The design of a communication protocol for a reconfigurable wireless animal tracking mesh network

C.J. van der Spuy

22117156

Dissertation submitted in fulfilment of the requirements for the degree *Magister* in *Computer and Electronic Engineering* at the Potchefstroom Campus of the North-West University

---

Supervisor: Prof. A.S.J. Helberg

November 2015

Declaration

I, C.J. van der Spuy hereby declare that the dissertation entitled "The design of a communication protocol for a reconfigurable wireless animal tracking mesh network" is my own original work and has not already been submitted to any other university or institution for examination.

C.J. van der Spuy

Student number: 22117156

Submitted on the 25th day of November 2015 at Potchefstroom.

Acknowledgements

"The fear of the Lord is the beginning of wisdom, and knowledge of the Holy One is understanding" -Proverbs 9:10 NIV
All honour to God, for He is the way, the truth and the life.

Prof. Albert Helberg, my study leader for his guidance, support and continuous assistance.

Dr. Melvin Ferreira for his objective view and willingness to help.

Financial support of National Research Foundation (NRF), Technology and Human Resource for Industry Programme (THRIP) and the Telkom Centre of Excellence (CoE) at the North-West University.

The TeleNet research group members, for their friendship, assistance and sometimes the needed distraction.

My family and friends for their inputs, support and prayer.

Special thanks to my parents, Charles and Sandra van der Spuy. You have carried me here, and supported me into a better life.

Elizabeth, being the best wife in the world. Helping me, when I felt helpless; giving me hope, when all hope was lost; and loving me beyond reason.

Abstract

Animal poaching, especially rhino poaching has become a crisis in South Africa. As part of the initiatives to combat rhino poaching, rhinos are tracked using electronic equipment such as sensor devices. Sensors that are used to track wildlife, e.g. rhinos, are expensive to attach and replace. For this reason, it is desirable to replace the sensors as seldom as possible. The sensors are energy efficient, limiting the transmission power and reducing the connection range. Our project partner, YRless international working in the field of animal tracking, requires a system that can overcome these constraints.

Due to the recent advances in computing, communication, sensing and miniaturisation of devices, the Unmanned Aerial Vehicle (UAV) environment has been receiving significant attention. A sensor network implementing UAVs can be utilised to perform any power draining operations and extend the connection range.

The focus of this research is the design of a communication protocol that operates in the UAV system.

The protocol is designed by combining well-known protocols (Ad Hoc On Demand Distance Vector (AODV) and Dynamic Source Routing (DSR)) and network architectures (Delay-Tolerant Networks (DTNs)). The protocol is then formally specified. As verification and validation, the identified protocol components' (includes network functions such as *routing update*, *landing update*, *data send* and *position update*) functionalities are tested using a Java simulation.

Further, the performance of the protocol is tested using a Monte Carlo simulation. A test network is simulated with induced transmission and node failures. The performance regarding the delay and reliability are recorded. As an elaboration of the performance tests, the influence of collisions in the network is also tested.

As a result, we see that node failures have a more detrimental effect on the performance of the protocol than transmission failures. The *data send* network function is the most reliable network function that verifies our design choices.

The functional and performance tests demonstrate that the protocol satisfies the initial

requirements set by the industry partner.

Keywords: *Animal Tracking, Communication Protocol, Rhino Poaching, UAV, Wireless Mesh Networks*

Opsomming

Stropery, veral renosterstropery, het 'n krisispunt in Suid-Afrika bereik. Elektroniese toerusting, byvoorbeeld sensor toestelle, vorm deel van verskeie inisiatiewe om renosterstropery te bestry deur wild, soos renosters, op te spoor en te moniteer. Bogenoemde sensors is egter duur om aan wild aan te heg en om te vervang. Om hierdie rede, is dit wenslik om die vervanging van sensors tot 'n minimum te beperk. Die energie-doeltreffende sensors, het 'n beperkte versendings drywing wat die konneksie afstand verkort. Ons projek vennoot, YRless international, wat werksaam is met die opsporing van diere, benodig 'n stelsel wat hierdie beperkings kan oorkom.

Onlangse vooruitgang in rekenaarverwerkingstegnologie, kommunikasiestelsels, opsporingstoestelle en miniaturisasie van toestelle verseker dat Onbemande Lugtuig (OLT) navorsing toenemende aandag geniet. 'n Sensor netwerk wat van OLT's gebruik maak kan aangewend word om energie dreinerende funksies uit te voer en om konneksie afstande te verleng.

Die fokus van hierdie navorsingsprojek is die ontwerp van 'n kommunikasie protokol wat werksaam is binne die OLT sisteem.

Die protokol is ontwerp deur die kombinering van bekende protokolle (*AODV* en *DSR*) en netwerk argitekture (*DTNs*). Die protokol word ook formeel gespesifiseer. Die geïdentifiseerde protokol komponente (insluitend netwerk funksies soos *routing update*, *landing update*, *data send* en *position update*) se funksionaliteite word getoets deur gebruik te maak van 'n Java simulasie.

Die prestasie van die protokol word verder getoets met behulp van 'n Monte Carlo simulasie. 'n Toets netwerk met geïnduseerde versending en nodus mislukkings is geïmplementeer. Die prestasie ten opsigte van vertraging en betroubaarheid is aangeteken. Hoe boodskap botsings die netwerk affekteer vorm ook deel van die prestasie toetsings.

Gevolglik sien ons dat node mislukkings 'n meer skadelike uitwerking op die prestasie van die protokol het. Die *data send* funksie is die mees betroubare netwerk funksie wat ook as verifikasie van die ontwerpskeuses dien.

Die funksionele en prestasie toetse toon dat die protokol voldoen aan die vereistes opgestel deur die projek vennoot.

Sleutelterme: *Dier Sporing, Draadlose Ineengeskakelde Netwerk, Kommunikasie-protokol, Onbemande Lugtuig, Renosterstropery*

Contents

List of Figures

List of Tables

List of Acronyms

ABSOLUTE Aerial Base Stations with Opportunistic Links for Unexpected & Temporary Events

AODV Ad Hoc On Demand Distance Vector

CERATIN Communication protocol for a Reconfigurable Animal Tracking Network

CoE Centre of Excellence

CRC Cyclic Redundancy Check

DLC Data Link Control

DSDV Destination-Sequenced Distance-Vector Routing

DSR Dynamic Source Routing

DTN Delay-Tolerant Network

FIFO First In First Out

FSM Finite State Machine

GPS Global Positioning System

GUI Graphical User Interface

ISO International Organisation for Standardisation

MANET Mobile Ad Hoc Network

NRF National Research Foundation

OSI Open System Interconnection

PDU Protocol Data Unit

RF Radio Frequency

THRIP Technology and Human Resource for Industry Programme

TTL Time To Live

UAV Unmanned Aerial Vehicle

WPAN Wireless Personal Area Network

List of Symbols & Subscripts

List of Symbols

d	Sight distance
h	Antenna altitude
s	Distance

List of Subscripts

m	Metres
km	Kilometres
mi	Miles
ft	Foot

Chapter 1

Introduction

In this research a door is opened into the tracking of wildlife using autonomous Unmanned Aerial Vehicles (UAVs). There are many possibilities and variations, here one system is proposed as a starting point for other research. The system originated from the rhino poaching problem but has many aspects that can be used in other energy-constrained networks.
In this chapter background regarding the rhino poaching problem is given. A system as a solution is then proposed. From the system, the research problem is stipulated. The assumptions and project scope is presented to keep the research focussed. Lastly, the methodology is given in the form of a chapter overview.

1.1 Introduction

First the origin of the research is presented. Here the rhino poaching problem is discussed.

1.1.1 The Rhino Poaching Problem

Animal poaching, especially rhino poaching has become a crisis in South Africa. From 2008, numbers of poached rhinos have increased excessively. The table below shows the increase in the number of rhinos poached each year in South Africa [12].

Table 1.1: Rhinos poached each year in South Africa

Year	2008	2009	2010	2011	2012	2013	2014	2015*
Quantity	83	122	333	448	668	1004	1215	987

From Table 1.1 we see that the rhino poaching numbers still increases although many anti-poaching campaigns are launched. Rhinos are mainly poached for their horns. Rhino horns are used as traditional Chinese medicine in countries such as China, Taiwan, South Korea and Vietnam [13]. Consumers in Asia are willing to pay extremely high prices for rhino horns; therefore more and more rhinos will be poached even though international trading is illegal. Braam Malherbe a veteran eco-warrior, extreme conservationist and 50/50 television presenter has proposed potential solutions to combat rhino poaching [14]. These solutions include:

Safe rhino dehorning: Remove the rhinos' horns safely without threatening their lives. The poachers will not go through the trouble of poaching rhinos if the rhinos do not have horns.

Educating the rhino horn consumers: Rhino horn has no proven medical value, and if the consumers knew this they would not bother paying such large amounts.

Rhino horn poisoning: Non-lethal poison is injected into the horn of the rhino. When humans ingest the poisoned horn, it causes stomach problems and severe headaches.

Legalise international trade of the rhino horn: At this moment rhinos are more worth dead than alive. Private rhino owners have to pay lots of money to keep the rhinos alive, by protecting them from poachers. If the owners were able to dehorn rhinos and legally

*Unofficial 31 October

sell the horns, money would be available for protection and conservation.

Taking the fight to the poachers: If the chance that the poacher will get caught increases fewer poachers will attempt to acquire rhino horns. Strategies such as game reserve patrols and animal tracking systems can be implemented to catch poachers.

In this research, we will focus on the tracking and monitoring of rhinos in near real-time using electronic equipment.

1.1.2 The UAV Environment

UAVs have been present for many years [15]. It has been due to recent miniaturisation of computer and navigation devices that research in the UAV environment received significant attention. Technical research and development include ABSOLUTE, ANCHORS, AVIGLE and projects run by Facebook, Amazon and Google (detail discussion in section 1.2). The UAVs are getting smaller and smaller; having more complex systems with higher capabilities. We will assume these capabilities and use the information around the UAV environment in this research.

1.1.3 YRless

YRless International [16] uses strap-on sensors to track farm stock and wildlife. Their current system operates as follows (portrayed by figure 1.1):

Sensors attached to animals send animal behaviour updates to the base station. The base station determines if the farmer or ranger should be notified. Other updates such as position monitoring data are also sent to the servers. The connections between the sensors and the base station are achieved using Radio Frequency (RF) transceivers and the connection from the base station to the notification device and servers are via a cellular network.
The attached sensors are usually around the animals' necks. This is a problem when

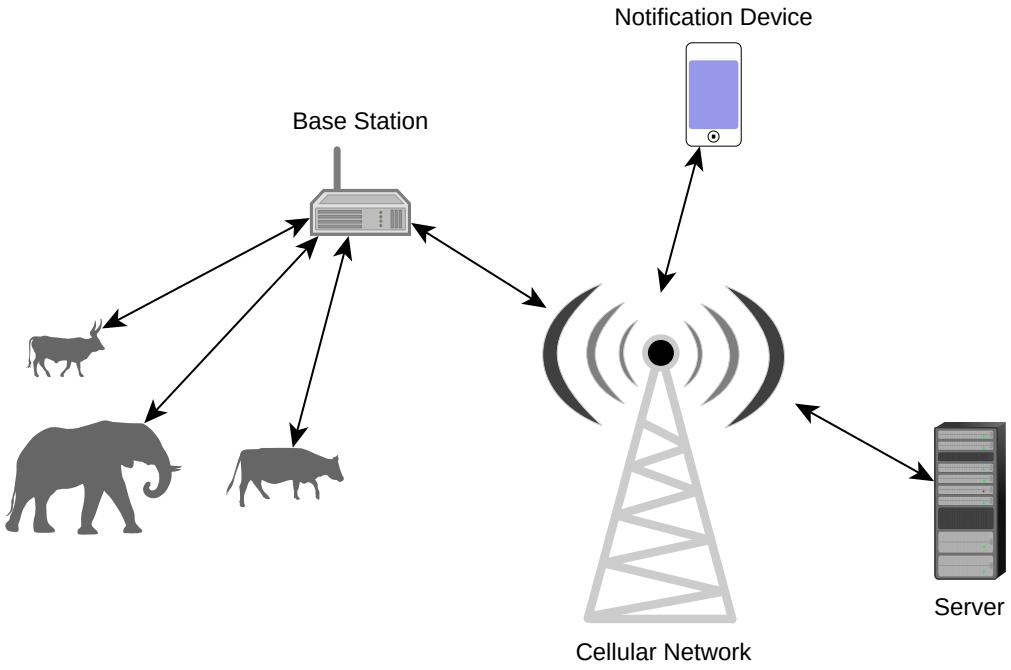


Figure 1.1: The current YRless system

tracking rhinos. The circumference of the neck of a rhino is similar to the circumference of its head; for this reason, the sensor cannot be strapped around the neck of a rhino. As a result of the growing horn, if the sensor is placed into the horn, the horn may break off, or the sensor is grown out in a few years. The sensor is consequently strapped to the leg of the rhino, reducing the sensor's communication distance.
It is costly to find and dart a rhino to attach the sensor, and therefore has to be done as few times as possible. Energy efficient sensors with reduced connection ranges are used to increase the battery life, extending the time before the sensors have to be replaced.
YRless experienced that the placement and energy constraints of the sensors reduced the maximum communication distance to $800m$. YRless requires a solution to their currents system's problems.

The following requirements were stated by YRless:

- A system that can provide greater coverage (covering an area of a game reserve) is required.
- It should be built on their current infrastructure. The sensors on the animals are low powered; therefore the intricate tasks should be performed by the system.
- The system should have a near real-time response.
- Near-autonomous. Frequent human interaction is undesirable because various park rangers are associated with rhino poachers.

1.1.4 Discussion

Looking at the current YRless system and its constraints regarding rhino tracking, a modified system is needed to combat rhino poaching. With the advantages UAV systems provide it can be used to overcome these constraints. Existing work relating UAV networks and wildlife networks is presented next.

1.2 Related Work

1.2.1 ZebraNet [1]

This is a wireless sensor network that collects tracking data of zebras, used to determine their behaviour and improve human understanding of zebras in general.
ZebraNet uses its custom tracking collars in a large wild area implementing peer-to-peer flooding and direct protocols. Data collected by each collar are flooded to the rest. The direct protocol sends data over a longer distance to a mobile base station.
Characteristics:

- Mobile base station for data collection.

- The focus is on the architecture for energy efficiency and storage (the producing of the collars).
- Nodes connect and sync without needing a base station.
- Highly mobile, and always operating.
- Data gathering network (not for emergency response).

1.2.2 SaamiNet [2]

The project was developed to provide internet services such as email and cached web access to Saami reindeer herders. The Saami herders move their reindeer yearly through parts of Norway, Sweden, Finland and Russia. Most of these areas are without any internet connection. The network makes use of Delay-Tolerant Networks (DTNs) to offer connections.

1.2.3 Air Shepherd [3]

A project was launched that will investigate the use of UAVs to protect wildlife especially rhinos. The project includes research done by the University of Maryland in the US that uses the poachers' known past positions, terrain details and animal behaviours to determine a possible poaching hazard.

1.2.4 ABSOLUTE [4]

Aerial Base Stations with Opportunistic Links for Unexpected & Temporary Events (ABSOLUTE) is a system that makes use of UAVs or described as aerial base stations to provide broadband connections to disaster areas or areas temporarily in need of a very high throughput. The network is built up of aerial, terrestrial and satellite communication links. The ABSOLUTE system focuses on rapid deployment (less than an

hour) with a temporary service. It is an expensive, complex system focussing on providing a broadband connection with no tracking aspect.

1.2.5 ANCHORS [5]

The ANCHORS system is used to inspect crisis areas that are unsafe for humans, for example, nuclear hazard incidents and extensive disaster areas. The interconnected network of ground and aerial autonomous unmanned vehicles uses sensors, actuators and cameras to report the damage back to the mission control. The system also provides a more efficient ad-hoc network between the emergency personnel and the technical systems.

1.2.6 AVIGLE [6]

A near real-time virtual reality built from pictures supplied by a network of UAVs is the basis of the AVIGLE system. The network of UAVs can also provide additional mobile radio communication where existing cell coverage is insufficient.
Architects and emergency services can use the virtual reality in situations where it is dangerous for humans.

1.2.7 Facebook [7]

As an attempt to provide Internet access to remote areas, Facebook has launched a project that will make use of solar-powered UAVs as atmospheric satellites. The UAVs will fly at a high altitude for long durations. The UAV is only in the prototype phase and test flights started March 2015.

1.2.8 Amazon [8]

Amazon is developing a parcel delivery system named Prime Air incorporating UAVs. They strive to deliver packages in less than 30 minutes.

1.2.9 Google

Google has more than one project regarding this research.

- [17] Titan Aerospace is a company Google acquired that specialises in developing solar-powered UAVs. Google is using these UAVs for the same reason as Facebook: wide area internet delivery.
- [18] is a parcel-delivery UAV system similar to Amazon.
- Google is funding a directly related project to combat rhino poaching using a UAV system. The system has three steps. [19].
  - The command centre launches the UAV with a predetermined flight path. It is then connected to the law enforcement units.
  - If the UAV detects poachers or tagged animals the command centre and law enforcement units are notified. The UAV continues the surveillance.
  - The law enforcement units intercept the poachers using the GPS location in the notification and details from the command centre.

1.2.10 Discussion

Wildlife networks such as ZebraNet are used for monitoring. Data is only collected when researchers bring the mobile base station (data collection device) in connection range of the collar network. SaamiNet is designed to provide Internet access, and no tracking component is included. Air Shepherd predicts where a poaching may occur.

The Google-funded rhino tracking system's UAV's are only launched occasionally (the system is not active all the time). ZebraNet, SaamiNet, Air Shepherd and the Google-funded rhino tracking systems are not designed to track the animals and provide emergency response.
ABSOLUTE, ANCHORS, AVIGLE, the Facebook project, the Amazon project and the similar Google projects are examples of work done in the UAV environment. From this, we see the possibilities of UAV deployment. These systems are not designed as wildlife tracking systems but offer useful insight into the use of UAVs in the animal monitoring environment.

1.3 System Proposal

Using the requirements from YRless, the constraints of wildlife tracking and the possibilities of UAV systems, the following system is proposed by the research team:

- Make use of relays that will enable the sensors to connect to a base station over a sufficient distance.
- Implement it as a mobile relay network. In a stationary approach, a relay must be installed every 800 metres resulting in an unappealing sight. A solar rechargeable drone[†] used as a relay will also be able to achieve a higher altitude, giving it a greater coverage area.
- A centralised network is used. The base station controls and configures the network.
  - YRless' existing system uses a base station.
  - A central point easily determines the positions of the drones.
  - Calculations such as position identification and routing information must be determined by a device with a reliable power source.

[†]Here defined as an autonomous UAV that is able to hover at a fixed elevation and can fly around, for example, a quadcopter.

  - The network must be able to connect to a cellular network. The base station is used to decode the data received from the drones to determine the severity of the notification.

- Drone landing spots are determined such that neighbouring drones can communicate when hovering above landing spots.
- The positioned drones will track the rhinos and give near real-time distress signals to the base station.
- The drone network is reconfigured at specified time intervals (not frequent - once or twice a day), and drones are repositioned to form the network topology.
- To save energy, not all the drones will be airborne when transferring data. This is a significant constraint on the network that causes intermittent connectivity between nodes.

The formal and detailed description of the system is presented in chapter 2.

The research team identified two research areas required by the system:

The protocol The focus of this research.

The positioning of the nodes Provided in another study [20].

1.4 Research Goal

Two goals are specified as the focus of this research:

1. Design and specify a communication protocol for the proposed drone coverage system.
2. Determine how well it will operate in the proposed system.

1.5 Assumptions

Here are the research assumptions made to have a stable starting point.

- The specifications of the proposed drone coverage system are assumed as described in chapter 2.
- Information from [20] is accurate. This information includes the node positions and their routing information (two disjoint routes from each drone to the base station). The expected information is formally described in section 2.4 item 4.

1.6 Project Scope

Considering the research goal and assumptions, the following work is in the scope of this research.

- The protocol for the proposed drone coverage system must be designed and specified.
- The animals to be tracked will not be characterised by the study. The protocol will not change its operation according to the characteristics of the animals that are tracked.
- The specific properties and behaviour of the animals will not be considered by the research. Although the position of the sensor on the animal is a constraint, it is considered in the proposed drone coverage system and will not affect the design of the protocol.
- The positions of the nodes' landing spots are not in the scope of work. YRless already have a system that determines the position of the nodes (landing spots in this instance).

- The initial positioning of the drones will not be controlled by the protocol. Drones are manually positioned before the network is operating for the first time. The protocol will take over after that and control the elevation of the drones.
- Long-term behaviour of the nodes will not be characterised, e.g. the routing protocol will not determine the routing paths differently if some nodes are utilised more often than others during the past three months.
- The drone network will not be developed and implemented as part of this study.
- Acquiring or manufacturing the hardware enabling the protocol, is also beyond the scope of work.

1.7 Research Methodology and Document Overview

In this chapter, we discussed where the problem originated. As the solution, a new system is proposed using the related work. The focus of this research was then derived from the system: design and evaluate the protocol (further referred to as Communication protocol for a Reconfigurable Animal Tracking Network (CERATIN)[‡]) for the new system. To maintain focus we made assumptions and constructed a project scope.
The research methodology can be illustrated as presented in figure 1.2. The proposed drone coverage system's components and how it operates is the main assumption of this research and are therefore discussed in detail in the next chapter (chapter 2). Chapter 2 will also serve as an elaboration of the requirement specification.
Before we could design CERATIN, knowledge was gathered and presented in chapter 3 as the literature study.
Using the requirement specification and the literature study an analyses is formed and presented in chapter 4. Chapter 4 includes the design process and formal specification. The specified protocol is implemented in chapter 5.

[‡]The term CERATIN was formed as an acronym of the new protocol's description and being similarly pronounced as the term *keratin*. Rhino horn is composed of a protein called *keratin*.

From chapter 5 we see the protocol does operate in the proposed drone coverage system, but how well does it perform? This is answered by chapter 6. The research is concluded in chapter 7 and future work is presented.

The way components were constructed in chapters 4 and 5 were compared with their relating elements in chapter 3. This serves as verification to the research.
Chapters 5 and 6 answer the question: Were the research problem and requirements in chapters 1 and 2 met? It is regarded as validation of the research.

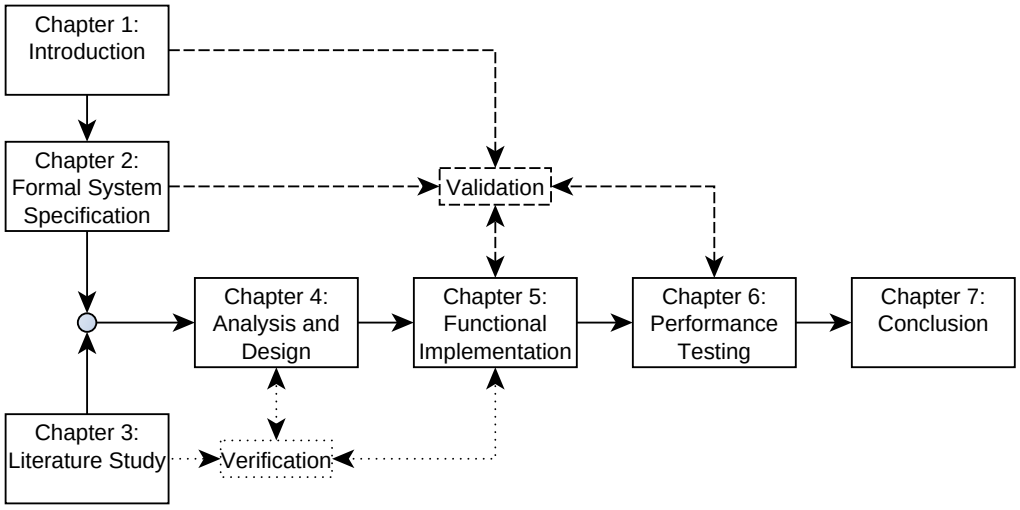


Figure 1.2: Research methodology

Chapter 2

Formal System Specification

The proposed drone coverage system is analysed and formally specified in this chapter. This is an elaboration of the specification in section 1.3. The formal system specification is also considered the requirement specification for the protocol.

2.1 Network Illustration

First an illustration (portrayed by figure 2.1) of the deployed system and its components are presented and discussed.

The whole game reserve is divided into drone tracking areas (small circles in figure 2.1) with radii of $800m$. The drone landing spots (squares) are located in the centre of these areas. The base station's position (indicated with a black marker) is then determined in such a way to be able to connect to a cellular network. The base station is stationary and has a more reliable power source. It will therefore be executing the more intricate operations and control the network. At deployment, the base station will control the configuration of the network and act as the data centre. The base station sends data to

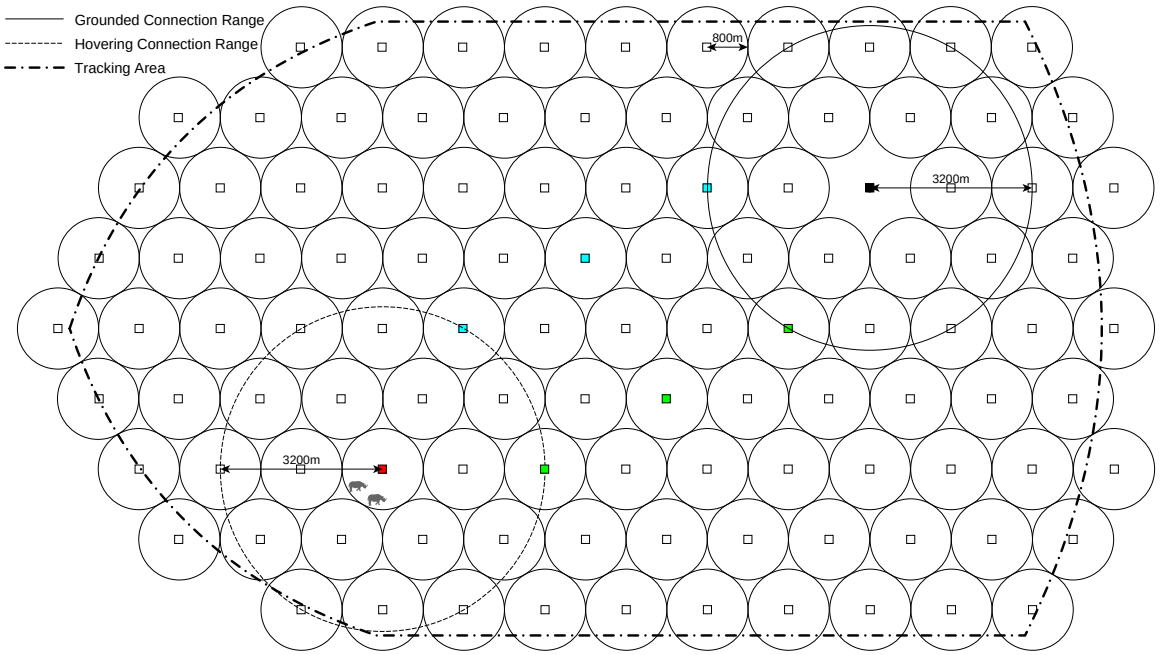


Figure 2.1: Proposed network

a server or a notification device at applicable times.

The drones are mobile and will not perform power draining operations. The red (drone actively tracking the rhinos), blue (primary route relay drone) and green (secondary route relay drone) markers indicate the drones. When a drone is grounded, it will have a communication range of 800m, and will not be able to connect to the drone at the neighbouring landing spot. When a drone is airborne, it will be able to connect to a drone 3200m away. It is also assumed that the base station has a communication range of 3200m.

2.2 Technical Specification

A proposed technical specification of the network components is presented to understand how the network operation influenced by the components' capabilities.

2.2.1 Drone

- Consists of typical UAV components, for example, motors, stabilisers, battery, etc. to ensure drone mobility and making it autonomous.
- YRless RF transceiver to provide communication.
- Solar panel, enabling the drone to recharge without interaction.
- Global Positioning System (GPS). It must be able to determine its position.

2.2.2 Base Station

- Solar panel (more powerful than the drone's).
- Battery. The base station may also be connected to a power grid.
- YRless RF transceiver as communication to the drones.
- Cellular network communication to connect to the servers or notification devices.
- GPS

2.2.3 Collar

- No GPS to save power
- YRless RF transceiver for communication.
- Sensors (heart-rate monitor, accelerometer, etc.) for data collection that can be used to identify alarms.

2.3 The System Without a Protocol

Before the operation of the full system is discussed, the system without a protocol is presented. The required role of the protocol becomes apparent.

- The game reserve is divided into landing spots.
- The drones are positioned but have no knowledge of the network.
- The drones are unable to connect with each other (Out of range).
- The base station has all the network information.
- The drone closest to the rhinos actively monitors their positions.

To identify the functions and services the protocol must provide, a full operational model is discussed.

2.4 Operational Model

This is a description of the full system presented as a operation life cycle.

1. A game reserve or farm is hosting rhinos to be tracked.
2. The game reserve is divided into landing spots as shown in figure 2.1.
3. The best position for the base station is determined.
    - The base station must be able to connect to a server or notification device via a cellular network.
    - A more central position will cause shorter network paths.
4. All drone positions and their routing information are determined by the base station:

- The positions determined in steps 2 and 3 are used in relation to the rhino positions.
- Two shortest (least hops) disjoint paths are determined.
- The two paths are determined in such a way that every drone will also have two communication paths to the base station (this excludes one drone neighbouring the base station).

5. The drones are positioned on their landing spots.
   - Only initially; the drone system will reposition itself hereafter.
6. The base station sends out configuring signals.
7. Upon receiving its configuring signal, the drone will ascend and update its routing table.
8. After the network is configured, the drones will return to the ground.
9. If a drone has data to transmit, it will ascend and transfer data to the next drone according to its routing table.
10. The data will propagate through the network towards the base station in a wave formation (causing intermittent connectivity further described in section 2.5).
    - The data includes distress signals and rhino position updates.
    - A distress signal is sent from the base station to the server or notification device.
11. When the drones have to relocate, the base station will send out a relocation signal.
    - The position updates are used to determine the next positions of the drones.
    - Step 4 is re-executed.
12. Upon receiving its relocation signal, the drone will ascend and update its position data and relocate itself.

13. After all the drones are relocated the configuration procedure (starting with step 4 and excluding step 5) will be re-executed.

YRless executes steps 2 and 3. Another study [20] determines step 4. Step 5 is performed by a deployment team. Steps 6 to 12 are controlled by the protocol and are considered part of the protocol requirement analysis.

2.5 Intermittent Connectivity

One of the key constraints that result in a unique protocol is the intermittent connectivity. Here we describe the cause and reason for the intermittent connection on the network.

Waves propagate as a result of diffraction, refraction, reflection or absorption [21]. This propagation effect is greater with higher frequency waves. Antennas transmitting waves above 30 MHz requires a line-of-sight to their receivers [22]. These waves only have a line-of-sight up to the horizon because of the curvature of the earth. The distance from the antenna and the horizon is called radio horizon. The formula to calculate this distance is:

$$d_{mi} = \sqrt{2h_{ft}} \tag{2.1}$$

Converted to the metric system:

$$d_{km} \approx 3.57\sqrt{h_m} \tag{2.2}$$

From equation (2.2) it can be seen that a small change in the altitude of an antenna has a large influence on the sight distance to the horizon. In an ideal case with no obstacles the altitude of a drone should be

$$h_m \approx (\frac{d_{km}}{3.57})^2 = (\frac{3.2}{3.57})^2 \approx 0.8m \tag{2.3}$$

A drone does not have to ascend high to reach the theoretical connection range of $3200m$. In a game reserve, the drone elevation will have to be higher because of obstructions such as mountains, hills and bushes.

If the drones are elevated, they will have extended transmission ranges. Fewer drones are then needed for greater coverage. The drones will be unable to hover for inordinate lengths of time because it is a power draining task. Therefore, if a drone has to transmit data it first has to elevate to extend its transmission range to reach the next drone. When transmitting data, a drone will not be connected to its destination but only to its next hop. A drone is therefore not constantly connected and sends data at irregular intervals, hence the intermittent connectivity.

2.6 Discussion

It is now clear how the network is supposed to operate and what role the protocol plays in this operation. Before we can analyse the requirements and present a protocol design, knowledge has to be gathered. The next chapter presents knowledge required to address the research goal.

Chapter 3

Literature Study

A research goal was identified: Design a protocol for the proposed drone coverage system. The system was described in chapter 2. In the literature study chapter, information is presented to solve the problem. It includes: Describing a telecommunication network fundamentals and its components. A data link layer, Stop-and-Wait protocol is then presented. Further routing protocols for Mobile Ad Hoc Networks (MANETs) are presented namely Ad Hoc On Demand Distance Vector (AODV) and Dynamic Source Routing (DSR). Then a typical network architecture for MANETs namely DTN is given. Lastly the chapter is concluded with a discussion.

3.1 Telecommunication Network [9]

The word telecommunication means to communicate over a distance (the Greek term *tele* means "distant" [23]). Communication is the sharing or transmission of information between entities. Communication systems have five basic components namely *senders*, *receivers*, *messages*, *transmission media* and *protocols* [9] as portrayed by figure 3.1.

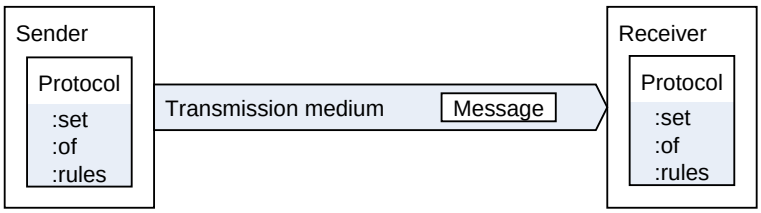


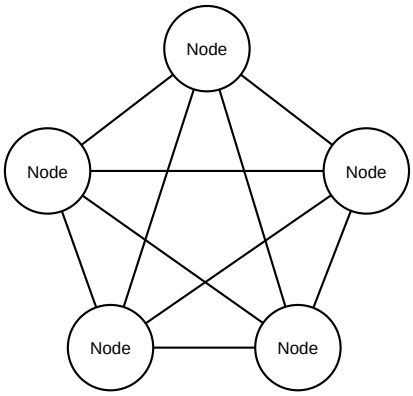
Figure 3.1: Five components of a communication system

- *Sender:* The source entity of the message that is communicated.
- *Receiver:* The destination entity of the message that is communicated.
- *Message:* The collection of information that is communicated.
- *Transmission medium:* The physical path along which the message travels. The transmission media are divided into two main categories namely wired and wireless. Wired media include twisted pair cables, coax cables and fibre optics. Wireless media make use of waves to carry the information (there is no physical connection). Wireless media include radio waves, microwaves and infrared.
- *Protocol:* For the entities to 'understand' each other they have to follow a set of rules. The set of rules is called the protocol of the network.

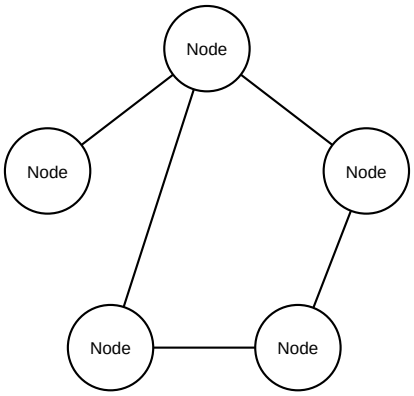
When more entities (from now called nodes) need to communicate or over a further distance, a network is formed. The way a network is laid out physically is referred to as the network topology. The four network topologies are *mesh*, *star*, *bus* and *ring*.

Mesh (figures 3.2a and 3.2b)

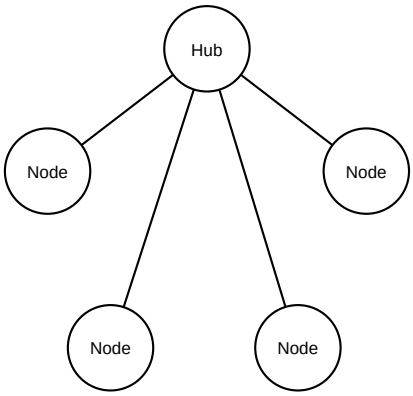
With a *full mesh*, every node has a dedicated connection to every other node in the network. This topology is robust but requires a lot of cables and ports when using a wired transmission medium. Wireless media can be used in a *full mesh* because no physical cabling is needed only different channels. It is seldom possible for every node



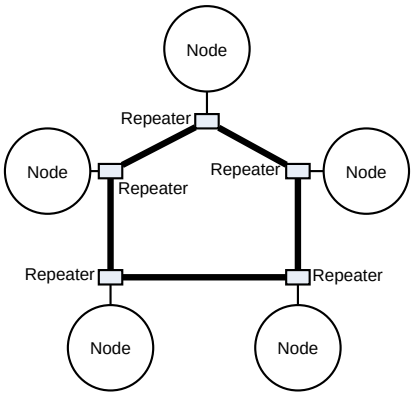
(a) Full mesh



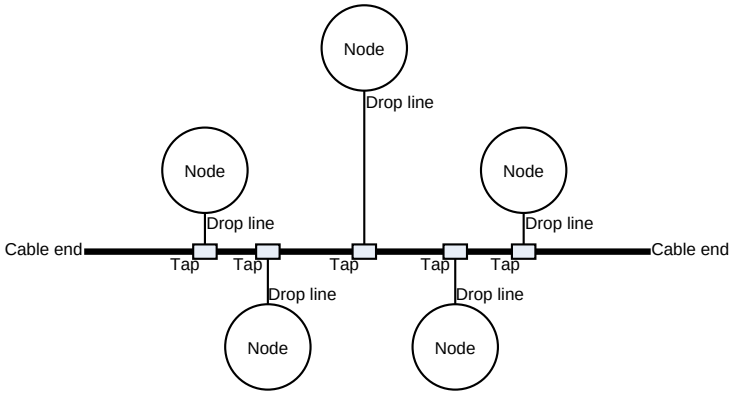
(b) Partial mesh



(c) Star



(d) Ring



(e) Bus

Figure 3.2: Network topologies [9]

to be connected to every other node and therefore a *partial mesh* is formed. Due to mobility and layout of MANETs their network topology is usually a partial mesh.

Star (figure 3.2c)

Each node is connected to the other nodes via a central point usually called a hub. With a *star topology*, less cabling is used and is, therefore, less expensive than the *full mesh*. When the central point of the network breaks, all connections are lost.

Ring (figure 3.2d)

Data is sent along the ring until a repeater recognises that the message is intended for the node directly connected to it.

Bus (figure 3.2e)

The *mesh* and *star topologies* have point-to-point connections between the nodes whereas the *bus topology* is multipoint. Taps and drop lines connect the nodes to the backbone cable.

3.2 Network Model

When a complex communication system is developed the networking tasks are divided into layers. Each layer contains the appropriate protocol(s). This is called protocol layering. The set of defined layers forms the network model.

The International Organisation for Standardisation (ISO) [24] developed a network model named Open System Interconnection (OSI). OSI can be used to construct a new communications network and its protocols.

OSI is defined with seven layers named the *physical*, *data link*, *network*, *transport*, *session*, *presentation* and *application layer*.

Physical Layer Responsible for transferring the bits from one node to another.

Data Link Layer Ensures data delivery between two nodes that is directly connected (neighbouring nodes).

Network Layer Controls the path the data travels through the network. It creates a routing link between the source and destination.

Transport Layer It controls to what extent data is delivered from the source to the destination. The transport layer, for example, controls if the network is state-oriented, connection-oriented or connectionless.

Session Layer It establishes an isolated interaction or period between application processes of different nodes.

Presentation Layer Data arrives at the presentation layer with different formats. This layer then 'translates' the data to a format usable by the application layer. 'Translation' may include encryption, decryption, compression and character code translation.

Application Layer The applications of a node are given network capabilities. This layer includes the network connection to applications such as the web browser, email and internet chat.

When constructing a network model two properties must be present: 1) In bidirectional communication a layer should perform two inverse tasks. 2) Each node should have the same object under each corresponding layer (forming a logical link).

3.3 Data Link Layer [9]

As discussed in section 3.2 the data link layer controls the communication between two neighbouring nodes. The basic control functions include *framing* and *flow and error control*. These two functions form Data Link Control (DLC).

Framing

This is used to organise and pack the data from the network layer into a frame for improved transmission. Framing divides the data (errors on large data result in longer retransmissions) and makes the frames distinguishable from each other (using a delimiter as boundaries).

Flow and error control

Data acceptance of the neighbouring receiver node takes variable durations. When it happens slowly, an error will occur when data is sent too fast. Time is wasted when data is sent too slow when data acceptance is fast. Flow control is the sending of data at optimal intervals. Data can become corrupt during transfer (failure occurs). Error control is implemented to identify the failure and then correct or report it.

3.3.1 Stop-and-Wait Protocol

It is a relatively simple protocol implementing flow and error control. A sender adds a Cyclic Redundancy Check (CRC) (generated from the frame data) to the frame and sends it. The receiver then tests the frame using the CRC. If the data is received successfully and correctly, it can reply with an acknowledgement. The sender can send the frame according to how fast it receives an acknowledgement (flow control). If an acknowledgement is not received a failure occurred (error control).
Sequence and acknowledgement numbers are also added to the frame to avoid data

duplication and incorrect ordering.

The working of the stop-and-wait protocol is described using Finite State Machines (FSMs)* in figures 3.3 and 3.4.

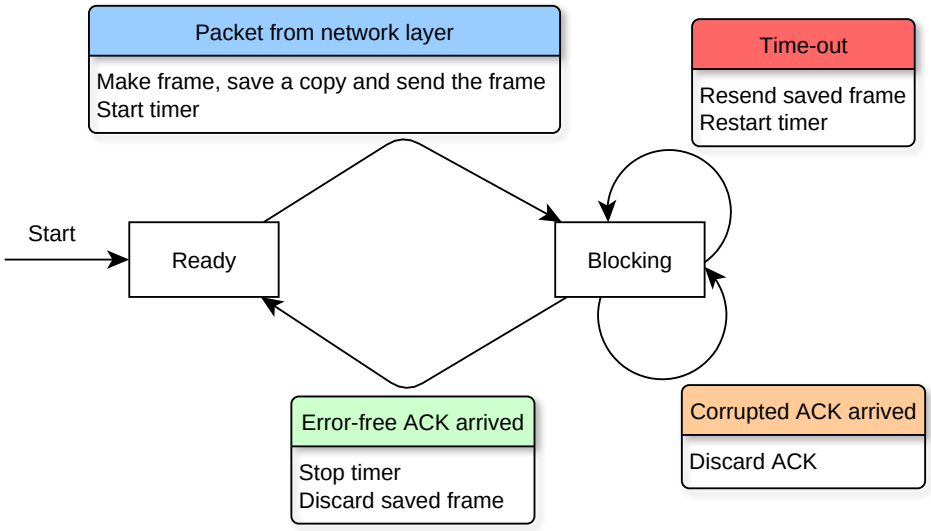


Figure 3.3: Stop-and-wait protocol sender FSM

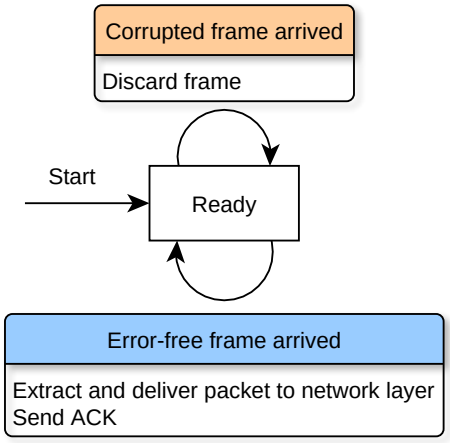


Figure 3.4: Stop-and-wait protocol receiver FSM

*A FSM in this study is defined as follows: A sender and a receiver FSM represent the whole node, and not the sender or receiver unit of the node, e.g. an intermediate node will receive and send a message but will only be considered as a receiver in the FSM. In summary, the node that transmits the first message is the sender the rest are the receivers.
Every FSM is implemented as a Mealy machine where the output relies on the current state and action.

3.4 Network Layer

When sending a message from a source to a destination with multiple intermediate nodes, the message's path should be determined first. Routing protocols, usually contained in the network layer, are implemented to form the path.
Here two routing protocols namely *AODV* and *DSR* are discussed. These routing protocols provide useful information regarding this study, for they were designed with a MANET in mind.

3.4.1 AODV [10]

AODV was developed for mobile ad hoc networks. It was regarded as the solution to the problems other routing protocol experience with MANETs. The mobility and ad hoc constraints required a dynamic routing protocol. Based on the table driven but not highly dynamic proactive protocol, Destination-Sequenced Distance-Vector Routing (DSDV), AODV was created.

Here are the characteristics of AODV [10,25]:

Table driven The path a data packet travels is determined by routing tables hosted locally on each node.

On demand/Reactive The route is set up only when data needs to be sent and not in advance.

Distance vector The nodes in the network contains paths (not necessary the full route to the destination) of nodes across the network.

Ad hoc No existing network architecture is needed before executing.

Wireless and mobile friendly

Performs link state checks (hello messages) AODV periodically sends hello messages to detect if link failures occurred between neighbouring nodes.

Sequence number utilisation Sequence numbers are used by AODV to retain route freshness and prevent loops.

Single route Native AODV only implements one route because it is difficult to identify whether another route is available if one fails. (Multiple routes can be determined).

Uniform packet sizes Other protocols such as DSR uses variable sizes because its route is contained inside the packets.

Route maintenance AODV incorporates error messages as a maintenance mechanism.

AODV basic operation

The operation of AODV is described and presented as finite state machines (figures 3.5 and 3.6):

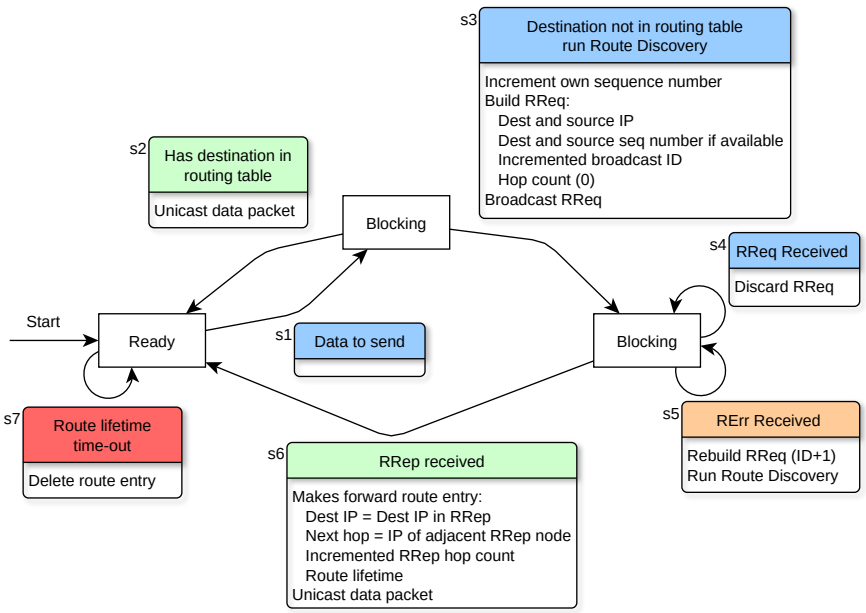


Figure 3.5: AODV sender FSM

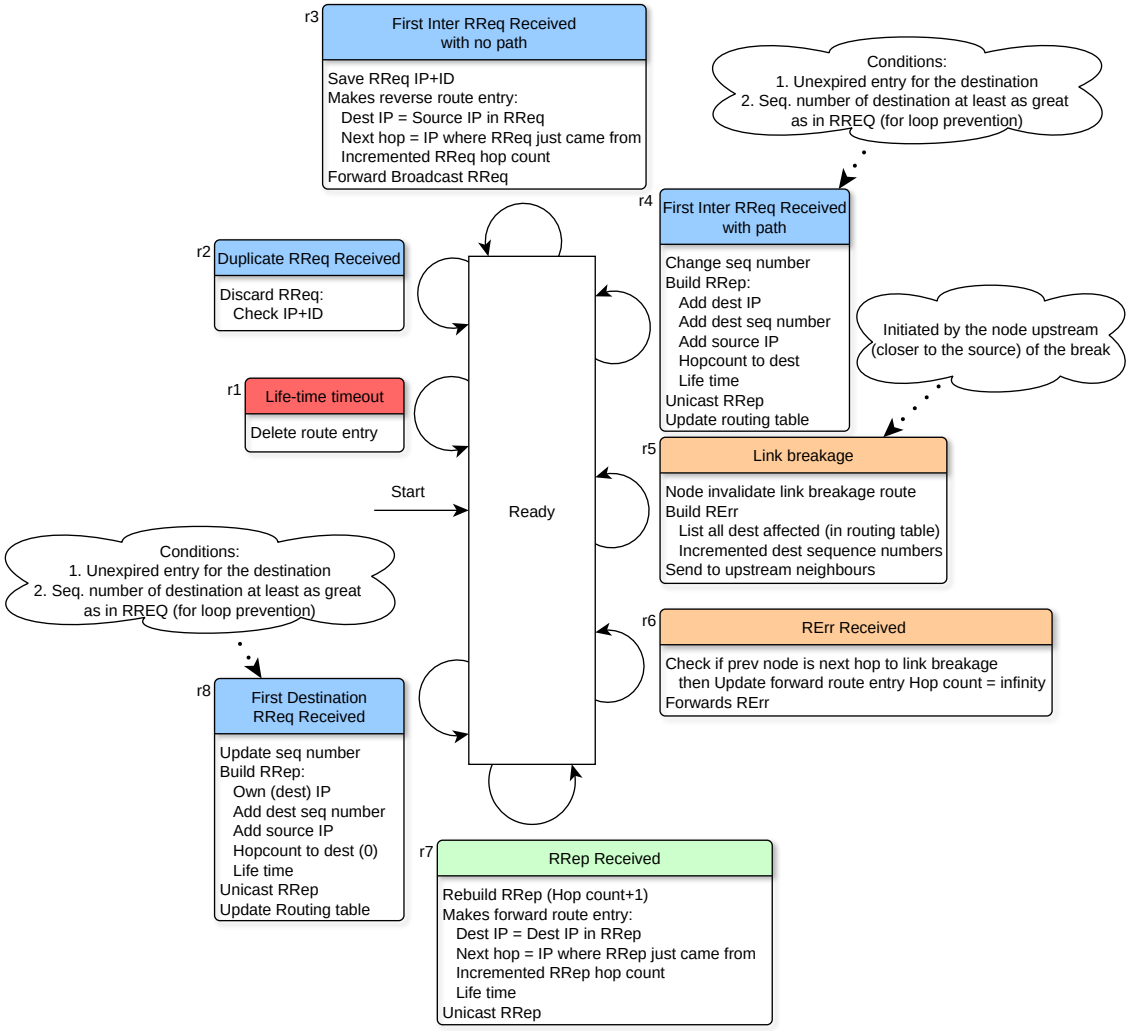


Figure 3.6: AODV receiver FSM

Figures 3.5 and 3.6 description:

Route discovery is initiated when data needs to be sent (s1), and the source node does not have a route entry for the destination (s3). If the source contains a route entry for the destination (s2), the data is unicasted.

The source node builds the route request packet and broadcasts it (s3). If a receiving intermediate node has a route entry for the destination, it updates the sequence number for the destination and informs the source with a route reply (r4). If an intermediate node does not have a route entry it updates its routing table and broadcasts the route request (r3). The route request propagates through the network until it reaches the des-

tination. Upon receiving a route update, the destination updates its sequence number and routing table (r8). It builds a route reply packet and informs the source (r8).
As a route reply packet travels through the network, a forward route entry is made in the nodes (r7). A reverse route entry is made when a new route request is received (r3). When a link breakage occurs the node breakage identifier node sends a route error to all the affected nodes (r5). Nodes receiving the route error invalidate the infected route entry and forwards the route error (r8). The source node will re-initiate the route discovery (s5).
Each route entry has a Time To Live (TTL). When the TTL expires its route entry is discarded (r1,s7). When the source node receives its route request (s4) or a node receive a duplicate route request (r2) the packet is discarded.
Hello messages are used by AODV for nodes to locally offer connectivity information. Every node periodically broadcasts a hello message while the receiving nodes update their routing tables. If a neighbour node does not receive a hello message from a specific node after a predefined time-out, the neighbour node deletes that specific node's address from its routing table. This function does not depend on the state of the node, but rather on how much time has passed.
AODV also uses route reply acknowledgement packets that are sent in reply to route reply packets. This function is only activated when the underlying protocols do not provide an assured delivery.

Well-known technologies, such as ZigBee and DigiMesh, developed to allow one to create Wireless Personal Area Networks (WPANs), also use AODV as the base of their routing protocol [26,27].

3.4.2 DSR [11]

As AODV, DSR was designed with MANETs in mind. Also dynamic, DSR can only function in a network consisting of maximum 200 nodes. This is mainly because the full route is contained within the packet header (because of the source routing ap-

proach).

Here are the characteristics [11, 25]:

Source routing The route of a packet is contained inside the packet. The source therefore determines the route.

On demand/Reactive The route is set up only when data needs to be sent and not in advance.

Distance vector The nodes in the network contains paths (not necessary to the destination) of nodes across the network.

Ad hoc No existing network architecture is needed before executing.

Wireless and mobile friendly

No link state checks Errors are detected when packet forwarding is unsuccessful.

Route cache This function allow the route discovery to execute faster, and a lot of network utilisation is saved (fewer nodes are active during route discovery - saves power)

Multiple routes A packet may travel different routes. The sources determine the routes, but a node can check for other routes in its cache when failed.

Variable packet size DSR uses variable sizes because its route is contained inside the packets and is not always the same lengths.

Route maintenance DSR incorporates error messages as a maintenance mechanism.

The operation of DSR is described using FSMs (figures 3.7 and 3.8):

Figures 3.7 and 3.8 description:

Route discovery is initiated when a source node does not have a cached path to the destination (s3) but has data to send (s1). The source node builds a route request and

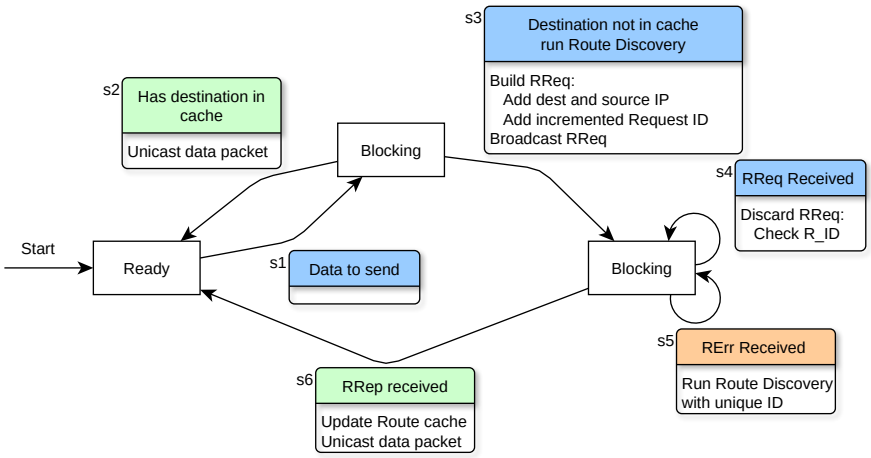


Figure 3.7: DSR sender FSM

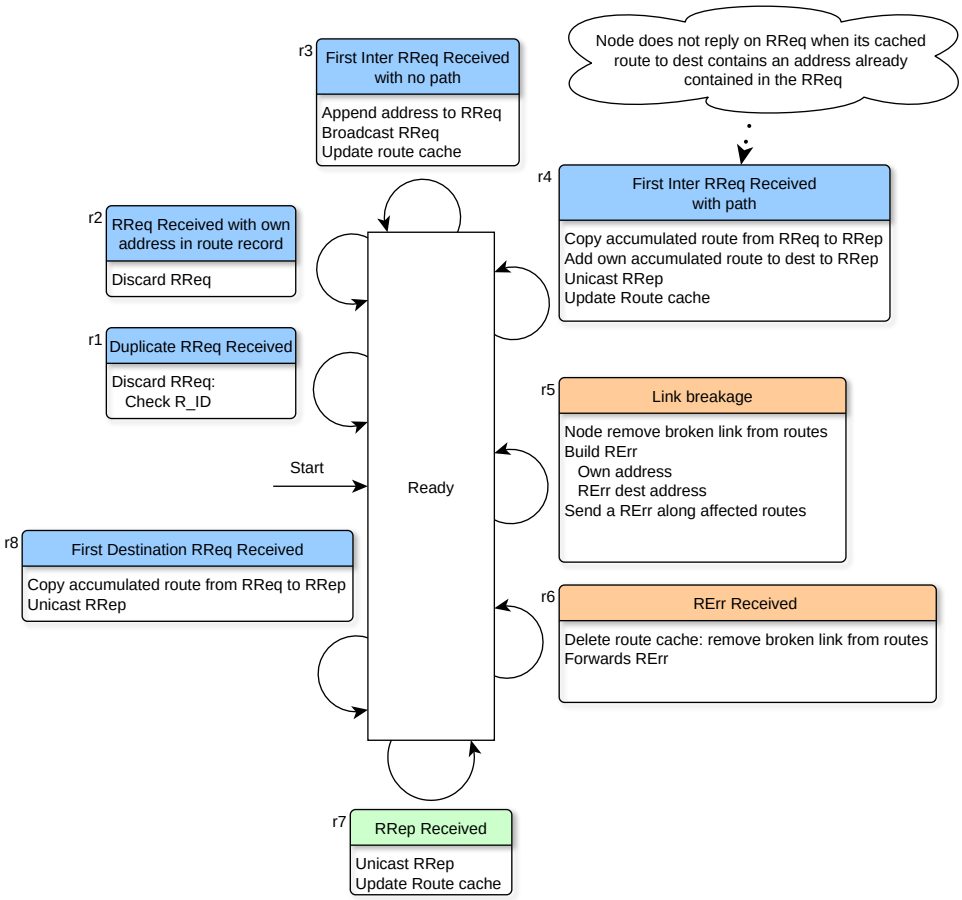


Figure 3.8: DSR receiver FSM

broadcasts it (s3). An intermediate node not containing the destination in its cache appends its address to the route request and broadcasts the route request (r3). A node

containing a route to the destination will add that route to the route in the route request and inform the source with a route reply (r4). The destination node will copy the route from the route request to the route reply and in forms the source (r8). A route reply received by intermediate nodes is unicasted, and the nodes update their route cache (r7). When the source node receives a route reply, it sends the data along the route in the route reply (s6).
If a node encounters a link breakage, the unreachable node address is used to determine the affected nodes (r6). A route error is sent to the affected nodes and the cached routes containing the unreachable address are removed (r6). A node will delete a cached route that is affected by the received route error (r5). The source node will re-initiate the route discovery (s5).
A duplicate route request or one containing the received node's address is discarded (s4,r1,r2).

3.5 Network Architectures

Various highly mobile sensing networks use the DTN architecture. The drone coverage system has components that can be addressed with a DTN architecture.

3.5.1 DTN

DTNs [28] (also known as disruption-tolerant networks) were originally designed for interplanetary communications. It is used in networks where an instantaneous connection from the source to the destination is absent for a long period.

The DTN architecture implements a bundle layer above the conventional transport layer. This bundle layer provides data storage for when the network is interrupted until a connection is available for data to be forwarded (also known as the store-and-forward mechanism). The layer implementation also provides interoperability to DTNs. It can therefore connect two regions with different sub-networks together with

a delay tolerant connection.

DTN was designed to have characteristics Internet architectures lack. These characteristics include:

- A connection from the source to destination does not have to exist for a given time to transfer data.
- Time-outs on network errors and error correction do not have to be stable.
- A lot of loss in transmission is accepted.
- It is interoperable.
- A fixed single path is not required for communication.

To achieve these characteristics DTNs were designed with the following functions:

- Message lengths are not fixed. The network transfers more data with longer messages when a connection is available.
- For interoperability, a naming syntax is used that supports a wide range of addressing and naming conventions.
- The nodes have high storage capacities to store more data for longer durations.
- Services are provided by many small module classes, delivery options and a smart calculation of data lifetime are used.

The application of a DTN was found to be advantageous to MANETs. The mobility of the nodes in MANETs results in a disruption or delay in connectivity. Wildlife networks are usually classified as a MANET.

Here are examples where DTNs are used in wildlife scenarios:

White Tail Deer habitat monitoring [29]

This is a wireless sensor network as a DTN that monitors the habitat of the White Tail Deer in North Canada. Sensors on animals and in the environment collect data and communicate it to the principle nodes (nodes with storage capabilities placed where the deer density is high). The data is then recovered from the principle nodes using a mobile data collector device.

Sufficient node density conditions for wildlife monitoring [30]

Light weight battery powered collars attached to animals, collect the data. The data is sent to a storage device (access point) when a connection is available. Data includes location, activity and biometric information. Collected data is used to form a tracking history of the animals. This research, however, focusses on the effect of access point density on data loss.

Seal-2-Seal [31]

Here the protocol is used to operate in a network where sensors attached to animals, are used to monitor their behaviour and interactions. As in [29] and [30] data is collected by the sensors are communicated to storage devices (here called sinks). Researchers will then occasionally collect the data from the sinks to be analysed.

DTNs are used in wildlife scenarios but only for monitoring and not for active or real-time tracking.

3.6 Discussion

Considering the specification of the system in chapter 2, the following conclusions are made.

This network's transmission media will be *wireless*. The network topology is incorporated as a *partial mesh* network (also called a mesh network).
From the network model, we see that this network consists of the first three OSI layers namely the *physical*, *data link* and *network layer*.
The *physical layer* assumes the current YRless RF communication and is not part of this study.
The *data link layer* will implement a version of the *Stop-and-Wait protocol*. It is simple and provides error and flow control necessary for this implementation. The *data link* is described and implemented in this study because the network layer relies on it. The focus is rather on the network layer protocol.
The major design lies in the network layer. More specifically the protocol in the network layer. Looking at the network layer routing protocols and the system specification we see that CERATIN is rather a communication protocol. The base station determines the routes of the data. CERATIN should only use this routing information and enable communication. Although CERATIN is not a routing protocol, services and mechanisms of AODV and DSR can be used and include:

Source routing (DSR) The base station has all the routing information and can therefore include the routing path in the packet.

Table driven (AODV) When data is sent from the source drone to the base station, the drone does not have the whole route. Therefore, a table driven approach can be followed.

Route maintenance (AODV and DSR) Both protocols notify the source node of a link failure using an error message.

Some aspects are not included in the routing protocols but can be described by the DTN architecture. During data send not all the drones will be connected, to save energy. This causes a disruption in connection that is characteristic of a DTN.

The acquired knowledge can now be analysed to form a protocol design.

Chapter 4

Analysis and Design

The requirements stated in chapter 2 and literature in chapter 3 are analysed to form a protocol design. First a discussion of the design process is given. In the design process, the constraints and requirements are specified, the network functions are then determined and synthesised in finite state machines, and lastly the protocol is formally specified.

4.1 The Design Process

The design process that was followed is based on the design processes and protocol structures discussed in [32], [33] and [34].
[32] uses a modified waterfall development process. This waterfall development process displays the stages specific to protocol development. It is untidy and difficult to follow but has beneficial aspects such as a predetermined requirement analysis.
The phases presented in [33] are uncluttered and will be used as the primary design process. [34] discusses phases of a structured protocol design.

These processes and structures were modified and combined to fit this implementa-

tion. This altered process is displayed in figure 4.1.

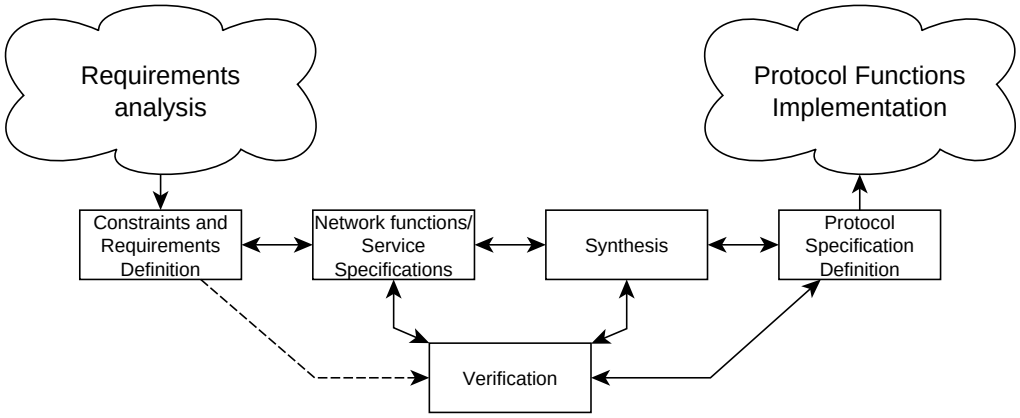


Figure 4.1: Adapted design phases

Chapter 2 contains the requirement analysis. Now the constraints and the requirements are defined based on that analysis. A network functions/service specification is developed and then further described and synthesised as finite state machines. A formal protocol specification is then defined.
The Network functions/Service Specifications, Synthesis and Protocol Specification Definition phases are verified against the defined Constraints and Requirements. If the phase could not be verified, the previous phase is inspected and modified.
After the Protocol Functions Implementation phase is verified the protocol is implemented in a network environment. This implementation is discussed in chapter 5.

4.2 Constraints and Requirements

The requirement analysis (chapter 2) and the literature study (chapter 3) are considered to formally specify the constraints and requirements of the protocol.

4.2.1 Wireless and Mesh Network

The system specification dictates that the system operational environment should be wireless with a mesh topology. A wireless network is more unstable than a wired network. AODV and DSR were designed for wireless mesh networks. Many of the characteristics of CERATIN are based on AODV and DSR.

4.2.2 Network Model

The network for the proposed system requires physical communication, data delivery between two neighbouring nodes and the transferring of data along specific routes. Therefore, only the physical, data link and network layer are designed.

4.2.3 Base Station - Drone Environment

The specifications state that the network is centralised (section 1.3). With all the routing information at the base station, a source routing approach can be followed. When data is sent from the base station, source routing from DSR can be used but when data is sent to the base station, another mechanism should be used. Luckily AODV hosts such a mechanism called the table driven approach. This approach has the following advantages over source routing:

- A failure is repaired locally by a table driven approach.
- An extra field in the packet for the full route is not required.

4.2.4 Intermittent Connectivity (network disruption)

Section 2.5 described the intermittent connectivity in the network. The intermittent connectivity creates disruptions in the network. From the literature study, we see

that the DTN architecture can be used in a disruptive network, such as the store-and-forward mechanism. DTNs transfer data only when a connection is available. The network specification requires a near real-time response, therefore has to create a connection, resulting in an actor network*. As an actor-network, a requirement is that the network functions must be capable of initiating some of the drones' flight functions to make their connection to the base station.

4.3 Network Functions

Considering the constraints and requirements, the following network functions are identified.

4.3.1 Supplied Service/Functions

The base station will contain the positions of the drones and a primary and secondary disjoint route from each drone to the base station. The positioning and routing algorithms are determined in another study [20] and will be supplied to the communication protocol.

4.3.2 Route Update

The drones' routing tables have to be updated before they can communicate. The base station containing the routes will determine each drone's routing table and send it in such a way that all the drones receive it (The drones will be grounded initially and not able to send data to the next drone). When a link failure occurs, the base station should be informed. A single routing table that is not received can disrupt the network. This will allow the base station to recalculate the routes and routing tables before continuing

*In an actor-network, the network functions are able to perform physical tasks (manipulate the hardware).

with the route update transmissions.

4.3.3 Landing Update

After the routing tables are configured the base station must be able to send out a landing update. The landing update will only contain a landing instruction. It will be unicasted to all the drones in the network in such a way that every drone is reached.

4.3.4 Data Send

Data will be transferred from a drone to the base station every few hours when transmitting status updates or anytime when transmitting distress signals. The drones will be required to ascend regularly. If the data is transferred to the base station in a wave formation, the drones will use less power.
The wave formation works as follows:

- If a drone (drone one) has data to transmit, it will ascend and transfer data to the next drone (drone two).
- Upon receiving the data, the next drone (drone two) will ascend and also transfer data to its next drone (drone three).
- When data is received by drone three, drone one will return to the ground and delete the data.
- While data travels through the network, two drones should be in possession of the data, should the next drone lose all connections the previous drone can transmit the data using another route.
- In summary, each time a drone receives data and ascends the drone two hops back will return to the ground.

4.3.5 Position Update

The *position update*[†] is similar to the *route update* in the sense that the delivery of the update is crucial. If one drone did not receive it, it will not reposition itself and may disrupt the network.
After the drones receive the position update[‡], and the base station acknowledges it, they will receive a fly signal as a repositioning trigger. The repositioning trigger is implemented the same way as the *landing update*.

4.4 Synthesis

Finite state machines are employed in formal validation, protocol synthesis and conformance testing of a protocol [34]. It will also show the limits of the network functions and how they will operate in a machine.
While CERATIN is synthesised, the FSM result is compared with its corresponding component in the literature study. If the component is not verified, it is redesigned. Here follows the final constructed FSMs.

4.4.1 Physical Layer

YRless international has an existing sensor communication system that will be used as a physical layer.

4.4.2 Data Link Layer

The data link layer protocol is not the focus of the research but has to be implemented for the network to operate correctly (It transfers data to and from the network layer

[†]'*Position update*' indicates the network function
[‡]'Position update' refers to the message named position update

and gives notifications if failures occurred). It is therefore part of the design process.

It is implemented as a *Stop-and-Wait protocol* with minor alterations. FSMs in figures 4.2 and 4.3 describe the modified protocol operation.

All the network functions as identified in section 4.3 will operate the same on the data link layer.

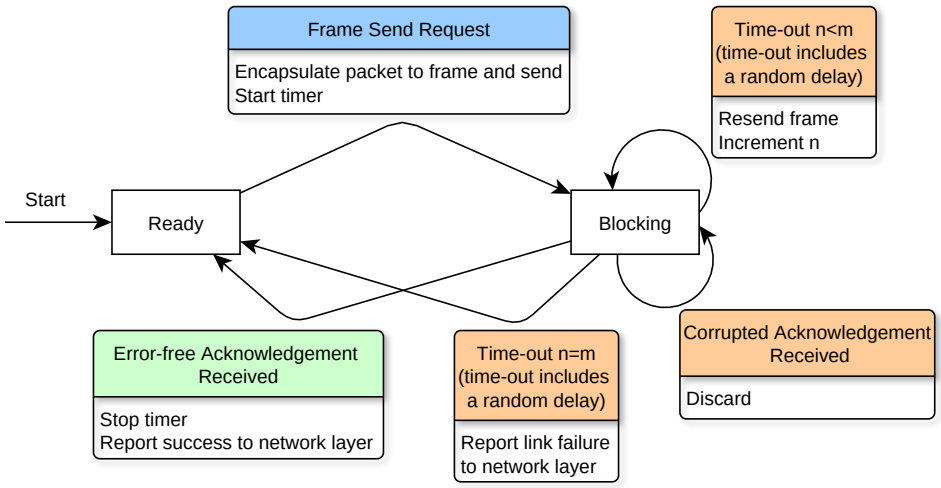


Figure 4.2: Data link (Sender)

Figure 4.2 description:

Frame send request from network layer: The data link layer is requested to send data. It takes the packet from the network layer and creates a frame. The frame is sent to the next hop, and the resend timer is started. The FSM is now in the blocking state.

Time-out n<m: When an n^{th} time-out occurs with $n < m$, where m is the maximum number of retries, the frame is resent, and n is incremented. A random delay is added to the duration of the time-out (also referred to as the *random added delay mechanism*). When a collision of frames occur the retransmission is not certain to collide because it will most likely happen at different times.

Corrupted acknowledgement received from physical layer: All corrupted acknowledgements are discarded.

Time-out $n = m$: If the data link tried to send the frame the maximum amount of times without success a link failure is reported to the network layer.

Error-free acknowledgement received from physical layer: The frame delivery was successful if an error free acknowledgement arrived. The timer is stopped. Success is reported to the network layer.

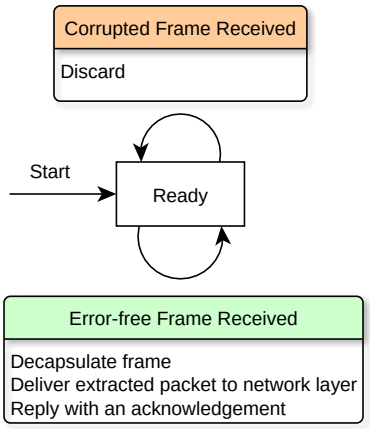


Figure 4.3: Data link (Receiver)

Figure 4.3 description:

Corrupted frame received from physical layer: All corrupted frames are discarded.

Error-free frame received from physical layer: When an error-free frame arrives, the packet is extracted and delivered to the network layer. An acknowledgement is assembled and replied to the data link sender.

4.4.3 Network Layer

This section serves as the synthesis of the network functions of the new protocol (CERATIN).

Route Update

The base station sends out the first route update (the first address in its pre-obtained routes that is one hop away) and waits for the acknowledgement before sending the next route update. It will send route updates starting with the least hops, waiting for its acknowledgement, continuing until the last drone that is the most hops away. If an error occurs the routes and routing tables will be recalculated before proceeding. A drone receiving the route update will ascend to extend its transmission range to reach the next drone. When all the drones acknowledged that they received the route updates the base station will initiate the *landing update* network function. The route that the update must follow is encapsulated inside the message (similar as implemented in DSR). This will allow the message to follow the same route back, and the base station will know that the route is set up correctly.

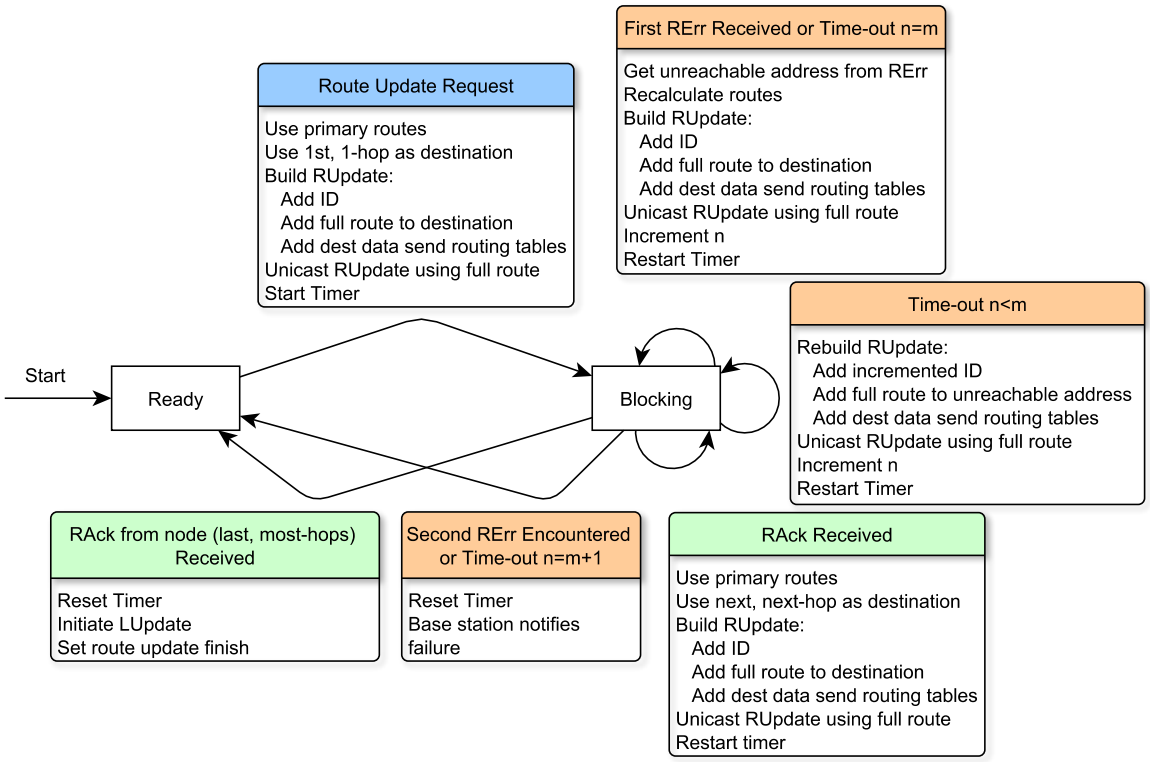


Figure 4.4: *Route update* (Sender - base station)

Figure 4.4 description:

Starts on ready state and waits for a *route update* request.

Route update request: The base station considers the predefined primary routes. The full route from the base station to the drone is determined. It builds a route update packet containing an ID, a full route to the destination and the destination's *data send* routing tables. The ID and the destination address combined are always unique. The route update packet is unicasted using the full route field. A timer is started that is used to estimate when a packet is lost. The base station goes into a blocking state.

First route error packet received or time-out $n = m$: n is the number of time-outs with m the maximum. The base station extracts the address of the unreachable drone from the route error packet. For a time-out, the unacknowledged route update destination address is set to be the unreachable address. It then recalculates the routes to facilitate the route failure. The base station constructs a new route update with a unique ID destination combination by incrementing the ID. The new route update packet is unicasted using the full route field, n is incremented, and the timer is restarted.

Time-out $n < m$: A new route update packet with its incremented ID is unicasted to the unacknowledged address. n is incremented, and the timer is restarted.

Route acknowledgement packet received: The FSM reacts the same as when a *route update* request is initiated. The only difference is that it uses the next, next hop node from the predefined routes.

Second route error encountered or time-out $n = m + 1$: The timer is reset, and a notification is created because too many failures occurred.

The last (the last node with the most hops) route acknowledgement packet received: The timer is reset, and the base station initiates the *landing update* FSM and registers that the *route update* is finished.

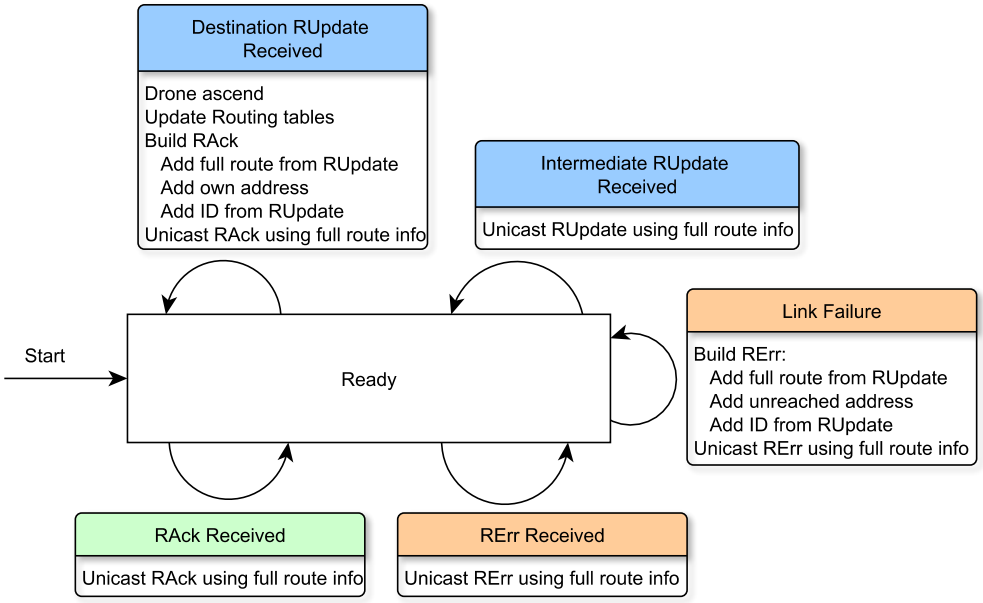


Figure 4.5: *Route update* (Receiver - drones)

Figure 4.5 description:

The FSM starts and is always in the ready state.

Destination route update received: The drone will first ascend to extend its coverage range and connect to the preceding node. It will update its routing tables by extracting data from the route update packet. The acknowledgement is then constructed. It adds the full route from the route update, its address and the ID from the route update. The route acknowledgement is then unicasted using the full route field.

Intermediate route update packet received: The route update packet is unicasted using the full route field.

Link failure occurs: A route error is constructed. The drone adds the full route from the route update, the unreachable address and the ID from the route update. The route error is then unicasted using the full route field from the packet.

Route error packet received: The route error packet is unicasted using the full route field from the packet.

Route acknowledgement packet received: The route acknowledgement packet is unicasted using the full route field from the packet.

Landing Update

The base station starts sending an update to every drone. If a drone receives its intended updates, it will return to the ground. Intermediate drones will forward a received landing update. It is, therefore, important that the base station sends the landing updates to the drone the furthest away and ending with the closest drone. The routes are also encapsulated inside the messages and are adjusted to the link errors that occurred during the *route update*.

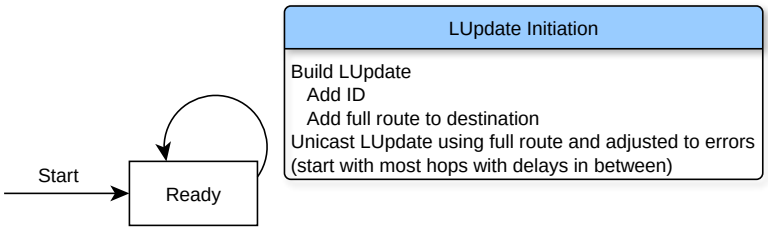


Figure 4.6: *Landing update* (Sender - base station)

Figure 4.6 description:

The FSM starts and is always in the ready state.

Landing update is initiated: The base station builds a landing update packet. It contains an ID and the full route to the destination. The full route was adjusted to failures that occurred during the *route update*. The landing update is then unicasted starting with the drones with the most hops. It unicasts the next packet after a delay to avoid collisions.

Figure 4.7 description:

The FSM starts and is always in the ready state.

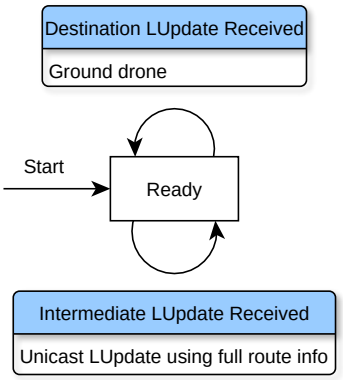


Figure 4.7: *Landing update* (Receiver - drones)

Destination landing update packet received: The drone is grounded.

Intermediate landing update packet received: The landing update packet is unicasted using the full route field from the packet.

Data Send

The *data send* function is initiated when a drone has to transmit status updates and distress signals. The source drone will ascend and transmit the data to the next drone (drone two). Upon receiving the data packet, drone two ascends and will reply with an acknowledgement with a hop count of 0. Drone three (drone receiving a data packet from drone two) will ascend and reply to drone two with an acknowledgement packet with a hop count of 0. Drone two will increase the hop count to 1 and send it to the source drone. The source drone will ground when receiving the 1 hop acknowledgement. When a link failure occurs, the drone will use its secondary route in its routing table. The base station will also send a second acknowledgement with a hop count of 1 to notify the drone one hop away to land.
Each time a drone receives a data packet, it will add its address to the route history field. The acknowledgement will follow the route history field containing the drone addresses to the source.

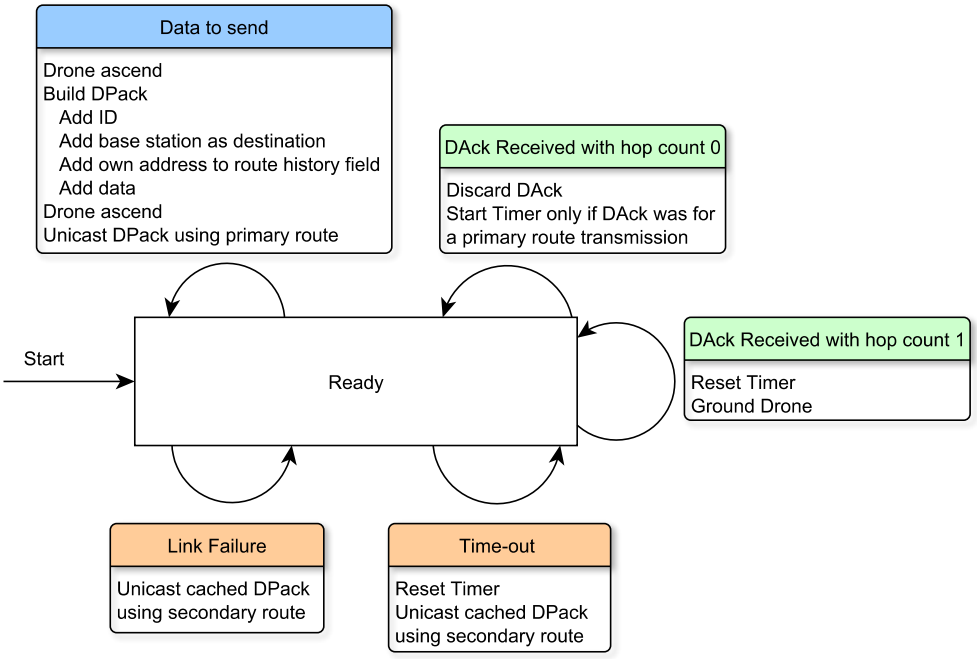


Figure 4.8: *Data send* (Sender - a drone)

Figure 4.8 description:

The FSM starts and is always in the ready state.

A drone has data to send: It builds a data packet containing an ID, the base station as the destination, its address in the route history field and the data. The drone then ascends to be able to connect to the next and unicasts the packet using the information in its routing table.

Data acknowledgement packet received with a hop count of 0: This is an indication that the current drone is the prior communication node, and the packet is discarded. If the acknowledgement packet is the address that is a primary route entry a timer is started to determine if the next drone (primary address) was able to forward the packet. A timer is not started for a packet sent to the secondary route entry drone to prevent loops.

Data acknowledgement packet received with a hop count of 1: The timer is reset and the drone grounds (It is the second preceding hop).

Time-out: This means that the next drone (primary address in the routing table) was not able to forward the data. The drone resends the data packet using its secondary route entry.

Link failure: The data link layer notifies the network layer that a link failure occurred. Data is then sent using its secondary route entry.

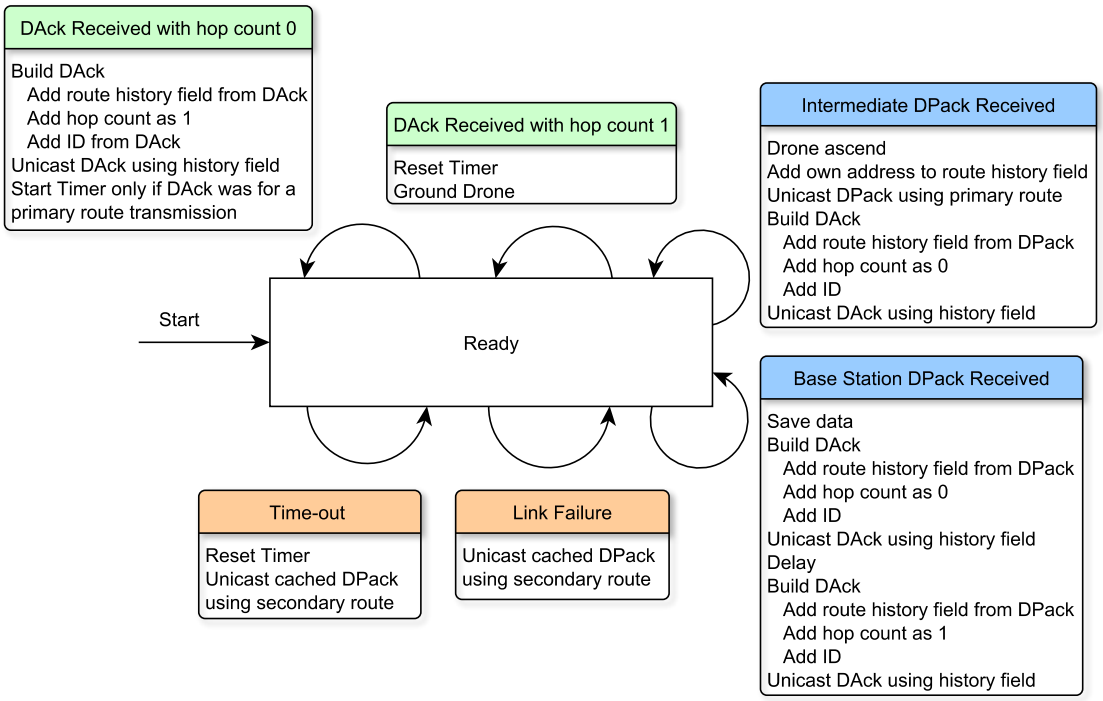


Figure 4.9: *Data send* (Receiver - drones and base station)

Figure 4.9 description:

Data acknowledgement packet received with hop count of 0: The drone rebuilds the data acknowledgement packet containing the ID from the received packet, the hop count of 1 and the route history field. The packet is unicasted using the route history field. If the acknowledgement was from a primary routing table entry, the timer is started.

Data acknowledgement packet received with hop count of 1: The drone is the second preceding hop, and the drone grounds. The timer is also reset.

Intermediate data packet received: The drone will ascend to be able to communicate to the preceding and next drone. It will add its address to the route history field and unicast the data packet using its routing table information. The drone builds the data acknowledgement packet containing the ID from the received packet, the hop count of 0 and the route history field. The packet is unicasted using the route history field.

Base station received a data packet: The data is extracted from the packet and stored. The base station builds two data acknowledgement packets containing the ID from the received packet, the hop count of 0 and 1 consecutively and the route history field. The packets are unicasted using the route history field with a delay in between.

Link failure: The data link layer notifies the network layer that a link failure occurred. Data is then sent using its secondary route entry.

Time-out: This means that the next drone (primary address in the routing table) was not able to forward the data. It resends the data packet using its secondary route entry.

Position Update

The *position update* is implemented the same way as the *route update*. The drones will receive their next position data rather than their routing tables. Another difference is the repositioning trigger. It is implemented the same way as the *landing update* and is also included in the *position update* FSM rather than in a separate one.

Figure 4.10 description:

The FSM starts on ready state and waits for a *position update* request. It ignores the rest.

Position update request: It considers the predefined primary routes. The full route from the base station to the drone is determined. It builds a position update packet containing an ID, a full route to the destination and the destination's position data. The ID and the destination address combined are always unique. The position update packet

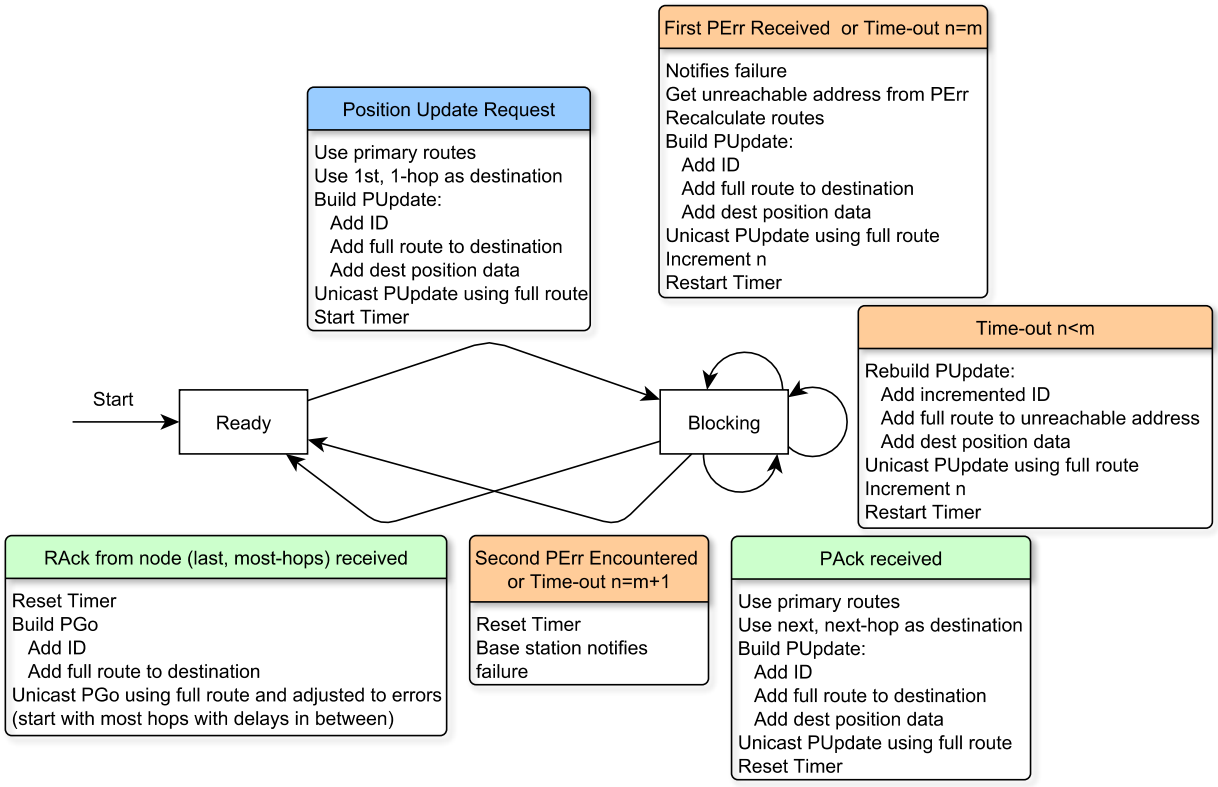


Figure 4.10: *Position update* (Sender - base station)

is unicasted using the full route information. A timer is started that is used to estimate when a packet is lost.

First position error packet received or time-out $n = m$: n is the number of time-outs with m the maximum. This means that one of the drones did not receive its position data and will not be able to reallocate itself, and therefore a notification is created. The base station extracts the address of the unreachable drone from the position error packet. For a time-out, the unacknowledged position update destination address is set to be the unreachable address. It then recalculates the routes to facilitate the route failure. The base constructs a new position update with a unique ID destination combination by incrementing the ID. The new position update packet is unicasted using the full route field, n is incremented, and the timer is restarted.

Time-out $n < m$: A new position update packet with its incremented ID is unicasted to the unacknowledged address. n is incremented, and the timer is restarted.

Position acknowledgement packet received: It reacts the same as when a *position update* request is initiated. The difference is that it uses the next, next hop node from the predefined routes.

Second route error encountered or time-out $n = m + 1$: The timer is reset, and a notification is created because too many failures occurred.

The last (the last node with the most hops) position acknowledgement packet received: The base station builds a position update packet. It contains an ID and the full route to the destination. The full route was adjusted to failures that occurred during the *position update*. The position go packet is then unicasted starting with the drones with the most hops. It unicasts the next packet after a delay to avoid collisions.

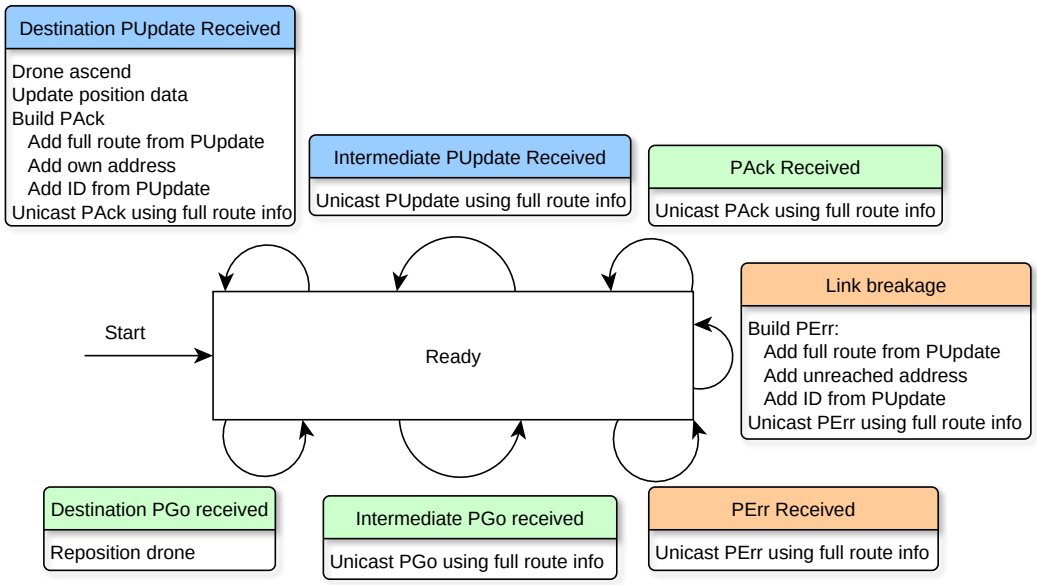


Figure 4.11: *Position update* (Receiver - drones)

Figure 4.11 description:

The FSM starts and is always in the ready state.

Destination position update packet received: The drone will first ascend to be able to communicate to the preceding node. It will update its position data that are extracted from

the position update. The acknowledgement is then built. It adds the full route from the position update, its address and the ID from the position update. The position acknowledgement is then unicasted using the full route field.

Intermediate position update packet received: The position update packet is unicasted using the full route field.

Link failure occurs: A position error is built. It adds the full route from the position update, the unreachable address and the ID from the position update. The position error is then unicasted using the full route field from the packet.

Position error packet received: The position error packet is unicasted using the full route field from the packet.

Position acknowledgement packet received: The position acknowledgement packet is unicasted using the full route field from the packet.

Destination position update packet received: The drone is repositioned.

Intermediate position update packet received: The position update packet is unicasted using the full route field from the packet.

4.5 Formal Protocol Specification

According to [33] the components of a formal protocol specification are:

- Communication service
- Protocol entity specification
- Communication interfaces
- Interactions
- Message formats

4.5.1 Communication Service

The communication service is the collection of the basic services the protocol has to supply. It is divided into two sections namely service specification and specification of the behaviour aspects of a protocol.

Service Specification

The service specification is based on service primitives such as *request*, *response*, *indication* and *confirm*. During connection establishment, all four primitives are used, but in data transfer only *request* and *response*. This protocol will utilise all the primitives in the data link layer to ensure data transfer. The network layer will be incorporated as data transfer, using only the *request* and *response*.

Specification of the behaviour aspects of a protocol

The behaviour aspects of a protocol ensure that a particular implementation of a protocol is specified. This specification limits the features and keeps it simple. It prevents one from over specifying the protocol as prescribed in [34]
The behaviour aspects include:

- Temporal ordering of interactions
  The *data send* can only be executed when the *route update* function has completed.
- Parameter range
  Parameter range is to describe the size of the messages or packets. It is done in section 4.5.5
- Selecting values of parameters
  These parameters are specified for the functions to be able to control and process messages. A selected value is the maximum data size: 256 bytes for one packet.

- Coding of Protocol Data Units (PDUs)
  How are the data units coded and what are their formats? These data units refer to the transmissions. The transmissions (messages) formats are described in section 4.5.5.

- Liveness properties
  What interactions will certainly happen? E.g. it is certain that when the *route update* is finished the *landing update* will execute.

- Real-time properties
  It specifies the amount of time the network should be utilised. CERATIN does not provide services to users that need real-time connectivity such as web browsing. It is specified that the network should be near real-time for the rangers to act in time when an alert is received.

4.5.2 Protocol Entity Specification

The logic of the protocol is defined in this section. The logic is described in terms of FSMs. Three entities are described namely the sender, receiver and the channel.

Sender and receiver entities

Refer to section 4.4

Sender - route update The base station is responsible for initiating and controlling the *route update* network function.

Receiver - route update All the drones will receive their routing tables.

Sender - landing update The base station will inform the drones to land.

Receiver - landing update The receiving drones will land.

Sender - data send The drone that has data to transmit will initiate the *data send* network function.

Receiver - data send The base station and the drones receiving the data packets. Receiving drones will forward the data.

Sender - position update Base station initiates and controls the *position update* network function.

Receiver - position update All drones will acknowledge the receiving of the position update and will reposition itself when receiving the position go packet.

Channel entity

It is designed as First In First Out (FIFO) channel. The first message that enters the channel is the first message that exits. Multiple messages are handled as a queue and not a stack.

4.5.3 Communication Interfaces

The collaborations of the layers and functions. The collaborations of the layers are called external interfaces, and the collaborations of the functions are called internal interfaces. Figure 4.12 depicts the designed protocol's communication interfaces.

External interfaces:

Network layer - data link layer (g) The encapsulation from packets into frames and decapsulation from frames to packets. Another interaction is when the data link layer notifies the network layer that a link failure occurred.

Data link layer - physical layer (h) The encapsulation from frames into bit strings and decapsulation from bit strings to frames.

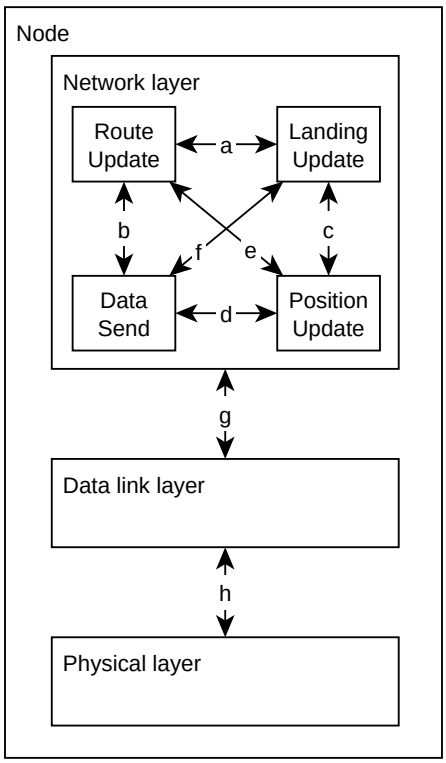


Figure 4.12: Communication interfaces

Internal interfaces:

Route update - landing update (a) After the *route update* function is completed the landing update function should be initiated.

Route update - data send (b) No direct interface.

Landing update - position update (c) No direct interface.

Data send - position update (d) No direct interface.

Route update - position update (e) No direct interface.

Data send - Landing update (f) No direct interface.

4.5.4 Interactions

As a network model is constructed each component in another node's model has corresponding objects. It can be seen as the logical link between the layers and sub-layers of the node's models.

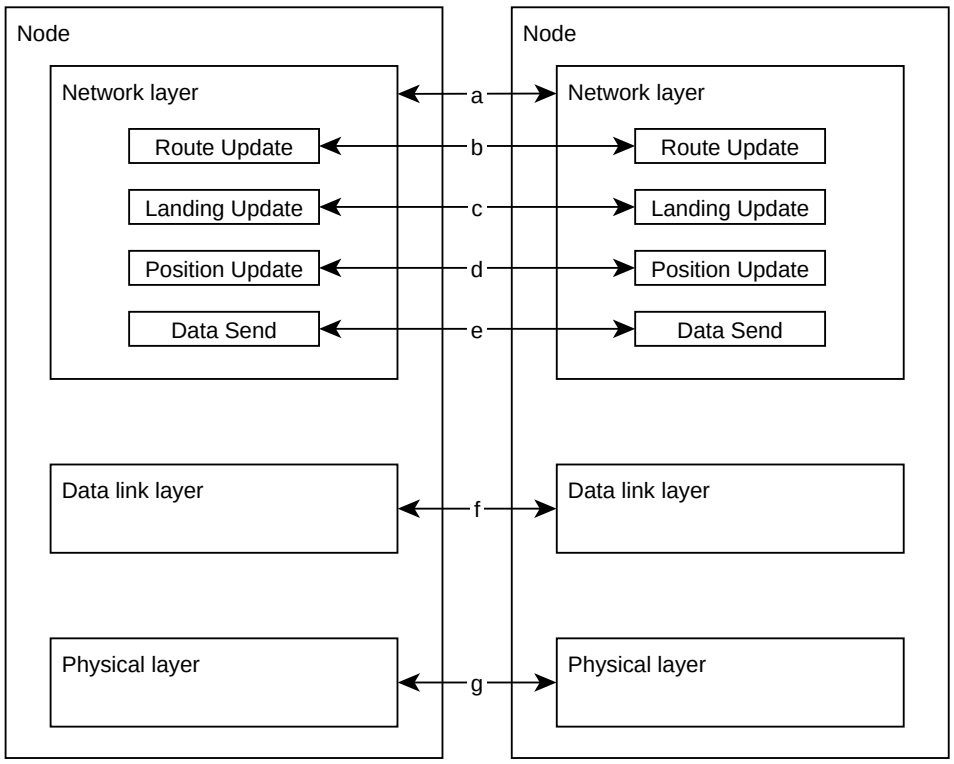


Figure 4.13: Interactions

Protocol interactions:

Network layer - network layer (a) A packet. The network layer is only aware that a packet arrived from another node's network layer.

Route update - route update (b) It interacts with a route update (containing the routing information), route acknowledgement and route error packet.

Landing update - landing update (c) Landing update packet

Position update - position update (d) Position update, position acknowledgement, position error and position go packets.

Data send - data send (e) Data packets and data acknowledgement packets.

Data link layer - data link layer (f) Frames

Physical layer - physical layer (g) Bit strings

4.5.5 Message Formats

The messages are presented in the FSMs in section 4.4. This section is a formal and technical presentation of the message format.

The message types are encoded to hexadecimal values as follows:

Table 4.1: Encoded values of messages types

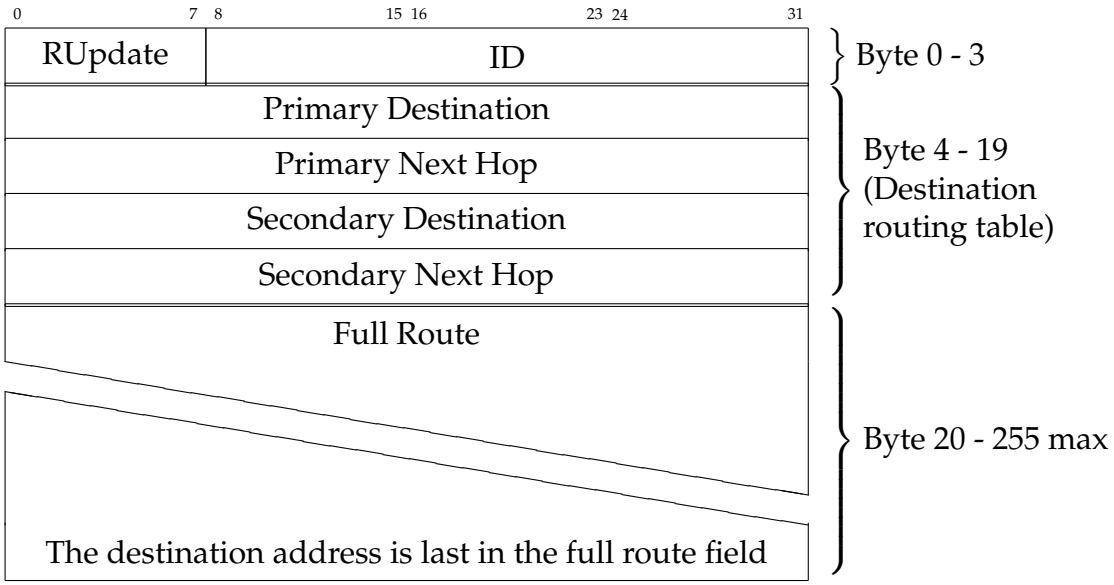
Message type	Encoded value	Message type	Encoded value
	Network layer		
RUpdate	00_H	DAck	21_H
RAck	01_H	PUpdate	30_H
RErr	02_H	PAck	31_H
LUpdate	10_H	PErr	32_H
DPack	20_H	PGo	33_H
	Data link layer		
SRequest	$F0_H$	Ack	$F1_H$

The ID field of the messages is three bytes long (ranging from 000001_H to $FFFFFF_H$) which allows a specific messages to have a maximum of 16777215 unique IDs. IP version 4 addresses are used in these message formats. An IP version 4 address can be written as eight hexadecimal values e.g. $C4.FC.BD.52$ resulting in 4 bytes. The longitude and latitude uses a 4 byte number each.

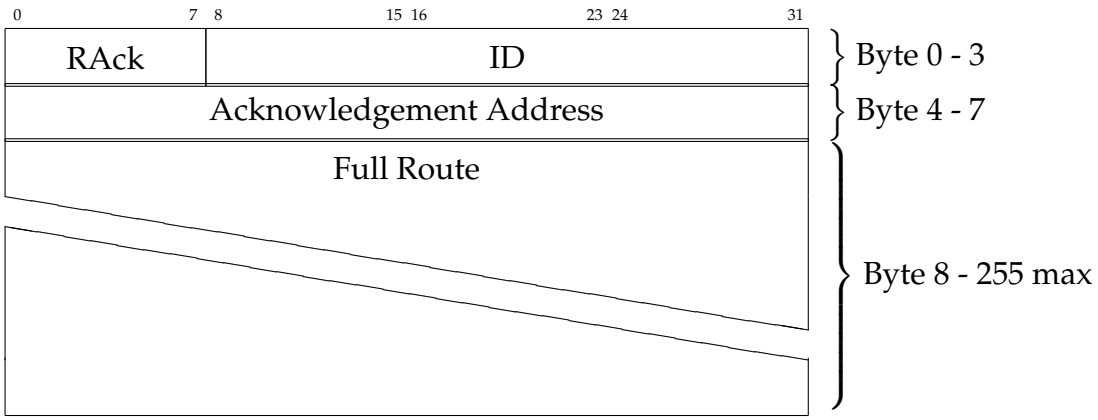
Network layer

The network is designed to allow at least 50 nodes which mean the full route field should be able to support 50 addresses. The full route field is 236 bytes long, and therefore supports 59 IP version 4 addresses of 4 bytes each.

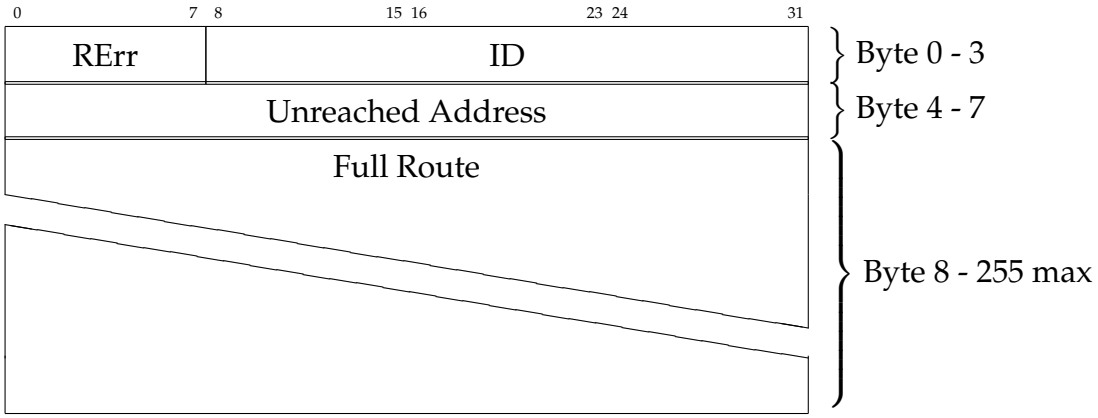
Route update:



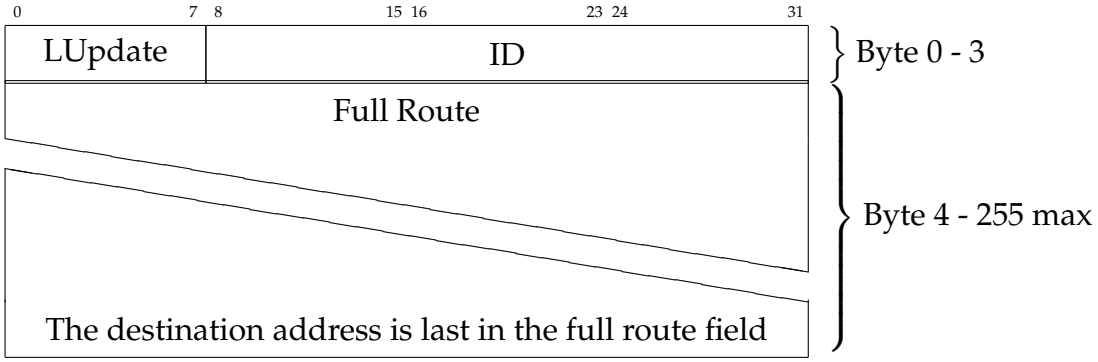
Route acknowledgement:



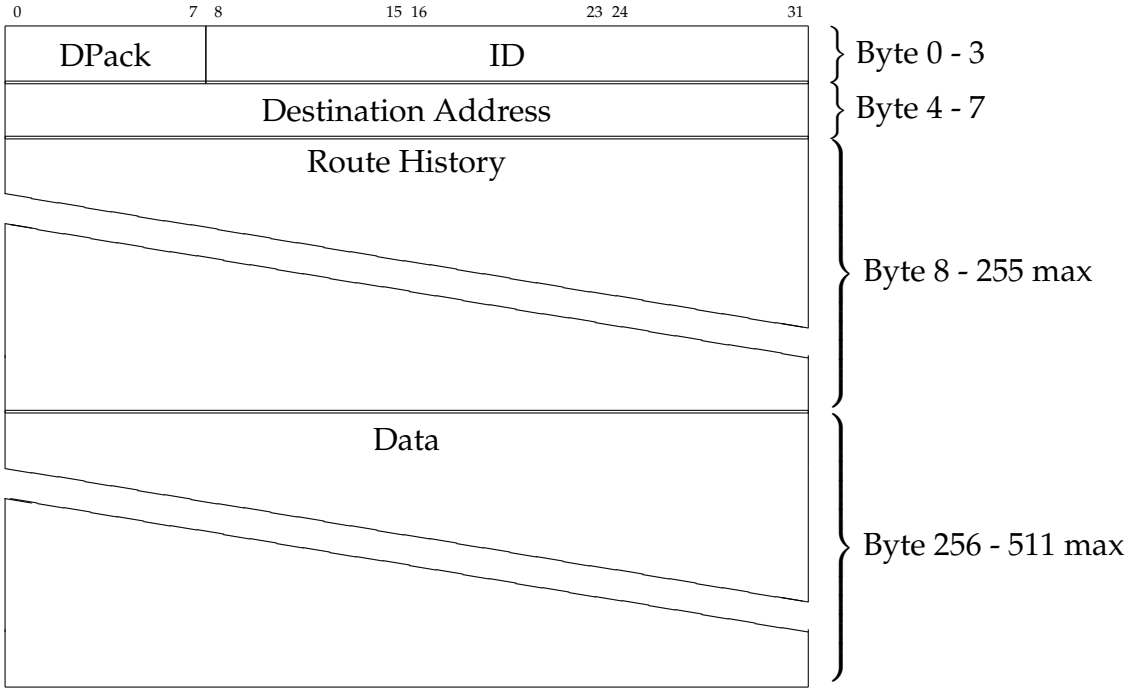
Route error:



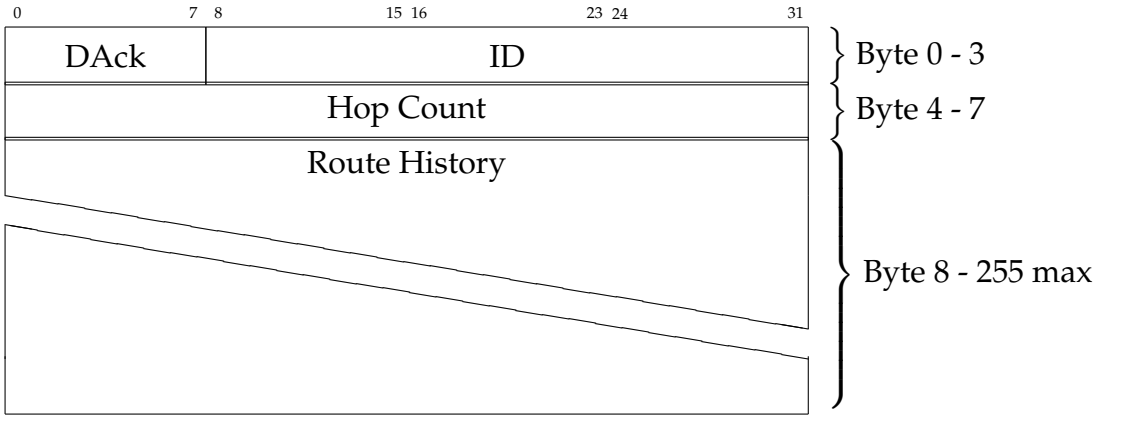
Landing update:



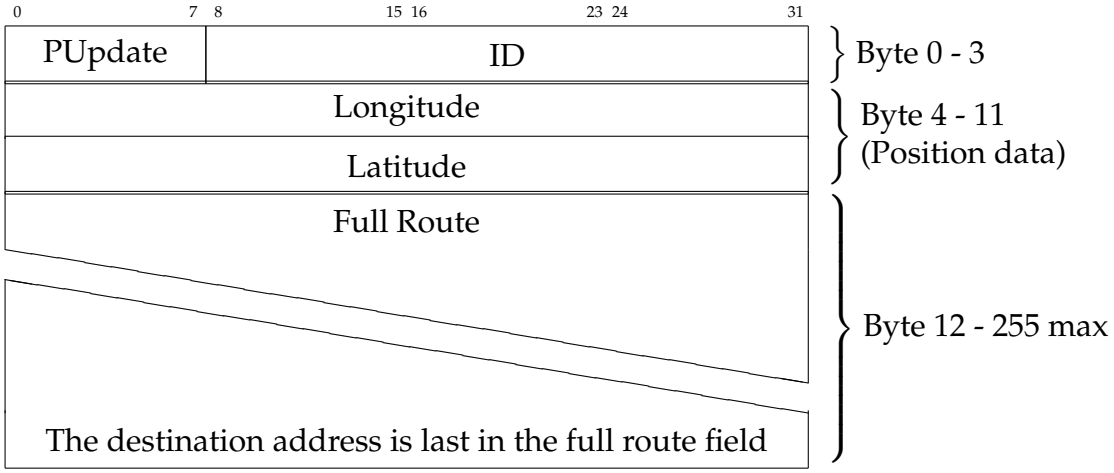
Data packet:



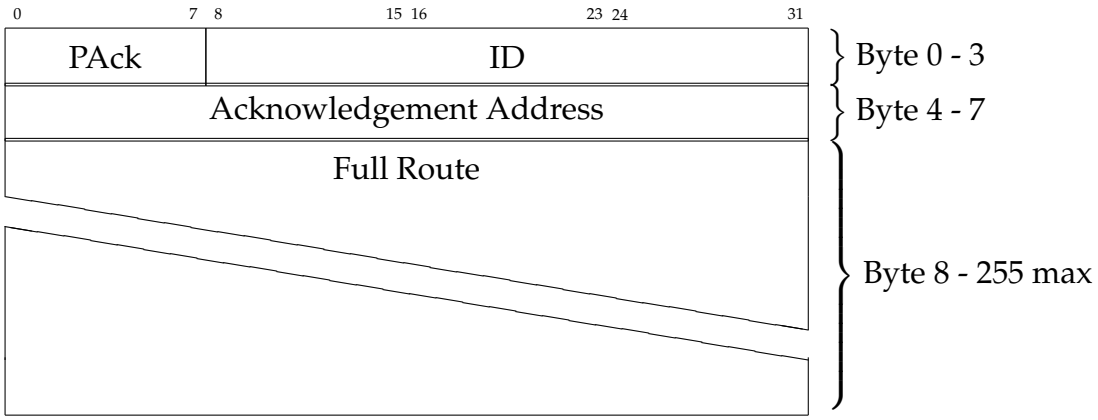
Data packet acknowledgement:



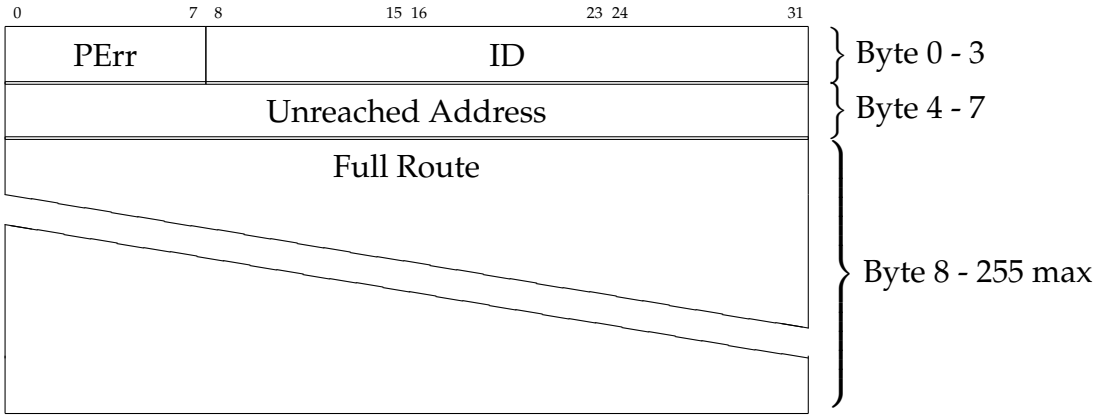
Position update:



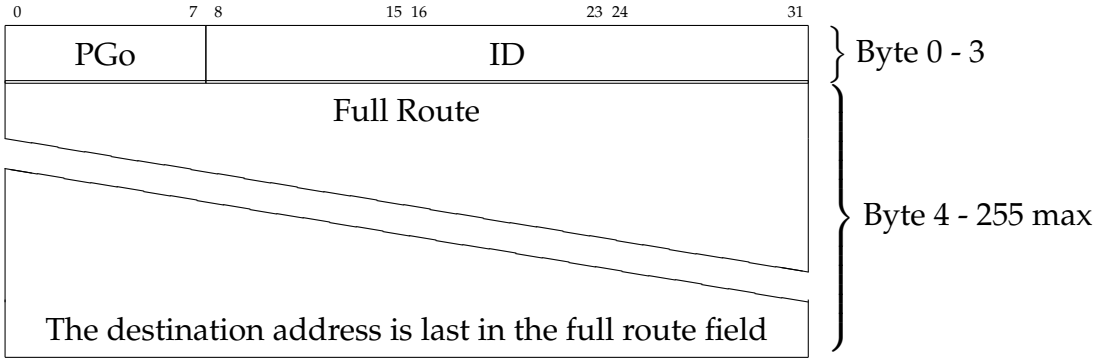
Position acknowledgement:



Position error:



Position go:

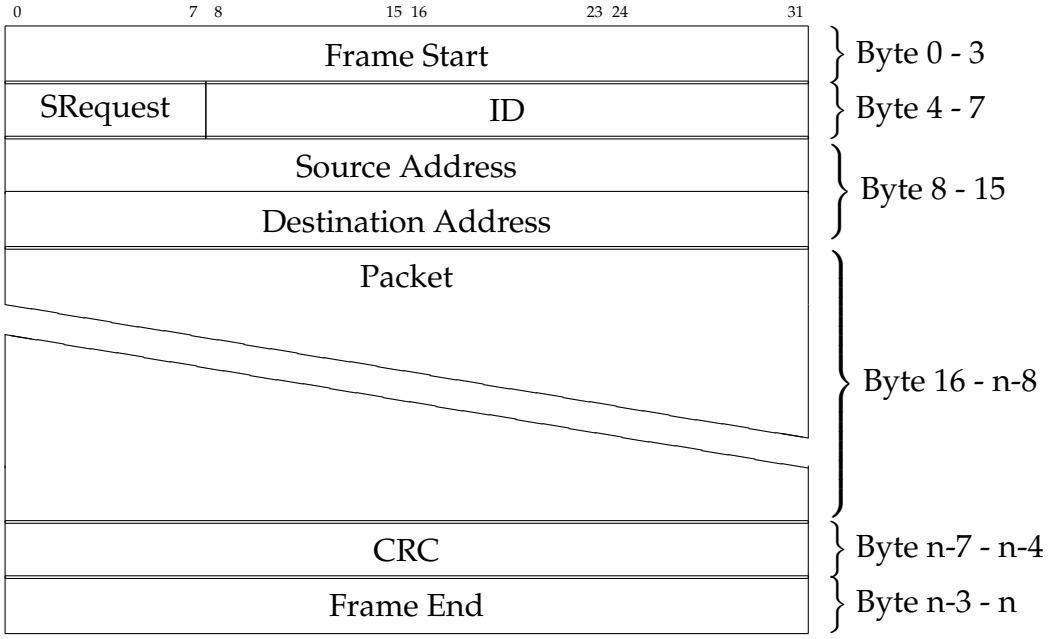


Data link layer

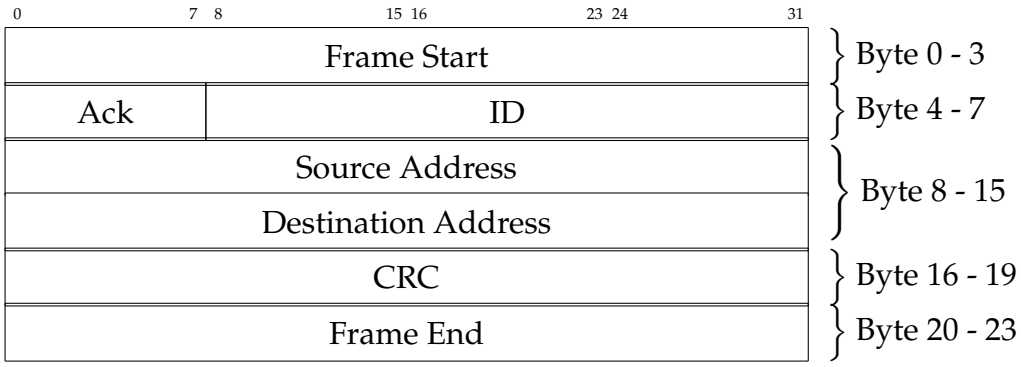
The frame start and end fields use a 32 bit flag. This is a relatively long flag to reduce the need of bit stuffing[§]. The 32 bit CRC is determined by the encoder (sender) using the frame data (dataword) and the standard CRC-32 polynomial as divisor (also referred to as the generator). The decoder (receiver) then determines if an error occurred in the received message using the codeword (dataword and CRC code) and the generator.

[§]Adding a bit in the frame data where a frame start or end flag sequence is encountered. This process is implemented to inform the receiver that the encountered sequence is not a frame start or end flag. [9]

Send request:



Acknowledgement:



4.6 Discussion

In the design process; the constraints and requirements are specified, the network functions are then determined.
The network functions that were determined from the constraints and requirements are:

- *Route update*, where the drones' routing tables are configured.
- *Landing update*, where the base station will inform the drones to return to the ground.
- *Data send*, when a drone has to transmit status updates and distress signals to the base station.
- *Position update*, where the base station will inform the drones to fly to a new position.

Each of these network functions was described in FSMs as verification and to be implemented in a simulation or machine.
In the analysis and design, each component is verified against its origin in the literature study, e.g. when source routing is implemented it is compared with its original implementation in DSR. For example, if it has the same behaviour (resulting in variable length messages), the design process is continued.

As a further verification, the network functions of CERATIN are implemented and tested using a simulation in the next chapter.

Chapter 5

Functional Implementation

The protocol specification is implemented as a simulation to verify the network functions and test their coherence. It further serves as validation that the protocol operates as prescribed by its requirements.

5.1 Introduction

The coherency of the network functions was tested by implementing them in a simulation environment. The focus of the simulation was to implement and test the functions. Java was used as the programming language in this simulation. It is an Object Oriented Programming Language which is suitable for modular network implementations.
The supplied services, e.g. the drones' and base station's positions, and the drones' routes were specified in text files. The main class of the program dynamically creates a base station object and drone objects. The base station and drone classes inherit from node class (The base station and drones are nodes). Sender and receiver objects running on different threads were used to implement the connection channel between the nodes. Node ranges and the base station address were initialised in a global class. The

message class shows the contents of the different messages, and the main class also controlled the Graphical User Interface (GUI) used to record results. The simulation consisted of ±2200 lines of code.
The finite state machines presented in section 4.4 were implemented on the nodes to verify that they will perform the desired operations and to test their coherency.

5.2 FSM Implementation

The FSMs were implemented as switch statements. A switch statement is shorter and more structured than a nested if statement (an if statement is a popular conditional statement). When a single code block (matching a certain condition) of many has to be executed, a switch statement is used.

Algorithm 1 is a switch statement based on the FSM displayed in figure 5.1. The algorithm is presented as pseudo code for simplicity.

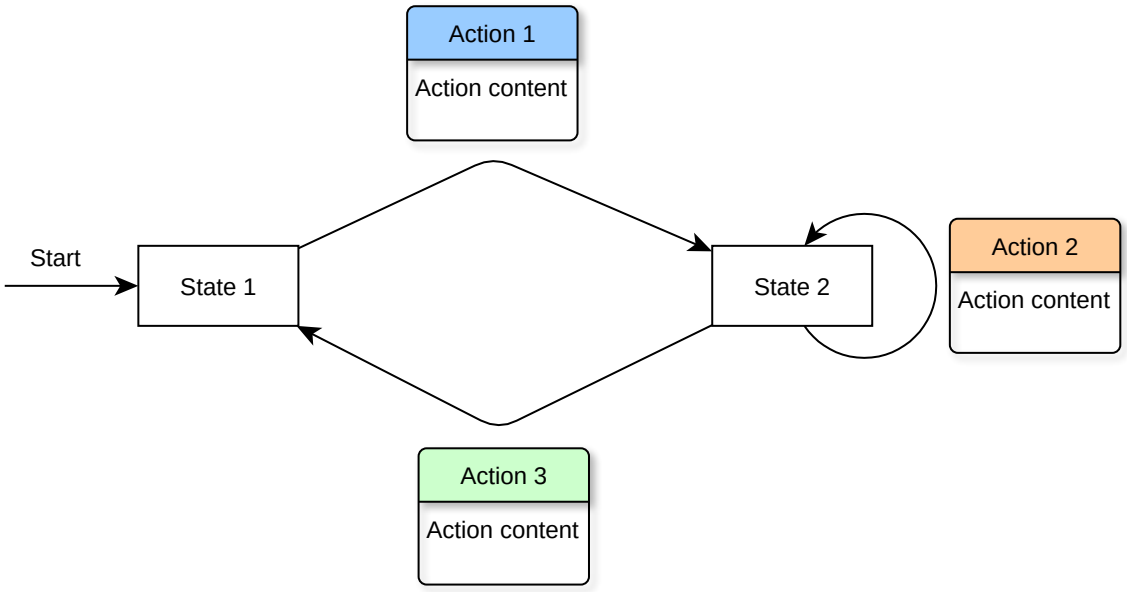


Figure 5.1: Finite State Machine

Algorithm 1 FSM as a switch statement

Initialise:

Create list containing all actions

Create list containing all states

Create a map that matches all runnable actions to their states

Set current state as State 1

Running the FSM:

switch (Action to run)

case Action 1**:**

  if Action 1 is matched with current state in map **then**

    Action 1 content;

    Current state = State 2;

  end if

case Action 2**:**

  if Action 2 is matched with current state in map **then**

    Action 2 content;

    Current state = State 2;

  end if

case Action 3**:**

  if Action 3 is matched with current state in map **then**

    Action 3 content;

    Current state = State 1;

  end if

default:

  Unknown action was run

end switch

The FSMs described in section 4.4 show the basic logic and decision making when an event occurs. They do not show the algorithms used to detect their events. These algorithms first had to be formulated.

5.3 Key Algorithms

Here the algorithms are discussed that are not shown by the FSMs. Most of the algorithms are implemented into more than one FSM. The affected FSMs are also presented. For the Java implementation of the algorithms refer to appendix A

5.3.1 Determining the routes for the base station (Full routes to the drones)

Route update (Sender)
Landing update (Sender)
Position update (Sender)
Initially, the base station only has the routes from the drones to the base station. The full route from the base station to the drone is the reverse of the supplied routes.

5.3.2 Send packets starting with routes with the least hops

Route update (Sender)
Landing update (Sender)
Position update (Sender)
The supplied routes are not in any specific order. This algorithm calculates the primary route lengths and sorts them from short to long. The base station sends the updates, following the sorted list.

5.3.3 Calculating the drones' routing tables

Route update (Sender)
The routing tables are calculated by removing the first address from the supplied routes.

5.3.4 Updating the routes according to a route failure (Recalculate routes)

Route update (Sender)
Position update (Sender)
All the routes are changed to follow the path around the unreachable address. The changed routes are used to create the update packet paths and the routing tables of the drones.

5.3.5 Determine if all drones received the packet

Route update (Sender)
Position update (Sender)
The base station keeps track of the sent and received messages. Every sent message is unique and has a unique acknowledgement. The base station determines if the number of sent and received message pairs is the same as the number of drones.

5.3.6 Drone determining if it is the destination or intermediate

Route update (Receiver)
Landing update (Receiver)
Position update (Receiver)
The drone is the destination if its address is the last in the full route field of the update packet. If not the drone is regarded as an intermediate node.

5.3.7 Identify a network link failure

Route update (Receiver)
Position update (Receiver)

A network link failure is encountered when the data link layer was unable to deliver the packet to the neighbouring node.

5.4 Functional Testing

Two networks were set up for the simulation tests. Network one (figure 5.2) has one base station and 9 drones, and network two (figure 5.3) has one base station and 26 drones.

Network one

The inputs (text files contents) are shown in table 5.1 (the node positions) and table 5.2 (the routing information). The coordinates are in metres, with x horizontally and y vertically. x increases to the right and y downwards. Figure 5.2 depicts the physical layout of network one, and serves as a visualisation and is not an output of the simulation.

Table 5.1: Network one node positions

Drone	Coordinate $(x(m), y(m))$	Drone	Coordinate $(x(m), y(m))$
a	$(13520, 5210)$	f	$(8050, 6970)$
b	$(12340, 7950)$	g	$(4860, 2210)$
c	$(9830, 8070)$	h	$(1970, 2020)$
d	$(10710, 5680)$	i	$(3250, 4580)$
e	$(8750, 3540)$	Base	$(6270, 4680)$

Table 5.2: Network one routing information

Drone	Primary route	Secondary route	Drone	Primary route	Secondary route
a	adeB	abcfB	f	fB	-
b	bdeB	bcfB	g	gB	giB
c	cfB	cdeB	h	hgB	hiB
d	deB	dfB	i	iB	-
e	eB	-	-	-	-

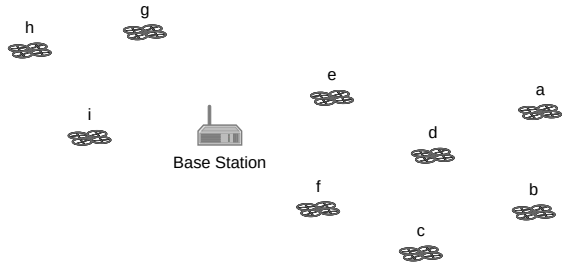


Figure 5.2: Network one

Network two

The inputs are shown in table 5.3 (the node positions) and table 5.4 (the routing information). Figure 5.3 depicts the physical layout of network two and serves as a visualisation and is not an output of the simulation.

Table 5.3: Network two node positions

Drone	Coordinate $(x(m), y(m))$	Drone	Coordinate $(x(m), y(m))$
a	$(5764, 10736)$	o	$(22000, 11316)$
b	$(6523, 6419)$	p	$(20951, 12931)$
c	$(5109, 8451)$	q	$(19303, 10549)$
d	$(8983, 6052)$	r	$(18067, 12325)$
e	$(10161, 11047)$	s	$(17019, 9620)$
f	$(8093, 8902)$	t	$(15296, 11437)$
g	$(2780, 10285)$	u	$(14921, 8530)$
h	$(30089, 9782)$	v	$(12862, 9782)$
i	$(32353, 11349)$	w	$(13012, 6915)$
j	$(29611, 12424)$	x	$(22462, 9212)$
k	$(27318, 10629)$	y	$(19609, 8253)$
l	$(26868, 13173)$	z	$(21808, 6334)$
m	$(24883, 11074)$	Base	$(10951, 8222)$
n	$(23872, 13254)$	-	-

Tests were formulated, and performed on these two networks to verify and validate the network functions. The formulated tests will allow one to investigate each network function by answering the following questions:

- Do all the drones receive the correct routing table?
- Did all the drones return to the ground after finishing the route updates?

Table 5.4: Network two routing information

Drone	Primary route	Secondary route	Drone	Primary route	Secondary route
a	afB	acbdB	n	nprtvB	noqsuwB
b	bdB	bfB	o	oqsuwB	oprtvB
c	cfB	cbdB	p	prtvB	pqsuwB
d	dB	-	q	qsuwB	qrtvB
e	eB	evB	r	rtvB	rsuwB
f	fB	fdB	s	suwB	stvB
g	gafB	gcbdB	t	tvB	tuwB
h	hkmoqsuwB	hjlnprtvB	u	uwB	uvB
i	ijlnprtvB	ihkmoqsuwB	v	vB	vwB
j	jlnprtvB	jkmoqsuwB	w	wB	-
k	kmoqsuwB	klnprtvB	x	xysuwB	xoprtvB
l	lnprtvB	lmoqsuwB	y	ysuwB	yqrtvB
m	moqsuwB	mxysuwB	z	zysuwB	zxoprtvB

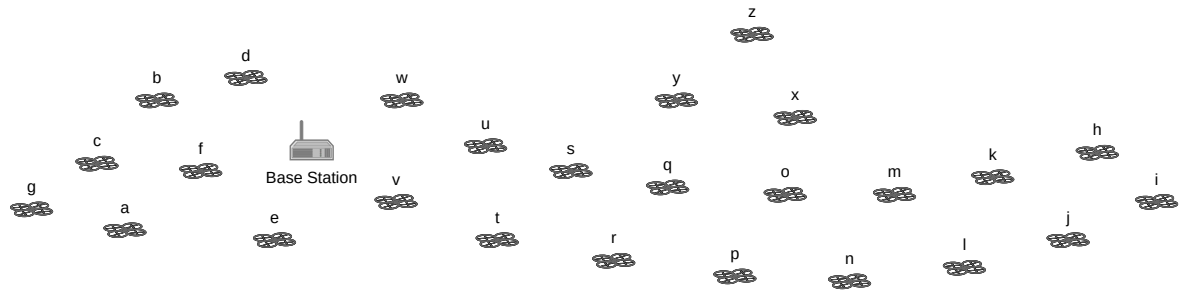


Figure 5.3: Network two

- Does the base station receive data from every drone?
- Do all the drones receive a position update?

The results of the functional tests are recorded as follows (recording methods):

1. *Route update* request is initiated in the base station. The number of drones that received the routing table is recorded.
2. After the route updates are completed the number of drones that returned to the ground are recorded.

3. *Data send* is then initiated in every drone, and the next hop node is recorded and compared to its primary entry in its routing table.

4. The primary next hop node is forced to remain grounded. *Data send* is initiated in every drone, and the next hop node is recorded and compared to its secondary entry in its routing table.

5. In methods 3 and 4 the number of successful data deliveries to the base station are recorded.

6. *Position update* request is initiated in the base station. The number of drones that received the position update is recorded.

The mentioned recording is done by evaluating the GUI. As an example the *data send* wave formation is shown in appendix B.

Different scenarios are used to determine how the network functions will handle networks with complications. Above mentioned functional tests were performed in the following scenarios:

- The ideal case (no forced link failures).
- Link failure before *route update*.
- Link failure during *data send*.

5.4.1 The ideal case

In this scenario, no link failure is induced into the network. The ideal test serves as a baseline, verifies the designed protocol and validates that the specification is met.

Table 5.5: Ideal test results

Recording method	Network one	Network two
1. Drones received a route table	9	26
2. Drones returned to the ground	9	26
3. Drones with correct primary route	9	26
4. Drones with correct secondary route	9	26
5. Drones the base station received data from	9	26
6. Drones received a position update	9	26

The ideal case discussion

- All the drones received a routing table. This is an indication that the base station utilised the routes to the drones correctly, and the routes were followed correctly. The base station - drone environment is also correctly implemented.
- All the drones were grounded after finishing the route updates. This demonstrates that the drones did receive the landing updates in the correct order and forwarded them correctly.
- Both the network's drones' primary and secondary routes are set up correctly. This is an indication that the base station calculated each drone's routing table correctly and was successfully received by all drones.
- The base station received data from all the drones. This is an indication that the wave formation is correctly implemented.
- The *position update* is correctly implemented because all the drones received a position update.

The fact that the network functions as a whole and that one function follows correctly after the other is an indication that the functions are operating coherently.

5.4.2 Link failure before *route update*

A drone is forced to remain grounded before the *route update* request is initiated. This will serve as verification that the route recalculation is correctly implemented.

Table 5.6: Link failure before *route update* test results

Recording method	Network one	Network two
1. Drones received a route table	9	26
2. Drones returned to the ground	9	26
3. Drones with correct primary route	8	25
4. Drones with correct secondary route	8	25
5. Drones the base station received data from	8	25
6. Drones received a position update	8	25

Link failure before route update discussion

- All the drones received a routing table. The drone that did not ascend also received a route table, but could not acknowledge it. The drones with more hops than the grounded drone also received routing tables. This is an indication that the base station recalculated the routes to the drones correctly and that they were successfully delivered.
- All the drones were grounded after finishing the route updates. This shows that the drones did receive the landing updates in the correct order and forwarded them correctly.
- Both the network's drones' primary and secondary routes are set up correctly, except the grounded drone. The base station also received data from all the drones, except the grounded drone. This is an indication that the base station recalculated each drone's routing table correctly and all drones received it successfully.
- The *position update* is correctly implemented because all the drones received a position update except the grounded drone.

5.4.3 Link failure during *data send*

This scenario will test if the *data send* function can handle link failures.
A drone is forced to remain grounded before data is sent from the drones and the results for *data send* are recorded (recording method 5).

Table 5.7: Link failure during *data send* test results

Recording method	Network one	Network two
5. Drones the base station received data from	8	25

Link failure during data send discussion

- The base station received data from all the drones, except the grounded drone. This is an indication that the wave formation is correctly implemented and that *data send* function can handle complications such as link failures.

5.5 Discussion

The finite state machines were implemented in a simulation environment to test coherency of the network functions. A modular programming language; Java was used. Two networks were set up for the simulation tests and further verify and validate the designed protocol. The tests were performed on three scenarios, named the ideal case, link failure before *route update* and link failure during *data send*.
The first scenario's test results show that the network functions operate coherently and correctly.
When a link failure was forced before *route update*, the network test results show that the base station recalculated the routes correctly, and the network responded accordingly.
The third scenario's test results show that the *data send* network function can handle

complications such as link failures.

The behaviour of each network function was evaluated against the requirements in chapter 2. If they are found to be the same, it is considered valid, for example, the requirements state that the *data send* should operate in a wave formation. It is observed to be the case (appendix B).

This chapter verified that the first research goal: design and specify a communication protocol for the proposed drone coverage system; was achieved. To reach the second goal, i.e. determine how well it will operate in the proposed system; the following tests (in chapter 6) were conducted.

Chapter 6

Performance Testing

The second research goal is to determine how well the protocol performs. This is achieved by conducting performance tests. The tests are performed using a Monte Carlo simulation. The setup and results are then presented and discussed.

6.1 Protocol Performance Metrics

The performance or quality of service a protocol offer can be measured in four ways namely *bandwidth*, *reliability*, *delay* and *jitter* [9].

6.1.1 Bandwidth

The bandwidth of a network is how much (capacity) information the network can handle. How much information is actually transferred (throughput) is always less than the bandwidth and is a much more reliable measurement of a network's performance.

This network is not user reliant. The capacity is therefore not a good estimation of how

well the protocol performs. A more appropriate measurement is if the data is delivered and not how much data the network can deliver.

6.1.2 Reliability

The reliability of a network is to what extent the information is transferred safely. A network that loses many packets is not reliable.

CERATIN should be reliable. Reliability tests will be performed on the protocol.

6.1.3 Delay

How long does it take for information to propagate through the network? It is considered more desirable for a network to have as low as possible delay.

Delay is a measurement that can be used to estimate the drone flight times and therefore battery life. It is therefore a favourable performance test measurement.

6.1.4 Jitter

The jitter of a network is to what extent the delay of the network varies. It is usually determined by one transmission (the same flow). Does one portion of the information transmitted have a different delay than the other? Does the destination receive the packets in the wrong order?

The drone system has many real world factors contributing to the jitter and can therefore be more accurately determined when implemented. In this case, jitter is reliant on the system and not the protocol.

6.2 Monte Carlo Simulation

The performances of the protocol will be determined using a Monte Carlo simulation. A Monte Carlo simulation is a way to determine the output of a complex system without having to characterise or describe the system in a model [35]. The input is random samples with fixed parameters, and the output is described/explained using a statistical analysis.

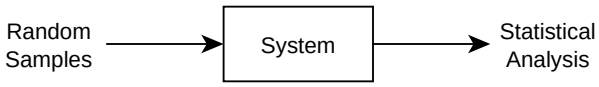


Figure 6.1: Monte Carlo analysis

In this case, the system is the simulated drone network model, and the input is communicational scenarios.

The more samples you have, the higher the accuracy of the results.

6.3 Simulation Setup

The Java simulation used in the functional implementation was modified and improved to form a Monte Carlo simulation (the program code is provided in appendix C). The random sample inputs of the Monte Carlo simulation, in this case, are induced failures that occur in the network. The statistical analysis is done on the delays and reliability that the protocol delivers.
The probability that a failure will occur is controlled, but when and where it will occur are random.
The simulation starts by running a network function multiple times at a low failure probability, and the delay and reliability are recorded. When complete, the probability is increased, and the network function is run again. When the ending probability is reached the recorded delays and reliabilities in relation to their probabilities are pro-

duced to be analysed.

The simulation setup with its interactions is portrayed in figure 6.2. Its description follows.

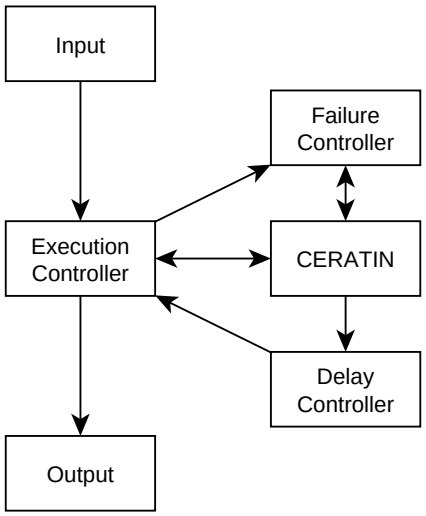


Figure 6.2: Performance testing simulation setup

Input

The simulation inputs are:

- Node positions
- Routing data for the base station
- Estimated network delays
- Probability inputs for every network function
  - Starting probability
  - Probability increment
  - Ending probability
  - Number of runs per probability

Execution controller

Manages the whole simulation.

- It supplies the failure probability to the failure controller and runs the network function the supplied number of times.
- The duration of the function error is collected for every run.
- The duration (delay) and errors (reliability) with their corresponding probability are produced as output.

Failure controller

- It receives the probability of failure from the execution controller.
- When a transmission occurs or a node is used the failure controller determines whether it should fail using algorithm 2.
- Using algorithm 2 each transmission or node has a deterministic probability to fail but the time and place are random.

Algorithm 2 How a failure is determined

random = random number between 1 and 10000; {10000 is used to have a probability of 4 decimals}
if $(random \leq probability \times 10000)$ **then**
  Failure does occur;
else
  Failure does not occur;
end if

CERATIN

This component is the implemented protocol into the simulated network.

- The execution controller starts the network function.
- When the function is finished or an error occurred the execution controller is notified.
- When the protocol uses a node or transmits data, it interfaces with the failure controller to determine the working of the node or the success of a transmission.
- The durations of the network elements are communicated to the delay controller. The durations are defined constants that are summated every time the network encounters a delay. The simulation would run too long if it was executed in real-time. As a result, the execution time of the simulation code does not have an effect on the outcome.

Delay controller

All the delays are recorded during the execution of a network function and passed on to the execution controller.

Output

A Matlab script file is created containing the analysis data and graph plotting functions.

6.4 Testing Configuration

For consistency, a test network was used consisting of a base station and seven drones as figure 6.3 depicts. The position coordinates are in metres. The distance between communicating drones (e.g. between e and b) are 3150m (calculated using equation 6.1 from [36]), which is smaller than 3200m but larger than 800m. The drone will be able to communicate when elevated but not when grounded.

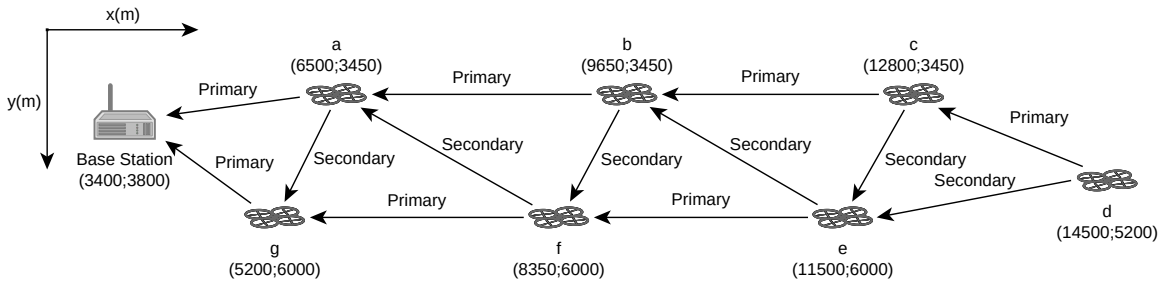


Figure 6.3: Test network

$$s_{eb} = \sqrt{(x_e - x_b)^2 + (y_e - y_b)^2} = \sqrt{(11500 - 9650)^2 + (6000 - 3450)^2} \approx 3150m \quad (6.1)$$

The routing information provided are:

Table 6.1: Performance tests routing information

Drone	Primary route	Secondary route
a	aB	agB
b	baB	bfgB
c	cbaB	cefgB
d	dcbaB	defgB
e	efgB	ebaB
f	fgB	faB
g	gB	-

The routing information is used by the base station to determine the node's routing tables and the paths to the nodes. All the delays are discussed next.

Only delays that influence the network were considered. Delays, for example, the deletion of buffers and the time a drone takes to land were not considered. The estimated network delays are:

Table 6.2: Estimated network delays

Delay name	Duration (time units)
DRONE_ELEVATE	10
DATA_LINK_TRANSMISSION	1
DATA_LINK_TIMEOUT	12
BUILD_PACKET	0.1
RECALCULATE_ROUTES	0.2
LANDING_UPDATE_NEXTSEND_DELAY	1
DATA_SEND_DACK_REPLY_DELAY	1
PGO_NEXTSEND_DELAY	1

DRONE_ELEVATE

Estimated time a drone takes to elevate. This value may vary, depending on the drone and its environment. Consider the 3drobotics drone, Solo, which has an ascent speed of $10m/s$ in stabilise mode and $5m/s$ in fly mode [37]. Solo will therefore take 1 to 2 seconds to ascend to a height of $10m$.

DATA_LINK_TRANSMISSION

Duration of a data link transmission. The largest data link packet according to the specification in section 4.5.5 is 536 bytes or 4.288 kb. The wireless Microcontroller YR-less is using in their current system is part of the Si100x series. The data rates of these Microcontrollers are between 0.123 and 256 kbps [38].

DATA_LINK_TIMEOUT

Duration the data link waits before resending the frame when a data link failure* occurred. It is determined by adding the duration of a forward data link transmission, the duration of a drone elevation and the duration of a reply data link transmission.

*A data link failure is when a frame is not acknowledged by the neighbouring node.

BUILD_PACKET

How long a node takes to build a packet.

RECALCULATE_ROUTES

How long the base station takes to recalculate the routes when a network link failure[†] occurred.

LANDING_UPDATE_NEXTSEND_DELAY

The base station waits before it sends consecutive landing update packets. It prevents message collisions and ensures effective propagation.

DATA_SEND_DACK_REPLY_DELAY

The base station waits before it sends consecutive data acknowledgement messages.

PGO_NEXTSEND_DELAY

The base station waits before it sends consecutive position go messages. It prevents message collisions and ensure effective propagation.

Most of these delays will change when applied to a practical system, but the protocol behaviour will stay the same. When the system is tested in the real-world, the delays can be changed and simulated to get the optimum performance.

The testing setup and results specific to each network function are presented next. Each network function setup has four components:

[†]A network link failure is when all the data link transmissions for the specific packet fails / three data link failures occurred.

- Probability inputs
  These inputs will ultimately determine how many times the network function is executed and at what probability of failure.
- Influencing delays
  Only delays (from those discussed above) that influences the current network function is presented.
- When is the network function completed?
  This is considered as the network function success.
- When does a total failure occur?
  Used to determine if the network function was unsuccessful.

6.4.1 Route Update

Setup

Probability inputs

- Starting probability = 0
- Probability increment = 0.005
- Ending probability
  - Transmission failure = 1
  - Node failure = 0.5
- Number of iterations per probability = 100

Influencing delays

- DRONE_ELEVATE

- DATA_LINK_TRANSMISSION
- DATA_LINK_TIMEOUT
- BUILD_PACKET
- RECALCULATE_ROUTES

When is the routing update network function completed?

The duration of a run is recorded when the base station detects that all the drones received their routing data. A drone that is unreachable and not part of the recalculated routes is not considered here.

When does a total failure occur?

When two network failures occur, it is considered a total failure. The routes are recalculated only once.

Results

The *route update* was ran 20000 times for transmission failures and 10000 times for node failures. A *route update* with no failures has 32 transmissions with a duration of 102.7 time units.

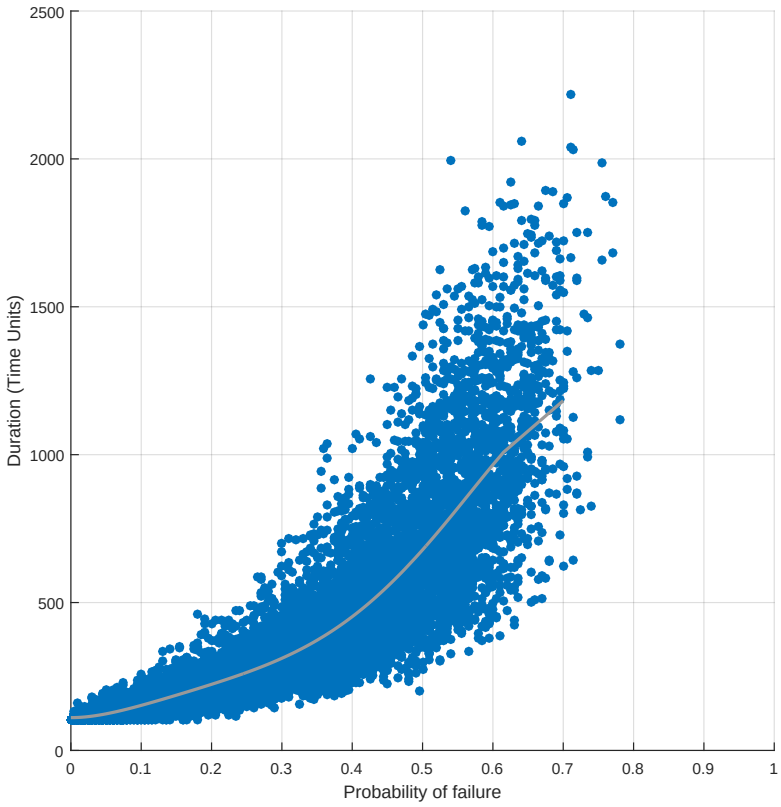


Figure 6.4: Successful *route update* durations for transmissions undergoing failures

Figure 6.4 discussion:

The duration of the *route update* network function in relation to the probability of transmission failure is shown. At a low probability, the data is compact because it is where only data-link failures occur and the link is just re-established. Higher probabilities result in network failures, and the base station should be informed and the routes are recalculated. A larger difference in delay is observed.

The solid line shows the average. We see that the increase in the probability of transmission failure has an exponential increase in duration (more network functions have to recalculate their routes).

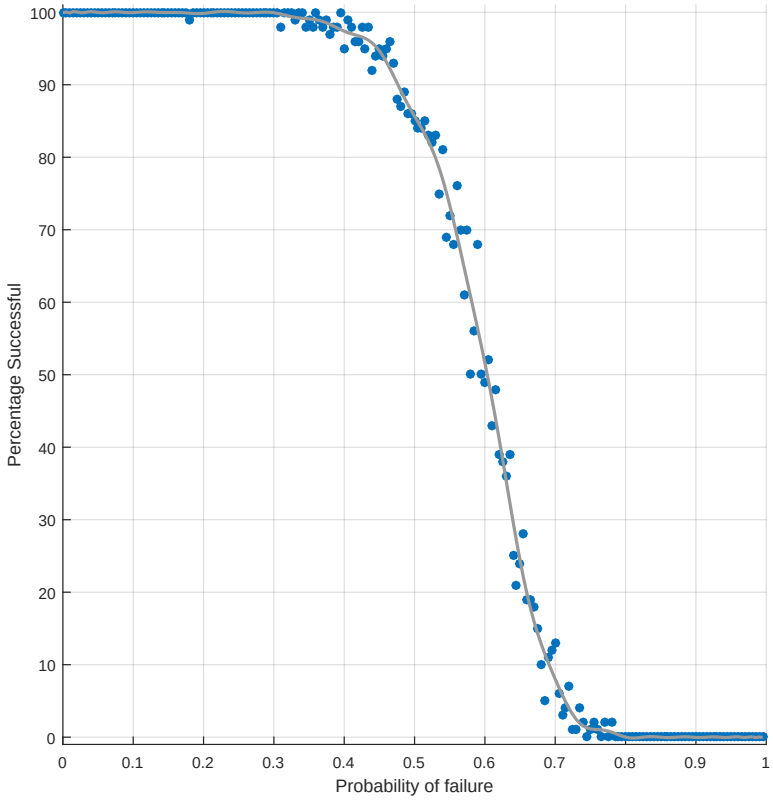


Figure 6.5: Percentage successful *route updates* for transmissions undergoing failures

Figure 6.5 discussion:

Figure 6.5 gives the percentage of successful *route update* executions in relation to the probability of transmission failures.
Up to a 0.3 probability of transmission failure, *route updates* are successful but decrease dramatically to 0.8.

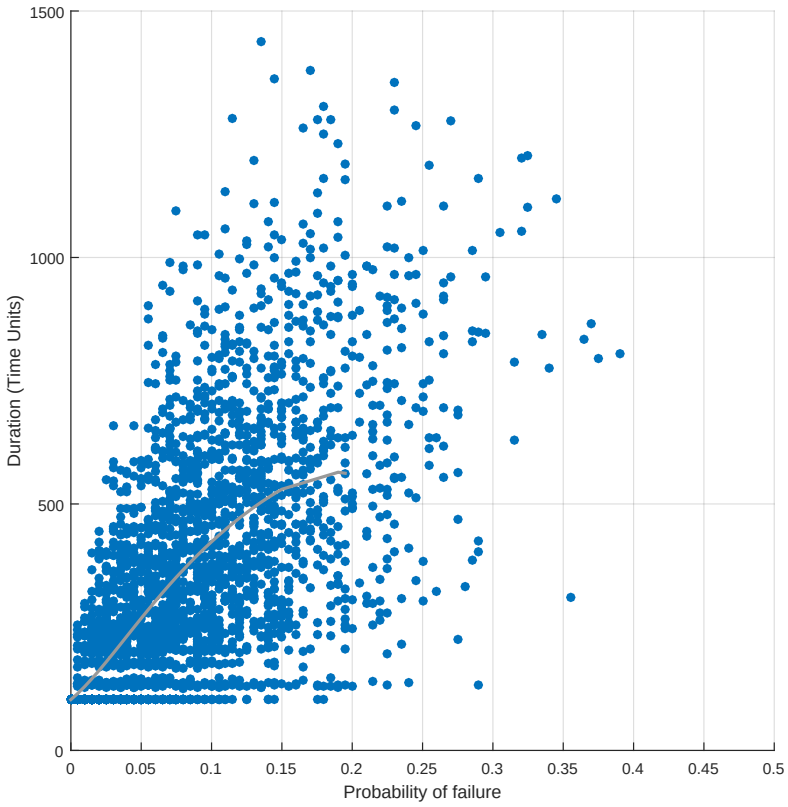


Figure 6.6: Successful *route update* durations for nodes undergoing failures

Figure 6.6 discussion:

The duration of the *route update* network function in relation to the probability of node failure is shown. We see discrete jumps in smaller durations. The data at 102.7 time units (resembling a line at the bottom) is where no failure occurred. The data resembling a second line (at $\pm$135 time units) represents when a failure occurred, the base is notified and routes are recalculated. The data resembling a third line (at $\pm$177 time units) is when an error was found, and the acknowledgement got lost; the base station encounters a time-out and resends the packet. The solid line shows the average.

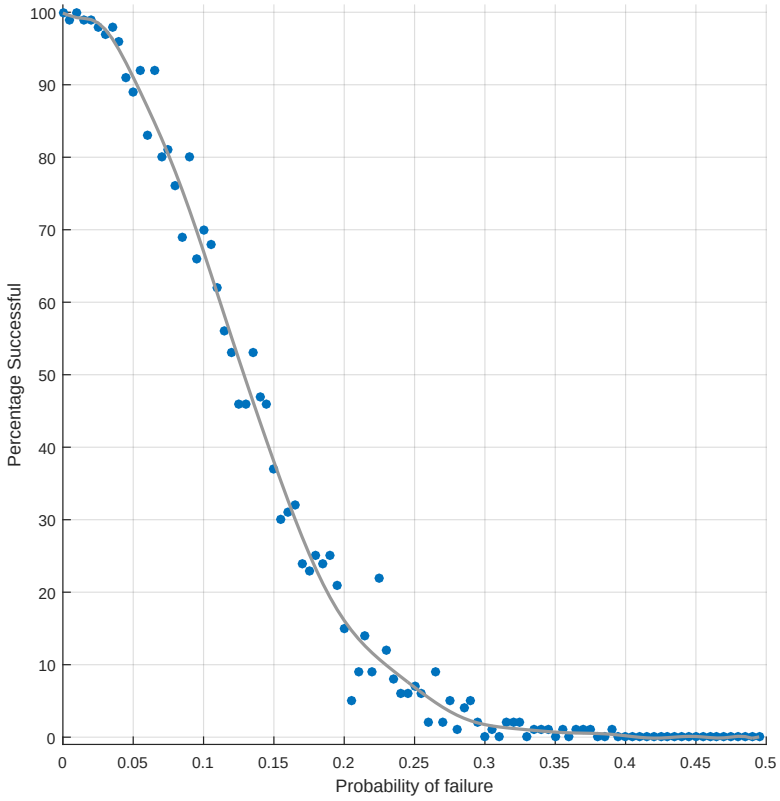


Figure 6.7: Percentage successful *route updates* for nodes undergoing failures

Figure 6.7 discussion:

Figure 6.7 gives the percentage of successful network function execution in relation to the probability of node failure.
Node failures have a dramatic effect on the reliability of the network.
In figure 6.6 we see that the average starts to level out at a ± 0.13 probability of node failure. If we compare this observation with figure 6.7, we see that at a ± 0.13 probability of node failure 50% of the *route update* functions have failed. More functions fail than succeed after a probability of node failure of 0.13 thus valid deductions on network performance after a probability of node failure of 0.13 cannot be made.

6.4.2 Landing Update

Setup

Probability inputs

- Starting probability = 0
- Probability increment = 0.005
- Ending probability
  - Transmission failure = 1
  - Node failure = 0.5
- Number of iterations per probability = 100

Influencing delays

- DATA_LINK_TRANSMISSION
- DATA_LINK_TIMEOUT
- BUILD_PACKET
- LANDING_UPDATE_NEXTSEND_DELAY

When is the landing update network function completed?

All the drones have landed.

When does a total failure occur?

When a network failure occurs. The *landing update* has no mechanism built in for the base station to identify and correct failures. The network function is regarded as unsuccessful when one network failure occurs.

Results

The *landing update* was ran 20000 for transmission failures and 10000 times for node failures. A *landing update* with no failures has 16 transmissions with a duration of 8.7 time units.

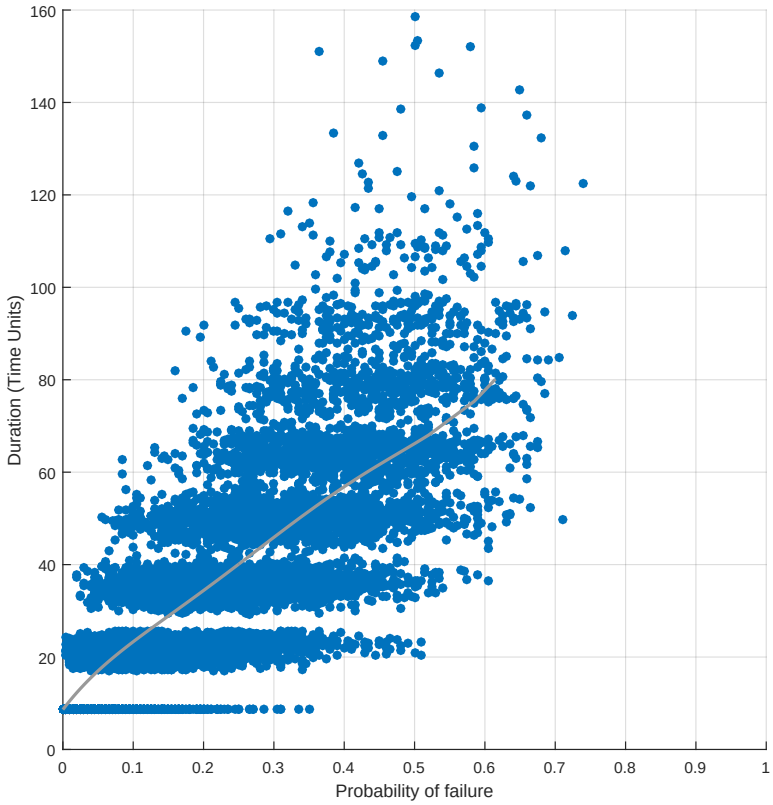


Figure 6.8: Successful *landing update* durations for transmissions undergoing failures

Figure 6.8 discussion:

The duration of the *landing update* network function in relation to the probability of transmission failure is shown. When a data link failure occurs, a fixed amount of time is added. With multiple data link failures, multiple discrete times are added. The *landing update* network function is also completed faster because it does not include the elevation of drones which is the biggest delay factor.

The average is shown by the solid line to present the data distribution.

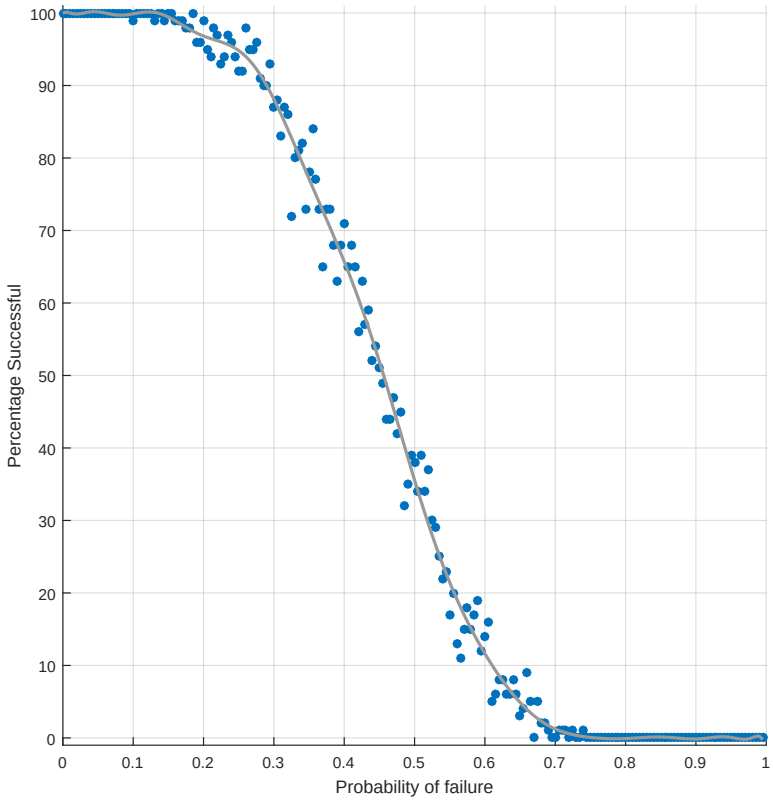


Figure 6.9: Percentage successful *landing updates* for transmissions undergoing failures

Figure 6.9 discussion:

Figure 6.9 gives the percentage of successful network function execution in relation to the probability of transmission failure. It does not have to be as reliable as the *route update*; this network function sends landing signals, and no data will be lost.
Up until a 0.15 probability of transmission failure, most *landing updates* are successful but decrease dramatically to a probability of transmission failure of 0.7. The failure rate of the *landing update* is less than the *route update* because the *landing update* has no acknowledgement and repair.

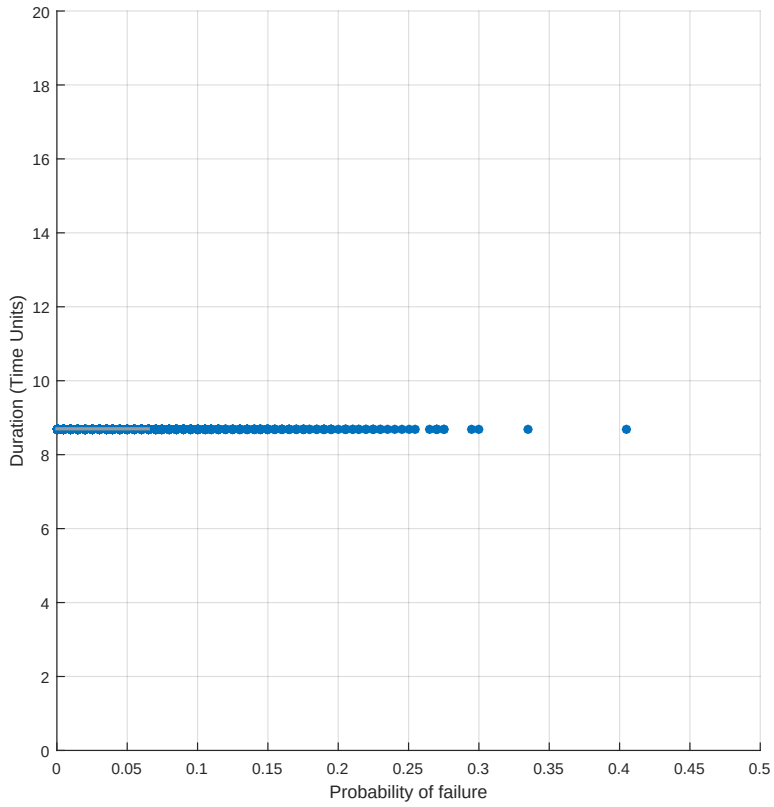


Figure 6.10: Successful *landing update* durations for nodes undergoing failures

Figure 6.10 discussion:

The *landing update* network function does not repair or recalculate the route if a network failure occurs. Therefore when a node fails, the *landing update* function encounters a total failure.

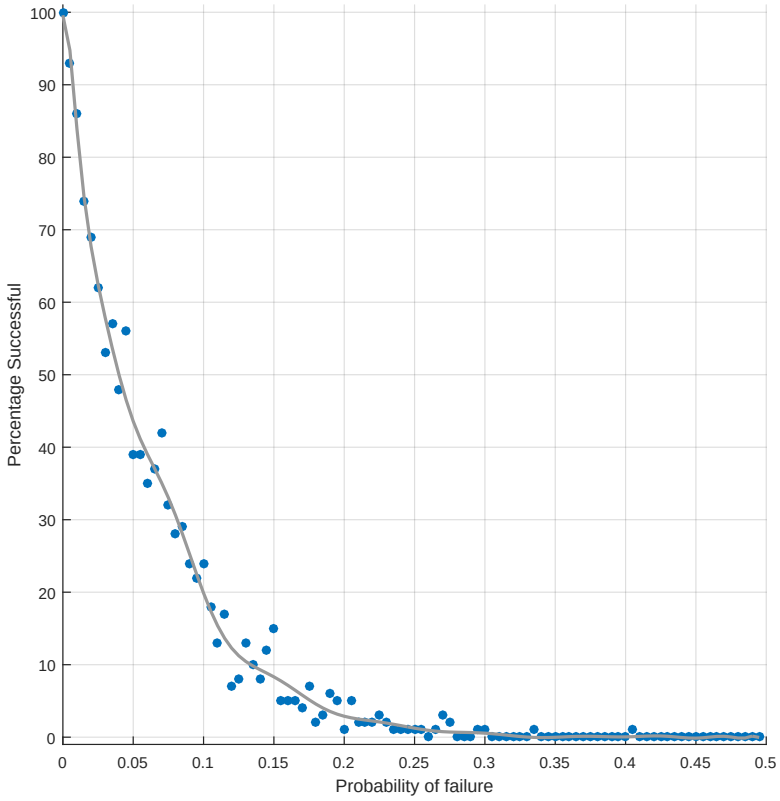


Figure 6.11: Percentage successful *landing updates* for nodes undergoing failures

Figure 6.11 discussion:

The success percentage decreases straight away. At a 0.25 probability of node failure, approximately all *landing updates* are unsuccessful.

6.4.3 Data Send

Setup

Probability inputs

- Starting probability = 0

- Probability increment = 0.005
- Ending probability = 1
- Number of iterations per probability = 100

Influencing delays

- DRONE_ELEVATE
- DATA_LINK_TRANSMISSION
- DATA_LINK_TIMEOUT
- BUILD_PACKET
- DATA_SEND_DACK_REPLY_DELAY

When is the data send network function completed?

When the base station receives the data.

When does a total failure occur?

When the wave of drones is unable to forward the data. One drone's primary and secondary route must fail then the previous drone's (if available) secondary route must also fail for a total failure to occur.

Results

Data send was ran 20000 times for transmission and node failures. The *data send* function with no failures has 12 transmissions with a duration of 45 time units.

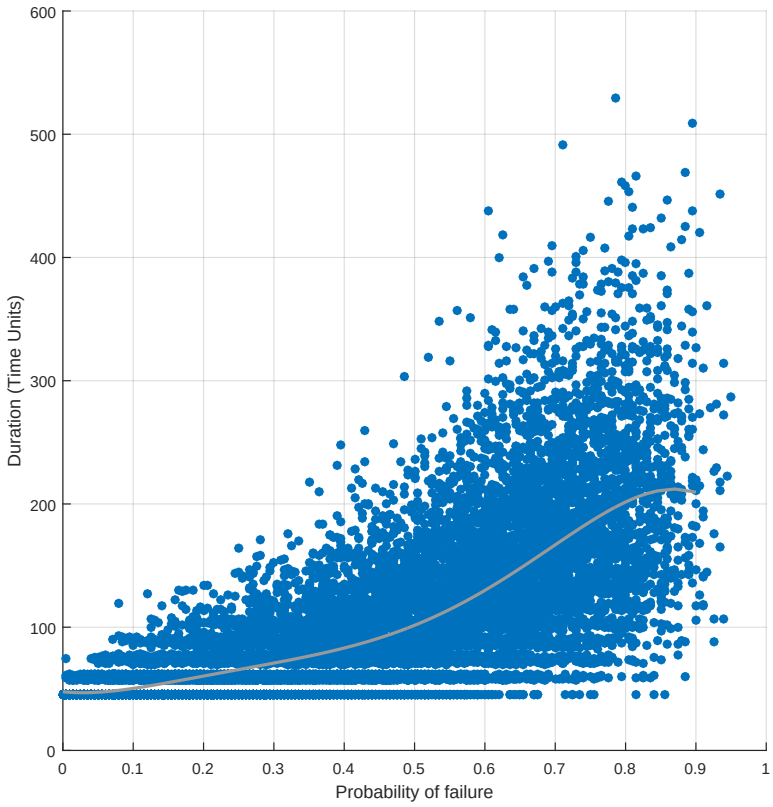


Figure 6.12: Successful *data send* functions durations for transmissions undergoing failures

Figure 6.12 discussion:

The results show discreet behaviour at low duration times this can be explained when a data link fails a discreet time duration is spent. This does not appear at higher durations due to the *random added delay mechanism*.

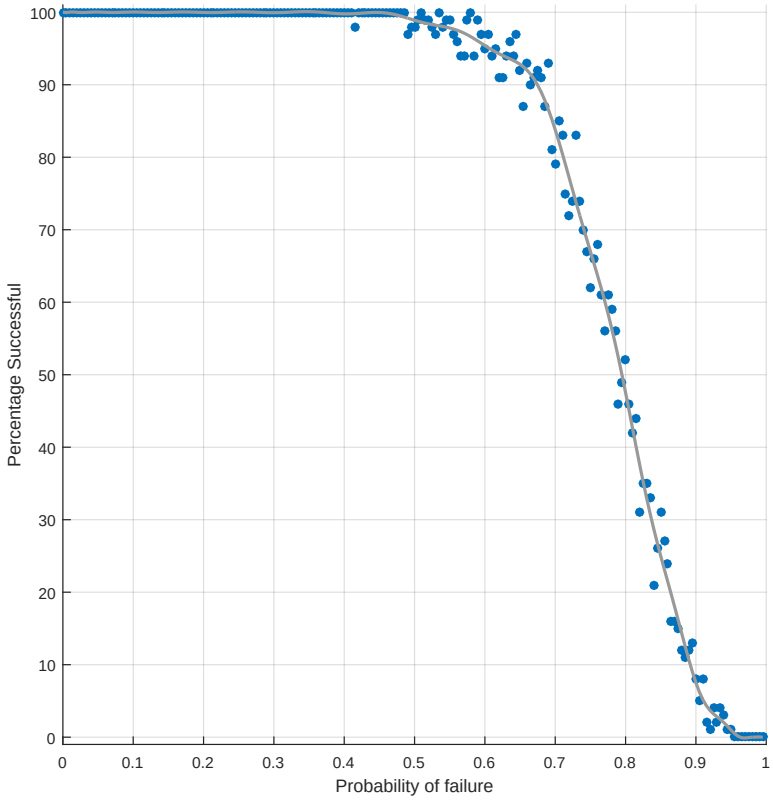


Figure 6.13: Percentage successful *data send* functions for transmissions undergoing failures

Figure 6.13 discussion:

The percentage of successful *data send* functions only start declining at a probability of transmission failure of ± 0.49. *Data send* is more reliable than the other network functions. This is because the *data send* function has more mechanisms for route repair.

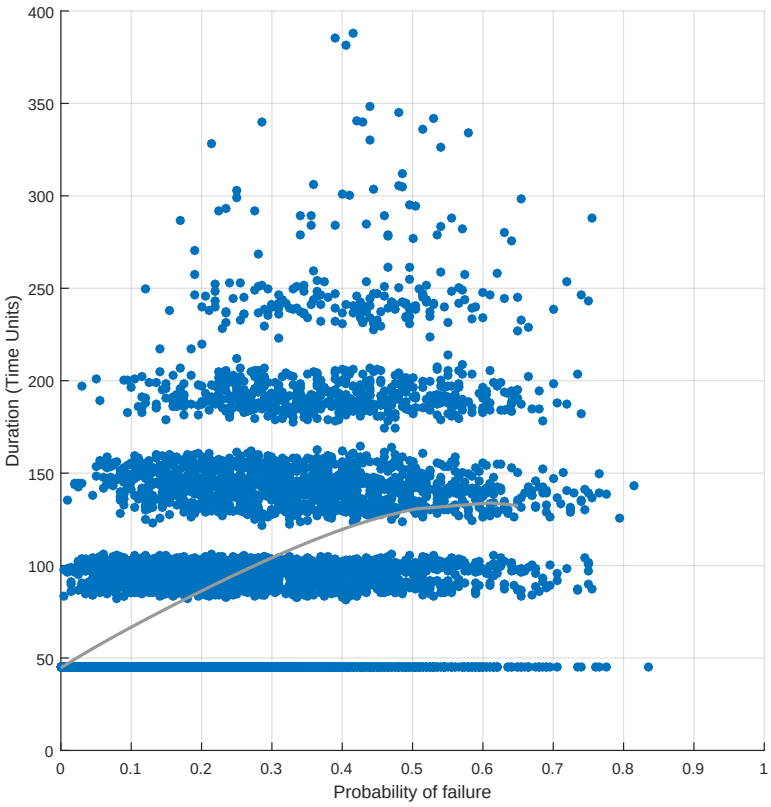


Figure 6.14: Successful *data send* function durations for nodes undergoing failures

Figure 6.14 discussion:

Discrete jumps can be noticed in the data. The data at 45 time units (resembling a line at the bottom) is where no failure occurred. The data resembling a second line (at ± 83 time units) is when one node failure occurred. The data resembling a third line (at ± 98 time units) is when one node failure occurred that resulted in a path that is one hop longer. As the failures and added hops increase the duration also increase in discrete jumps.

The same average levelling can be seen starting at a probability of node failure of ± 0.4. At ± 0.4 probability of node failure, 50% of the functions failed (depicted in figure 6.15), therefore only the average duration *data send* functions up to 0.4 probability of node failure is conclusive.

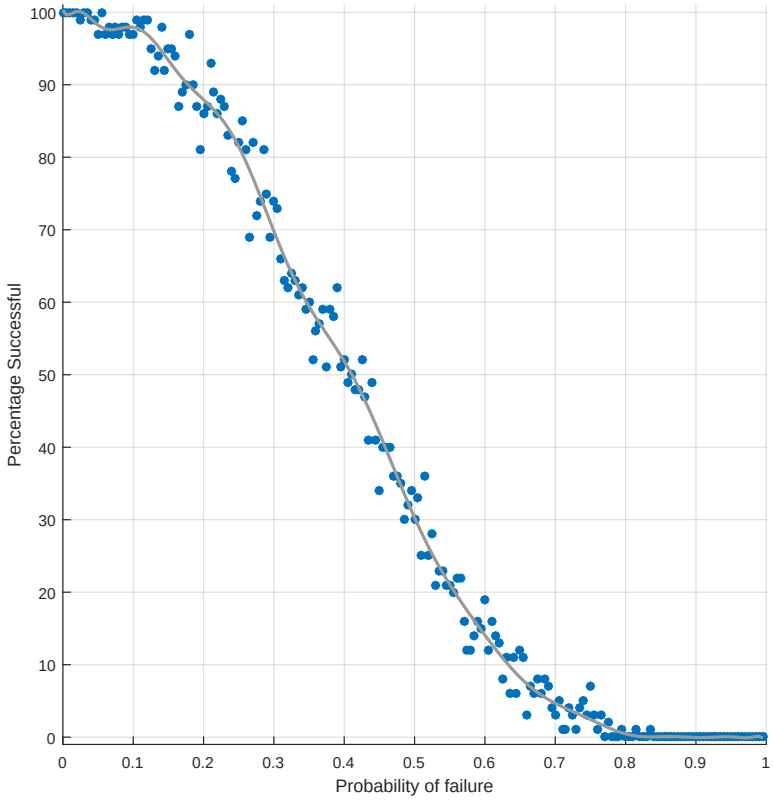


Figure 6.15: Percentage successful *data send* functions for nodes undergoing failures

Figure 6.15 discussion:

A lot more *data send* functions are failing with node failures than with transmission failures but still performs better than all the other functions.

6.4.4 Position Update

Setup

Probability inputs

- Starting probability = 0

- Probability increment = 0.005
- Ending probability = 1
- Number of iterations per probability = 40

Influencing delays

- DRONE_ELEVATE
- DATA_LINK_TRANSMISSION
- DATA_LINK_TIMEOUT
- BUILD_PACKET
- RECALCULATE_ROUTES
- PGO_NEXTSEND_DELAY

When is network function completed?

When all drones received their position information and the repositioning signal (position go).

When does a total failure occur?

When two network failures occur during the position update transmissions. The routes are only recalculated once.
Alternatively, when a network failure occurs during the position go messages.

Results

The *position update* was ran 20000 times for transmission failures and 6000 times for node failures. A *position update* with no failures has 48 transmissions with a duration

of 111.4 time units.

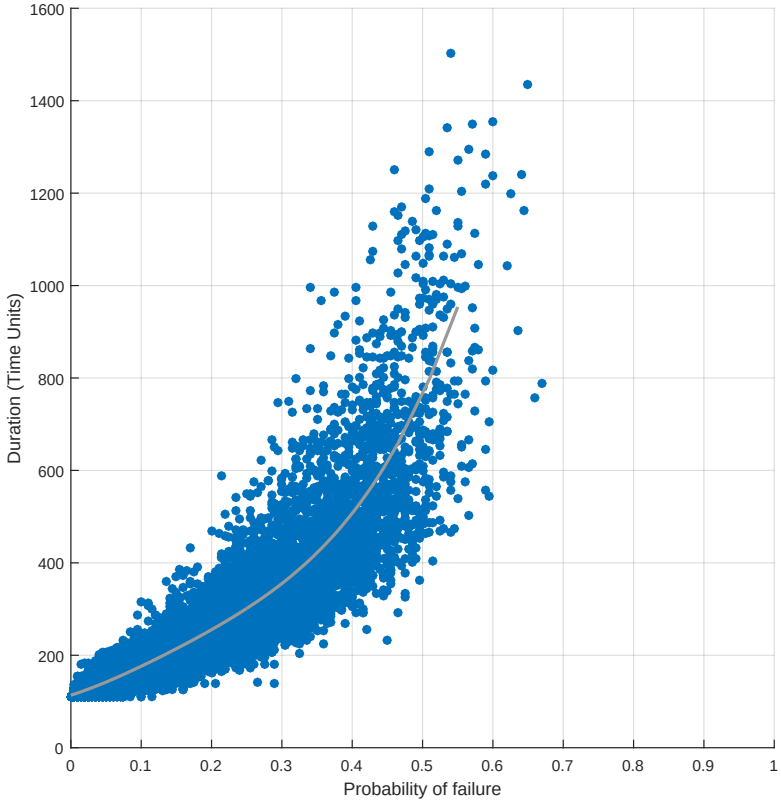


Figure 6.16: Successful *position update* durations for transmissions undergoing failures

Figure 6.16 discussion:

Position update is a combination of the *route update* and the *landing update*. It sends position update packets the same as route update packets and position go packets the same as landing update packets. It therefore has the shape of the *route update* but fails much earlier because of the 'position go packet' implementation.

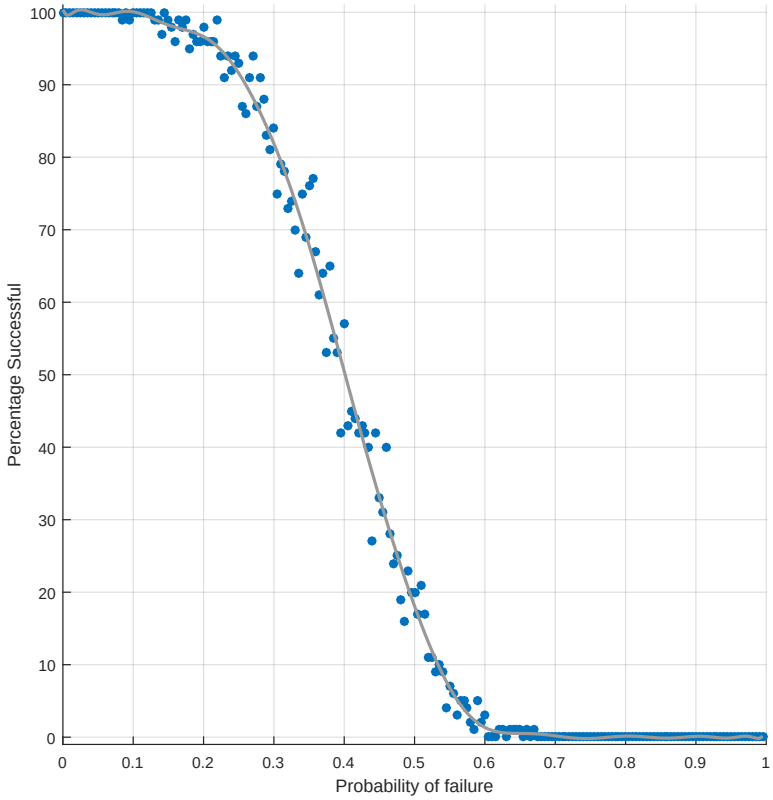


Figure 6.17: Percentage successful *position updates* for transmissions undergoing failures

Figure 6.17 discussion:

The *position update* starts to fail after a probability of transmission failure of 0.08 and approximately all functions are unsuccessful at a probability of 0.67.

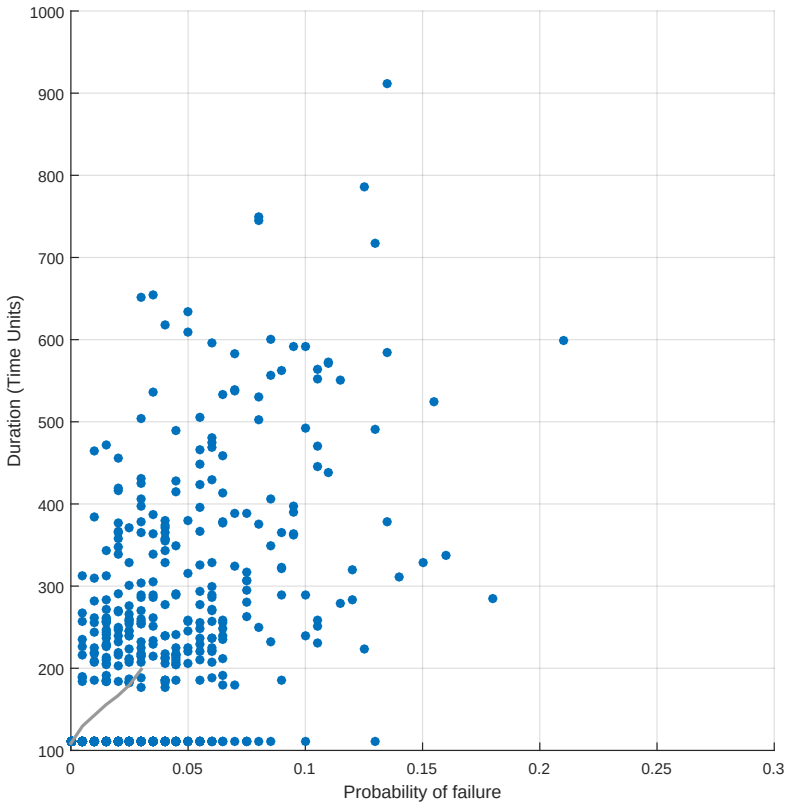


Figure 6.18: Successful *position update* durations for nodes undergoing failures

Figure 6.18 discussion:

At time units 123.7 the successful *position updates* are shown that experienced no failure. Data shown at a greater number of time units is when the *position update* function was successful but experienced multiple nodes failures.

In this case, the average stops at ± 0.03. We see that at ± 0.03 in figure 6.19 50% of the functions have failed. Figure 6.18 does not show the inconclusive values for the delay.

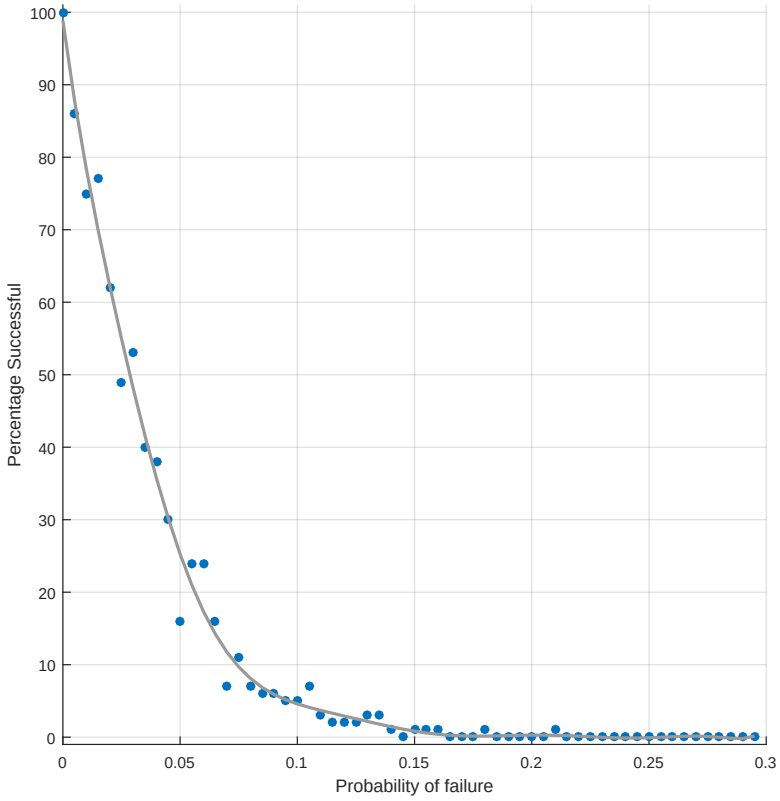


Figure 6.19: Percentage successful *position updates* for nodes undergoing failures

Figure 6.19 discussion:

One can observe the dramatic effect the position go packet transmission has on the *position update* network function.

6.5 Collision Tests

Recall, in section 2.4 it is stipulated that [20] determines the paths to utilise the least number of drones from the rhinos to the base station. To achieve this, nodes are shared between paths from the base station to different rhino groupings. As a result collisions may occur where two branches merge.

Here the qualitative influence of the data transmission with collisions is tested.

Setup

A network was constructed that contained branches. The network is shown in figure 6.20

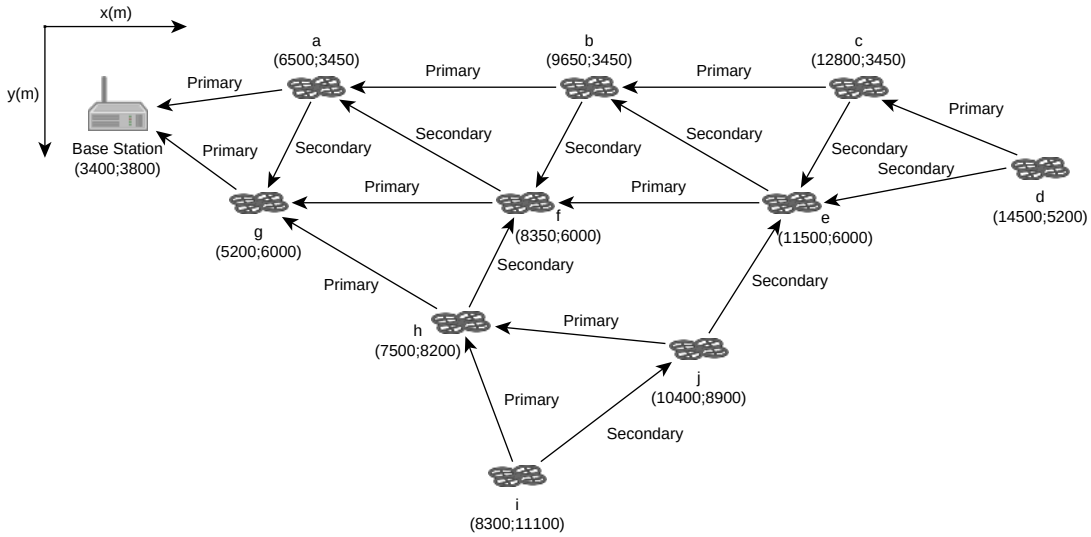


Figure 6.20: Collision testing network

The routing information provided are:

Table 6.3: Base station routing information for collision testing

Drone	Primary route	Secondary route
a	aB	agB
b	baB	bfgB
c	cbaB	cefgB
d	dcbaB	defgB
e	efgB	ebaB
f	fgB	faB
g	gB	-
h	hgB	hfgB
i	ihgB	ijhgB
j	jhgB	jefgB

When, for example, *data send* is initiated in nodes *e* and *i* simultaneously, theoretically when no failures occur there will be a collision at node *g* (*e* and *i* is the same amount of hops away from *g*). In practice, a collision will not occur when one drone ascends longer or when a failure occurs in one path since these will cause delays that differ between paths.
This network also does not carry lots of data therefore lowering the probability of a collision. The probability of a collision in this network was chosen as 0.1. A link failure is re-established three times when broken. The probability for re-establishment was chosen as a third of 0.1 resulting in ± 0.03.

To test the influence of collisions on the network, and the mechanism implemented to reduce the collisions (*random added delay mechanism* formulated in section 4.4.3), three cases were constructed:

1. *No Collisions* No alterations are made to the network to establish a baseline. The *data send* network function is initiated simultaneously in nodes *e* and *c*.
2. *Collisions Data send* is initiated in nodes *e* and *i* simultaneously to test when collisions occur on its primary route. A collision will occur at node *g* in this case.
3. *Collisions; no random time-out* The random time added to the retransmissions mechanism is deactivated to test its influence.

The collisions were tested in *data send* network function therefore the same delays, starting and total failure identification were implemented as mentioned in section 6.4.3.

Results

All three test cases are executed on the network, and their results are plotted on the same graph to form a comparison. First the delay of the network is presented then its reliability. The reliability graph is enlarged to emphasise the effect of the test cases.
Data send was ran 20000 times with transmission failures.

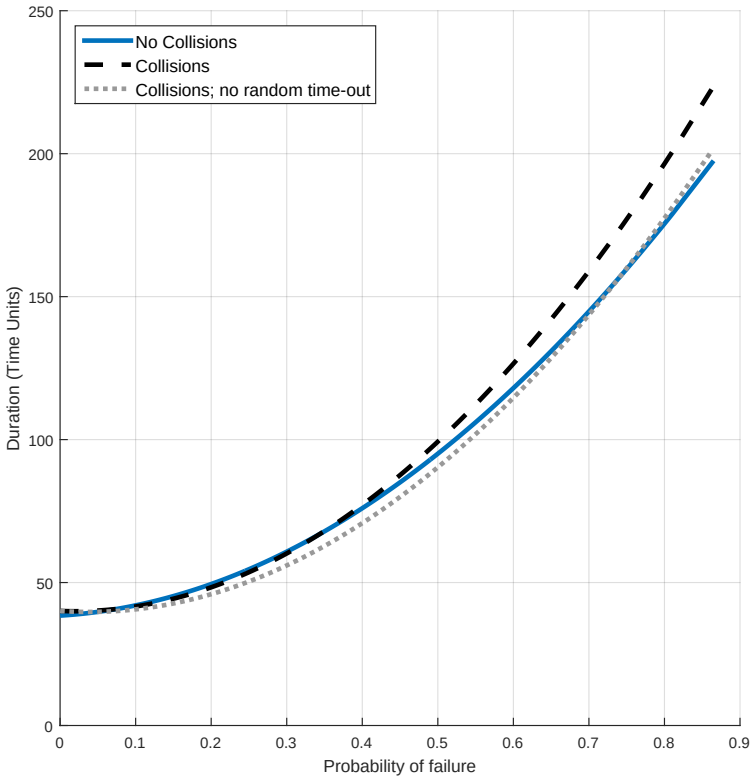


Figure 6.21: Successful *data send* functions durations undergoing transmission failures for the collision test cases

Figure 6.21 discussion:

Comparing *No Collisions* and *Collisions*: The duration of the two cases are more or less the same initially. As the probability of transmission failure increases the average delay of the network with collision becomes more than the network without collisions. Comparing *Collisions* and *Collisions; no random time-out*: Regarding delay, the network performs better without the *random added delay mechanism* in this case. When retransmission occurs on any node, a random delay is added. With multiple delays added, the total duration of communication over the network is increased. The total delay is then more than the delay added when a collision occurs in a single node. This only worsens as the probability of transmission failure increases. Networks with more branches will result in more collisions.

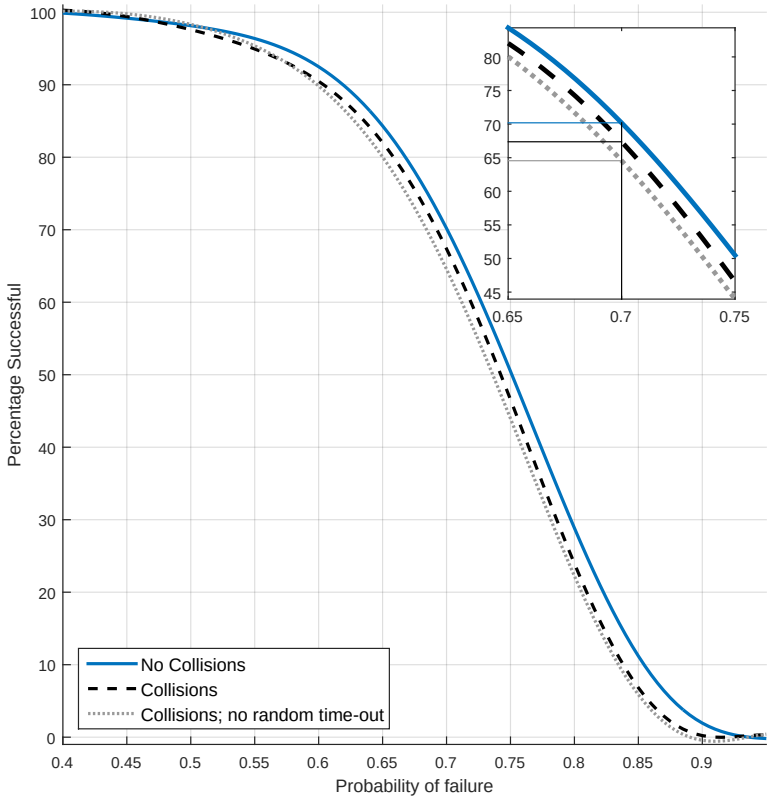


Figure 6.22: Percentage successful *data send* functions undergoing transmission failures for the collision test cases

Figure 6.22 discussion:

Comparing *No Collisions* and *Collisions*: When collisions occur in the network, the reliability is affected. To make the influence more apparent, the percentage of success is compared at a probability of transmission failure of 0.7. The success percentage with no collisions is 70.2% in comparison to the success percentage with collisions of 67.4%. Comparing *Collisions* and *Collisions; no random time-out*: Although the delay is not improved by the *random added delay mechanism* in this case, the reliability is. At a probability of transmission failure of 0.7, the success percentage with collisions is 67.4% in comparison to the success percentage with collisions no random time-out, of 64.6%.

6.6 Chapter Summary and Discussion

The second research goal was to determine how well CERATIN operates in the proposed system. To reach this goal, a Monte Carlo simulation was constructed implementing a test network. Using the Monte Carlo simulation, the performance of each network function was determined. Transmission and node failures were induced into the network, and performance metrics such as delay and reliability of each network function were recorded. The results of each network function are summarised using the following categories (the results are shown in table 6.4):

1. The delay of the network function with no failures.
2. The probability of transmission failures‡ when not all of the network functions executed successfully.
3. The probability of transmission failures‡ when all the network functions failed.
4. The probability of node failures‡ when not all of the network functions executed successfully.
5. The probability of node failures‡ when all the network functions failed.

Table 6.4: Protocol performance test results summary

Category	*Route update*	*Landing update*	*Data send*	*Position update*
1. (time units)	102.7	8.7	45	111.4
2. (probability of transmission failure)	0.3	0.1	0.49	0.08
3. (probability of transmission failure)	0.8	0.75	0.95	0.67
4. (probability of node failure)	0	0	0.03	0
5. (probability of node failure)	0.4	0.3	0.84	0.15

‡At a higher probability of failure, more failures occur in the network. A network function performs better than the other network functions if it can handle more failures (executes successfully at a higher probability of failure) .

From table 6.4 we see that the *landing update* executes the fastest in ideal conditions, followed by *data send*, *route update* and lastly *position update*. During the *landing update* network function, no acknowledgements are sent from the drones resulting in a shorter duration.
The values for categories 2 to 5 are larger for *data send* than the other network functions. This means *data send* starts to fail and fails completely at higher probabilities of failure. *Data send* is therefore the most reliable network function.
Looking at the difference between categories 2 and 4, and categories 3 and 5 for each network function we see that node failures have a more significant impact on the reliability of the network than transmission failures.

As an elaboration of the performance tests, the influence of network collisions was tested. Three cases were constructed: *No Collisions* (1), *Collisions* (2) and *Collisions; no random time-out* (3). We observe that collisions do influence the network, and although the *random added delay mechanism* improves the reliability, it prolongs the delay in the tested network.

The next chapter is presented to conclude the research.

Chapter 7

Conclusion

In this chapter, a summary of the research is presented, and how the research goals were addressed with our findings are provided. Lastly, future work is recommended.

7.1 Research Summary

The research was started by presenting problems regarding rhino poaching in South Africa. Using the possibilities of UAV deployment a drone coverage system was proposed. From the system, the research goals were derived:
Design and specify a communication protocol for the proposed drone coverage system (detailed discussion in chapter 2), and determine how well it will operate in the proposed system.
The protocol was designed in chapter 4 using information obtained from literature, presented in chapter 3. The protocol was tested and verified as presented in chapter 5. How well the protocol operates in the proposed system was answered by chapter 6.

7.2 Addressing the Research Goals

The first and main goal is to design and specify a communication protocol for the proposed drone coverage system.
A structured method from [33] was used to design and specify the protocol in chapter 4. The design choices are discussed next.
Section 2.4 serves as an operational model for the proposed drone coverage system. The operational model is also used as a requirement specification. We developed CERATIN as the solution protocol to address steps 6 to 12 of the operational model. The other steps of the model are determined by our project partner YRless and in another study [20]. Here we present steps 6 to 12 and how they were addressed:

6. The base station sends out configuring signals.
7. Upon receiving its configuring signal, the drone will ascend and update its routing table.

Steps 6 and 7 are implemented in the *route update* network function.

8. After the network is configured, the drones will return to the ground.

Step 8 by the *landing update* network function.

9. If a drone has data to transmit, it will ascend and transfer data to the next drone according to its routing table.
10. The data will propagate through the network towards the base station in a wave formation (causing an intermittent connectivity further described in section 2.5).
    - The data includes distress signals and rhino position updates.
    - A distress signal is sent from the base station to the server or notification device.

Steps 9 and 10 are implemented in the *data send* network function.

11. When the drones have to relocate, the base station will send out a relocation signal.
    - The position updates are used to determine the next positions of the drones.
    - Step 4 is re-executed.
12. Upon receiving its relocation signal, the drone will ascend and update its position data and relocate itself.

Steps 11 and 12 are implemented in the *position update* network function.

The update network functions are implemented to send messages form the base station to the drones. The source routing approach in DSR is used to determine the paths of the update messages.
Data send must be reliable and use less energy; therefore the table driven approach of AODV in a DTN is used. The table driven approach repairs a broken link locally. An error message therefore does not have to propagate through the network wasting the drones' power. The DTN enables the network to propagate data through the network without having all drones (from the source to the destination set up a connection) ascend.
Other key design features include: The wave formation during *data send* (1), random retransmission time-out delay (2), link failure reporting (3) and the protocol's control over drones (4).

1. At any time when data is sent from a drone to the base station, two nodes are airborne and contain the data. With this feature, the propagation of data from a drone to the base station imitates a wave. The leading node can send data to its primary route entry or when failed to its secondary route entry. If the preceding node does not receive an acknowledgement that the leading node could forward the data, it can send the data via its secondary route entry. This feature improves the reliability of the network beyond the other network functions.

2. This *random added delay mechanism* is implemented to reduce the influence of collisions in the network.

3. In the *route and position update* network functions, a broken link is reported back to the base station using an error message. It is crucial that the drones receive the routing and position data. The base station is notified even if this feature aggravates the network delay.

4. The protocol is designed to control the drones to achieve features like the wave formation.

The second goal was to determine how well the new protocol, CERATIN will operate in the proposed system.
It was addressed by conducting performance tests using a Monte Carlo simulation. Transmission and node failures were induced into the network, and the delay and reliability of each network function were recorded. Here is a summary of the findings:

- *Route update* The increase in the probability of transmission failures has an exponential increase in duration and the network starts to fail after a 0.3 probability of transmission failure. Node failures increase the delay significantly starting at a low probability of node failures. All *route update* network functions fail at a probability of node failure of 0.4.

- *Landing update* The *landing update* network function does not have a repair or failure reporting feature. It therefore starts to fail at a lower probability of failure and has a much shorter delay.

- *Data send* The *data send* function is more reliable than the other network functions because it has more mechanisms for route repair. It is the main network function, responsible for reporting the rhino positions and giving poaching alerts. The other network functions were designed to assist the *data send* network function.

- *Position update* The *position update* network function is a combination of the *route update* and the *landing update*. It therefore behaves the same as the *route update*

but fails much earlier because of the *landing update's* impact.

An observation in every network function is that node failures have a more detrimental effect on the performance of the protocol than transmission failures.

As an elaboration of the performance tests, the qualitative influence of network collisions is tested. We observe that collisions do influence the network, and although the *random added delay mechanism* improves the reliability, it prolongs the delay in the tested network.

In conclusion, the functional and performance tests demonstrate that the protocol satisfies the protocol requirements specification formed from the initial requirements set by the industry partner.

7.3 Future Work

We suggest that the operation of drones in a wilderness/uncontrolled environment should be tested and evaluated before improving or continuing work on the protocol. As UAV systems improve, this will be a viable solution but the current technology of deploying drones in a wilderness/uncontrolled environment is a big risk.

Here are identified protocol design improvements:

- Larger messages to operate in bigger networks. CERATIN only implements 59 drones.
- When two branches meet, data can be sent simultaneously (in one packet) using piggybacking.
- When a drone is not active for a certain amount of time, it should land or reposition itself depending on its last instruction.
- A received data packet by the base station contains the full route it followed from

the source. The base station can use that route to determine whether a drone failed (a drone failed when the base station's stored route and the received route does not match). If the same drone failed multiple times, the base station could create a notification.

The simulation can be modified to determine the exact number of transmissions and ascends made by every drone. These results can be used to calculate the energy consumption of each drone for optimisation.

Real-world recorded inputs should be provided to the performance testing simulation in order to optimise the system. It is therefore future work to deploy and implement the system.

7.4 Research Closure

The research originated from the rhino poaching problem but has many aspects that can be used in other energy-constrained networks. The wave formation with its intermittent connectivity can be implemented in other networks. The change in connection range does not necessary have to be nodes that ascend and land, it can be nodes that increase and decrease their transmission power. Nodes can be in a sleeping mode, and when data needs to be transferred, increase their transmission power only for a while. Some nodes can return to their sleeping mode before the data reaches the destination. This will save energy and the reduce need of large buffer sizes as in DTNs.

We sincerely hope that the door we opened (tracking of wildlife using autonomous UAVs) stays open to solve the rhino poaching problem and prevent their extinction.

Bibliography

[1] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," *ACM Sigplan Notices*, vol. 37, no. 10, pp. 96–107, 2002.

[2] A. Doria, M. Uden, and D. Pandey, "Providing connectivity to the saami nomadic community," *generations*, vol. 1, no. 2, p. 3, 2009.

[3] J. Coetzee. (2015, Mar.) These ground-breaking drones want to prevent rhino poaching in sa. [Online]. Available: http://memeburn.com/2015/03/these-ground-breaking-drones-want-to-prevent-rhino-poaching-in-sa/

[4] (2015, Aug.) Aerial base stations with opportunistic links for unexpected temporary events. [Online]. Available: http://www.absolute-project.eu/

[5] (2012, Jun.) Anchors. [Online]. Available: http://www.anchors-project.org/index.php/de/index.html

[6] (2015, Aug.) Avigle. [Online]. Available: http://www.fsd.rwth-aachen.de/English/Research/Avigle.php

[7] G. Warwick. (2015, Mar.) Facebooks uav flies, builds on developments in solar power. [Online]. Available: http://aviationweek.com/technology/facebook-s-uav-flies-builds-developments-solar-power

[8] (2015, Aug.) Amazon prime air. [Online]. Available: http://www.amazon.com/b?node=8037720011

[9] A. B. Forouzan, *Data Communications and Networking*, 5th ed. McGraw-Hill, 2013.

[10] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," Tech. Rep. RFC 3561, Jul. 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3561.txt

[11] J. Broch, D. B. Johnson, and D. A. Maltz, "The dynamic source routing protocol for mobile ad hoc networks," Tech. Rep. RFC 4728, Mar. 1998. [Online]. Available: https://tools.ietf.org/html/rfc4728

[12] (2015, Aug.) Stop rhino poaching. [Online]. Available: http://www.stoprhinopoaching.com/

[13] A. Taylor, K. Brebner, R. Coetzee, H. Davies-Mostert, P. Lindsey, J. Shaw, and M. 't Sas-Rolfes, "The viarility of legalising trade in rhino horn in south africa," 2014.

[14] B. Malherbe. (2013, May) The rhino wars - a comprehensive overview of the situation and the solutions.

[15] E. Darack. (2011, May) A brief history of unmanned aircraft. [Online]. Available: http://www.airspacemag.com/photos/a-brief-history-of-unmanned-aircraft-174072843/

[16] (2015, Aug.) Yrless international: Animal tracking. [Online]. Available: http://www.yrless.co.za/animals/animalsGI.html

[17] G. Warwick. (2015, Jun.) Google perseveres after titan uav crash. [Online]. Available: http://aviationweek.com/technology/google-perseveres-after-titan-uav-crash

[18] B. Stevenson. (2014, Aug.) Google unveils uav parcel delivery concept. [Online]. Available: http://www.flightglobal.com/news/articles/google-unveils-uav-parcel-delivery-concept-403160/

[19] N. Ungerleider. (2012, Dec.) The google-funded drones that hunt illegal hunters. [Online]. Available: http://www.fastcompany.com/3003870/google-funded-drones-hunt-illegal-hunters

[20] I. Venter, "Development of a uav placement algorithm to establish redundant communication between nodes," M. Eng. thesis, North-West University, Potchefstroom, South Africa, Nov. 2015.

[21] D. Halliday, R. Resnick, and J. Walker, *Fundamentals of physics extended*. John Wiley & Sons, 2010, vol. 1.

[22] L. Frenzel, *Principles of electronic communication systems*. McGraw-Hill, Inc., 2007.

[23] (2015, Sep.) Dictionary.com. [Online]. Available: http://dictionary.reference.com/browse/tele-

[24] (2015, Sep.) International organization for standardization. [Online]. Available: http://www.iso.org/iso/home.html

[25] B. D. Shivahare, C. Wahi, and S. Shivhare, "Comparison of proactive and reactive routing protocols in mobile ad hoc network using routing protocol property," *International Journal of Emerging Technology and Advanced Engineering. Website: www.ijetae. com (ISSN 2250-2459, Volume 2, Issue 3*, 2012.

[26] S. C. Ergen, "Zigbee/ieee 802.15.4 summary," *UC Berkeley, September*, vol. 10, p. 17, 2004. [Online]. Available: http://pages.cs.wisc.edu/~suman/courses/707/papers/zigbee.pdf

[27] Libelium, "Waspmote digimesh," Tech. Rep., 2012. [Online]. Available: http://www.libelium.com/downloads/documentation/waspmote-digimesh-networking_guide.pdf

[28] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture," Tech. Rep. RFC 4838, Apr. 2007. [Online]. Available: https://tools.ietf.org/html/rfc4838

[29] A. Tovar, T. Friesen, K. Ferens, and B. McLeod, "A dtn wireless sensor network for wildlife habitat monitoring," in *Electrical and Computer Engineering (CCECE), 2010 23rd Canadian Conference on*. IEEE, 2010, pp. 1–5.

[30] S. Ehsan, M. Brugger, K. Bradford, B. Hamdaoui, and Y. Kovchegov, "Sufficient node density conditions on delay-tolerant sensor networks for wildlife tracking and monitoring," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*. IEEE, 2011, pp. 1–6.

[31] A. Lindgren, C. Mascolo, M. Lonergan, and B. Mcconnell, "Seal-2-seal: A delay-tolerant protocol for contact logging in wildlife monitoring sensor networks," in *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*. IEEE, 2008, pp. 321–327.

[32] H. König, *Protocol Engineering*. Springer Science & Business Media, 2012.

[33] P. Venkataram, S. S. Manvi, and B. S. Babu, *Communication protocol Engineering*. PHI Learning Pvt. Ltd., 2014.

[34] J. H. Gerard, *Design and validation of computer protocols*. USA: Prentice Hall, 1991.

[35] S. Raychaudhuri, "Introduction to monte carlo simulation," in *Simulation Conference, 2008. WSC 2008. Winter*. IEEE, 2008, pp. 91–100.

[36] J. Steward, *Calculus*, 6th ed. Thomson Brooks/Cole, 2009.

[37] (2015, May) Solo specs: Just the facts. [Online]. Available: https://3drobotics.com/solo-gopro-drone-specs/

[38] S. Labs. (2013) Si1000/1/2/3/4/5. [Online]. Available: https://www.silabs.com/Support%20Documents/TechnicalDocs/Si1000.pdf

Appendix A

Key Algorithms

The key algorithms from section 5.3 as implemented in Java are presented here.

A.1 Determining the routes for the base station (Full routes to the drones)

Implementation of the algorithm discussed in section 5.3.1:

```
//supply the function with the drone ID and primary or secondary routes.
private String getRouteTo(String ID, Map<String, String> fullRoutes) {
    String route = fullRoutes.get(ID);
    if (!route.equals("-")) {                    //is the route valid?
        route = new StringBuilder(route).reverse().toString();//reverse
    } else {
        route = "-";
    }
    return route;
}
```

A.2 Send packets starting with routes with the least hops

Implementation of the algorithm discussed in section 5.3.2:

```
private void makeRoutesReady() {
    //Create two maps that will contain the route lengths.
    Map<String, Integer> primaryRouteLengths = new HashMap<>();
    Map<String, Integer> secondaryRouteLengths = new HashMap<>();
    //Insert route lengths into tables
    for (String key : tableOfRoutes.keySet()) {
       primaryRouteLengths.put(key, tableOfRoutes.get(key).get(0).length());
       secondaryRouteLengths.put(key, tableOfRoutes.get(key).get(1).length());
    }
     // Sorting and rewrite:
    Map<String, Integer> sortedPrimaryRouteLengths = sortByComparator(
    primaryRouteLengths, true);
    Map<String, Integer> sortedSecondaryRouteLengths = sortByComparator(
    secondaryRouteLengths, true);
}

private static Map<String, Integer> sortByComparator(
Map<String, Integer> unsortedMap, boolean order) {
    List<Map.Entry<String, Integer>> list = new LinkedList<>(
    unsortedMap.entrySet());
    // Sorting the list based on route lengths.
    Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
        public int compare(Map.Entry<String, Integer> o1,
         Map.Entry<String, Integer> o2) {
          if (order) {
              return o1.getValue().compareTo(o2.getValue());
          } else {
```

```
                return o2.getValue().compareTo(o1.getValue());

            }
        }
    });
    // Maintaining insertion order with the help of LinkedList
    Map<String, Integer> sortedMap = new LinkedHashMap<>();
    for (Map.Entry<String, Integer> entry : list) {
        sortedMap.put(entry.getKey(), entry.getValue());
    }
    return sortedMap;
}
//When an update must be sent to the next drone the key is
//the next one contained in the sorted list but the values
//are used from the tableOfRoutes list:
String key =
sortedPrimaryRouteLengths.entrySet().iterator().next().getKey();
```

A.3 Calculating the drones' routing tables

Implementation of the algorithm discussed in section 5.3.3:

```
//supply the function with the drone ID and primary or secondary routes.
private String getRoutesOf(String ID, Map<String, String> fullRoutes) {
    String route = fullRoutes.get(ID);
    if (!route.equals("-")) {                //is the route valid?
        route = route.substring(1);          //cut away first address
        route = route.substring(0, 1);       //keep first address
    } else {
        route = "-";
```

```
    }
    return route;
}
```

A.4 Updating the routes according to a route failure (Re-calculate routes)

Implementation of the algorithm discussed in section 5.3.4:

```
//First get all routes of nodes affected by failure.
String nextHopFromError;
String unreachableAddress = packetToProcess.get("unreachable_address");
HashMap<String, ArrayList<String>> nextHopFromErrorRoutes = new HashMap<>();
System.out.println("nextHopFromErrorRoutes" + nextHopFromErrorRoutes);
for (String key : tableOfRoutes.keySet()) {
    String currentPrimRoute = tableOfRoutes.get(key).get(0);
    String unreachableAddressPrimRoute
    = tableOfRoutes.get(unreachableAddress).get(0);
    if (currentPrimRoute.contains(unreachableAddress) &&
    currentPrimRoute.length() > unreachableAddressPrimRoute.length()) {
        nextHopFromError =
        String.valueOf(currentPrimRoute.charAt(currentPrimRoute.length() -
        unreachableAddressPrimRoute.length() - 1));
        ArrayList<String> list = new ArrayList<>();
        list.add(0, tableOfRoutes.get(nextHopFromError).get(0));
        list.add(1, tableOfRoutes.get(nextHopFromError).get(1));
        nextHopFromErrorRoutes.put(nextHopFromError, list);
    }
}
```

```
//Change routes to accommodate failure.
HashMap<String, ArrayList<String>> newTableOfRoutes = new HashMap<>();
for (String key1 : tableOfRoutes.keySet()) {
    String currentPrimaryRoute = tableOfRoutes.get(key1).get(0);
    String currentSecondaryRoute = tableOfRoutes.get(key1).get(1);
    for (String key2 : nextHopFromErrorRoutes.keySet()) {
        String nextHopFromErrorPrim
        = nextHopFromErrorRoutes.get(key2).get(0);
        String nextHopFromErrorSec
        = nextHopFromErrorRoutes.get(key2).get(1);
        if (currentPrimaryRoute.contains(nextHopFromErrorPrim)) {
            String changedRoute
            = currentPrimaryRoute.replaceAll(nextHopFromErrorPrim
            , nextHopFromErrorSec);
            ArrayList<String> list = new ArrayList<>();
            list.add(0, changedRoute);
            list.add(1, key1.equals(key2) ? "-" : currentSecondaryRoute);
            newTableOfRoutes.put(key1, list);
        }
    }
}
//Replace changed routes.
for (String key : newTableOfRoutes.keySet()) {
    String newPrimaryRoute = newTableOfRoutes.get(key).get(0);
    String newSecondaryRoute = newTableOfRoutes.get(key).get(1);
    addRowToTableOfRoutes(key, newPrimaryRoute, newSecondaryRoute);
}
//Clear unreachable address
addRowToTableOfRoutes(unreachableAddress, "-", "-");
```

A.5 Determine if all drones received the packet

Implementation of the algorithm discussed in section 5.3.5:

```
//When packet is sent a unique id is created.
sentMessages.put(packetIdBuilder(PacketType.R_UPDATE, outPacket), outPacket);
//Function creating the unique id.
private String packetIdBuilder(PacketType type,
Map<String, String> packetInProcess) {
    return type.name()
    + ","
    + destinationAddress(packetInProcess.get("route_from_base_to_dest"))
    + ","
    + packetInProcess.get("id")
    + ","
    + packetInProcess.get("route_from_base_to_dest").length();
}
//Function determining the destination from the route.
private String destinationAddress(String route) {
    String r = new StringBuilder(route).reverse().toString();
    r = r.substring(0, 1);
    return r;
}
//The test to determine if all drones's acknowledgements were received.
if (_droneList.size() - amountAcksReceived(receivedMessages_RUpdate) == 0) {
    runFSM_RUpdate(Action_RUpdate.LAST_RACK_RECEIVED, null);
}
//Function that counts the number of acknowledgements in comparison to the
//sent packet ids.
protected int amountAcksReceived(List<String> messages) {
    int ctr1 = 0;
```

```
    List<String> contains = new ArrayList<>();
    try {
        for (Iterator<String> it = messages.iterator(); it.hasNext(); ) {
            String rack = it.next();
            String[] ar1 = rack.split(",");
            if ((ar1[0].equals("R_ACK") || ar1[0].equals("P_ACK"))
            && !contains.contains(ar1[1])) {
                ++ctr1;
            }
            contains.add(ar1[1]);
        }
    } catch (Exception e) {
        System.out.println(e);
    }
    return ctr1;
}
```

A.6 Drone determining if it is the destination or intermediate

Implementation of the algorithm discussed in section 5.3.6:

```
//The function returns a boolean "true" if the current drone is the
//destination. "false" is returned when the drone is intermediate.
public Boolean isDestination(String route) {
    String r = new StringBuilder(route).reverse().toString();
    r = r.substring(0, 1);
    return r.equals(this.getAddress());
}
```

A.7 Identify a network link failure

Implementation of the algorithm discussed in section 5.3.7:

```
//A data link object is created for the route update transmission.
//If the data link object executes and returns a "false" a network
//link failure is encountered.
DataLink ru_dataLink = new DataLink(outType, getAddress(),
nextHop(outPacket.get("route_from_base_to_dest")), outPacket);
if (!ru_dataLink.execute()) {
    System.out.println(getAddress() + " encountered network link failure");
    _base.timeoutEmulatorRUpdate(outPacket);
}
```

Appendix B

Functional Testing Results

Figure B.1 depicts the GUI of the simulation when started. One can see the 3200m radius circle around the base station (B) and the 800m radii around the drones (they are grounded). In the upper-right corner, an input box is shown that is used to prevent specific drones from ascending. Left clicking on the base station initiates the route update (figure B.2) followed by the *landing update*. Right clicking initiates the position update network function. Left clicking on the drone will initiate data send to the base station. Data send on a drone will only execute if its routing table was updated.
Figures B.3 to B.12 show a data send wave formation. Data send was initiated in drone i.
All the abilities of the simulation are not shown, e.g. the data send wave formation when bypassing a failed drone and the recalculation of the routes when a drone fails during the *route update* and position update network functions.

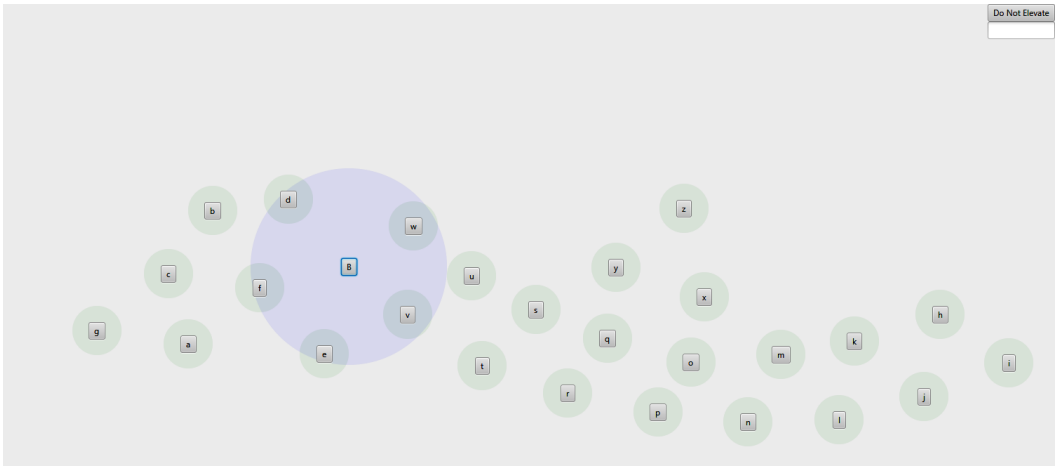


Figure B.1: Simulation start

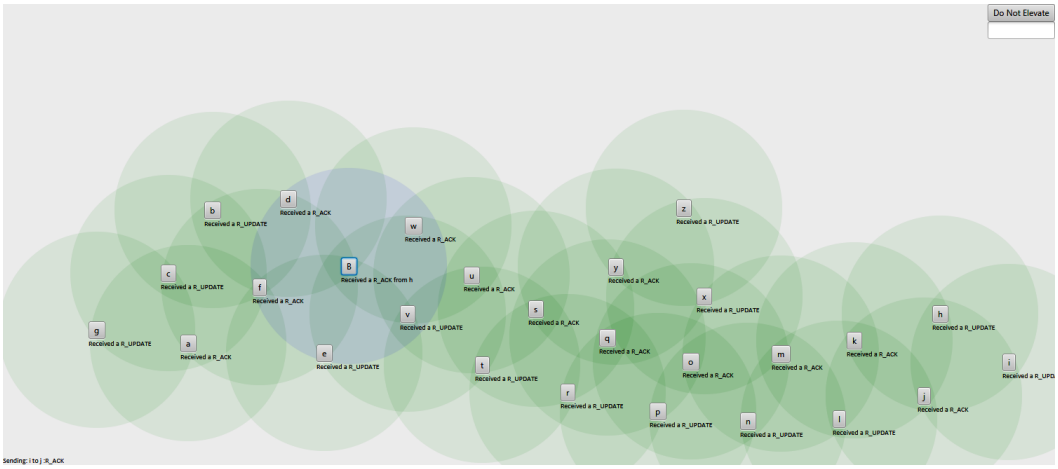


Figure B.2: *Route update* in progress

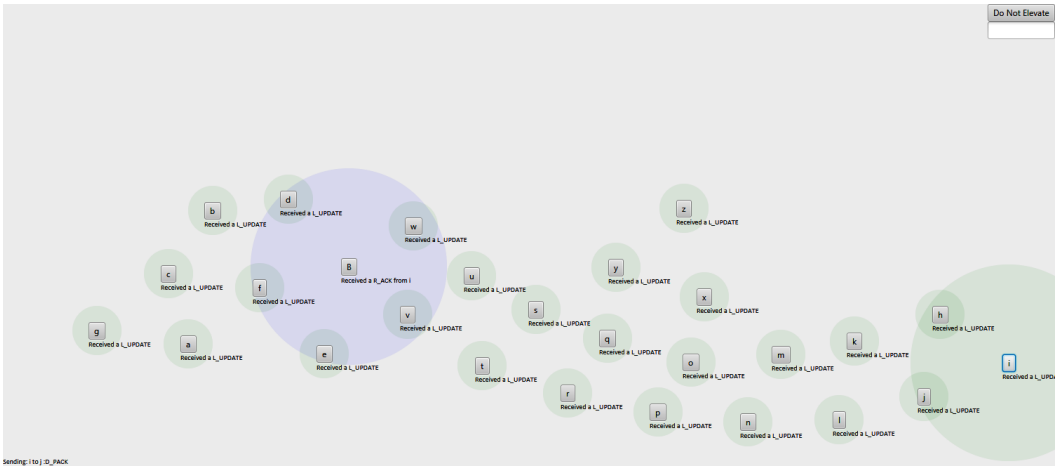


Figure B.3: Node i sending data

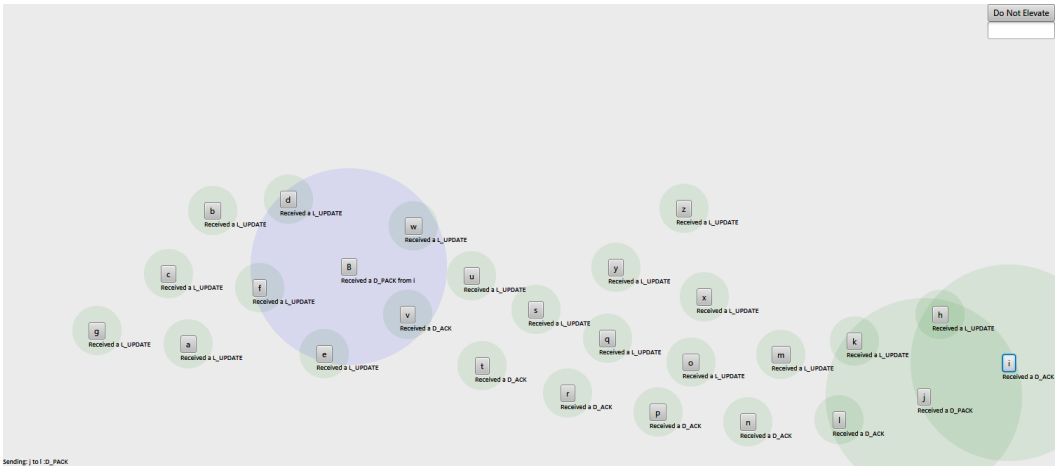


Figure B.4: Node j sending data

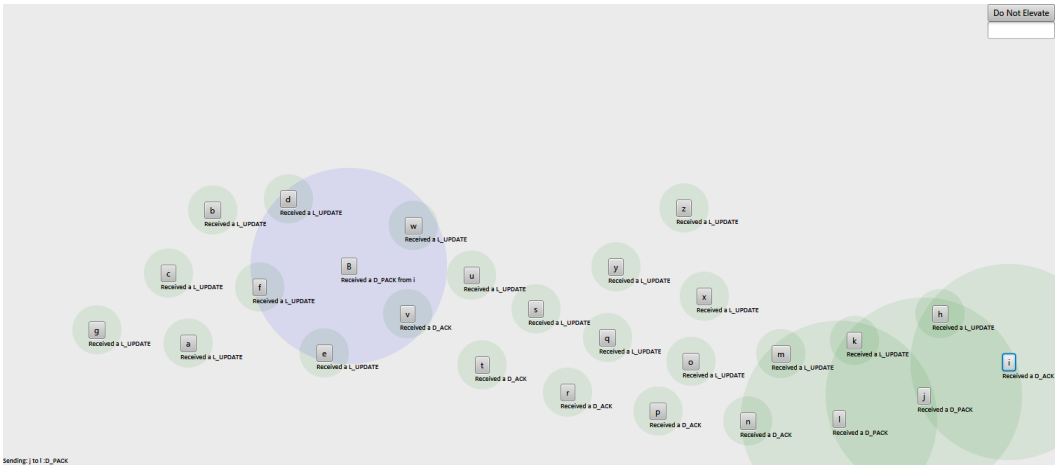


Figure B.5: Node l sending data

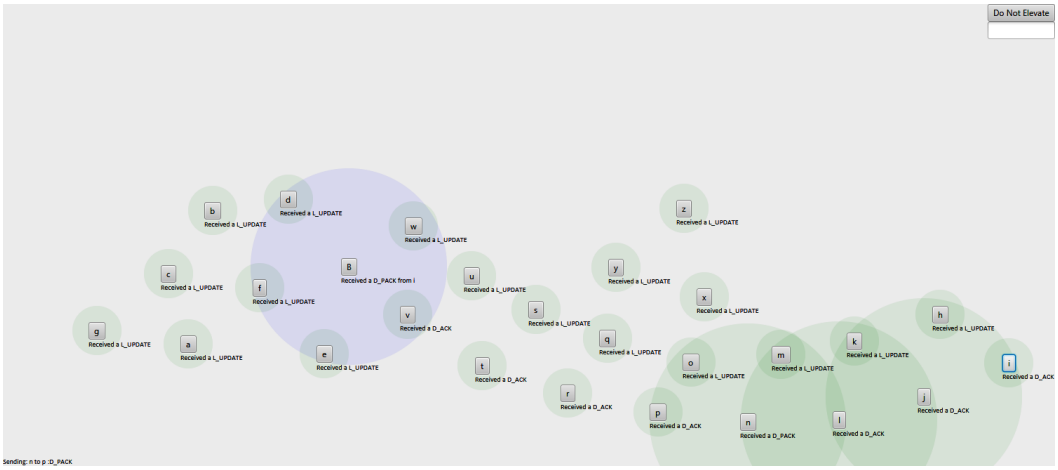


Figure B.6: Node n sending data

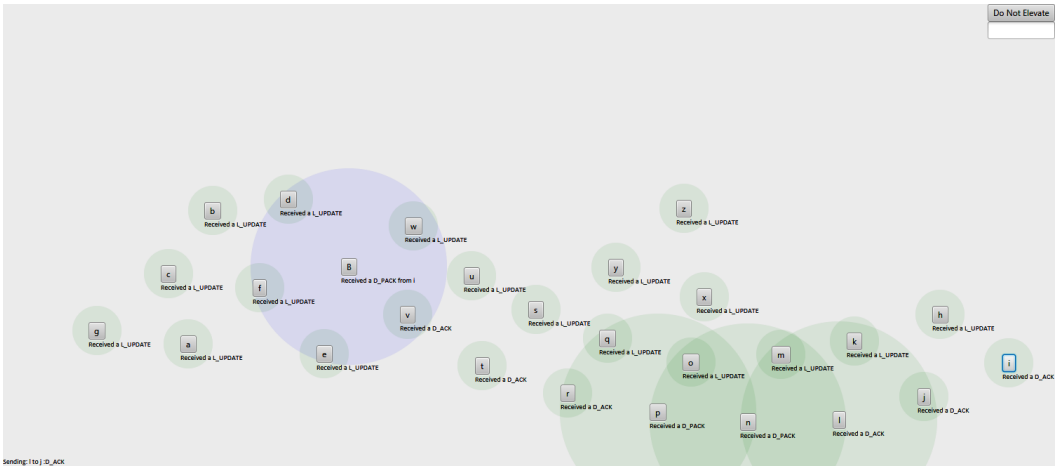


Figure B.7: Node p sending data

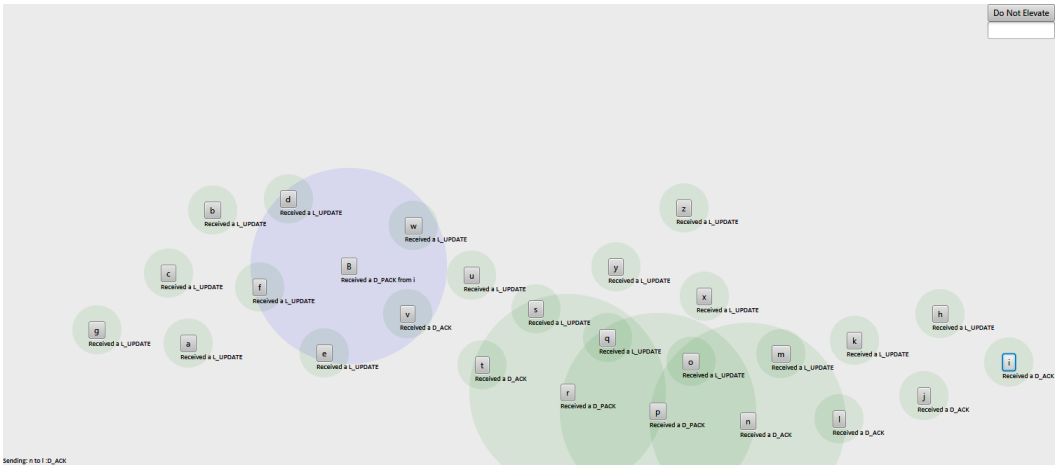


Figure B.8: Node r sending data

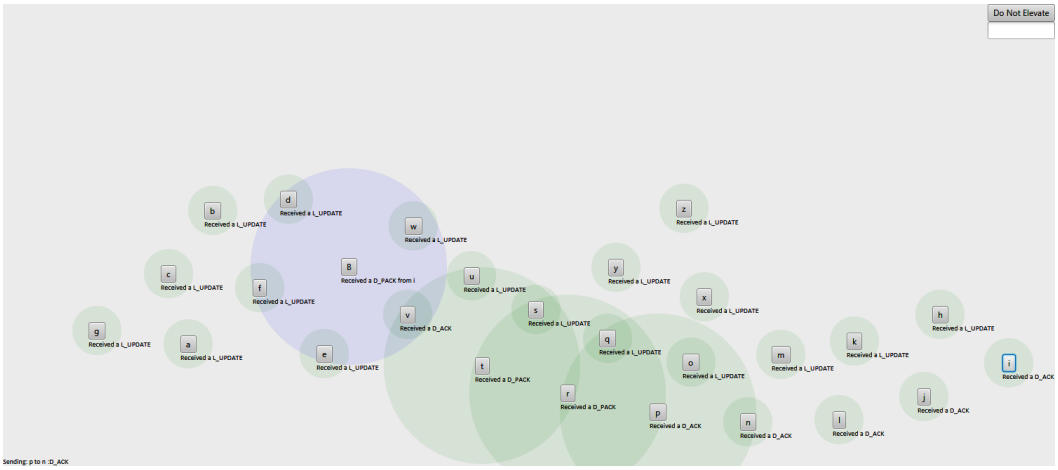


Figure B.9: Node t sending data

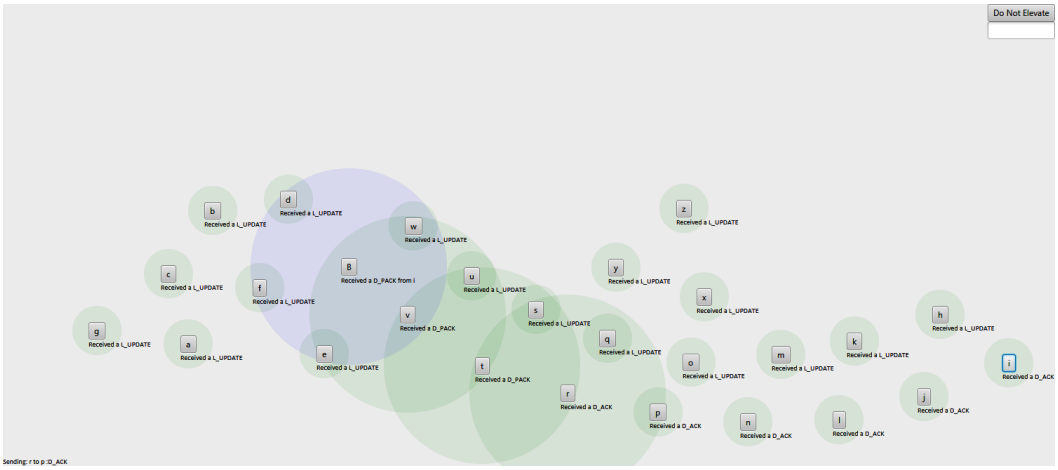


Figure B.10: Node v sending data

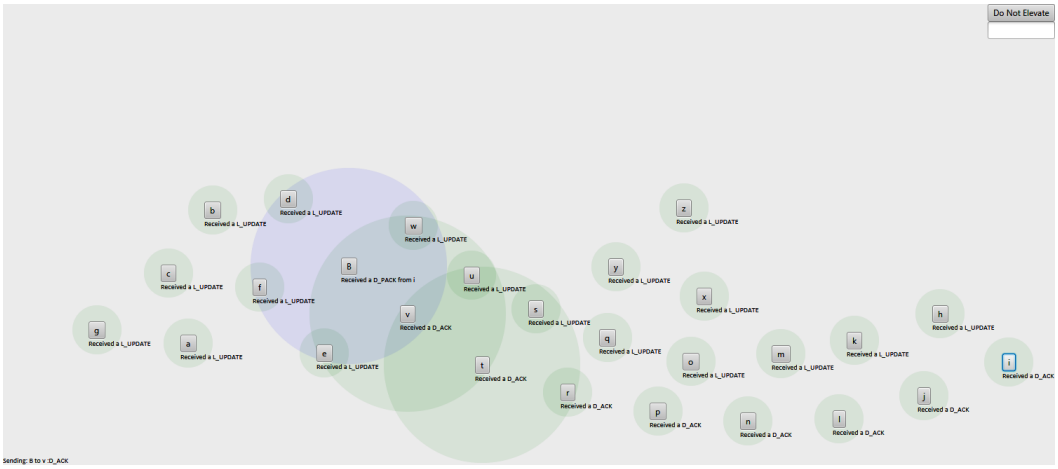


Figure B.11: Node t landing

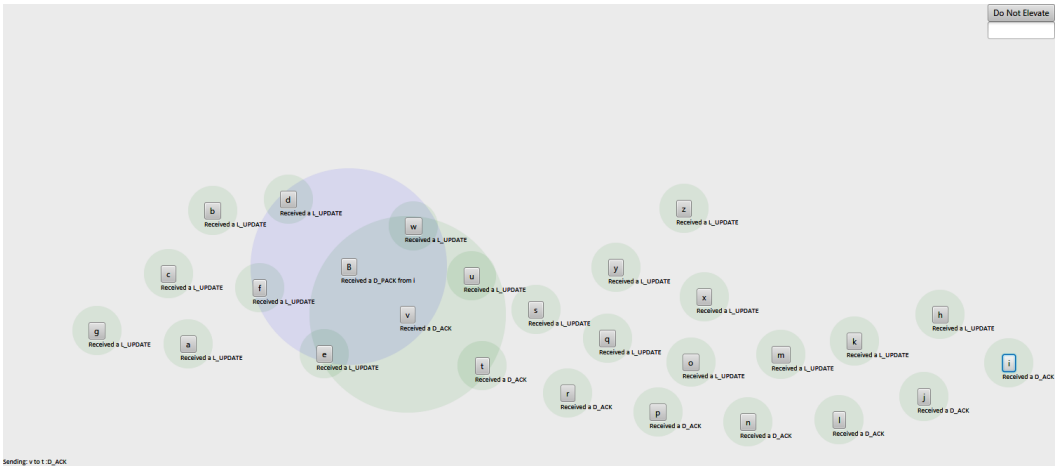


Figure B.12: Node v landing

Appendix C

Performance Tests Program Code

The program code is supplied electronically on the provided CD.
It requires the installation of IntelliJ IDEA as a requisite.