed $Seed_1 = 123456$ and potentially improve the results a second abbreviated, in compute time, run was done using the seed $Seed_2 = 4321$. Figures 7.13, 7.14, 7.15, 7.16, 7.17, 7.18, and 7.19 show these results.

Additionally, in the abbreviated simulation the linear case was extended down to 1km to provide a larger comparison of the difference in geometry between a rectangular and linear mesh.

7.3.2 Packets vs Nodes by Mesh Size

A calculation of the average number of packets in the system during the duration of the several runs was done. These averages were then extracted and graphed Figures 7.5 and 7.13 show the case of a grid network while Figures 7.6 and 7.14 show a linear network. These provide some insight into how the number of packets in a system correlates with the number of nodes, the size, and geometry of a system. The data was scaled by $1/(N*5)$ where N is number of packets and 5 is the number of recurring

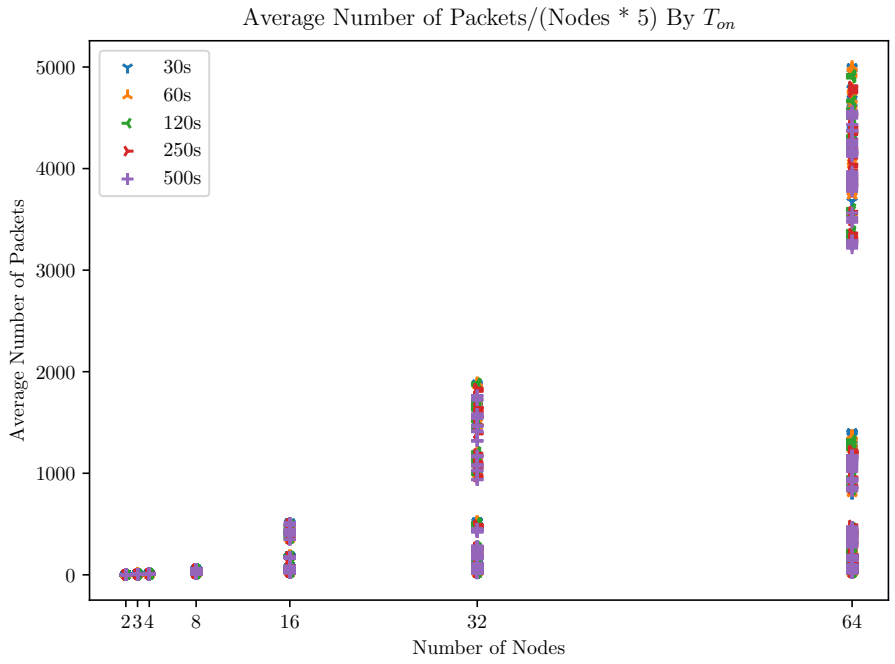

Figure 7.7: Scatter plot of packets broken out by T_{on}. Random number $Seed_1$

packets sent from each node.

The number of packets was normalized by number of nodes times 5. Five is the number of primary packets produced in each run. The normalization makes sense based on the Equation 4.9.

7.3.3 Variation of Parameters

Figures 7.7, 7.8, 7.9, 7.10, 7.11, 7.15, 7.16, 7.17, 7.18, and 7.19 show similar scatter plots broken out by the various parameters T_m, T_r, T_p, and bq_{mult} as described in Section 6.1.3.

It was expected that these parameters could have obvious effects on the number of packets in the system. The results indicate that the variations in these parameters produce very similar patterns as those seen in Figures 7.5, 7.6, 7.13, and 7.14. While these parameters deserve consideration it is fairly clear that number of nodes combined with mesh size appear to be the dominating factors.

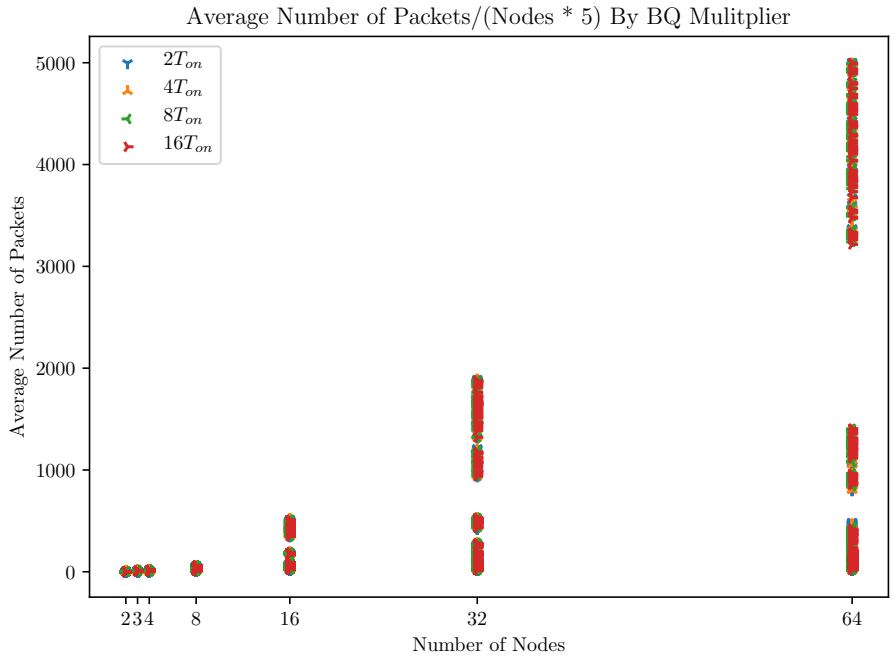

Figure 7.8: Scatter plot of packets broken out by bq_{mult}. Random number $Seed_1$

7.3.4 BQ, TQ, and VL Sizes

The sizes of BQ, TQ, and VL are theorized in Equations 4.12, 4.13, and 4.14. To validate these the graph in Figures 7.12 and 7.20 were produced. For the TQ and BQ the size remains bounded, on the order of 10 for the maximum number. It is also confirmed that the BQ is usually getting cleared out before the simulation finishes. As expected the VL can grow very large. In the simulation there was no size bound on the VL and `NodeReceipt` packets are counted (which don't need to be placed in the VL in an actual implementation). The simulation is reporting a somewhat higher number of packets than needed in a real implementation.

Of all the results in the simulation this one is very compelling. It very convincingly validates that long term use of the SBNPL will require fairly minimal on device storage making it viable for not only the reference ESP32 device but potentially for lower end devices with less memory. The Equations 4.12, 4.13, and 4.14 are thus validated and

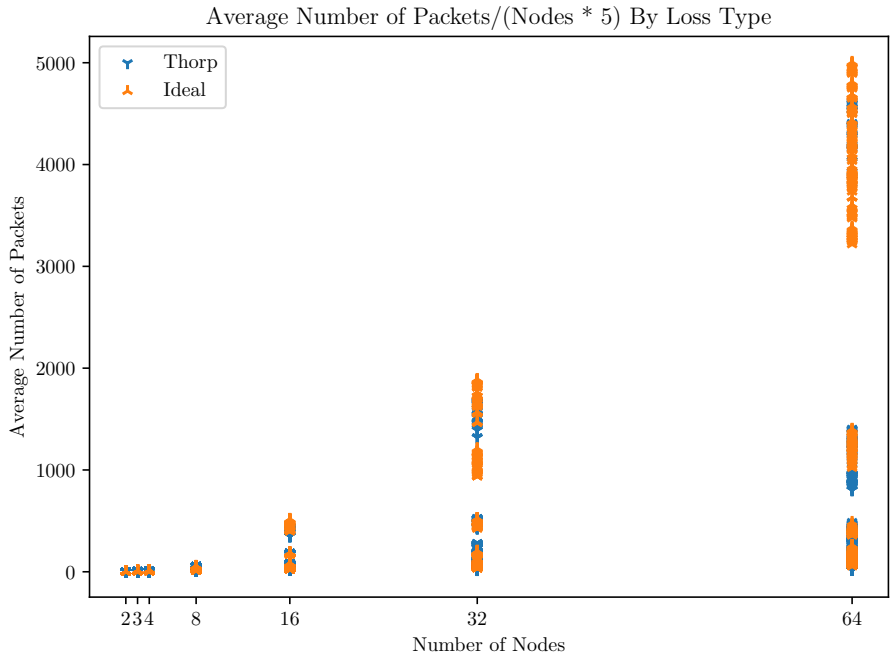

Figure 7.9: Scatter plot of packets broken out by loss model. Random number $Seed_1$

can be used to quantitatively estimate a lower bound on memory requirements.

7.3.5 Linear vs Grid

An intuition about the efficiency of the system is that a linear or near linear organization of nodes will be more efficient, resulting in fewer packets in the system.

Comparing Figures 7.5 and 7.6 it is observed that the linear case produces fewer packets as suggested by Equation 4.9. This is additionally supported by Figures 7.13 and 7.14 of the abbreviated run.

7.4 Round Trip Time

In order to understand the round trip time for a `DataPacket` and the return `EndReceipt` additional instrumentation was added to the abbreviated run. The result of this can be seen in Figure 7.21. A curve labeled NLM Based Round Trip was added

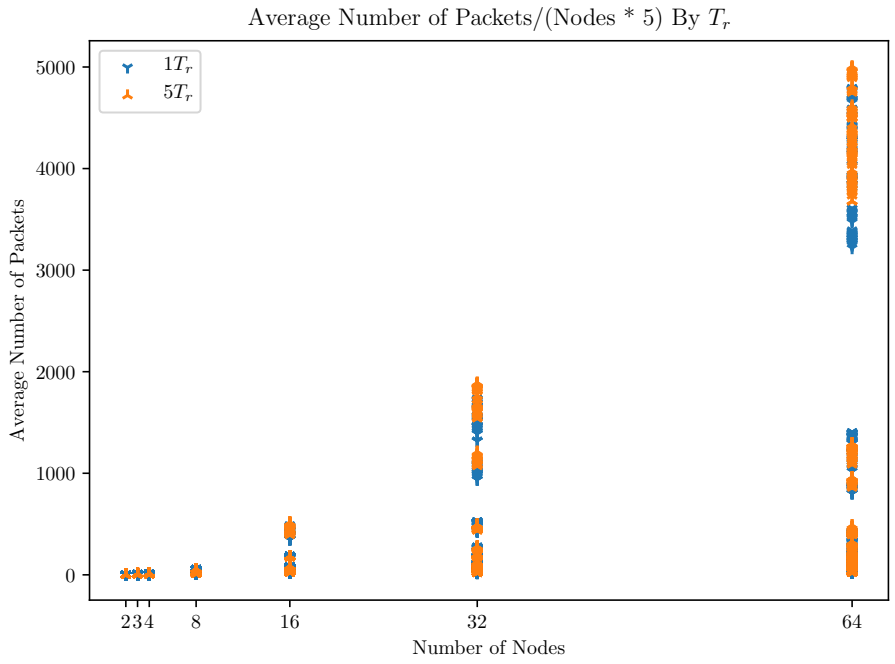

Figure 7.10: Scatter plot of packets broken out by T_r. Random number $Seed_1$

to represent Equation 4.9 assuming the best case. This curve represents using the NLM model of time which was not easily extricable from the source code and does not clearly match the Semtech time on air calculation shown in Section 4.1.1. The source code was not documented making reverse engineering the details time consuming and error prone. However, this appears, within the limits of how accurately the code was interpreted to validate 4.9. The NLM also used input parameters that were undocumented leading to input that cannot be satisfactorily reconciled with settings on Semtech hardware. The transmit time in NLM is calculated as (packet size in bytes * 8) / datarateBPS. In the simulations the datarateBPS is set to 300. This value was used to produce the "NLM Based Round Trip" curve.

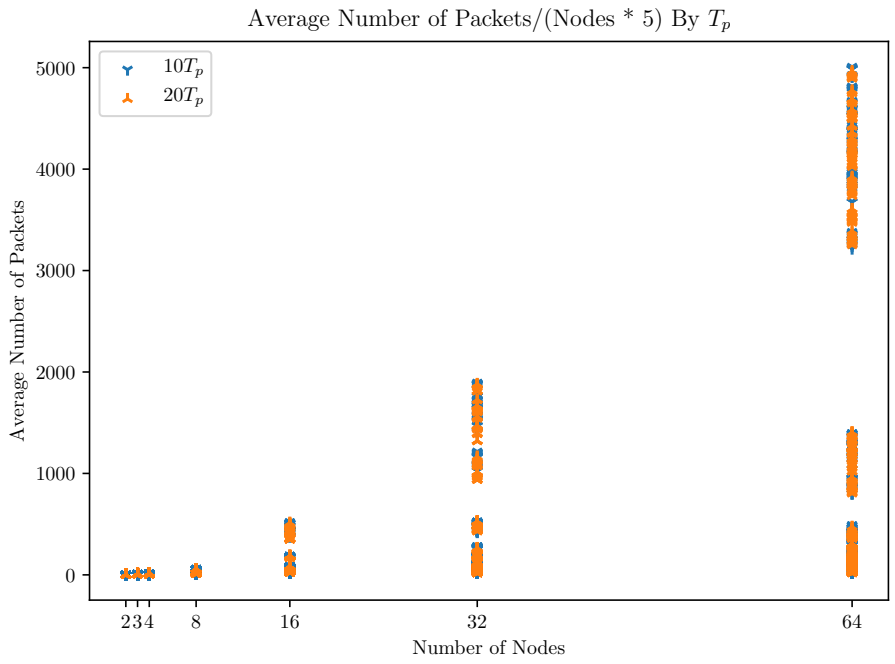

Figure 7.11: Scatter plot of packets broken out by T_p. Random number $Seed_1$

7.5 Reliability

The number of `DataPacket` and `EndReceipt` packets that made it to their destinations was tracked. Figure 7.22 shows the results for the abbreviated run. Because `DataPackets` are placed in the BQ and rebroadcast until a `EndReceipt` is received they enjoy a somewhat higher percentage of delivery. The primary goal of the SBNPL is delivery of data packets to the destination. The results show a high percentage of delivery for all variations of parameters explored. As the number of nodes increases the delivery rate tends to fall off but remains high.

7.6 Corruption Detection

A single run was done using 10 nodes with a probability of random corruption set as $p = 0.1$. Figure 7.23 shows the result of detection of corruption using a SHA-256 cryptographic hash. As can be seen it was simple to detect all the cases of intentional

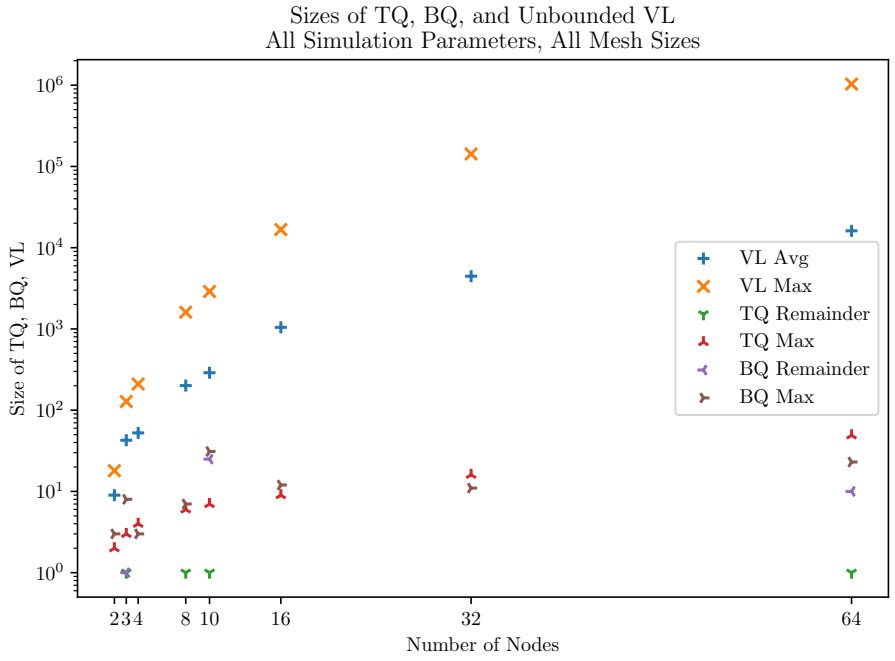

Figure 7.12: TQ, BQ and unbounded VL sizes. Random number $Seed_1$

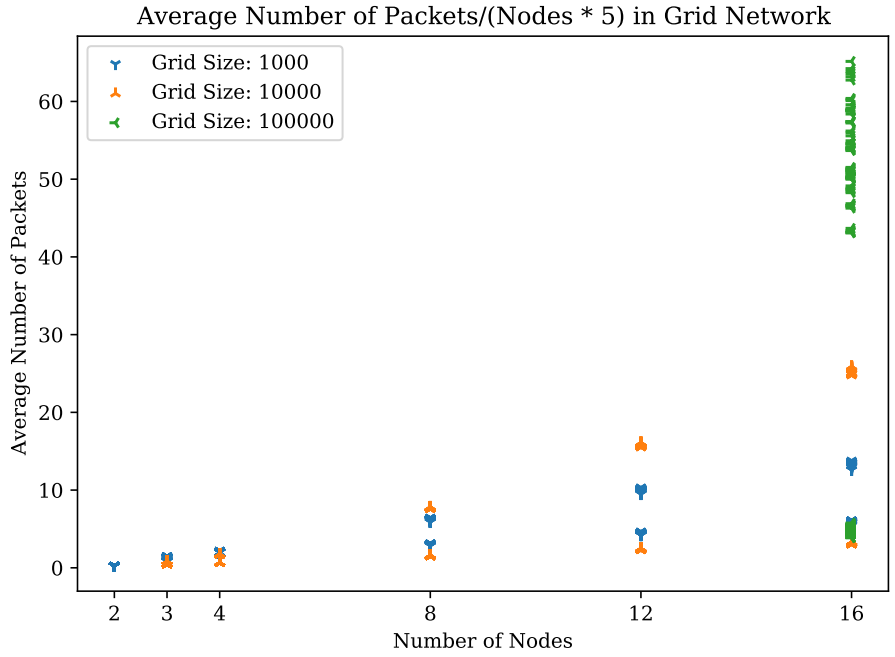

Figure 7.13: Scatter plot of packets in a Grid network. Random number $Seed_2$

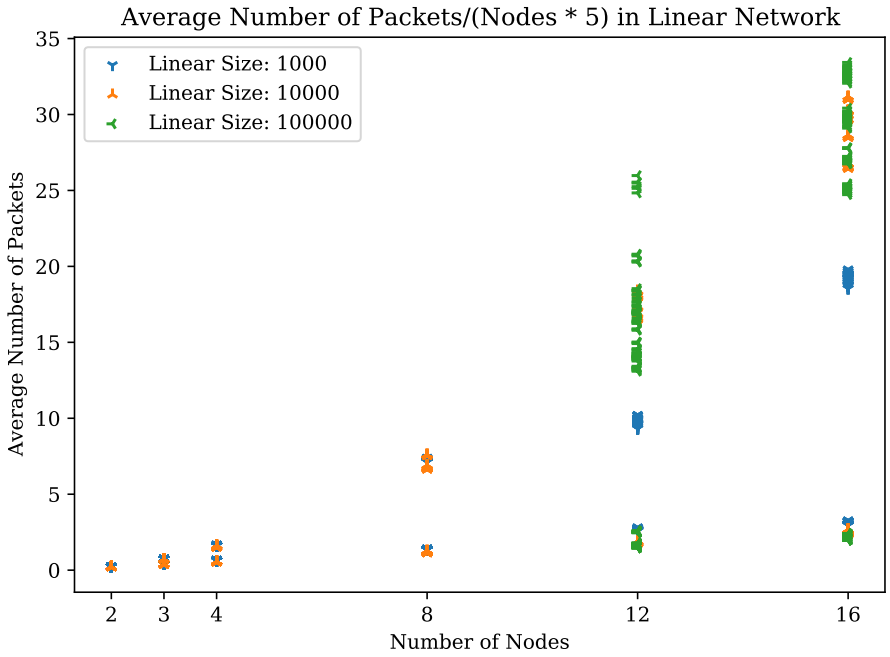

Figure 7.14: Scatter plot of packets in a Grid network. Random number $Seed_2$

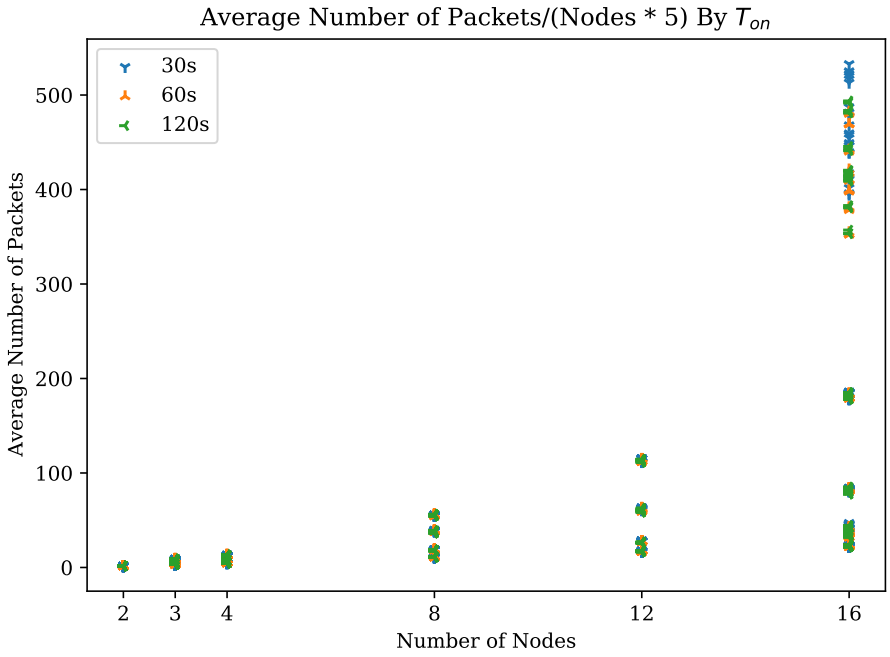

Figure 7.15: Scatter plot of packets broken out by T_{on}. Random number $Seed_2$

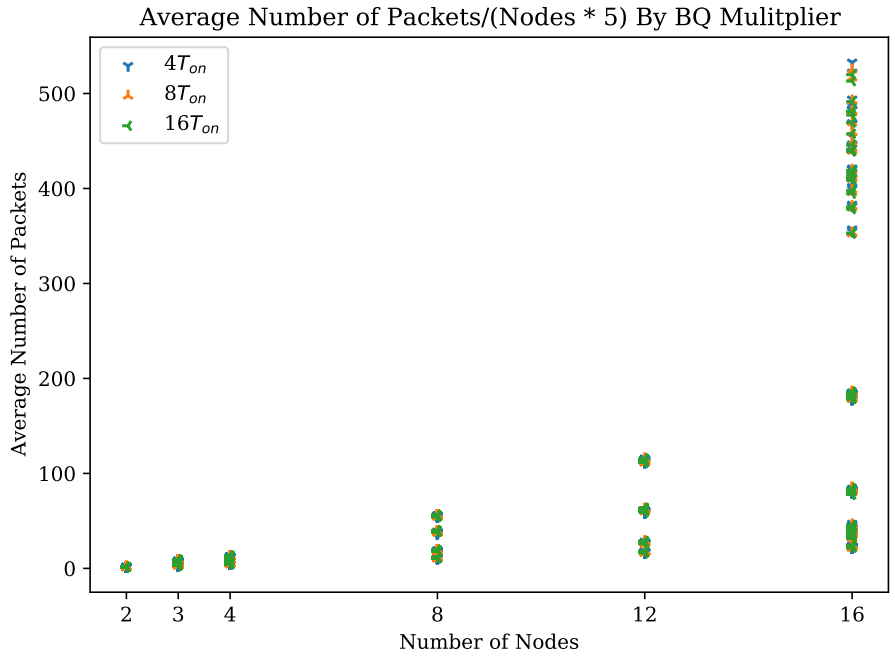

Figure 7.16: Scatter plot of packets broken out by bq_{mult}. Random number $Seed_2$

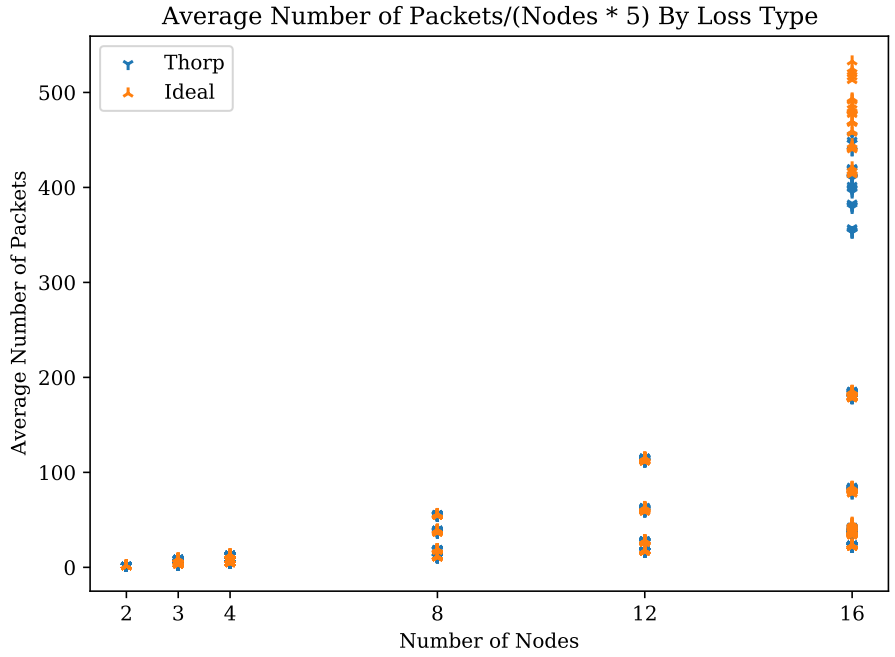

Figure 7.17: Scatter plot of packets broken out by loss model. Random number $Seed_2$

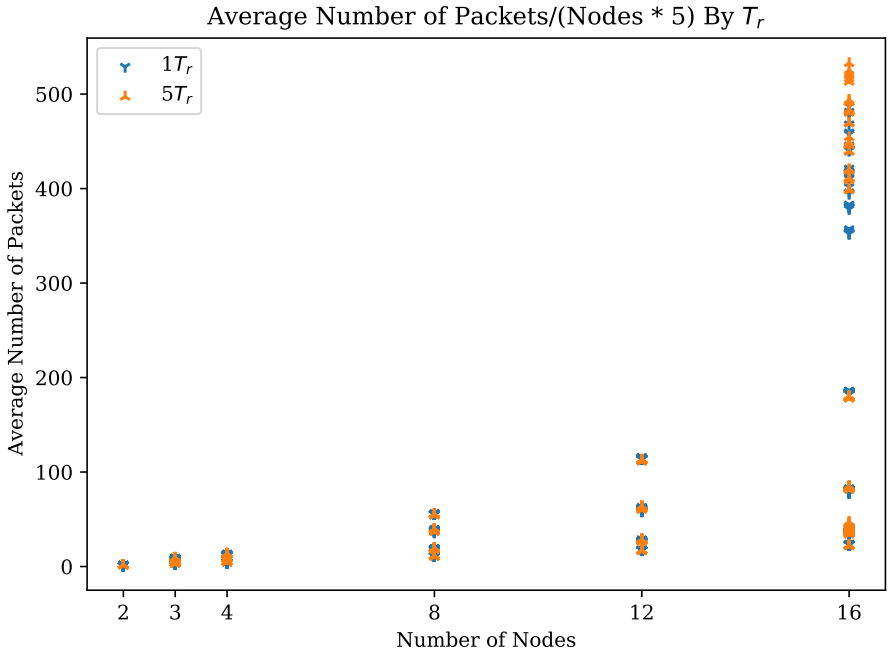

Figure 7.18: Scatter plot of packets broken out by T_r. Random number $Seed_2$

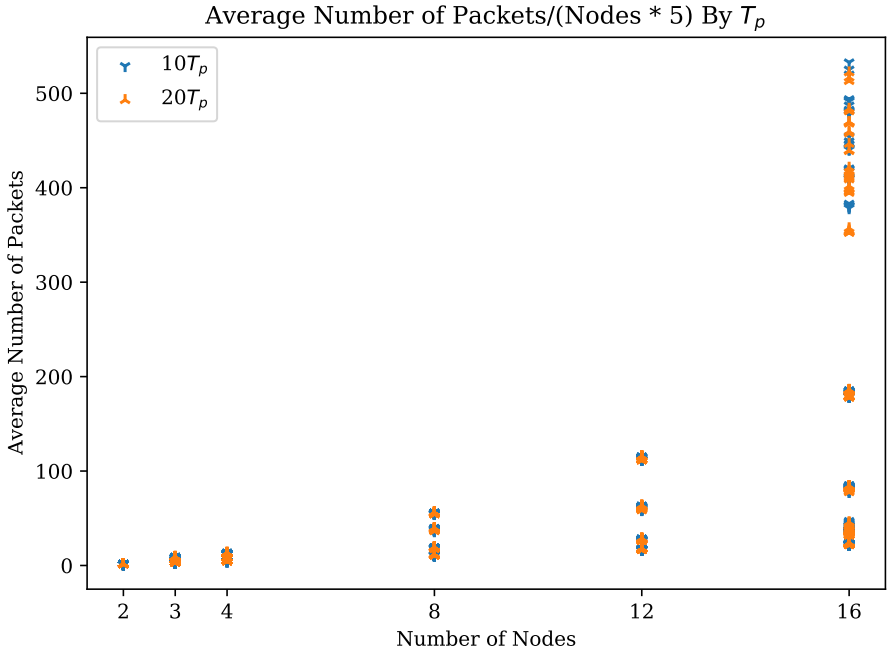

Figure 7.19: Scatter plot of packets broken out by T_p. Random number $Seed_2$

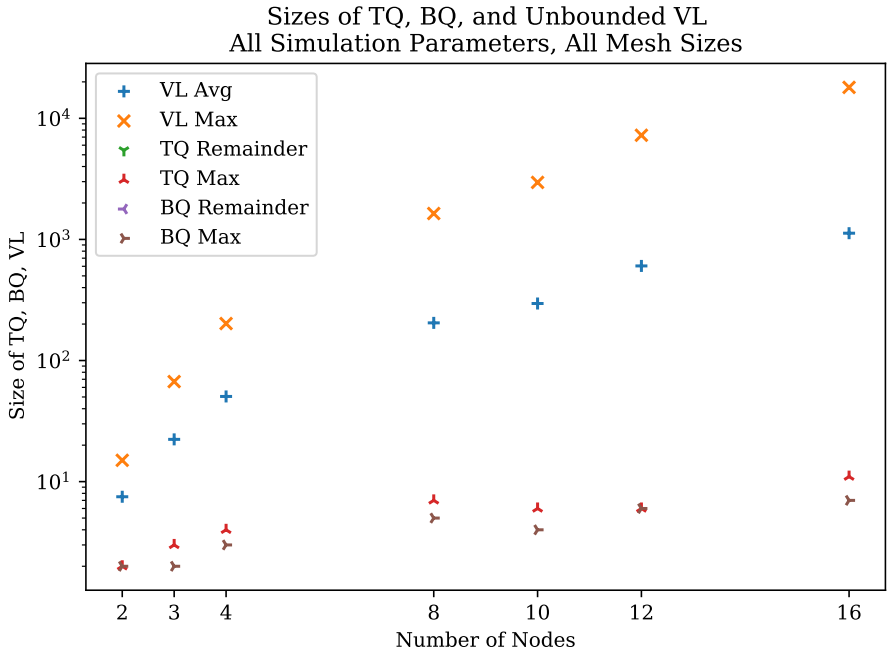

Figure 7.20: TQ, BQ and unbounded VL sizes. Random number $Seed_2$

corruption. Even though in this run 100% of the corrupt packets were detected, it is not expected that this would hold for higher corruption probabilities. When corruption exceeds a certain threshold it is assumed that transmission failure could be a result or that software detection would not work. There are many different scenarios that must be considered in the software for detecting corruption including packet truncation, packet header corruption, etc. It is even theoretically possible to corrupt the data and the cryptographic hash in such a way that the corruption is not detectable. This could be exploited to produce corrupted measurements. Data validation based on expected ranges should catch at least some of these cases. Detection of invalid measurements should be something that is not only viewed as a potential instrument failure, but also a potential cyberattack.

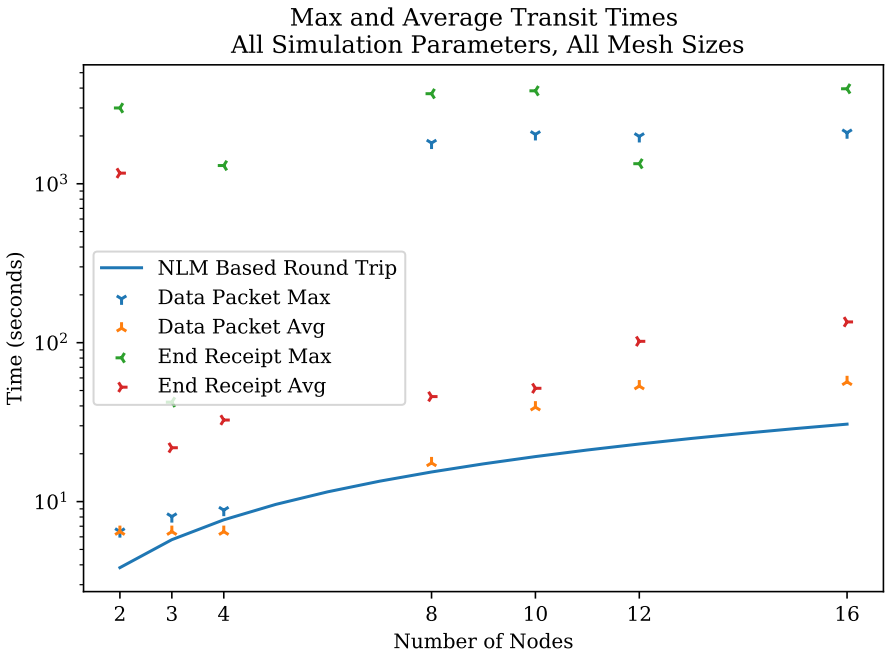

Figure 7.21: Round trip times for `DataPacket` and return `EndReceipt`. NLM Round Trip is a calculation of total round trip time, `DataPacket` and `EndReceipt` using Equation 4.9 and the NLM code calculation for transmit time. Random number $Seed_2$

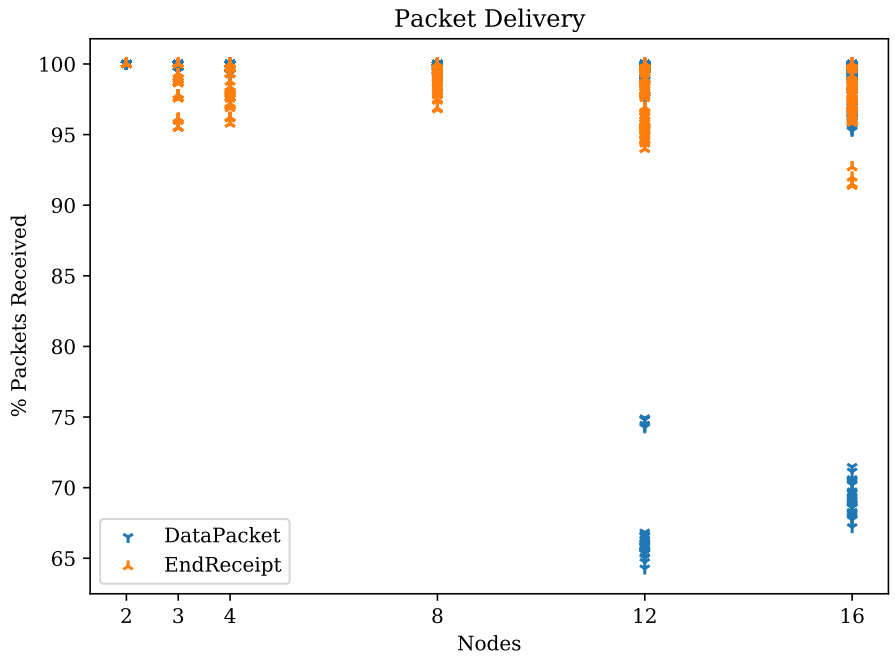

Figure 7.22: Percentage of `DataPacket` and return `EndReceipt` packets received. Random number $Seed_2$

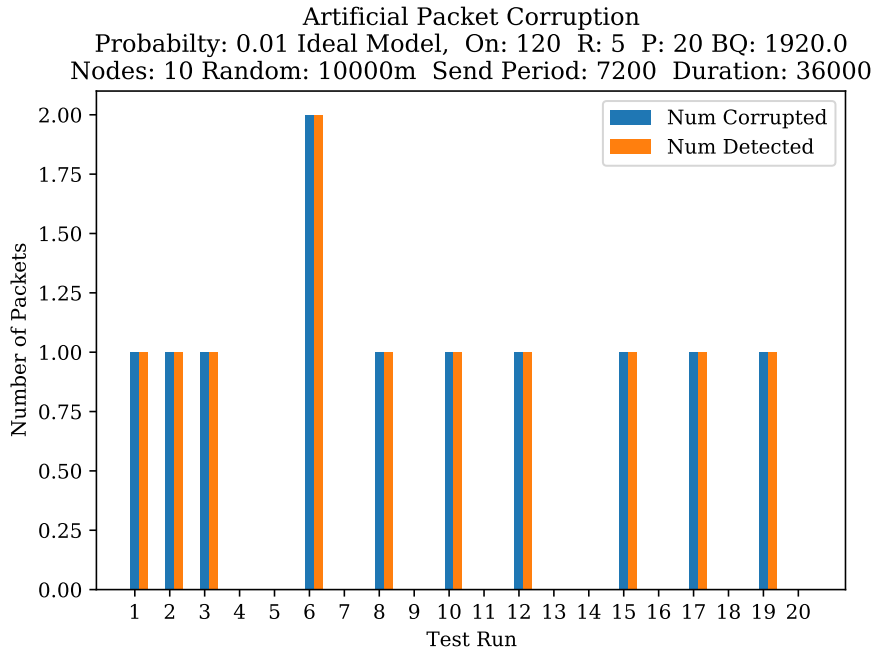

Figure 7.23: Detection of artificial corruption. Random number $Seed_2$

Chapter 8

Applications

Two of the primary benefits of this protocol are ease of implementation and deployment. It is inferred that a collection of devices using this protocol can be built with one or more set to connect to a WAN. Then it is simply a matter of distributing them, and turning them on. There are many potential applications to this protocol.

8.1 Environmental Sensor Networks

The motivating research for creating this protocol of building a sensor network for the purpose of monitoring water quality with an eye toward creating a tool for the management of water resources is an obvious application.

Other networks of environmental sensors are also an obvious use of this protocol. For example, a network of sensors collecting temperature, humidity, air quality, etc. could be deployed in a city or in any other area where the collection of environmental data would be beneficial. In areas that are subjected to periodic poor air quality a network of sensors could produce a map of poor air quality, allowing residents of the area with health issues to opt to temporarily relocate to areas of better air quality.

Another possibility is to predict the likelihood of flash floods. In certain parts of the world rain may fall in one area producing a flood in another where no rain has fallen. A network of rain gauge sensors could be created to predict flash flooding notifying residents ahead of the time, allowing them time to prepare or evacuate or perhaps trigger a flashing warning in areas where water is allowed to flow over a roadway.

Other options may include keeping track of open parking spaces in a lot or garage.

8.2 Other Possibilities

Other options are also envisioned. For example a network of sensors that check continuity on a livestock electric fence could potentially be deployed. In some areas fences can span multiple miles. Knowing there is a break and approximately where it is located could prevent livestock from escaping saving time in tracking them down.

Another option is tracking wildlife. Machine vision techniques have been employed to track wildlife by species and individuals. This is a less intrusive method than traditional tagging and radio collar methods. Assuming the power requirements for doing the vision recognition can be met, transmitting information throughout the network could be accomplished using this protocol, assuming that the amount of data is reasonable [18].

The ability to lose connectivity for some period of time and then connect again offers the possibility of mobile data collection units. A mobile unit may need to create a somewhat different BQ strategy than what is outlined and modeled, but there is no conceptual issue preventing caching data for long periods. This could allow things such as mobile air, water (both surface and subsurface), and land based systems.

Chapter 9

Future Work

There are a number of possibilities that could be examined to expand on this simulation.

9.1 Real World Usage

This protocol while simulated has yet to be implemented on real hardware. Doing so with a reasonable sized network of from 3-10 devices as a first pass validation of the protocol would be an obvious first step.

Expanding to more extensive networks and pushing the limits of long range communication, perhaps along a river or a large body of water would be a natural progression.

9.2 Simulations

In the simulation that was run only the LoRa modem was used. It would be reasonable to run simulations using the FSK/OOK modem as a comparison and/or use the FSK/OOK modem in real devices.

Usage with other RF technology would be useful. Conceptually the protocol is valid for any communication system. The benefits of usage in other types of RF communications could be explored.

For a linear or near linear network, for example along a river, some loss is expected and was seen in my simulations. Extra nodes, perhaps in a staggered configuration could be simulated to see what improvement could be made in packet delivery.

Additional examination of the protocol, focusing on parameter tuning may yield

improved performance. A more detailed analysis of geometry effects would also be useful. For example star, concentric, and regular grids could potentially be analyzed. These may suggest improvements to the SBNPL.

9.3 Comparison to ENDCAST

ENDCAST[27] was an inspiration for SBNPL. A cursory comparison of ENDCAST and SBNPL suggests that the performance of both may be comparable. In terms of reliability SBNPL would seem better owing to the BQ. It would be interesting to do a direct comparison of the two with and without the BQ, which could easily be added to ENDCAST.

Chapter 10

Conclusion

The new method SBNPL has been shown to be straight forward to implement, reliable, and secure-able on a simulated LoRa IoT devices. A mathematical basis for throughput, number of packets in the system and storage requirements has been derived and presented and a simulation was done with results that show SBNPL works well under the real world oriented simulation of noise and distance.

The modeling of this protocol was challenging, partly because of a lack of deep understanding of networking hardware and protocols, partly due to unanticipated software and hardware limitations.

The protocol appears to work very well and should deliver on ease of implementation and real world ease of deployment. As such it is worth considering for at least the case of creating a low cost LoRa IoT measurement network.

Sizes of the TQ, BQ, and VL required by SBNPL to get nearly 100% primary packet delivery are small enough to be implemented on typical IoT microprocessors making this protocol a viable choice for creating a simple broadcast mesh network on inexpensive hardware.

The time required to transmit a packet through a mesh of modest size (less than 32 nodes) it relatively small. This makes it viable for small networks. This appears to fall within the guidelines of The Things Network[30] for small amounts of data.

The ability to lose connectivity for some period of time and then connect again offers the possibility of mobile data collection units. A mobile unit may need to create a somewhat different BQ strategy than what is outlined and modeled, but there is no conceptual issue with holding data for long periods. This could allow things such as mobile air, water (both surface and subsurface), and land based systems.

The possibility of using SBNPL in remote regions where regulatory restrictions may not need to be considered suggests possibilities for use in very large sensor networks.

In more urban areas smaller networks that can connect to a Wi-Fi Internet hot spot could be dispersed over large areas as well.

While SBNPL may not be the perfect solution for all IoT broadcast networking needs, it provides an attractive solution to ease of implementation, deployment, and the potential for securely transmitting sensor data to the cloud.

References

[1] Augustin, A., Yi, J., Clausen, T., and Townsley, W. A study of lora: Long range & low power networks for the internet of things. *Sensors 16*, 9 (Sep 2016), 1466.

[2] Bor, M., Vidler, J., and Roedig, U. Lora for the internet of things.

[3] Dias, J., and Grilo, A. Multi-hop lorawan uplink extension: specification and prototype implementation. *Journal of Ambient Intelligence and Humanized Computing* (Feb 2019).

[4] Durresi, A., Paruchuri, V., Iyengar, S., and Kannan, R. Optimized broadcast protocol for sensor networks. *IEEE Transactions on Computers 54*, 8 (2005), 1013–1024.

[5] esp32 series datasheet. `https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf`.

[6] FreeRTOS. `https://aws.amazon.com/freertos/`, 2012.

[7] Gao, J., Guibas, L., Milosavljevic, N., and Hershberger, J. Sparse data aggregation in sensor networks. *2007 6th International Symposium on Information Processing in Sensor Networks* (2007).

[8] Jones, C., Bonsignour, O., and Arroyo, T. *The economics of software quality*. Addison-Wesley, 2012.

[9] Kleinrock, L., and Tobagi, F. Packet switching in radio channels: Part i–carrier sense multiple-access modes and their throughput-delay characteristics. *IEEE Transactions on Communications 23*, 12 (1975), 1400–1416.

[10] LimitedResults. Pwn mbedtls on esp32: Dfa warm-up, Jun 2019.

[11] Linnartz, J.-P. The aloha protocol. http://www.wirelesscommunication.nl/reference/chaptr06/aloha/aloha.htm.

[12] Lobo, S. Espressif iot devices susceptible to wifi vulnerabilities, Sep 2019.

[13] LoRaWAN® specification v1.0.3: LoRa alliance®.

[14] McKerns, M., and Aivazis, M. pathos: a framework for heterogeneous computing. http://trac.mystic.cacr.caltech.edu/project/pathos, 2010.

[15] Mckerns, M., Strand, L., Sullivan, T., Fang, A., and Aivazis, M. Building a framework for predictive science. *Proceedings of the 10th Python in Science Conference* (2011).

[16] Miller, C., and Valasek, C. Remote exploitation of an unaltered passenger vehicle, Aug 2015.

[17] Ni, S.-Y., Tseng, Y.-C., Chen, Y.-S., and Sheu, J.-P. The broadcast storm problem in a mobile ad hoc network. *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking - MobiCom 99* (1999).

[18] Norouzzadeh, M. S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M. S., Packer, C., and Clune, J. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences 115*, 25 (2018).

[19] NS-3 Consortium. The University of Washington NS-3 Consortium. https://www.nsnam.org/, 2020.

[20] Paruchuri, V., and Durresi, A. Stateless adaptive reliable broadcast protocol for heterogeneous wireless networks. *2015 IEEE 29th International Conference on Advanced Information Networking and Applications* (2015).

[21] Peng, W., and Lu, X. Ahbp: An efficient broadcast protocol for mobile ad hoc networks. *Journal of Computer Science and Technology 16*, 2 (2001), 114–125.

[22] R. Fielding, J. R. Hypertext transfer protocol (http/1.1): Message syntax and routing rfc(7230). https://tools.ietf.org/html/rfc7230.

[23] Russell, B., and Van Duren, D. *Practical Internet of Things Security*. Packt Publishing, 2018.

[24] SAFERTOS. https://www.highintegritysystems.com/safertos/, Jun 2019.

[25] What is LoRa®? https://www.semtech.com/lora/what-is-lora.

[26] Lora sx1276/77/78/79 datasheet, rev. 6. https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001OKs/Bs97dmPXeatnbdoJNVMIDaKDlQz8q1N_gxDcgqi7g2o, Jan 2019.

[27] Sooriyaarachchi, S., and Gamage, C. Endcast: mobile stateless data delivery in manets. *EURASIP Journal on Wireless Communications and Networking 2017*, 1 (2017).

[28] Sooriyaarachchi, S. J., and Gamage, C. D. A cell biology inspired model for managing packet broadcasts in mobile ad-hoc networks. *2015 17th International Conference on Advanced Communication Technology (ICACT)* (2015), 716–721.

[29] Stallings, W., and Brown, L. *Computer Security Principles and Practice*. Pearson Education, Inc., 2017.

[30] The things network duty cycle. https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html, Jul 2020.

[31] To, T.-H., and Duda, A. Simulation of lora in ns-3: Improving lora performance with csma. *2018 IEEE International Conference on Communications (ICC)* (2018).

[32] Tseng, Y.-C., Ni, S.-Y., and Shih, E.-Y. Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network. *IEEE Transactions on Computers 52*, 5 (2003), 545–557.

[33] Valerio, P. Top wireless standards for iot devices. `https://iot.eetimes.com/top-wireless-standards-for-iot-devices/`, Nov 2019.

[34] Vijayakumar, K. P., Ganeshkumar2, P., and Anandaraj3, M. Review on routing algorithms in wireless mesh networks. *International Journal of Computer Science and Telecommunications 3*, 5 (May 2012), 87–92.

[35] Vázquez, T., a Muñoz, S., Bellalta, B., and Bel, A. Hare: Supporting efficient uplink multi-hop communications in self-organizing lpwans. *Sensors 18*, 2 (Mar 2018), 1–29.

[36] Yu, Y., Li, Y., Tian, J., and Liu, J. Blockchain-based solutions to security and privacy issues in the internet of things. *IEEE Wireless Communications 25*, 6 (2018), 12–18.