Message Passing Interface for Simulation of Mobile Ad-hoc Networks

Project Report - 9th Semester Software Engineering

ds901e18

Cassiopeia
Department of Computer Science
Aalborg University

Copyright © Aalborg University 2014

Written in LaTeXfrom a template made by Jesper Kjær Nielsen.

Software Engineering
Aalborg University
http://www.cs.aau.dk/

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Message Passing Interface for Simulation of Mobile Ad-hoc Networks

Theme:
Distributed Systems

Project Period:
Fall Semester 2018

Project Group:
ds901e18

Participant(s):
Charlie Dittfeld Byrdam
Jonas Kloster Jacobsen

Supervisor(s):
Jiri Srba
Peter Gjøl Jensen

Copies: 3

Page Numbers: 41

Date of Completion:
January 10, 2019

Abstract:

A Mobile Ad-hoc Network is a decentralised wireless network that requires no pre-existing infrastructure to function. Nodes communicate in the network using radio, and require networking protocols to provide energy efficient communication. The aim of this project is to simulate the protocols, and to provide an alternative to real-life testing, as scaling a real-life test requires a significant amount of effort and investment of both money and time. The goal is to be able to perform repeatable experiments in a control topology environment with up to 1000 devices, faster than a real-life scenario.
We propose a C++ library for writing, and running, simulations of mobile ad-hoc networking protocols, using MPI. With this library, it is possible to write C++ implementations of communication protocols for mobile ad-hoc networks, such as ALOHA or LMAC, and perform repeatable experiments on these using our library, where we emulate the physical hardware, and simulate the radio communication between the emulated hardware with state-of-the-art modelling of link path loss, packet error and collisions.
We introduce the notion of virtual time, where we are able to skip long periods of inactivity when simulating wireless networking protocols.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Aalborg University, May, 2018

Charlie Dittfeld Byrdam
<cbyrda14@student.aau.dk>

Jonas Kloster Jacobsen
<jkja14@student.aau.dk>

Contents

Preface

This report documents the project work of a 9th semester group of Software Engineering Masters students at Aalborg University.

We would like to thank Jiri Srba and Peter Gjøl Jensen for their excellent supervision throughout this project. Additionally, we would like to thank Rasmus Liborius Bruun for his help with link modelling, as well as supplying us with data from the Reachi project.

Chapter 1

Introduction

A Mobile Ad-hoc Network (MANET) is a decentralised wireless network that requires no pre-existing infrastructure, such as routers or access points [10]. Instead, each node in the ad-hoc network are battery powered, and communicates directly with each-other using a radio. Due to this, MANETs rely on wireless networking protocols to provide energy efficient communication in the network. Because of the ad-hoc nature of MANETs, common applications consist of enabling communication in emergency situations such as natural disasters, or for military conflicts.

The goal of this project is to simulate wireless network protocols for communication in a mobile setting. Ideally the capabilities of a wireless network protocol are tested in a real-life scenario, using physical devices for radio communication. This poses interesting challenges such as scalability, and repeatability. Scaling a real-life test requires a significant amount of effort and investment of both money and time, and repeating the same test over and over becomes near impossible, the larger the scale. Our goal is to be able to perform repeatable experiments in a control topology environment with up to 1000 devices, faster than a real-life scenario.

Our project proposes a Message Passing Interface (MPI) C++ library for writing, and running, simulations of the network protocol behind the mesh communication in a MANET, using state-of-the-art modelling of link path loss [3] behind the wireless communication, to simulate packet loss and collisions caused by interfering transmitters, where the physical devices, and the communication between these, are emulated entirely using software. With our library, it is be possible to write a C++ implementation of communication protocols, such as Lightweight Medium Access Protocol (LMAC) [17] or Slotted ALOHA [14], using a simple interface header file resembling a traditional hardware interface, and perform simulations with these, where each physical device is emulated in real-time by different CPUs on the MCC compute cluster at AAU [16].

The primary contribution of this project is the centralised controller that facilitate wireless communication between the emulated physical devices. The centralised controller allow us to simulate wireless communication in virtual time, where the controller is able to skip periods of inactivity, reducing the time required to run real-time simulations. In addition to the centralised controller, we suggest two methods for optimising the computational time required for computing the link path loss, as this poses a significant scalability challenge.

1.1 Related Work

In "Modeling and Efficient Verification of Broadcasting Actors" [19] the authors present an extension to the actor-based modelling language Rebeca [15], that enable broadcast communication between actors (nodes), in order to allow modelling of MANETs. The authors provide a framework to model MANETs for a static topology, with no support for mobility. The same authors further extend Rebeca to add key features of wireless ad hoc networking, such as mobility (dynamic topologies), local broadcasting within a transmission range, and energy consumption in "Modeling and efficient verification of wireless ad hoc networks" [20]. The modelling language lacks features such as lossy transmissions (packet loss) and non-deterministic behaviour, and generally abstracts away from wireless communication, in order to focus on modelling and verification of MANET protocols. Our approach is generally more un-restrictive, as we allow protocol implementations in C++, whereas the paper is restricted to a fixed formalism. One of the major advantages of using Rebeca is that the modelling language makes it possible model check and verify MANET protocol implementations, and explore the full state space for the model of a protocol.

The paper "Modeling and Evaluation of Wireless Sensor Network Protocols by Stochastic Timed Automata" [21] proposes a method to analyse and evaluate Wireless Sensor Network (WSN) protocols using Stochastic Timed Automata, along with the non-deterministic behaviour of WSNs, such as lossy transmission and dynamic topologies. The authors utilise statistical model checking to evaluate the performance of WSN protocols, as well as checking the correctness of the protocols. This approach is somewhat similar to the approach in [19] and [20], but the protocols evaluated are instead modelled using the UPPAAL model checker, and does include non-deterministic behaviour. They show in the paper that their method is able to model and evaluate network topologies of up to 100 nodes by using statistical model checking in UPPAAL.

In the paper "Simulating MANETS: A Study using Satellites with AODV and AntHocNet" [9], the authors present a network simulator for satellite networks called SatSim. The authors argue that a satellite constellation can be thought of as an extreme example of a MANET. SatSim includes features such as a bit error rate, determined for each packet using a representative link budget, where packets are randomly dropped if the bit error rate exceeds a specific threshold. This approach is similar to ours in that we also use a link budget (the link path loss model) to compute the probability for packet errors. Our approach differs, in that we expand upon this by also simulating collisions, caused by interfering transmitters.

Chapter 2

Link Modelling

The goal of this chapter is to be able to estimate the probability, based on the topology of a wireless network, of how likely it is that packet loss will happen during a transmission between two nodes. Section 2.1 presents the method for simulating the link model in a MANET, Section 2.2 presents the method for computing the packet loss probability during a transmission, and Section 2.3 proposes two ways for optimising the computational time required to compute the link model.

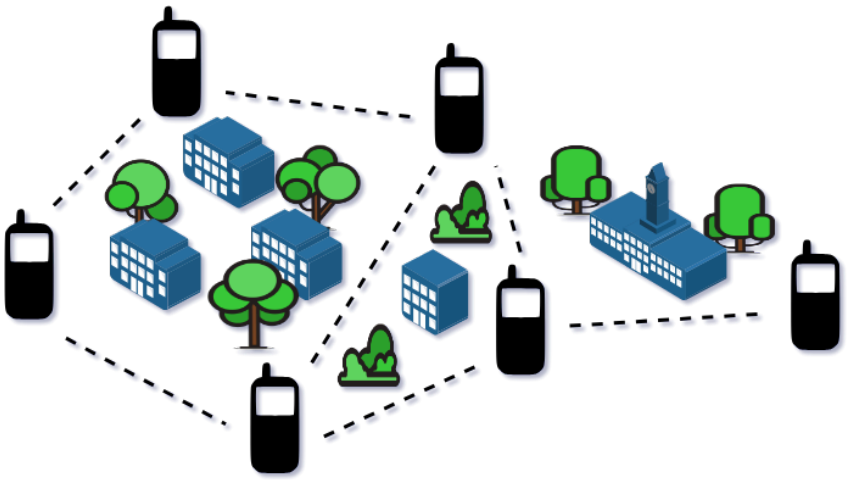


Figure 1: A sample wireless network topology.

Figure 1 shows a sample wireless network topology for a Mobile Ad-hoc Network (MANET). The network consists of mobile devices (nodes), and the communication between these (links). Nodes are "linked" with other nodes when they are able to communicate wirelessly. Wireless communication relies on the transmission and reception of electromagnetic waves [3, p. 10] (radio signals), and the strength, or quality, of a wireless link is described by the signal loss occurring when propagating the signal from transmitter to receiver, and is measured by the Received Signal Strength Indication (RSSI) (a negative value, where a value close to 0 is better). In [3], the term path loss is used to describe this signal loss, and is determined in part by the physical distance between transmitter and receiver, but also by physical objects and terrain, like buildings or forests. Section 2.1 will elaborate further on the path loss of a link, as well as present a method for simulating the path loss for a mobile network topology. Another major consideration for mobile networks is that the topology is dynamic. Nodes move around, causing existing links to disappear, or new links to appear, thus changing the topology of the network.

2.1 Link Path Loss

In this section, we present the method for simulating link path loss (the link model) from [3]. We want to simulate the performance of nodes in a MANET. The performance is, however, heavily dependent on network conditions and the capabilities of the technology [3, p. 10]. The author of [3] presents methods for evaluating the performance of a wireless networks, and proceeds to introduce methods for simulating path loss on a multi-link model, based on a real-world performance measurement. Section 2.1.2 will summarise the method for simulating path loss on a MANET, and Section 2.1.3 present the method, as well as an example of simulating the path loss on a network topology.

2.1.1 Units

In the following sections, we use different units of measurements. For distances, all measurements (e.g., the distance between two nodes) will be in meters. We describe the strength of a radio signal using dB [12], and the transmission power, as well as the RSSI as dBm [12]. The RSSI is on a logarithmic scale, and the closer the value is to 0 the better.

2.1.2 Path Loss

The path loss of a link is dependent on the distance between transmitter and receiver, as well as a stochastic shadow fading term. The path loss vector $\overrightarrow{l_{pl}}$ is the sum of two parts:

$\overrightarrow{l_d}$ A deterministic distance dependent part, which describes the mean signal attenuation at any given link distance (distance between transmitter and receiver), in decibel (dB). It is a vector of size n, where n is the number of links in the network.

$\overrightarrow{l_{fading}}$ A stochastic slow/shadow fading part, which specifies the local mean of the signal, in dB. The slow fading variable is, more or less, constant in the same local area, as it is caused by terrain, buildings, vegetation, and cars. It is a vector of size n, where n is the number of links in the network.

$$\overrightarrow{l_{pl}} = \overrightarrow{l_d} + \overrightarrow{l_{fading}} \tag{2.1}$$

The deterministic distance dependent path loss is an independent value, while the stochastic slow/shadow fading value depends on physical objects and terrain. This means that links existing in the same physical environment should have a similar slow fading. The similarity is modelled by introducing a correlation between the slow fading on different links:

Cross correlation (interlink) describes the correlation between two or more links in the same spatial (physical) environment (**spatial correlation**) [3, p. 16].

Auto correlation (intralink) describes the correlation in the development of slow fading for a single link over time (**temporal correlation**) [3, p. 15].

The path loss can be used, along with the transmission power in dBm to simulate the RSSI on a link.

2.1.3 Simulating Link Path Loss

In this section, we summarise how path loss and RSSI values are simulated for a MANET according to [3], and we present how we expect to use this in our own simulations. Again, note that the values used throughout this section are based on on-site measurements made with the Reachi devices in the Philippines and are heavily dependent on the environment, as described in [3].

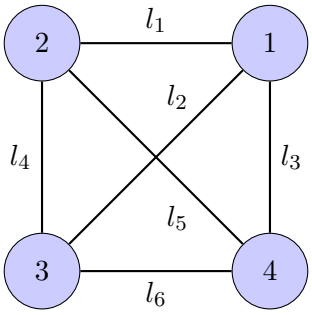


Figure 2: Sample graph **G** with 4 nodes and 6 links.

Figure 2 is a sample network topology **G** consisting of 4 nodes, with 6 unique links. In the sample, the links l_1 and l_4 are both 100 meters long, and the angle between them is 90°. Links are undirected and show equal loss in both directions. The length of a l link will be denoted by the function $d(l)$, $nodes(l)$ is a function that return the set of nodes of a link l, $\mathbf{N}$ is the set of nodes in the network topology, and finally, the angle between two unique links, k and l, will be denoted by the function $\theta(k, l)$, where k and l are unique links that share a common node ($k \neq l$ and $nodes(k) \cap nodes(l) \neq \emptyset$).

The graph **G** would be equivalent to the following link vector:

$$\overrightarrow{l_{\mathbf{G}}} = \begin{bmatrix} l_1 & l_2 & l_3 & l_4 & l_5 & l_6 \end{bmatrix}^T \tag{2.2}$$

The length of the unique link vector of a fully connected network is denoted by the function $len(\overrightarrow{l})$.

$$len(\overrightarrow{l}) = \sum_{i=1}^{|\mathbf{N}|-1} i = \frac{|\mathbf{N}|(|\mathbf{N}|+1)}{2} - |\mathbf{N}| \tag{2.3}$$

First, we compute the distance dependent part of the path loss. The deterministic distance dependent part $\overrightarrow{l_d}$ is obtained for each unique link in the network by:

$$\overrightarrow{l_d} = \begin{bmatrix} 10\gamma \log_{10}(d(l_1)) - c \\ 10\gamma \log_{10}(d(l_2)) - c \\ \vdots \\ 10\gamma \log_{10}(d(l_n)) - c \end{bmatrix} \tag{2.4}$$

where the path loss exponent $\gamma = 5.5$, the constant offset $c = -18.8$, $d(l_i)$ is the distance between the two nodes of the link, in meters. The path loss exponent is obtained by Total Least Squares (TLS) regression in [3], and the constant offset is the signal strength of a link with $d(l) = 1$, in dB.

For our sample network $\mathbf{G}$, we can compute the distance dependent part as follows:

$$\overrightarrow{l_{d,\mathbf{G}}} = \begin{bmatrix} 55\log_{10}(d(l_1)) - 18 \\ 55\log_{10}(d(l_2)) - 18 \\ \vdots \\ 55\log_{10}(d(l_6)) - 18 \end{bmatrix} = \begin{bmatrix} 92 \\ 100.2 \\ \vdots \\ 92 \end{bmatrix} \tag{2.5}$$

With the distance dependent part computed, we can move on to the stochastic slow fading part. This part is a bit more complicated, as it is not an independent value like the distance dependent part. Instead, the values depend on the correlation between links, and as such, we need the correlations presented in Section 2.1.2. First, we introduce the spatial correlation. $\mathbf{C}$ is the correlation matrix determining the correlation coefficient between links. $\mathbf{C}$ is a quadratic symmetric matrix of size $M \times M$, where $M = len(\overrightarrow{l})$.

$$\mathbf{C} = \begin{cases} r(k,l) & \text{if } k \neq l \text{ and } nodes(k) \cap nodes(l) \neq \emptyset \\ 1 & \text{if } k = l \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The function $r(k,l)$ is an auto-correlation function, and it is parameterised based on the Reachi measurements in [3].

$$r(k,l) = 0.595e^{-0.064*\theta(k,l)} + 0.092 \tag{2.7}$$

With this, we can generate the correlation matrix for our sample graph $\mathbf{G}$ as:

$$\mathbf{C_G} = \begin{bmatrix} 1 & 0.125 & 0.094 & 0.094 & 0.125 & 0 \\ 0.125 & 1 & 0.125 & 0.125 & 0 & 0.125 \\ 0.094 & 0.125 & 1 & 0 & 0.125 & 0.094 \\ 0.094 & 0.125 & 0 & 1 & 0.125 & 0.094 \\ 0.125 & 0 & 0.125 & 0.125 & 1 & 0.125 \\ 0 & 0.125 & 0.094 & 0.094 & 0.125 & 1 \end{bmatrix} \tag{2.8}$$

The correlation matrix $\mathbf{C}$ is used to create the covariance matrix $\mathbf{\Sigma} = \sigma^2\mathbf{C}$ by multiplying the correlation matrix with a standard deviation of $\sigma = 11.4$ dB squared. Next, we draw an Independent and Identically Distributed (IID) multivariate Gaussian vector, with the standard deviation $I = 1$.

$$\overrightarrow{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_{len(\overrightarrow{l})} \end{bmatrix}^T \sim N(0, I) \tag{2.9}$$

The independent variables in the vector are made dependent by multiplication with the lower triangular Cholesky decomposition [5, p. 143][13, p. 100] (Algorithm 1) of the covariance matrix $\mathbf{Q} = cholesky(\mathbf{\Sigma})$.

$$\overrightarrow{l_{fading}} = \mathbf{Q}\overrightarrow{x} \tag{2.10}$$

For our sample graph $\mathbf{G}$, this could be:

$$\overrightarrow{l_{fading,\mathbf{G}}} = \mathbf{Q_G} \cdot \begin{bmatrix} 0.122 \\ -1.087 \\ 0.684 \\ -1.075 \\ 0.033 \\ 0.744 \end{bmatrix} = \begin{bmatrix} 1.390 \\ -12.466 \\ 6.170 \\ -13.837 \\ -0.158 \\ 6.413 \end{bmatrix} \tag{2.11}$$

Note that as $\overrightarrow{x}$ is stochastic, the values in Equation 2.11 are a random realisation of the stochastic shadow fading part.

Next, the shadow fading part is expanded to include the temporal correlation. The vector $l_{fading}(t)$ describes the shadow fading path loss at time t. As the distance dependent part is not time dependent, it needs no further modifications.

$$\overrightarrow{l_{fading}}(t+\Delta t) = \overbrace{\mathbf{Q}(t+\Delta t)\overrightarrow{x}}^{spatial correlation} \overbrace{\sqrt{1-\rho_{\Delta t}} + \overrightarrow{l_{fading}}(t)\rho_{\Delta t}}^{temporal correlation} \tag{2.12}$$

The temporal correlation is computed based on the temporal correlation coefficient, $\rho_{\Delta t}$, describing the correlation after both transmitter and receiver have moved $|d_t|$ and $|d_r|$ meters, respectively, and with a decorrelation distance of 20 meters.

$$\rho_{\Delta t} = e^{-\frac{|d_t|+|d_r|}{20}\ln(2)} \tag{2.13}$$

Again, coming back to our sample graph $\mathbf{G}$, and assuming that all devices would have moved 10 meters in the same direction (preserving angles and distances between links), the stochastic shadow fading part at $t = 1$ would determined by:

$$\overrightarrow{l_{fading,\mathbf{G}}}(t=1) = \mathbf{Q_G} \cdot \overrightarrow{x} \cdot \sqrt{1-\rho_{\Delta t}} + \overrightarrow{l_{fading,\mathbf{G}}}(t=0) \cdot \rho_{\Delta t} \tag{2.14}$$

where $\rho_{\Delta t} = e^{-\frac{10+10}{20}\ln(2)} = 0.5$, $\overrightarrow{x}$ would be a new realisation of the Gaussian vector, and $\mathbf{Q_G}$ would be recomputed based on the new locations of the nodes, should the angles between the nodes have changed.

With this, the vectors $\overrightarrow{l_d}$ and $\overrightarrow{l_{fading}}$ can be combined as in Equation 2.1, to generate the path loss vector $\overrightarrow{l_{pl}}$ for all unique links in the network.

$$\overrightarrow{l_{pl,\mathbf{G}}}(t=0) = \overrightarrow{l_{d,\mathbf{G}}} + \overrightarrow{l_{fading,\mathbf{G}}}(t) = \begin{bmatrix} 93.39 \\ 87.734 \\ 98.17 \\ 78.163 \\ 100.042 \\ 98.413 \end{bmatrix} \tag{2.15}$$

All that remains is to subtract the path loss for a particular link l, from the transmission power of the transmitting node, to compute the RSSI in dBm. $RSSI_{dBm}(n, m, t = 0)$ denotes the RSSI on the link $l_{n,m}$ between nodes n and m at time t, in dBm.

$$RSSI_{dBm}(n, m, t = 0) = tx_{power} - l_{pl,l_{n,m}}(t) \tag{2.16}$$

For example, if node 1 in our sample graph $\mathbf{G}$ transmits with a transmission power of 26 dBm to node 2, the RSSI on the receiving node would be $RSSI_{dBm}(n_1, n_2) = 26 \text{ dBm} - l_{pl,\mathbf{G},l_{n_1,n_2}}(t) = -64.610$ dBm. In the following section, we show that a dBm of -64.610 on a link will have a probability for packet loss of approximately 0.001 %, given that no other nodes transfer at the same time.

2.2 Packet Error Probability

Now that we are able to simulate the RSSI when transmitting from one node to another, we need to be able to simulate whether the packet should arrive on the receiving node. With wireless radio communication, there is a chance that a receiving node may not receive the entire packet correctly, which is why we need some way of computing the probability of this happening. For this, we introduce the packet error probability. The computations in this section are derived from [6], as well as personal communication with the author of [3].

2.2.1 Radio Model

First we calculate the noise power $P_{N,db} = thermal\_noise + noise\_figure$ in dB. The noise power is calculated with the thermal noise and noise figure, and is the level of background noise affecting the wireless radio communication. For the Reachi devices we assume that the $thermal\_noise = -119.66$ dB and the $noise\_figure = 4.2$ dB.

Next we need to add the noise from interfering transmissions happening at the same time. We do this by adding the sum of the RSSI from interfering transmitters to the noise power $P_{N,dB}$, giving us $P_{NI,dB}$ on the link between the receiving node n_r and the transmitting node m_t. The set of currently transmitting nodes are denoted by $nodes_t$ and the function $RSSI_{dBm}(n, m)$ denotes the RSSI on the link between nodes n and m.

$$P_{NI,db}(n_r, m_t, nodes_t) = 10 \log_{10} \left(10^{\frac{P_{N,dB}}{10}} + \sum_{m \in nodes_t - \{m_t\}} 10^{\frac{RSSI_{dBm}(n_r, m)}{10}} \right) \tag{2.17}$$

Note that as both the noise power $P_{N,dB}$ and the RSSI is in dB (a logarithmic scale), we first need to convert the values to linear scale, before computing the sum of the noise and interference power, and then convert the value back into logarithmic scale (dB).

Next we calculate the signal to noise (and interference) ratio γ_{dB} in dB. The ratio γ_{dB} is the ratio between signal and noise, that compares the level of the signal to the level of the background noise (including the interference from other transmissions), and is computed by subtracting the noise power $P_{NI,db}$:

$$\gamma_{dB}(n_r, m_t, nodes_t) = RSSI(n_r, m_t) - P_{NI,dB}(n_r, m_t, nodes_t) \tag{2.18}$$

We use this to compute the bit error probability P_b:

$$P_b(n_r, m_t, nodes_t) = \frac{1}{2} erfc \left(\sqrt{\left(\frac{10^{\frac{\gamma_{dB}(n_r, m_t, nodes_t)}{10}}}{2} \right)} \right) \tag{2.19}$$

which we finally can use to compute the packet error probability P_p. The packet error probability is the probability that we experience a bit error for any of the bits in our packet, during

transmission.

$$P_p(n_r, m_t, nodes_t, packetsize) = 1 - (1 - P_b(n_r, m_t, nodes_t))^{packetsize} \tag{2.20}$$

2.2.2 Example

In Section 2.1, we computed the link model for a sample network topology $\mathbf{G}$. Equation 2.21 shows a vector containing the RSSI, in dBm, for each of the six links in the network, assuming a transmission power of 26 dBm. For the following, it is assumed that $thermal\_noise = -119.66$ dB, the $noise\_figure = 4.2$ dB, and $packetsize = 160$ bits (which is the size of a header packet for the Reachi devices).

$$\overrightarrow{RSSI_{dBm,\mathbf{G}}} = \begin{bmatrix} -67.390 \\ -61.734 \\ -72.170 \\ -52.163 \\ -74.042 \\ -72.413 \end{bmatrix} \tag{2.21}$$

If we assume that n_2 is currently listening, and the currently transmitting nodes $nodes_t = \{n_1, n_4\}$. What is the probability for packet error on the link between nodes n_2 and n_1 with interference from node n_4? First, we compute the noise power $P_{NI,db}$, according to Equation 2.17:

$$P_{NI,db}(n_2, n_1, nodes_t) = 10 \log_{10} \left(10^{\frac{(-119.66+4.2)}{10}} + 10^{\frac{-74.042}{10}} \right) = -74.041 \tag{2.22}$$

We subtract the noise power $P_{NI,db}$ from the RSSI to get the signal to noise (and interference) ratio γ_{dB}:

$$\gamma_{dB}(n_2, n_1, nodes_t) = -67.390 - (-74.041) = 6.651 \tag{2.23}$$

With which we can compute the bit error probability:

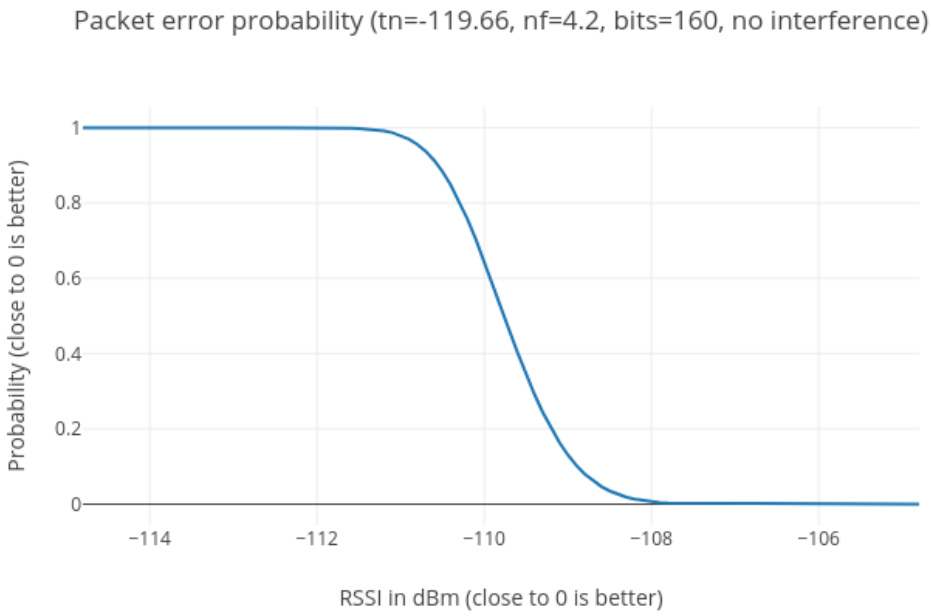
$$P_b(n_2, n_1, nodes_t) = \tfrac{1}{2} erfc \left(\sqrt{\left(\frac{10^{\frac{6.651}{10}}}{2} \right)} \right) = 0.000536 \tag{2.24}$$

Finally we can compute the packet error probability using the bit error probability:

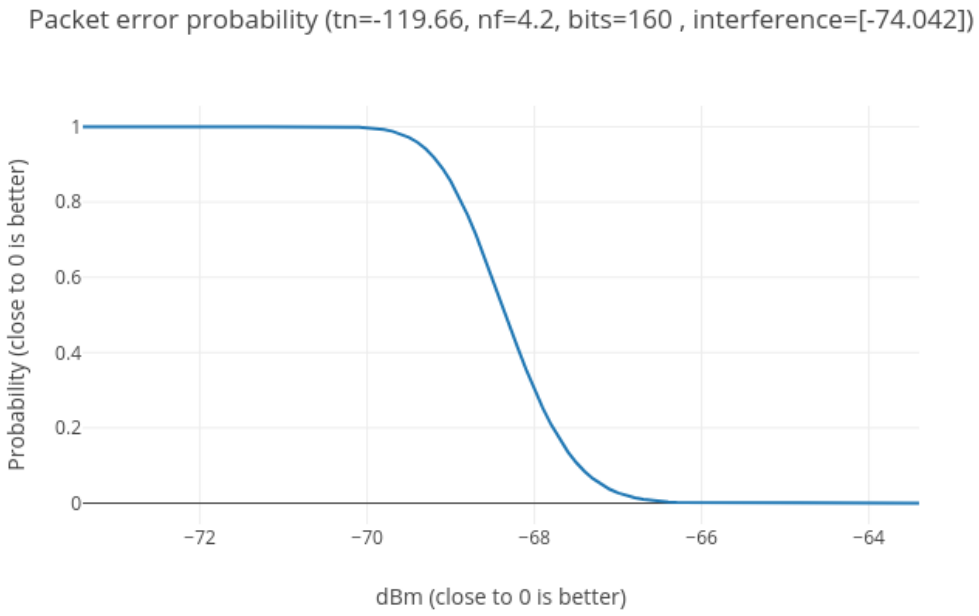
$$P_p(n_2, n_1, nodes_t, packetsize) = 1 - (1 - 0.000536)^{160} = 0.082 \tag{2.25}$$

This gives us a 8.2 % probability that we will experience packet error on the link from n_2 to n_1, with a single interfering transmitter, which is a significant difference in relation to the same transmission with no interfering transmitters. To demonstrate this, Figure 3a shows the probability for packet error with no interfering transmitters. According to the figure, an RSSI of approximately -103.0 dBm would have a probability for packet error close to zero, and an RSSI of approximately -110.0 dBm would have a probability for packet error close to 100 %. If we look back to the RSSI vector, $\overrightarrow{RSSI_{dBm,\mathbf{G}}}$, we see that the RSSI on the link between nodes n_1 and n_2 is -67.390 dBm, which is significantly better than the -103.0 dBm we see in the graph, but with just a single interfering transmitter, the probability for packet error increases to 8.2 %, which corresponds to what we see in Figure 3b, where an RSSI of approximately

−66.0 dBm is required for a probability for packet error close to zero, with a single interfering transmitter.



(a) Probability for packet error on a link with no interfering transmitters.



(b) Probability for packet error on a link with a single interfering transmitter.

Figure 3: Probability for packet error with and without interfering transmitters.

2.3 Optimising the Link Model

In this section we explore two methods for optimising the computational time required for computing the link model introduced in Section 2.1. Section 2.3.1 presents the Cholesky decomposition, along with the issues we have experienced in using the decomposition. In Section 2.3.2, the clustering addition to the link model is presented, along with a formalised problem statement and discussion of the choice of algorithm, and its implementation.

2.3.1 The Cholesky Decomposition

In Section 2.1.3, we utilised the Cholesky decomposition on the covariance matrix in the stochastic shadow fading part. The Cholesky decomposition is a decomposition algorithm for symmetric, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose, and is primarily used for solving systems of linear equations [13]. In this Section, we present and describe the Cholesky decomposition, as well as the problems the decomposition creates for our computation time of the stochastic shadow fading part of the link model, as well as possible ways for us to optimise our usage of the Cholesky decomposition. Algorithm 1 contains a pseudo code description of the Cholesky decomposition.

Algorithm 1: Cholesky decomposition

Result: The Cholesky decomposition of the input matrix

```
1  Function Cholesky(matrix, N):
2      result ← empty matrix of size N × N
3      for n ← 0, n < N do
4          for m ← 0, m < n + 1 do
5              sum ← 0
6              for i ← 0, i < m do
7                  sum ← sum + result_{n,i} · result_{m,i}
8              if n = m then
9                  if matrix_{n,n} − sum ≤ 0 then
10                     throw error; matrix is not positive-definite
11                 results_{n,m} ← √(matrix_{n,n} − sum)
12             else
13                 results_{n,m} ← 1/result_{m,m} · (matrix_{n,m} − sum)
14     return result
```

The first issue we have found with the Cholesky decomposition, or more specifically with the covariance matrix, is that the covariance matrix is not guaranteed to be a positive-definite matrix. The covariance matrix is based on the relation between links in the network, which means that whether the matrix is positive-definite or not is entirely based on the network. To work around this, we have employed a tool called NearPD [4] to transform our covariance matrix into a new matrix that has the positive-definite property, while minimising the Frobenius norm [18] of the difference between the original and the new matrix. This significantly increases the time required to compute the link model, however, which leads us to our next major issue: The computational time required by the Cholesky decomposition itself. The computational time required for the NearPD tool and the Cholesky decomposition can be seen in Table 1.

The Cholesky decomposition has a computational complexity of $O(n^3)$ [13]. With a fully

connected network of 1000 nodes, we would have a $\frac{1000(1000+1)}{2} - 1000 = 499500$ (Equation 2.3) unique links, which means that our correlation (and covariance) matrix would be of size 499500×499500. This is a major issue, as we would like to be able to compute the link model in a relatively short amount of time. To combat this issue, we have two possible solutions: removing links with a distance over a certain threshold, and clustering nodes that are very close to each-other. We present the first solution in this section, and clustering in Section 2.3.2.

Nodes	Links	NearPD	Cholesky
10	45	3 milliseconds	<1 millisecond
20	190	88 milliseconds	3 milliseconds
30	435	1 seconds	34 milliseconds
40	780	6 seconds	200 milliseconds
50	1225	32 seconds	720 milliseconds
60	1770	113 seconds	3 seconds
70	2415	285 seconds	6 seconds
80	3160	584 seconds	12 seconds
90	4005	22 minutes	26 seconds
100	4950	35 minutes	45 seconds
110	5995	75 minutes	93 seconds
120	7140	127 minutes	155 seconds
130	8385	235 minutes	250 seconds
140	9730	Timeout	...

Table 1: Computation time measurement for NearPD and Cholesky decomposition.

Table 1 shows time measurements from running the cholesky decomposition computation with different network topology sizes with the NearPD tool. The measurements shows that the NearPD tool is not fast enough, as at only 130 nodes with 8385 links the tool takes 235 minutes to compute the positive-definite matrix. This is not fast enough and a better solution will have to be found. The cholesky decomposition computation speed is also problematic, as the goal is 1000 nodes and already at 120 nodes the decomposition takes more than 2 minutes.

Distance Threshold

In Section 2.1, we saw that the distance dependent path loss has significantly more importance in the total path loss than the stochastic shadow fading part. In Equation 2.5, we see that the distance path loss is 92 dBm for links of 100 meters, and 100.2 dBm for the diagonal links of 141.42 meters, and in Equation 2.11, we see (stochastic) path loss values between -13.837 and 6.413. This means that, entirely based on the distance part of the link model, according to the formula for computing packet error probability in Section 2.2, with a distance of 1000 meters, and a transmission power of 26 dBm, we would have a link path loss of 147 dBm (disregarding the stochastic shadow fading part of the path loss), which in turn would mean that the RSSI on the receiving end of the link would be $26 - 147 = -121$, or equivalent to a packet loss probability of 99.999 % (assuming the noise figure and thermal noise is the same as in Section 2.2, and the packet size is 160 bits). Hence removing links that are far away can potentially reduce the size of the correlation matrix significantly. To show this, we ran an experiment, where we generated 1000 nodes with random locations in a 25×25 kilometre area, created links between these nodes based on their location, and iteratively scaled the maximum distance allowed between nodes (the distance threshold) and calculated the link path loss, based

on the distance threshold. The results of this experiment can be seen in Table 2. Recall that a fully connected network topology of 1000 nodes would have a total of 499500 links. With a distance threshold of 1000 meters, we get a total of 2346 links in our 25×25 kilometre area, which is a significant improvement as we would be able to compute the Cholesky decomposition in approximately six seconds (disregarding the positive-definiteness of the correlation matrix) according to Table 1.

Distance threshold	Links	Probability
200 meters	92	0.001 %
250 meters	139	0.001 %
300 meters	208	0.001 %
350 meters	296	0.001 %
400 meters	394	0.001 %
450 meters	490	0.001 %
500 meters	592	6.000 %
550 meters	723	82.333 %
600 meters	846	99.999 %
650 meters	994	99.999 %
700 meters	1164	99.999 %
750 meters	1349	99.999 %
800 meters	1507	99.999 %
850 meters	1714	99.999 %
900 meters	1910	99.999 %
950 meters	2102	99.999 %
1000 meters	2346	99.999 %

Table 2: Results from the distance threshold experiments.

2.3.2 Clustering

The second option for optimising the computation of the link model, is to introduce clustering of nodes. As mentioned in Section 2.1.2, links existing in the same physical environment should experience similar shadow fading path loss. This means that we are be able to cluster nodes with a minimum loss of precision, provided that we minimise difference of the correlation (the angle) between links, before and after clustering.

Metric Space

A metric space is a pair $(\mathcal{X}, d)$ where $\mathcal{X}$ is a set and $\mathbf{d} : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ is a metric, satisfying the following axioms:

- Positivity: $d(x, y) \geq 0$
- Reflexivity: $d(x, y) = 0 \iff x = y$
- Symmetry: $d(x, y) = d(y, x)$
- Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Our chosen metric space is $\mathbb{R}^2$ using Euclidean distance.

Problem Statement

A set $\mathbf{N} \subseteq \mathcal{X}$, is provided together with a parameter k, $k \leq |\mathbf{N}|$. Initially, the set $\mathbf{N}$ is a set of clusters, each containing a single node. The goal is to find a set of clusters, such that the computation time required for computing the link model can be reduced, while minimising the loss of precision in the stochastic shadow fading part. We minimise the loss of precision by minimising the difference of the correlation ($\Delta\theta$) between links to any node in a cluster, to the centroid of the cluster. The centroid of a cluster x is defined as $cent(x) = \sum_{n \in x} n \div |x|$.

The problem is defined as follows:
For a metric space $(\mathcal{X}, d)$,

- Input: A set $\mathbf{N} \subseteq \mathcal{X}$ and a parameter k, $k \leq |\mathbf{N}|$.
- Output: A set of clusters $C \subseteq 2^{\mathbf{N}}$, such that

  I $\bigcup_{x \in C} x = \mathbf{N}$,
  II $\forall x, y \in C$, if $x \neq y$ then $x \cap y = \emptyset$, and
  III $|C| \leq k$

- Goal: Minimise the $\Delta\theta(C)$ function:

$$\Delta\theta(C) = \sum_{n_1,n_2,n_3 \in \mathbf{N}} \left(r(l_{n_1,n_2}, l_{n_1,n_3}) - r(l_{cent(C(n_1),cent(C(n_2)}, l_{cent(C(n_1),cent(C(n_3)}) \right)^2$$

Figure 4 illustrates the goal, where we want to minimise the correlation difference $\Delta\theta$ between the nodes n, m, and u (as well of the rest of the nodes), to the centroids of their respective clusters $C(n)$, $C(m)$, and $C(u)$, where $C(n)$ is the cluster in C that contains n, such that $n \in C(n)$

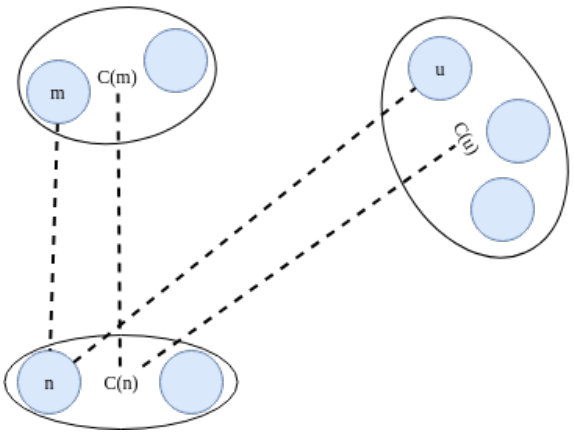


Figure 4: Minimising the difference of the correlation ($\Delta\theta$) between links to nodes in a cluster to the centroid of the cluster.

Approximation using OPTICS

We have chosen a greedy approach using the Ordering Points To Identify the Clustering Structure (OPTICS) algorithm [1] to approximate solutions for our clustering problem. OPTICS is an algorithm for finding density-based clusters, and the run-time of the algorithm is O($n \cdot$

run-time of a neighbourhood query) [1, p. 53]. We chose the OPTICS algorithm based on a survey on density-based clustering algorithms in [2].

The algorithm requires three parameters: our input set $\mathbf{N}$, ε, describing the maximum distance between two nodes to consider for clustering, and $MinPts$, describing the minimum number of nodes required to form a new cluster. For our case, the parameter $MinPts = 2$. With this, the OPTICS algorithm will return the set of clusters $C \subseteq 2^{\mathbf{N}}$, where any nodes not satisfying the $MinPts$ parameter will be a singleton cluster. Algorithm 2 presents the pseudo code description of our greedy approach, that repeatedly computes the a set of clusters C, until $|C| \leq k$, each time incrementing the ε variable by 10 meters.

Algorithm 2: Greedy approach using the OPTICS algorithm.

Result: A set of clusters $C \subseteq 2^{\mathbf{N}}$, where $|C| \leq k$

```
Function Cluster(N, k):
    ε ← 0 m
    C ← OPTICS(N, ε, 2)
    repeat
        ε ← ε + 10 m
        C ← OPTICS(N, ε, 2)
    until |C| ≤ k
    return C
```

Experiment

To measure the performance of our greedy approach, we conducted an experiment where we generated 1000 nodes with random locations in a 10×10 kilometre area, and ran the `Cluster` algorithm from Algorithm 2, with different values for the k parameter. The purpose of this experiment is to measure the computational time used to form a number of clusters $|C| \leq k$. The results of this experiment can be seen in Table 3.

Max clusters (k)	Resulting clusters	Links	Clustering
1000	1000	499500	244 milliseconds
900	884	390286	1 second
800	771	296835	2 seconds
700	692	239086	2 seconds
600	592	174936	3 seconds
500	463	106953	4 seconds
450	424	89676	4 seconds
400	389	75466	4 seconds
350	322	51681	5 seconds
300	299	44551	5 seconds

Table 3: Experiment results from benchmarking time required to cluster.

Chapter 3

Message Passing Interface

To be able to simulate radio communication using MPI, it is necessary to expose a series of functions, such that MANET nodes will be able to communicate with each other. This will be done in three parts. The first part is a set of hardware emulating functions to enable transmission and reception of data packets between nodes. This part is introduced in Section 3.1. The second part is a centralised controller, used to control the networking between the nodes. The controller is presented in Section 3.2. The third part is the Link Model Computer (LMC), used to continuously compute the link model. The Link Model Computer (LMC) will be responsible for computing the link model from Section 2.1 and making it readily available for the centralised controller. We introduce a separate compute node, whose only task is to compute the link model, in order to offload work from the controller. The LMC node computes the link model for the network topology, and whenever the topology of the network changes, the LMC has to recompute the link model.

The architecture of the Message Passing Interface can be seen in Figure 5.

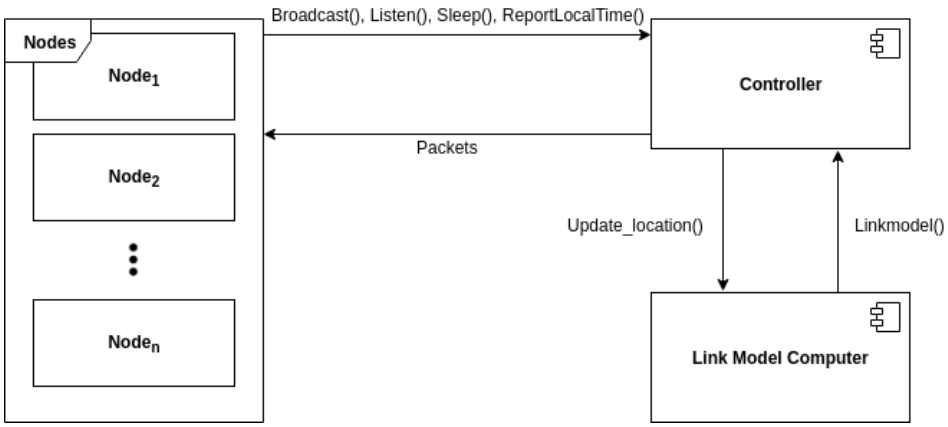


Figure 5: The architecture of the Message Passing Interface.

3.1 Hardware Emulation

In this section, we introduce the hardware functions required to emulate the radio hardware on a wireless sensor node. The three essential functions for hardware emulation is: broadcasting (Algorithm 3), listening (Algorithm 4), and sleeping (Algorithm 5). Additionally, a function for reporting the current local time (Algorithm 6) of the node to the controller is added. This function should be used whenever none of the above functions are applicable, and will allow the controller to continue processing requests from other nodes.

3.1.1 Time

To ensure that all nodes in the network are synchronised, we monitor the local time of every node. The controller described in Section 3.2 uses the local time of nodes to track when it is able to process nodes listening for transmissions. To do this, the controller requires regular time updates. These time updates are included in the MPI message sent to the controller from a node, whenever the node broadcasts, listens or sleeps. The local time of nodes are tracked in real-time, as the network protocols we simulate may rely on this.

As some protocols are designed to conserve power in a mobile device, nodes will spend a large amount of time sleeping, which can create time periods where nothing actually happens during a simulation. To counteract this, we introduce virtual time. Virtual time means that the controller will be able to skip periods of inactivity, which ensures that resources will not go unused, and we are able to speed up the time required to run the simulations.

3.1.2 Location

Recall that in Section 2.1, we use the location of the nodes to compute the link model for the wireless network topology. As we emulate the physical hardware of a mobile device, we also emulate the GPS location of the device. The current location of each node should be known by the centralised controller, and whenever the location of a node is updated, a location update should be sent to the controller.

3.1.3 Hardware Functions

The hardware functions listed in this section depend on the following local state. The local state is unique for each node.

clock ← **now**
localtime ← 0
id ← unique identifier

The local state for the hardware functions consist of three variables: clock, localtime, and id. The clock variable is used to measure the real-time spent by the node between calls to the hardware function. The localtime variable is used to track the local time of the node, as well as to enable virtual time (introduced in Section 3.1.1). Finally, the id variable is a unique identifier assigned to each node. A common element for all of the hardware functions is that we initially update the localtime variable with the elapsed time since the clock variable was reset last, before transmitting the current local time, as well as a duration, to the controller. The clock variable is reset at the end of all of the functions.

The `Broadcast` (Algorithm 3) function takes any arbitrary data packet and sends this to the controller using the MPI. The controller takes care of distributing the packet to neighbouring nodes, including computing the probability of the neighbouring node receiving the packet. The

Algorithm 3: The Broadcast Function.

```
Result: The time it took to broadcast the packet
1  Function Broadcast(packet)
2      localtime ← (now − clock) + localtime
3      duration ← transmission_time(|packet|)
4      m ← { transmit, id, localtime, duration, packet }
5      send m to controller
6      localtime ← localtime + duration
7      clock ← now
8      return duration
```

duration required for transmitting a packet is computed using the baud rate (the number of bits we can transmit per second [8]). This is sent to the controller, along with the packet, and finally we update the localtime variable with the duration, reset the clock variable and return the duration of the transmission.

Algorithm 4: The Listen Function.

```
Result: list of packets
1  Function Listen(duration)
2      localtime ← (now − clock) + localtime
3      m ← { listen, id, localtime, duration }
4      send m to controller
5      packets ← empty list
6      c ← await count from controller
7      for i ← 0 to c do
8          p ← await packet from controller
9          append p to packets
10     localtime ← localtime + duration
11     clock ← now
12     return packets
```

The `Listen` (Algorithm 4) function takes a duration and sends this, along with the updated local time, to the controller. The controller will return any packets the listening node have received within the duration.

Algorithm 5: The Sleep Function.

```
1  Function Sleep(duration)
2      localtime ← (now − clock) + localtime
3      m ← { sleep, id, localtime, duration }
4      send m to controller
5      localtime ← localtime + duration
6      clock ← now
```

The `Sleep` (Algorithm 5) function also takes an duration, and sends this, along with the updated local time to the controller.

Algorithm 6: The ReportLocaltime Function.

```
1 Function ReportLocalTime()
2     localtime ← (now − clock) + localtime
3     m ← { report-localtime, id, localtime, 0 }
4     send m to controller
5     clock ← now
```

Finally, the `ReportLocalTime` (Algorithm 6) function is included in the case where none of the above hardware functions are applicable. The function will send a message to the controller with the updated local time, and reset the `clock` variable.

3.2 Centralised Controller

To facilitate communication between nodes using the hardware interface, we introduce a centralised controller. The centralised controller works by gathering requests (broadcasting, listening, or sleeping) from all nodes and processing these. The pseudo code description of the controller can be seen in Algorithm 8.

The controller works by continuously awaiting requests from any of the nodes in the network on line 12. A call of any of the hardware functions described in Section 3.1 will result in a message arriving at the controller. Whenever a message has been received, the controller will update the local time of the sending node in the nodes variable on line 13. We keep track of the currently known local times for all nodes in order to know when we are ready to process listen requests.

If the request is a transmit request (line 14), we add the request to the transmissions list, to be processed later. Should the request be a listen request (line 17), we instead add a request to the listens list. Note that any call to the `Sleep` or `ReportLocalTime` functions are handled at line 13.

After processing the message, we compute the minimum local time of all the nodes stored in the nodes variable on line 20. The controllertime is used to check whether we are ready to process any of the listen requests stored in the listens variable. We iterate through these on line 21 and check if the end of the listen interval is greater than the minimum local time computed earlier. If this is the case, or if the listen request has already been processed, we continue iterating through the list. Otherwise, we begin iterating each transmission request stored in the transmissions list on line 25, and if the transmission interval is within the current listen interval (line 28), we check if the packet should be received by the listening node by calling the `ShouldReceive` function, and if yes, we add the packet to the list of packets to be sent to the listening node. We send the packets to the listening node on line 32.

The `ShouldReceive` function (Algorithm 7) contains a pseudo code description of how we can use the packet error probability function from Section 2.2, and defines the set of transmissions happening at the same moment ($nodes_t$).

Algorithm 7: The `ShouldReceive` Function.

Result: True if the transmission should be received.

```
1  Function ShouldReceive(l, t, transmissions, packetsize)
2      n_r ← l.id
3      n_t ← t.id
4      nodes_t ← empty Transmission list
5      foreach t′ ∈ transmissions - {t} do
6          if t.end > t′.start and t.start < t′.end then
7              append t′ to nodes_t
8      probability ← P_p(n_r, m_t, nodes_t, packetsize)
9      should-receive ← randomly choose based on probability
10     return should-receive
```

Algorithm 8: The `Controller` procedure.

```
1  Structure Message { action, source, localtime, duration, data }
2  Structure Transmission { source, start, end, data }
3  Structure Listen { source, start, end }
4  Structure Node { localtime }
5
6  procedure Controller()
7      transmissions ← empty Transmission list
8      listens ← empty Listen list
9      nodes ← map containing all nodes with id as key
10
11     repeat
12         m ← await Message from any node
13         nodes(m.source).localtime ← m.localtime + m.duration
14         if m.action = transmit then
15             t ← Transmission { m.source, m.localtime, m.localtime + m.duration, m.data }
16             append t to transmissions
17         else if m.action = listen then
18             l ← Listen { m.source, m.localtime, m.localtime + m.duration }
19             append l to listens
20         controllertime ← min_{∀n∈nodes}(n.localtime)
21         foreach l ∈ listens do
22             if l.end > controllertime then
23                 continue
24             packets ← empty Packet list
25             foreach t ∈ transmissions do
26                 if t.id = l.id then
27                     continue
28                 if t.start ≥ l.start and t.end ≤ l.end then
29                     if ShouldReceive(l, t, transmissions, |t.data|) then
30                         append t.data to packets
31             send |packets| to l.id
32             foreach p ∈ packets do
33                 send p to l.id
34         foreach l ∈ listens do
35             if controllertime > l.end then
36                 remove l from listens
37         foreach t ∈ transmissions do
38             if controllertime > t.end then
39                 remove t from transmissions
40     until protocol terminates
```

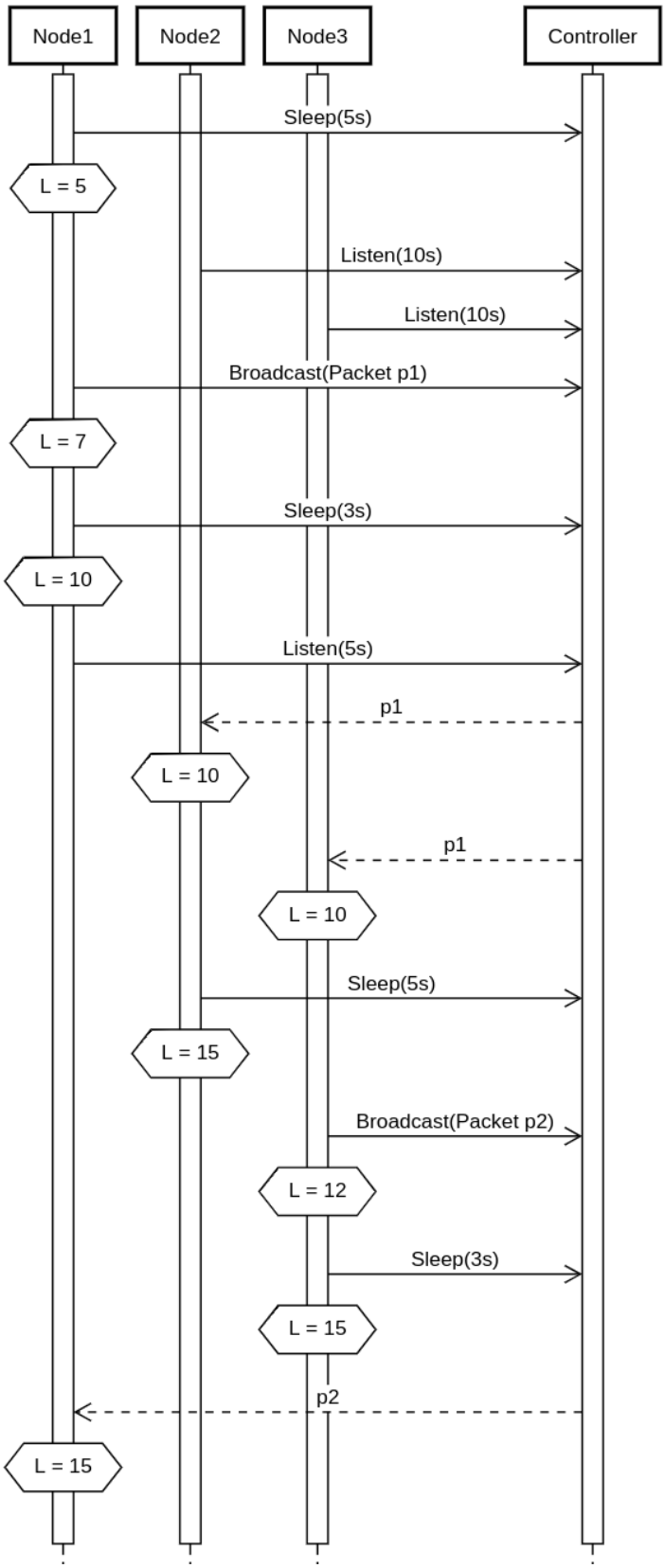


Figure 6: A sequence diagram showing an example communication flow between three nodes and the controller.

In Figure 6, we show a sample communication flow between three nodes and the centralised controller. The example is meant to show how the local time (L) of the nodes is continuously updated through the use of virtual time. For this example, we assume that a broadcast has a duration of two seconds, and the local time is initially L = 0. The figure demonstrates that whenever a node sleeps or broadcasts, the local time of the node is updated instantaneously, but when the node listens, its local time is updated only after receiving any packets transmitted within the listen interval.

3.3 Hardware Interface (hardware.h)

In this section we describe the design of the hardware (lower layer) interface, used by an upper layer protocol to simulate radio communication between nodes in a MANET. The section will serve as the documentation, as well as a programmers guide, for the `hardware.h` interface.

`void hardware::init(const mpilib::geo::Location &loc)`
Initialises the hardware functionality by initialising the MPI functionality, as well as registering the node with the MPI controller. The location is stored on the controller, and can later be updated by using the `set_location()` function. The location of a node is used to compute neighbourhood information, as well as the path loss experienced when transmitting data between nodes. This function has to be called exactly once, before calling any other hardware functions.

`void hardware::deinit()`
De-initialises the hardware functionality by un-registering the node from the MPI controller, as well as de-initialising the MPI functionality. This function has to be called exactly once, before terminating the protocol.

`std::chrono::microseconds`
`hardware::broadcast(std::vector<unsigned char> &packet)`
Transmit a vector of bytes. Algorithm 3 contains a pseudo code description of this function. Returns the duration of the transmission, in microseconds.

`std::vector<std::vector<unsigned char>>`
`hardware::listen(std::chrono::microseconds duration)`
Listen for data packets for a given duration of microseconds. Returns a vector of vectors of bytes. Algorithm 4 contains a pseudo code description of this function.

`void hardware::sleep(std::chrono::microseconds duration)`
Sleep for a given duration of microseconds. Algorithm 5 contains a pseudo code description of this function.

`void report_localtime()`
Report the current local time to the controller.

`unsigned long hardware::get_id()`
Gets the unique identifier of the node, assigned by the MPI library. This function will return 0, if the `init_hardware()` function has not yet been called.

`unsigned long hardware::get_world_size()`
Gets the total amount of nodes registered to the MPI controller. This function will return 0, if the `init_hardware()` function has not yet been called.

`bool hardware::set_location(const mpilib::geo::Location &loc)`
Updates the location registered on the MPI controller. Returns `true` if the location was successfully updated, and `false` if the location update failed on the controller, or if the `init_hardware()` function has not yet been called.

A sample C++ implementation of the Slotted ALOHA protocol using the hardware interface described above, can be found at Code 1 in Appendix A.

Chapter 4

Experiments

In this chapter we will present a pseudo code description of the Slotted ALOHA protocol [14], using the hardware functions from Section 3.1, and present an experiment showing the time required to run the simulation with a different number of time slots.

4.1 Slotted ALOHA

Slotted ALOHA [14] is a Time Division Multiple Access (TDMA) protocol, that enable communication between nodes in a wireless network, by randomly selecting a time slot to transmit, while listening in every other time slot. This method of selecting time slots for transmitting will have a high probability of collisions happening, where multiple nodes transmit at the same time, which make this protocol an ideal candidate for simulating collisions and packet loss.

Algorithm 9: The Slotted ALOHA protocol [14].

```
1  procedure SlottedALOHA(S)
2      secret ← a single node picks a secret message
3      repeat
4          selected ← randomly select a slot ∈ 1, 2, . . . , S
5          foreach current ∈ 1, 2, . . . , S do
6              if select = current then
7                  if node knows secret then
8                      duration ← Broadcast(select)
9                      Sleep(10s - duration)
10                 else
11                     Sleep(10s)
12             else
13                 if node knows secret then
14                     Sleep(10s)
15                 else
16                     secret ← Listen(10s)
17     until 1 hour
```

The Slotted ALOHA protocol shown in Algorithm 9 is a modified version of the protocol, where

the goal is to share a secret message between every node a the network. Initially, a single random node picks a secret message, and the protocol begins. The protocol is designed to conserve energy by sleeping as much as possible. If the node knows the secret message, it will broadcast the message when its selected time slot comes up, and sleep in any other time slot. If the node does not know the secret message, it will sleep in its selected time slot, and listen for the message in any other time slot.

4.1.1 Experiment

We ran an experiment with the Slotted ALOHA protocol from Algorithm 9. The experiment was conducted on a network topology of 33 nodes generated from GPS location data, from a field test conducted in the Philippines. The topology can be found in Appendix B. The goal of the experiment is to show the difference between virtual and real time, when running the protocol for an hour with a different number of time slots S.

Slots (S) / Time	5	6	7	8	9	10	11	12
Real	45 s	5 s	90 s	60 s	50 s	105 s	60 s	88 s

Table 4: Real time results from running the ALOHA protocol with different total time slots.

Table 4 show the real time duration the simulation took to run the Slotted ALOHA with different parameters for the time slots S. We see that the shortest simulation took 5 seconds with 6 time slots, and the longest took 105 seconds with 10 time slots. This demonstrates the non-deterministic nature of the Slotted ALOHA protocol, where the time required to complete the simulation heavily depends on the number of colliding transmissions and lost packets. The main takeaway from this experiment is that we are able to run the hour long Slotted ALOHA protocol in less than two minutes, with 33 nodes.

Chapter 5

Conclusion

In this report, we lay the groundwork for a C++ library for writing, and running, simulations of the network protocols behind the mesh communication in a MANET. We simulate radio communication between emulated devices, where we simulate packet errors and collisions, by using a state-of-the-art modelling of link path loss [3], accounting for both link distance and correlation between links.

Our library consist of three parts: A hardware interface header file, used for writing wireless network protocols, emulating the physical part of the devices in a MANET, a centralised controller facilitating communication between the emulated devices, as well as speeding up the time required to run the simulations by introducing virtual time, and finally, the link model, enabling us to simulate wireless radio communication, packet errors and collisions caused by interfering transmissions. Computing the link model is expensive, and because of this, we have implemented two methods to decrease the computational time required.

The first method is to introduce a distance threshold, limiting the maximum length allowed to form a link between two nodes, thus reducing the size of the correlation matrix in the link model. In Section 2.3.1, we found that a distance threshold of 1000 meters significantly reduced the amount of links we had to consider for our correlation matrix, while still showing a very high probability for packet loss (on links with a distance greater than 1000 meters).

In the second method, we present an algorithm for greedily approximating a solution to the clustering problem, defined in Section 2.3.2. We show the computational time required to form a number of clusters, while reducing the number of links we have to consider for our correlation matrix, but we have yet to quantify how well our clustering approximation performs.

We show in Chapter 4 how the library can be used to simulate the Slotted ALOHA protocol, running an hour worth of simulations in just a couple of minutes, real-time, with the ability of repeatedly running the simulation with different parameters.

5.1 Future Work

Computing the link model still poses a significant scalability challenge, and in general, scaling the library to reach our goal of running repeatable simulations with up to 1000 devices, will prove to be an interesting task in the future.

We want to implement, and experiment with, more advanced networking protocols such as the LMAC protocol [17] and the NeoCortec NeoMesh [11] protocol. The Slotted ALOHA protocol is incredibly simple, and the random selection of broadcasting time slot is a good way for us

to test the link model, as the protocol has a high probability of multiple nodes broadcasting at the same time, producing collisions.

We need to prove the correctness of the centralised controller. We introduce virtual time but to ensure that the controller works correctly, such that we do not miss any transmissions or receive transmissions that happened in the past, we need to formally prove this.

Additionally, we also need to quantify how well the clustering approximation from Section 2.3.2 works. We claim that we should be able to minimise the loss of precision when minimising the difference in correlation between links before and after clustering, but we need to quantify exactly how much precision we lose, when clustering the topology before computing the link model.

Bibliography

[1] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 49–60, New York, NY, USA, 1999. ACM.

[2] Rupanka Bhuyan and Samarjeet Borah. A survey of some density based clustering techniques. 03 2013.

[3] Rasmus Liborius Bruun. *Mobile ad-hoc network performance assessment based on simulation with scenario specific propagation and multi-link modelling.* Aalborg University, 2018.

[4] Cran. Nearest pd. https://github.com/cran/lmf/blob/master/R/nearPD.R. Accessed: 3. January 2019.

[5] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.).* Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[6] P. Massoud Salehi and J. Proakis. *Digital Communications.* McGraw-Hill Education, 2007.

[7] Wolfram MathWold. Positive definite matrix. http://mathworld.wolfram.com/PositiveDefiniteMatrix.html. Accessed: 10. Januar 2019.

[8] MathWorks. Baudrate. https://mathworks.com/help/matlab/matlab_external/baudrate.html. Accessed: 8. January 2019.

[9] A. Merts and A. Barnard. Simulating manets: A study using satellites with aodv and anthocnet. In *2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*, pages 1–5, Nov 2016.

[10] Morteza Mohammadi Zanjireh and Hadi Larijani. A survey on centralised and distributed clustering routing algorithms for wsns. volume 2015, 05 2015.

[11] NeoCortec. Noemesh. https://neocortec.com/, 2018. Accessed: 10. January 2019.

[12] The International Society of Automation (ISA). db vs. dbm. https://www.isa.org/standards-publications/isa-publications/intech-magazine/2002/november/db-vs-dbm/, 2001. Accessed: 28. September 2018.

[13] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[14] Lawrence G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5(2):28–42, April 1975.

[15] Marjan Sirjani, Ali Movaghar-Rahimabadi, Amin Shali, and Frank S. de Boer. Modeling and verification of reactive systems using rebeca. *Fundam. Inform.*, 63:385–410, 2004.

[16] Aalborg University. mccaau. https://sites.google.com/site/mccaau/home, 2018. Accessed: 10. January 2019.

[17] L.F.W. van Hoesel and Paul J.M. Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *1st International Workshop on Networked Sensing Systems (INSS)*, pages 205–208. Society of Instrument and Control Engineers (SICE), 2004.

[18] Eric W Weisstein. Frobenius norm. http://mathworld.wolfram.com/FrobeniusNorm.html, 2018. Accessed: 1. January 2019.

[19] Behnaz Yousefi, Fatemeh Ghassemi, and Ramtin Khosravi. Modeling and efficient verification of broadcasting actors. In *FSEN*, volume 9392 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2015.

[20] Behnaz Yousefi, Fatemeh Ghassemi, and Ramtin Khosravi. Modeling and efficient verification of wireless ad hoc networks. *Formal Asp. Comput.*, 29(6):1051–1086, 2017.

[21] Fengling Zhang, Lei Bu, Linzhang Wang, Jianhua Zhao, Xin Chen, Tian ZHANG, and Xuandong Li. Modeling and evaluation of wireless sensor network protocols by stochastic timed automata. *Electronic Notes in Theoretical Computer Science*, 296:261–277, 08 2013.

Glossary

baud rate

The baud rate is the rate at which information can be transferred as a wireless signal [8], and is equivalent to bits per second.. 20

conjugate transpose

The element in the i-th row and j-th column is equal to the complex conjugate of the element in the j-th row and i-th column for all indices i and j [13]. 13

lower layer

The software simulating radio hardware. 26

lower triangular matrix

A square matrix where all entries above the diagonal are zero [13]. 13

path loss

The signal loss inflicted by the propagation of a electromagnetic wave from transmitter to receiver [3, p. 10]. 3, 4, 5, 6, 7, 9, 14, 15, 26, 31

positive-definite matrix

A matrix is positive-definite if $\forall n \in M$, such that $eigenvalue(n) > 0$ [7]. 13, 14

symmetric

$a_{i,j} = a_{j,i}$ for $i, j = 0, \ldots, N-1$ [13]. 13

upper layer

The software implementation of a MANET protocol, e.g., an implementation of the LMAC protocol. 26

Acronyms

dB
decibel. 6, 7, 8, 10, 11, 37

dBm
dB relative to a milliwatt. 6, 9, 11, 14

GPS
Global Positioning System. 20, 30

IID
Independent and Identically Distributed. 8

LMAC
Lightweight Medium Access Protocol. 3, 31, 35

LMC
Link Model Computer. 19

MANET
Mobile Ad-hoc Network. 3, 4, 5, 6, 19, 26, 31, 35, 41

MPI
Message Passing Interface. 3, 19, 20, 26

OPTICS
Ordering Points To Identify the Clustering Structure. 16, 17

RSSI
Received Signal Strength Indication. 5, 6, 9, 10, 11, 14

TDMA
Time Division Multiple Access. 29

TLS
Total Least Squares. 7

WSN
Wireless Sensor Network. 4

Appendix A

Slotted ALOHA C++

```cpp
#include <random>
#include <ostream>

#include <mpilib/hardware.h>

int main(int argc, char *argv[]) {
    mpilib::geo::Location l{57.012668, 10.994625};
    hardware::init(l);

    auto id = hardware::get_id();
    auto slots = hardware::get_world_size();
    std::random_device rd{};
    std::mt19937 eng{rd()};
    std::uniform_int_distribution<unsigned long> dist{0ul, slots};

    for (auto i = 0; i < 4; ++i) {
        auto selected = dist(eng);
        for (auto current = 0; current < slots; ++current) {
            if (selected == current) {
                std::vector<unsigned char> data{0xBE, 0xEF};
                auto duration = hardware::broadcast(data);
                hardware::sleep(200ms - duration);
            } else {
                auto packets = hardware::listen(200ms);

                for (const auto &item : packets) {
                    std::cout << item << std::endl;
                }
            }
        }
    }

    hardware::deinit();
}
```

Code 1: Modern C++ implementation of the Slotted ALOHA protocol.

Appendix B

Slotted ALOHA Test Topology

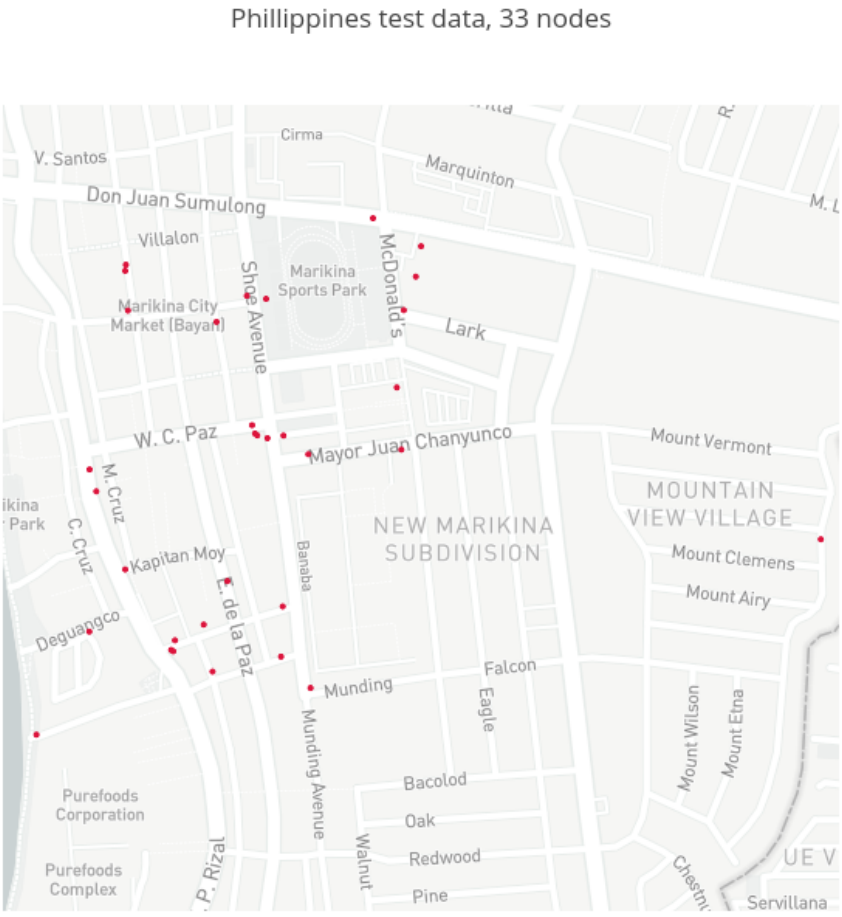


Figure 7: MANET topology from a field test with 33 nodes.