Fortentwicklung drahtloser Ad-Hoc Netzwerke durch Bluetooth-Kommunikation

Bachelorarbeit

von

Maxim Specianov

aus

Kemerowo

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

Oktober 2017

Betreuer:
Raphael Bialon, M. Sc.

Zusammenfassung

Mit dieser Arbeit beschäftige ich mich mit der Kurzstrecken-Kommunikationstechnologie Bluetooth und versuche eine Simulation für diese im PeerfactSim.KOM Simulator so realitätsgetreu wie möglich zu gestalten und implementieren. Da heutzutage in jedem Smartphone und in vielen anderen mobilen Geräten Bluetooth eingebaut ist und sogar viele Sensoren mit Bluetooth-Technologie ausgestattet sind, ist es meiner Meinung nach wichtig, Verbindungen mithilfe eines Simulators ausprobieren zu können und diese Simulationen zu analysieren im Hinblick auf die Verlustrate oder Anzahl der Hops, also Zwischenschritte, bis eine Nachricht an den Empfänger weitergeleitet wurde.

Danksagung

An dieser Stelle möchte ich mich herzlich bei meiner Familie bedanken, die mich stets unterstützt hat und für mich da war. Außerdem geht mein Dank auch an meine Freunde, die mir die Motivation gaben und jederzeit zur Verfügung standen.

Auch vielen Dank an meinen Betreuer Raphael Bialon, der mir viele hilfreiche Tipps geben konnte um mir das Arbeiten an der Bachelorarbeit zu erleichtern.

Zu guter Letzt möchte ich mich auch bei Jun.-Prof. Dr. Kalman Graffi für die Gelegenheit bedanken diese Arbeit schreiben zu dürfen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Heutzutage gibt es sehr viele Geräte die mit Bluetooth-Technologie ausgestattet sind. Allein die Tatsache, dass jedes aktuelle Smartphone in der Lage ist sich über Bluetooth mit anderen Geräten zu verbinden und Daten auszutauschen, zeigt, wie wichtig es ist diese Technologie weiter zu entwickeln und so gut wie es geht auszunutzen. Außerdem ist es immer gut Alternativen für Übertragungstechnologien zu haben die bei einem Ausfall von kabelgebundenen Netzwerken oder WLAN zur Verfügung stehen. Und überall, wo etwas getestet oder ausprobiert wird um zu sehen wie es sich verhält, bietet es sich an ein Programm zu haben, mit dem man diese Szenarien simulieren kann. Dies ist einerseits notwendig um die mit echten Tests verbundenen Kosten zu sparen. Andererseits braucht man für Test in größeren Dimensionen keine Sorgen zu haben, dass das vorhandene Test-Gelände zu klein wird und man nichts passendes in der Nähe findet was groß genug ist.

Aus diesen Gründen erscheint es für mich als sinnvoll den PeerfactSim.KOM Simulator [PS] mit einer Implementation zu erweitern, die das Simulieren von Bluetooth ermöglicht. Mit dieser werde ich auf der Netzwerkschicht die Struktur von Bluetooth nachbilden und anschließend mittels einiger Szenarien verifizieren. Doch vorher verschaffe

ich einen Überblick über die Funktionsweise von Bluetooth und die Motivation von Ad Hoc Netzwerken.

1.2 Überblick

Zu Beginn dieser Arbeit gehe ich auf einige aktuelle und verwandte Arbeiten ein, wobei ich auch grob auf PeerfactSim.KOM [PS] eingehe. Außerdem beinhaltet das Kapitel 2 auch ein paar Probleme und Hürden mit denen man zu kämpfen hat bei der Realisierung von Ad-Hoc Netzwerken mittels Bluetooth. Im Kapitel 3 schildere ich die Eigenschaften und Spezifikationen von Bluetooth. Außerdem beschreibe ich die Funktionsweise der Bluetooth Technologie und nenne einige bekannte und auch weniger bekannte Bereiche, in denen diese zum Einsatz kommt. Zu guter Letzt schließe ich das Kapitel mit einem Überblick über die Ad Hoc Netzwerke und Opportunistische Netzwerke ab. Als nächstes folgt im Kapitel 4 meine Vorgehensweise bei der Implementation, wie ich mich mit der Simulation von Bluetooth im PeerfactSim.KOM Simulator auseinander gesetzt habe. Weiterhin folgt im Kapitel 5 die Auswertung einiger Simulationen die der Verifizierung der Implementation dienen. Zuletzt fasse ich in einem Fazit im Kapitel 6 zusammen, inwiefern ich bei dem Thema Erfolg hatte und welche Herausforderungen oder Probleme der Bearbeitung im Weg standen.

Kapitel 2

Related Work

Der Inhalt dieses Kapitels bezieht sich auf mehrere themenverwandte Arbeiten. Diese haben sich ebenfalls mit Bluetooth und Ad-Hoc Netzwerken beschäftigt und dienen dazu einen Überblick zu schaffen, welche Arbeiten beziehungsweise Projekte schon gemacht wurden und welche Resultate bei diesen erzielt wurden.

Eine ziemlich aktuelle wurde 2017 im *International Journal of Advanced Computer Science and Applications* (IJACSA) mit dem Titel "A Review of Bluetooth based Scatternet for Mobile Ad hoc Networks" veröffentlicht. Verfasst wurde diese von Khizra Asaf und weiteren Mitgliedern der *Department of Science* an der *Gouvernment College University* in Faisalabad (Pakistan) [Kha17]. Sie handelt, wie der Titel schon sagt über einen Rückblick auf die Forschung im Hinblick auf Bluetooth Scatternetze. Als ein Scatternetz wird ein Netzwerk von mehreren Pikonetzen bezeichnet, wobei manche Geräte in mehr als einem Pikonetz Mitglied sind. Ein Pikonetz ist dabei ein kleines Netzwerk aus Bluetooth Geräten, wobei die genaue Erklärung hierfür im Kapitel 3.1.1 stattfindet.

Zum einen werden verschiedene Algorithmen beschrieben, die den Aufbau von Scatternetzen gewährleisten sollen. Zur Verwaltung des Netzwerkes in einer Baum Topologie werden oft Algorithmen wie TSF, SHAPER und SFX verwendet [Kha17, S.403]. "*Tree Scatternet formation (TSF)* kann verwendet werden, wenn alle Knoten in Kommunikationsreichweite sind oder in dynamischen Situationen, in denen TSF mithilfe von

Konnektivität, Effizienz und der Möglichkeit sich selbst zu reparieren, punkten kann." [Olu15] "Der SHAPER Algorithmus ist der erste flexible Algorithmus unter den Scatternet Aufbau-Algorithmen, der sich justieren kann und die Topologie in einem Multi-Hop Zustand nach Mobilität oder Katastrophen anpassen kann." [Aky04] SFX ist eine Weiterentwicklung des SHAPER Algorithmus, wobei manche von SFX unterstützten Prozesse ausgelassen wurden um die Komplexität zu verringern [Kha17]. "*Dieser* baut eine parallel Struktur auf und erlaubt einer beliebigen Anzahl an Bäumen gleichzeitig mit einem anderen Baum zu verschmelzen." [Kha17, S.404]

Neben den Baum Topologien gibt es auch Mesh Topologien, die zwar auf Grundlage der Baum-Topologien entstanden sind aber das Ziel haben die Länge der Routen in Standard Bäumen zu verkürzen. [Che13] Hierfür werden unter anderem die Algorithmen MTSF oder BlueMesh verwendet [Kha17].

"MTSF entfernt die Begrenzung des Netzes auf die Reichweite der Bluetooth Technologie und ermöglicht somit das erstellen von größeren Scatternetzen." [Kha17, S.404] Dieser Algorithmus besteht wie folgt aus drei Phasen. Die *Erkundungs-Phase*, in der die Knoten ihre Nachbarn kennenlernen, was ein zu erwartender Bestandteil in fast allen ähnlichen Algorithmen ist. Als nächstes folgt die Bildung von Pikonetz-Clustern. Als dritter Schritt werden die einzelnen Pikonetze untereinander verlinkt [Kha17].

Bei BlueMesh handelt es sich um ein Protokoll für die Scatternetz Einrichtung. Es unterstützt die Knoten dabei innerhalb des Mesh-Netzwerkes den kürzesten Weg unter vielen möglichen Pfaden zu finden [Chl04].

Es wird auch die hauptsächliche Hürde für Scatternetze klar, nämlich die Zusammenschaltung der einzelnen Pikonetze. An dieses Problem kann mit den vorher benannten Ansätzen heran gegangen werden. Im Unterschied zu der Arbeit von Khizra Asaf [Kha17], wird meine Arbeit keine feste Topologie beinhalten, da sich diese eher damit beschäftigt, dass Nachrichten in stark dynamischen Netzwerken weitergeleitet werden über Bluetooth. Denn erst bei Netzwerken, in denen sich die Geräte in Gruppen oder Clustern bewegen und meist nur die selben Knoten im Umkreis haben, eignet sich meiner Meinung nach eine feste Topologie.

Ein weiterer Artikel aus dem *Journal of Global Research in Computer Science* mit dem Titel “A Bluetooth mobile Ad Hoc Network Communication Topologies” handelt ebenfalls von einem Überblick über die Bluetooth Technologie als Verbindung für ein Netzwerk [Dr.14]. In diesem wird deutlich, dass die Knoten, die als Brücke zwischen zwei Pikonetzen fungieren eine Schwachstelle sind, die eine zuverlässige Arbeitsweise von Scatternetzen verhindern und Scatternetze aufgrund dieser Schwachstelle nicht als Ersatz für WLANs geeignet sind [Dr.14, S.14]. Der Unterschied zwischen dieser Arbeit und meiner ist, dass meine Implementation für einzelne, sich dauernd bewegende, Knoten gedacht ist und somit keine festen Gruppen oder Cluster,

In dieser Arbeit implementiere ich die bisher nicht vorhandene Bluetooth Kommunikation in dem PeerfactSim.KOM Simulator [PS]. Mithilfe dieser wird man dann in der Lage sein können verschiedene Szenarien zu simulieren und auszuwerten. Dazu gehören kleine Szenarien oder auch riesen große, durch die man sich einen Aufbau des Szenarios mit echten Geräten, was mit vielen Störfaktoren verbunden sein kann, sparen kann.

2.1 PeerfactSim.KOM

PeerfactSim.KOM [PS] ist ein Programm welches an der Technischen Universität Darmstadt [Dar] entwickelt wurde und nun auch an der Heinrich-Heine-Universität Düsseldorf [Dü] weiterentwickelt wird. Dieses ist dazu gedacht Peer-to-Peer Verbindungen zu simulieren und diese damit näher zu untersuchen.

Die Abbildung 2.1 zeigt die einzelnen Schichten, die im PeerfactSim.KOM [PS] berücksichtigt werden und welche Möglichkeiten es gibt Ergebnisse der Simulationen auszugeben. Wie man erkennen kann, konzentriert man sich hier dabei auf die Netzwerkschicht und die darüber liegenden Schichten und abstrahiert somit die Sicherungsschicht und die Bitübertragungsschicht da diese das Programm zu komplex gestalten würden.

Mithilfe der implementierten `Simulation Engine` besteht die Möglichkeit Log-Dateien, die mit Gnuplot [Gnu] in anschaubare Diagramme geplottet werden oder aber

auch Visualisierungen von bestimmten Szenarien zu erstellen. Außerdem ist es wichtig zu erwähnen, dass die `Simulation Engine` Event-basiert funktioniert, das heißt sobald ein neues Ereignis im simulierten Szenario ausgeführt werden soll, wird dieses in einen `Event Scheduler` an richtiger Stelle angeordnet, sortiert nach der Simulationszeit, und dann, sobald es dran ist, ausgeführt [Pus11, S.5-6]. Dies hat den Vorteil, dass die Simulation Engine nur von Event zu Event springt und nicht die Zeit dazwischen mit *warten* verbringen muss [Pus11, S.5]. Zur Veranschaulichung siehe Abbildung 2.2, in dieser erkennt man drei Simulationszeiten t1 – t3, wobei t1 die aktuelle Simulationszeit ist an der ein Event gerade ausgeführt wurde und t2 die Zeit ist, an dem das als nächstes auszuführende Event ist. Nach dem Ausführen des ersten Events wird die Simulationszeit auf t2 erhöht und die `Simulation Engine` behandelt das zweite Event.

Konfiguriert wird die Simulation Engine mithilfe von XML-Dateien, diese enthalten Variablen wie Seed, finishTime und Maße für die Welt. Außerdem lassen sich dort die einzelnen Klassen eintragen für die Layers, die im Szenario verwendet werden sollen. Auch die Bewegungs- und Platzierungsmodelle lassen sich hier anpassen [Pus11, S.79-88].

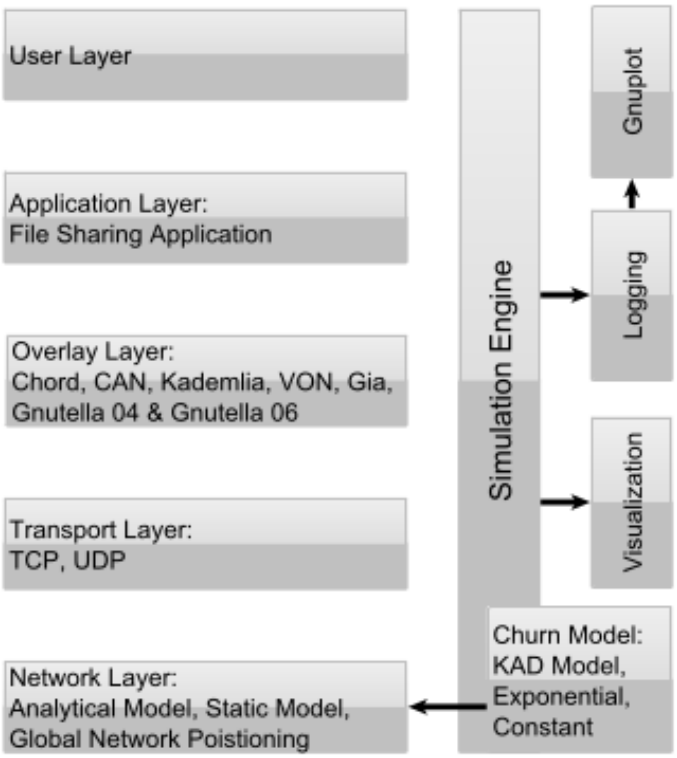

Abbildung 2.1: Layers des PeerfactSim.KOM [Pus11, S.3]

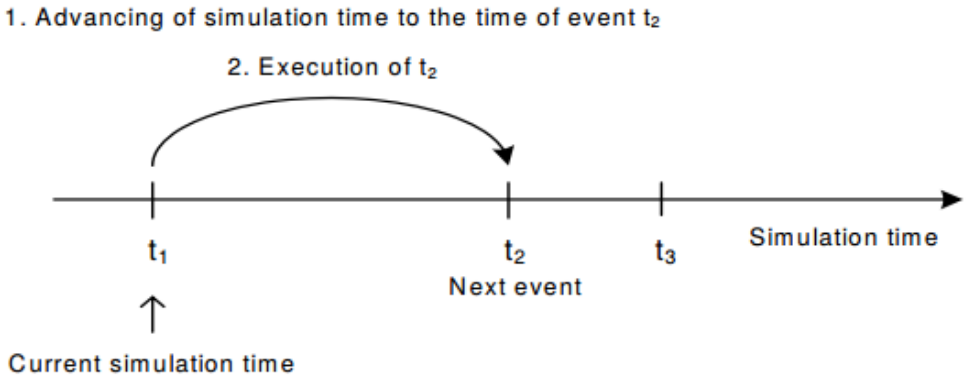

Abbildung 2.2: Fortschritt der Simulationszeit in Event-basierten Simulationen [Pus11, S.6]

Kapitel 3

Grundlagen

3.1 Bluetooth

3.1.1 Geschichte und Einsatzbereiche

Bluetooth ist eine drahtlose Verbindungstechnologie, die 1996 durch die Zusammenarbeit von Intel, Ericsson und Nokia geplant und ins Leben gerufen wurde [BS]. Benannt wurde die Technologie nach dem Dänischen König Harald “Bluetooth” Gormsson, der im zehnten Jahrhundert Dänemark und Norwegen vereint hatte. „Sein toter Zahn, welcher eine dunkel blaue/graue Farbe hatte, verpasste ihm den Spitznamen Bluetooth.“ [BS]

Die Technologie Bluetooth wurde einst entwickelt, um eine weitere Alternative zu kabelgebundenen Verbindungen, als zum Beispiel Infrarot zu schaffen und somit die Interaktion von mehreren Geräten untereinander zu vereinfachen. Wobei bei Bluetooth kein direkter Sichtkontakt vorhanden sein muss, da es sich hierbei um eine Funktechnologie handelt. Ein Netzwerk von mehreren Geräten, die mittels Bluetooth verbunden sind, wird als ein “Piconet” bezeichnet, was für ein sehr kleines Netzwerk steht. Nichtsdestotrotz ist es durchaus möglich auch ein großes Netzwerk mit Hilfe von Bluetooth zu erstellen

und so auch Ad-Hoc Netzwerke entstehen zu lassen, durch die Daten von einem Ende zum anderen gelangen können.

Heutzutage wird Bluetooth größtenteils für die Übertragung von Musik von Smartphones an Musikanlagen, Autoradios oder kabellose Kopfhörer genutzt. Außerdem bietet sich die Nutzung von Bluetooth bei Freisprechanlagen an. Außerdem gibt es zahlreiche Sensoren, die Bluetooth verwenden um ihre Daten zu verbreiten. Dazu gehören zum Beispiel Sensoren, welche die Temperatur messen oder mithilfe von Infrarot als Bewegungsmelder fungieren [HC].

3.1.2 Funktionsweise/Verbindungsaufbau

Bluetooth verwendet das 2.4 GHz ISM (Industrial Scientific Medical) Band, wobei in den meisten Ländern der Welt die Frequenzen 2400-2483.5 MHz lizenzfrei zur Verfügung gestellt werden [Blu99, S.18-19].

Bei jeder Bluetooth Verbindung zwischen zwei Geräten gibt es eine Zuweisung von Rollen, diese ist nämlich entweder Master oder Slave. Um eine stabile Verbindung zu ermöglichen, benutzt Bluetooth die *Gaussian Frequency Shift Keying* wobei Sender und Empfänger in regelmäßigen Abständen Frequenzsprünge machen. Wobei ein Zeitschlitz die Dauer von 625 µs hat. “Normalerweise deckt ein Paket nur einen Slot ab, doch es ist auch möglich, das es bis zu Fünf Slots abdeckt.” [Blu99, S.41] Damit in beide Richtungen gesendet werden kann, wird *Time-Division-Duplex* (*TDD*) verwendet. Eine Verbindung von mehreren Geräten mittels Bluetooth wird als Piconet bezeichnet. Dieses besteht aus einem Master Knoten, der gleichzeitig bis zu sieben aktive Slaves verwalten kann. Sobald diese Anzahl überschritten wird ist es möglich Slaves, die nicht mehr notwendig verbunden sein müssen, zu parken und später bei Bedarf wieder mit diesen in Kontakt zu treten und als aktiv einzustufen. Dabei bleiben diese Knoten weiterhin synchronisiert mit dem Master-Knoten. [Blu99, S.41-42] Ein Master Knoten kann auch gleichzeitig Slave eines anderen Pikonetzes sein. Sich überlappende Pikonetze können auch störungsfrei weiterhin kommunizieren, da jedes Pikonetz seine eigene zufällige Frequenzsprungfolge hat, die der Master vorgibt [Blu99, S.42]. Um Bit-Fehler bei der Übertragung vorzubeu-

h

Klasse	Max. Leistung	Reichweite im Freien
Klasse 1	100 mW	ca. 100 m
Klasse 2	2.5 mW	ca. 50 m
Klasse 3	1 mW	ca. 10 m

Tabelle 3.1: Abhängigkeit der Reichweite von Leistung [Com]

gen, werden die Header beispielsweise mittels *1/3-Vorwärtsfehlerkorrektur* versendet, wobei jedes Bit des Headers drei mal wiederholt wird und der Empfänger sich dann für den Bit-Wert entscheidet, der dabei zwei oder drei mal empfangen wird [Blu99, S.67].

Bei Bluetooth gibt es unterschiedliche Klassen, wobei die Einteilung nach Leistung geschieht und in verschiedenen Reichweiten der Empfänger resultiert, diese kann man der Tabelle 3.1 entnehmen. Wie man sieht beträgt die maximale Leistung dabei bis zu 100 mW und die maximale Reichweite im Freien ohne Hindernisse ca. 100 m [Com].

Inquiry Prozedur

Die *Inquiry Prozedur* ist der Ablauf des Verbindungsaufbaus zwischen zwei Bluetooth Geräten. Möchte ein Gerät sich mit anderen verbinden, schickt es eine Inquiry Nachricht. Alle Geräte, die diese Nachricht erhalten, antworten dann mit Ihren Daten und Adressen, dem sogenannten Inquiry Response, die wiederum von dem ersten Gerät verwendet werden können für den Verbindungsaufbau [Blu99, S.112].

Obwohl man denken könnte, dass es vielleicht Schwierigkeiten geben könnte wegen der Begrenzung von sieben aktiven Slaves pro Master, ist dies nicht so tragisch. Es macht nichts aus, dass man nur maximal sieben Slaves haben kann, denn jedes Limit höher als fünf erfüllt den Zweck. Ausgehend vom schlimmsten Fall bei sechs Slaves, die geradeso in Reichweite sind, bilden diese ein Hexagon. Bei einem weiteren Knoten in Reichweite gäbe es somit mindestens ein Slave, welches den neuen Knoten auch als Nachbarn hat und diesen in sein Pikonetz aufnehmen kann [Chl01].

3.2 Ad Hoc Netzwerke

Die Idee mobile Knoten miteinander kommunizieren zu lassen ohne eine feste Infrastruktur kam bereits in den 1970er Jahren zum Vorschein nachdem die Paket vermittelnde Mobilfunk Technologie eingeführt wurde [Kri05, S.2]. Der Begriff Ad Hoc kommt aus dem Lateinischen und bedeutet "für diesen Augenblick gemacht". Dies unterstreicht die spontanen Verbindungen der mobilen Knoten und ihre kurzfristige Dauer.

Als ein Ad Hoc Netzwerk werden Netzwerke bezeichnet, in denen sich die Geräte eigenständig mit anderen Geräten in ihrer Nähe verbinden und sich so erweitern. Sollen innerhalb dieses Netzwerkes Nachrichten versendet werden, so werden diese durch Weiterleitung und Zwischenstopps bei anderen Geräten im Netzwerk bis zum Zielknoten weitergereicht. Da dies in den Endgeräten selbst geschieht, ist der Energieverbrauch der am Ad Hoc Netzwerk beteiligten Knoten natürlich höher als bei Geräten, die nur ein Endknoten in einer festen Struktur sind.

Routing in Ad Hoc Netzwerken

Das Routing in Ad Hoc Netzwerken kann auf verschiedene Weisen bewältigt werden. Die wohl einfachste und bekannteste Möglichkeit ist das *Fluten*. Hierbei speichert man keinerlei Routing-Tabellen und sendet die Nachricht einfach an alle Knoten in Reichweite. Jeder Knoten, der die Nachricht bekommt, broadcastet diese weiter, vorausgesetzt er hat die selbe Nachricht nicht schon einmal erhalten. Dadurch wird gewährleistet, dass die Nachrichten keine Kreise im Netzwerk ziehen und irgendwann terminieren. Dieser Algorithmus zahlt sich aus, wenn das Netzwerk ständig im Wandel ist und es keinen Sinn ergibt Informationen über das Netzwerk zu speichern. Man muss lediglich wissen, welche Nachrichten man zuletzt schon einmal bekommen hat. Um ein “überfluten” der Netzwerkes zu verhindern, kann man diesen Algorithmus auch leicht abändern und eine Wahrscheinlichkeit hinzufügen, mit der ein Knoten die Nachricht weiterleitet an andere [NYCYSJP99].

Im Gegensatz dazu gibt es auch Distanzvektoralgorithmen. Diese sind dynamisch und besitzen Routing Tabellen, die den Weg zu ihren Nachbarn beinhalten. Diese Tabellen werden allen Nachbarn mitgeteilt, somit lernen Sie neue Pfade zu anderen Knoten und können diese bei sich speichern. Doch auch diese Art an Algorithmen hat ihre Probleme. Denn bei ungünstigen Situationen kann es hier zu Unendlichkeitsschleifen kommen [Tan03].

3.2.1 Opportunistische Netzwerke

Opportunistische Netzwerke (Opp Nets) sind wie der Name schon sagt, Netzwerke, die jede Opportunity, also Gelegenheit nutzen, um eine Nachricht näher an den Empfänger zu bekommen. Bei Opportunistischen Netzwerken legt man keinen Wert darauf, dass eine feste Route für eine Nachricht im Netzwerk existiert, es ist also möglich, dass Sender und Empfänger niemals Teil des gleichen Netzwerkes zur selben Zeit werden [Con06, S.1].

In Opportunistischen Netzwerken werden die Knoten also auch als „Träger“ von Nachrichten verwendet, wobei diese dann bis zur nächsten Gelegenheit zum Senden die Nachricht mit sich tragen und an einen weiteren Knoten verschicken, der hoffentlich irgendwann näher zum Empfänger kommt [Con06, S.2]. Dadurch entstehen Verzögerungen beim Übertragen und somit kann man Opp Nets, nur verwenden, wenn eine Anwendung mit Verzögerungen zurecht kommt.

Kapitel 4

Implementation in Peerfact

4.1 Vorgehensweise

Dieses Kapitel handelt von der Implementation der Bluetooth Kommunikation im PeerfactSim.KOM [PS] Simulator. Hier schildere ich meine Vorgehensweise und nenne Punkte, die ich dabei beachten musste.

Für die Implementation der Bluetooth-Kommunikation im PeerfactSim.KOM Simulator konnte ich Teile der bereits vorhandenen Klassen im `Wireless Package` verwenden und diese an die Bluetooth Spezifikationen anpassen. Zuerst habe ich versucht eine einfache Bluetooth Verbindung zu simulieren, die aus einem Master und einem Slave besteht. Dazu war es notwendig eine `Piconet` Klasse zu erstellen, in der klar definiert ist, welcher `DefaultMobilityHost` der Master ist und welcher der Slave ist. Doch hier hatte ich zu weit ins Detail gedacht, denn eine extra Klasse für Pikonetze ist nicht dringend, da es ausreicht, wenn nur der Knoten selbst seine Nachbarn und Verbindungen kennt. Denn die Bluetooth Spezifikation erlaubt es einem Knoten nur in einem Pikonetz Master zu sein. Somit werden die Slaves eines Knoten später schlicht in einer *LinkedList* verwaltet.

Also habe ich mich dazu entschieden, den Verbindungsaufbau von zwei Knoten zu ab-

strahieren und nur als eine Zeitverzögerung in `BluetoothNetLayer` einzubringen. Dazu verwende ich eine Dauer von 10 ms, da ich annehme, dass die Knoten sich stets in einem Zustand befinden, in dem sie aktiv nach neuen Geräten suchen und auch nach Inquiry Nachrichten von anderen Knoten lauschen. Die Synchronisierung der Frequenzen der Slaves mit dem Master wird hierbei auch vorausgesetzt und nicht weiter betrachtet, da diese in tieferen Schichten der Bluetooth Kommunikation stattfinden und der PeerfactSim.KOM Simulator die Kommunikation nur makroskopisch betrachtet.

Für die Klasse `BluetoothNetLayer` konnte ich die bereits von Tobias Korfmacher implementierte Klasse `MobilityNetLayer` als Vorlage benutzen, die ich nun entsprechend anpassen musste, damit es der Funktionsweise von Bluetooth entspricht. Zuerst die Änderung der Reichweite, welche ich zu pauschal für alle Geräte festgesetzt habe auf 100 m, da dies zu einem späterem Zeitpunkt detaillierter aufgedröselt werden kann um verschiedene Klassen und Versionen von Geräten zu realisieren. Es wird also zurzeit angenommen, dass jedes Gerät der ersten Klasse entspricht, siehe Tabelle 3.1.

Aber bevor man die `BluetoothNetLayer` überhaupt verwenden kann, müssen die Klassen `Host` und `DefaultHost` einen neuen Parameter *hasBluetooth* bekommen. Dieser ist ein *Boolean*, da es sich für die Unterscheidung am besten eignet. Dies ist notwendig um mit den Konfigurationsdateien später den Simulator so initialisieren zu können, dass die Kommunikation über Bluetooth nachgeahmt werden kann. Um Bluetooth Kommunikation zu simulieren, muss die Variable *hasBluetooth* auf *true* gesetzt werden, da andernfalls die `MobilityNetLayer` auf der Netzwerkschicht verwendet wird statt der `BluetoothNetLayer`. Denn auch die Klasse `MobilityFactory` wurde von mir so angepasst, dass diese wenn über die Konfigurationsdatei die Variable *bluetooth* mit *true* übergeben wird, entsprechend `BluetoothNetLayer` verwendet als `NetLayer` und auch allen `Host` Klassen mitteilt, dass es sich um eine Bluetooth Simulation handelt.

Für die Klasse `DefaultMobilityHost` habe ich eine Methode implementiert, die den Zufallsgenerator des Peerfacts verwendet um für jeden `DefaultMobilityHost` eine zufällige MAC-Adresse zu generieren, diese wollte ich für die Adressierung bei Nachrichten verwenden, doch diese sind auch eher überflüssig, da die bereits im PeerfactSim.KOM implementierten `NetIDs` sich besser eignen und es für die Simulation

selbst keinen Unterschied macht, ob dabei eine `NetID` verwendet wird oder die MAC-Adresse.

Da die zufällige Bewegung der `DefaultMobilityHosts` bereits mit der Klasse `RandomMovement` von Bjoern Richerzhagen implementiert ist und meinen Vorstellungen vollkommen entspricht, verwende ich diese so wie sie ist und brauche mich damit nicht weiter zu beschäftigen.

Die `BluetoothNetLayer` geht bei eintreffenden Paketen von den oberen Schichten wie folgt vor. Zuerst wird wie bei der `MobilityNetLayer` in einer Schleife durch alle anderen `BluetoothNetLayer` die Teil des `Subnets` sind durchgegangen und anhand der aktuellen Position in der Welt überprüft, ob diese sich in Reichweite befinden. Alle, die sich in Reichweite befinden, werden dann der Liste *neighbors* hinzugefügt. Als nächstes wird überprüft, ob der Empfänger der Nachricht bereits bekannt ist. Dazu werden die beiden Listen *isSlaveIn* und *piconet* überprüft.

Sowohl die Slaves des eigenen Pikonetzes, als auch die Master, mit denen ein Host vor kurzem eine Verbindung aufgebaut hatte, werden jeweils in eigene LinkedLists eingetragen. Bei der *isSlaveIn* Liste handelt es sich um eine aktuelle Liste der Master-Knoten, die den Host in Ihren Pikonetzen hinzugefügt haben. Möchte ein Host mal eine Nachricht versenden, wird in der *isSlaveIn*-Liste nachgeschaut, ob der Empfänger eventuell Master von diesem Host ist.

Die Liste *piconet* stellt das aktuelle Pikonetz eines Knotens dar und beinhaltet alle bereits verbundenen Slaves.

Mithilfe dieser beiden Listen kann man sicherstellen, dass die Verbindung zu diesen Knoten schon mal aufgebaut wurde und man somit bei den Events die *constructionTime* weglassen kann und man nur die *transmissionTime* hat. Dadurch wird auch nur ein mal gesendet und es entsteht kein unnötiges überfluten des Netzwerkes. Ist einer der Mitglieder mal außer Reichweite, entsteht nach dem *packetlossModell* ein Paketverlust und der Host wird zu den *parkedSlaves* hinzugefügt. Einer Liste, in der alle Geräte gespeichert werden, die jemals mit dem Host verbunden waren und anschließend geparkt wurden. Durch eine Abfrage, ob der Slave schon mal geparkt wurde, wird eine Redundanz in

der Liste verhindert. Sobald einer der geparkten Hosts wieder in Reichweite ist, wird er, solange es zu der gegebenen Zeit möglich ist, Teil des Pikonetzes. Dabei wird dieser aus der *parkedSlaves* Liste extrahiert und in die *piconet* Liste verlagert.

Ist der Empfänger weder in der *isSlaveIn*- noch in der *piconet*-Liste, muss die *neighbour* Liste überprüft werden ob der Empfänger vielleicht darunter ist, wenn ja, sendet man die Nachricht an ihn, wenn nicht, werden die Nachbarn darum gebeten die Nachricht weiterzuleiten. Um die Weiterleitung von Nachrichten zu simulieren, fügt der Sender der Nachricht für jeden einzelnen Nachbarn, den er in Reichweite hat, ein *BluetoothSimulationEvent.FORWARDMESSAGE*-Event in den Scheduler ein. Diese werden dann von der `BluetoothNetLayer` jeder dieser Nachbarn behandelt und bei weiteren vorhandenen Knoten rekursiv ausgeführt.

Erhält ein Knoten die Aufforderung eine Nachricht weiterzuleiten, überprüft dieser ob er die Nachricht bereits schon einmal erhalten hatte. Dies geschieht mithilfe einer HashMap, in der alle für die Weiterleitung bestimmten Nachricht gespeichert werden, wobei der Payload der Nachricht dem Key in der Hashmap entspricht. Wenn die Nachricht bereits bekannt ist, wird diese ignoriert, dies verhindert, dass Nachrichten im Netz ohne ein Ende weitergereicht werden.

Hat ein Knoten mal keine weiteren Nachbarn, an die er senden kann, wird ein *BluetoothSimulationEvent.TIMEOUT_EXPIRED*-Event vom eigenem Event-Handler bewältigt. Dabei wird die Nachricht gedroppt und dementsprechend an den `Monitor` weitergereicht.

Zusammenfassend ist es nun möglich auf der Netzwerkschicht des PeerfactSim.KOM Hosts mit einer `BluetoothNetLayer` auszustatten, sodass Nachrichten über diese verschickt beziehungsweise weitergeleitet werden. Dabei werden für jeden einzelnen Host alle Knoten, mit denen dieser in Kontakt gekommen ist, entsprechend gespeichert und verwaltet.

Kapitel 5

Simulationen

In diesem Kapitel werte ich einige Szenarien, die mithilfe des von mir implementierten Ansatzes ausgeführt wurden, aus. Diese Auswertung soll zeigen, ob und inwiefern die Bluetooth-Verbindung nun simuliert werden kann. Dabei gehe ich wie folgt vor. Zuerst lasse ich die Simulation mit unterschiedlichen Seeds und gleichem Szenario fünf mal durchlaufen um sicher zu stellen, dass der Code für alle Seeds gleichermaßen funktioniert und das Ergebnis kein Glücksfall ist. Diesen Ablauf wiederhole ich dann für andere Szenarien mit veränderten Variablen, wie einer anderen Weltgröße oder einer veränderten Anzahl an Hosts. Die Dauer der Simulationen lasse ich jedoch überall gleich bei 120 min, da die Knoten sich ohnehin zufällig bewegen und die Zeit somit keine entscheidende Rolle spielt. Um die vom PeerfactSim.KOM erstellten Daten der Szenarien zu veranschaulichen, verwende ich in diesem Kapitel zusätzlich Diagramme, die ich mit Gnuplot erstellt habe. Alle Boxplot Diagramme zeigen dabei jeweils die Ergebnisse der Fünf Durchgänge eines Szenarios. Jede X-Koordinate entspricht also einem anderen benutzen Seed.

5.1 Erstes Szenario

Im PeerfactSim.KOM Simulator wird die Größe der Welt in Millimetern definiert. Um eine bessere Lesbarkeit zu gewährleisten werde ich diese Angaben im folgenden umgerechnet in Meter nennen. Das erste von mir betrachtete Szenario ist eine quadratische Welt, mit der Größe $5000\,\mathrm{m} \times 5000\,\mathrm{m}$, was genau 25 Quadratkilometern entspricht. In dieser werden 50 Hosts platziert.

Da es sich hierbei um eine sehr große Fläche handelt, ist die Wahrscheinlichkeit, dass ein Host beim versenden seiner Nachrichten weitere Knoten in Reichweite hat, sehr gering. Noch geringer ist dann die Wahrscheinlichkeit, dass es sich dabei um den Empfänger der Nachricht handelt.

Die implementierte Reichweite beträgt 100 m. Mit der Kreisflächenformel $A = \Pi \times r^2$ beträgt somit die Fläche, die ein Knoten abdecken kann im Optimalfall circa

$$A = \Pi \times (0.1\,\mathrm{km})^2 = 0.0314\,\mathrm{km}^2.$$

Dies entspricht einem Teil von $\frac{0.0314\,\mathrm{km}^2}{25\,\mathrm{km}^2} = \frac{157}{125000}$ oder auch $1,256 \times 10^{-3} \quad (0,1256\%)$ der Gesamtfläche. Da ein Host die Möglichkeit hat 49 Hosts zu adressieren, beträgt die Wahrscheinlichkeit dafür, dass Sender und Empfänger sich in Reichweite befinden und eine Verbindung miteinander aufbauen können

$$P = \frac{1}{49} \times \frac{157}{125000} = 2,5632 \times 10^{-5} (0,002563\%).$$

Natürlich kommt dazu noch die Wahrscheinlichkeit, dass die Nachrichten weitergeleitet werden, diese wird aber kaum viel größer ausfallen. Somit sollten bei diesem Szenario so gut wie alle Pakete verloren gehen und nur wenige wenn überhaupt ankommen.

Bevor wir mit dem betrachten der Ergebnisse beginnen, verfasse ich hier noch eine kurze Erklärung zu Boxplot Diagrammen [Kro14]. Diese sollen einen Überblick darüber schaffen, in welchem Bereich die Daten liegen. Die Box beinhaltet dabei die mittleren 50%, der Strich in der Box ist der Median und die beiden äußeren Striche (Antennen) liegen

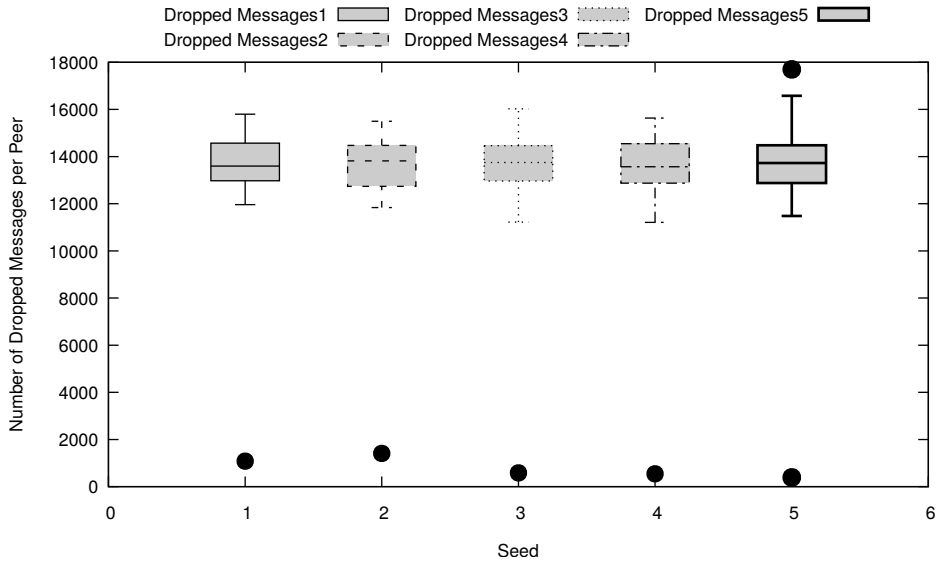

Abbildung 5.1: Durchschnittliche Anzahl gedroppter Nachrichten Szenario 1

bei den Werten, die am weitesten vom Median aber maximal eineinhalb mal so weit, wie die Box lang ist, entfernt sind [Gnu17]. Alle weiter außen liegende Punkte werden als Ausreißer mit einem Punkt gekennzeichnet.

Die Grafik 5.1 zeigt ein Boxplot Diagramm, wobei jeder Boxplot einem Durchlauf mit einem anderen Seed entspricht. Dieser Boxplot ist jeweils über die Anzahl der gedroppten Pakete aller Hosts generiert. Wie man erkennen kann liegen die meisten Punkte zwischen 12 000 und 16 000. Doch bei jedem dieser Boxplots gibt es auch jeweils einen Ausreißer, der weniger als 2000 verlorene Pakete hat. Doch nach betrachten der insgesamt gesendeten Nachrichten beziehungsweise den versuchten Sendevorgängen zeigt sich, dass diese Ausreißer generell nur so wenige Nachrichten gesendet haben siehe Diagramm 5.2.

An der Grafik 5.3 erkennt man, das jeweils ein Knoten die Gelegenheit hatte Nachrichten zu empfangen. Diese haben, je nach Seed, zwischen 400 und 1500 Nachrichten empfangen. Die Boxplots, die die anderen Knoten abbilden liegen jedoch alle bei Null. Geht man von den Werten aus Diagramm 5.2 aus, in dem man erkennen kann, dass jeder Host durchschnittlich 15 000 Nachrichten verschicken möchte, entspricht der Teil der angekommenen Nachrichten höchstens ca. $\frac{1500}{15000 \times 50} = \frac{1500}{750000} = \frac{1}{500}$ (0,2%)

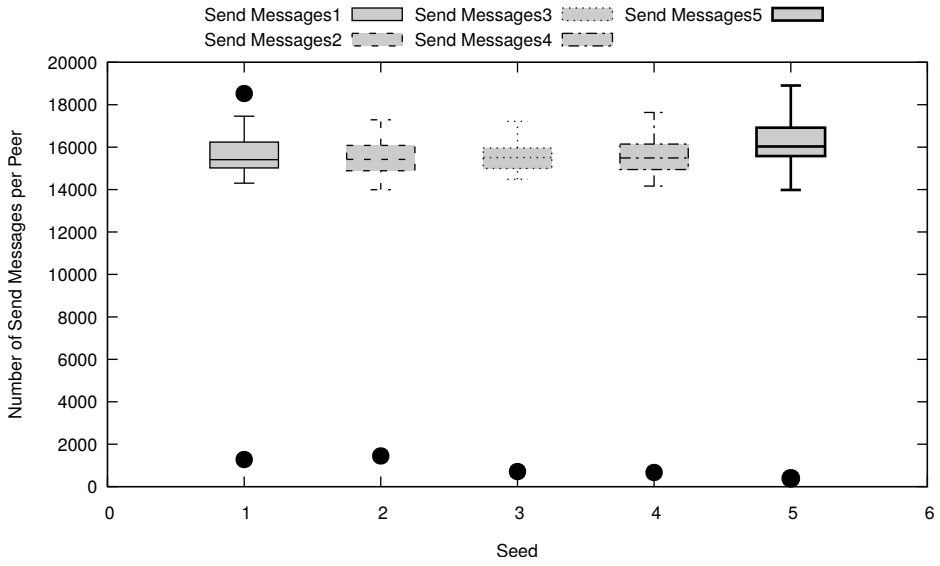

Abbildung 5.2: Durchschnittliche Anzahl gesendeter Nachrichten Szenario 1

In diesem Szenario erfüllen die Ergebnisse der Simulationen somit meine Erwartungen, da so gut wie alle Nachrichten gedroppt werden wie vorher berechnet.

5.2 Zweites Szenario

Als zweites Szenario habe ich mir überlegt eine viel kleinere Welt zu definieren, jedoch die Anzahl der Knoten nur gering zu verringern. Die Größe der Welt wurde auf $1000\,\mathrm{m} \times 1000\,\mathrm{m}$ gesetzt, was einer Fläche von einem Quadratkilometer entspricht, und dieses Mal mit 40 Hosts besetzt.

Bei dieser Größe ist es nun viel wahrscheinlicher, dass sowohl ein Sender und ein Empfänger sich in Reichweite befinden, als auch dass die Nachricht über mehrere Knoten an den Empfänger weitergeleitet wird.

Dieses Mal entspricht der Teil, der von einem Knoten abgedeckt wird $\frac{0.0314\,\mathrm{km}^2}{1\,\mathrm{km}^2} = \frac{157}{5000}$, also circa 3,14% der Gesamtfläche. Alle 40 Hosts wären somit in der Lage die ganze Welt abzudecken, was jedoch einem Abstand entspräche, der etwa doppelt so groß ist wie die

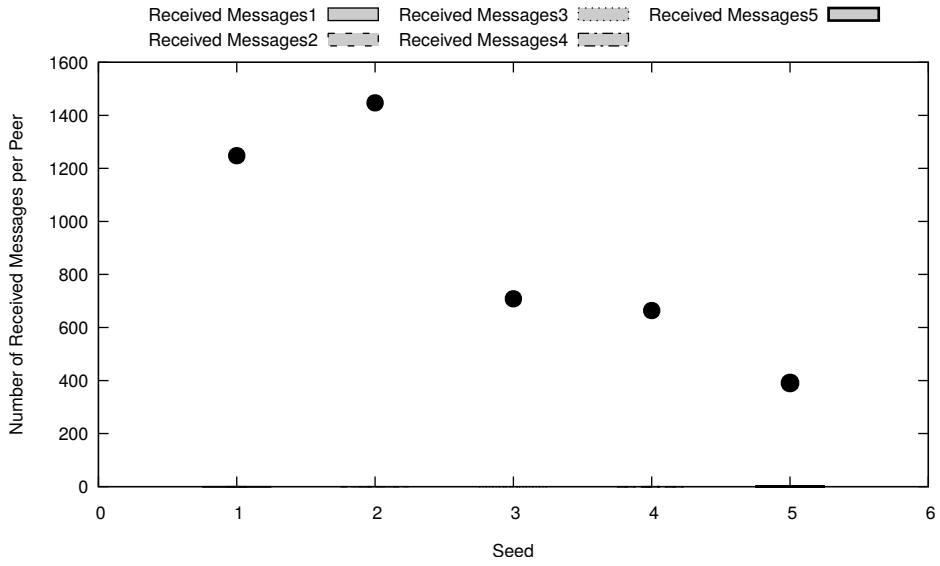

Abbildung 5.3: Durchschnittliche Anzahl empfangenen Nachrichten Szenario 1

unterstützte Reichweite. In diesem worst-case wäre kein Knoten in der Lage Nachrichten zu verschicken. Somit ist es immer noch zu erwarten, dass viele Pakete verloren gehen, die Anzahl der empfangenen Pakete doch auf jeden Fall höher als im ersten Szenario sein müsste.

An der Abbildung 5.4 kann man erkennen, dass die Anzahl der verlorenen Pakete von 40 000 bis 160 000 Stück breit verteilt ist. Dies ist schon einmal ein Zeichen dafür, dass Pakete von vielen Knoten an weitere Nachbarn weitergereicht werden können, da die Boxplots sonst einen niedrigeren Bereich abdecken würden. Betrachtet man dazu die Grafik 5.5, erkennt man an der hohen Anzahl der gesendeten Nachrichten, wobei der Median bei ca. 1×10^6 liegt, dass viele Knoten die Gelegenheit hatten eine Nachricht von anderen Knoten weiterzuleiten. Dies wird nämlich auch im Diagramm bei gesendeten Nachrichten berücksichtigt.

Und schließlich können wir den Boxplots über die empfangenen Nachrichten aus Abbildung 5.6 entnehmen, dass die Knoten im Durchschnitt deutlich mehr Nachrichten empfangen als Null. Um die Verteilung besser darzustellen, verwendet diese Abbildung eine Logarithmische Y-Skala. Dies bei der Verteilung dieser Werte praktischer, da man auf einer normalen Skala den Boxplot nur als einen Strich erkennen könnte. Man kann der

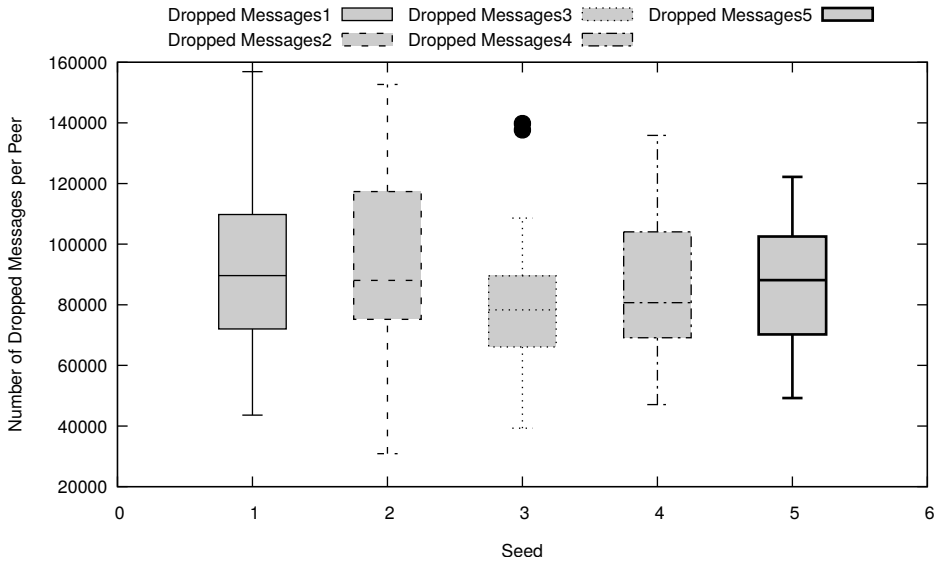

Abbildung 5.4: Durchschnittliche Anzahl gedroppter Nachrichten Szenario 2

Abbildung entnehmen, dass alle Knoten mindestens 100 Nachrichten und durchschnittlich circa 10 000 Nachrichten empfangen haben.

Den starken Ausreißer könnte man sich dadurch erklären, das dieser ein glückliches Bewegungsmuster hatte und somit mit vielen Knoten in Kontakt kommen konnte. Dies bestätigt auch das sortierte Diagramm 5.7, welches zeigt, wie oft die einzelnen Hosts aktive Bluetooth- Verbindungen mit anderen Hosts aufgrund der Reichweite abbrechen mussten und diese in ihre Liste der geparkten Geräte eingetragen haben. Im Fünften Durchgang dieses Szenarios hatte dabei ein Peer nämlich ca. $1,6 \times 10^6$ Mal diese Funktion benutzt, was klar aus der Menge hervor sticht.

5.3 Drittes Szenario

Als drittes Szenario werden 20 Hosts in einer $200\,\mathrm{m} \times 1000\,\mathrm{m}$ Welt gesetzt. In dieser schmalen Welt entspricht die von einem Knoten erreichbare Fläche $\frac{0.0314\,\mathrm{km}^2}{0.2\,\mathrm{km}^2} = \frac{157}{1000}$, also ca. 15,7% der Gesamtfläche. In diesem Fall ist also das erreichen des Empfängers sehr wahrscheinlich, denn allein die Wahrscheinlichkeit, dass Sender und Empfänger

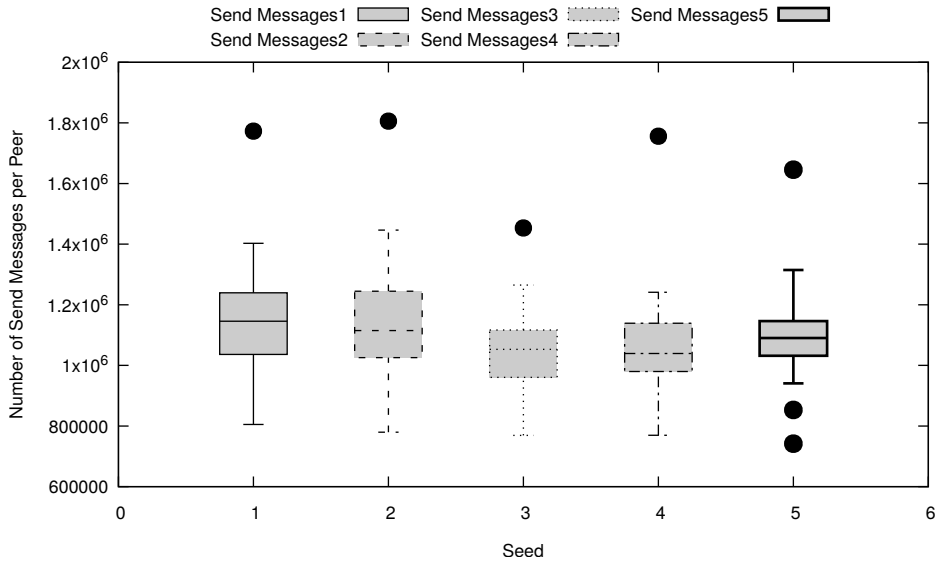

Abbildung 5.5: Durchschnittliche Anzahl gesendeter Nachrichten Sz 2

sich in Reichweite befinden beträgt

$$P = \frac{1}{19} \times \frac{157}{1000} = 8,2632 \times 10^{-3} (0,25632\%).$$

Da aber nun die Fläche viel kleiner ist und im besten Fall schon bei einer Kette von Neun Hosts eine Fläche von $0.185\,\mathrm{km}^2$ abgedeckt wird, was 92,5% der Welt entspricht, ist eine Weiterleitung der Nachrichten sehr wahrscheinlich. Die von mir mithilfe von GeoGebra erstellte Grafik 5.8 zeigt zwei sich schneidende Kreise, die Grundlage für meine folgende Rechnung waren. Bei diesen zwei Kreisen mit einem Radius von einem Zentimeter und einem Abstand von einem Zentimeter lässt sich die schneidende Fläche berechnen mithilfe von den beiden Kreissegmenten e und f, und dem Dreieck ABC. Addiert man die beiden Kreissegmente e und f, erhält man $0.52\,\mathrm{cm}^2 + 0.52\,\mathrm{cm}^2 = 1.04\,\mathrm{cm}^2$. Zieht man nun das Dreieck ABC, welches in beiden Segmenten liegt, ab, so kommt man auf $1.04\,\mathrm{cm}^2 - 0.43\,\mathrm{cm}^2 = 0.61\,\mathrm{cm}^2$ für die Schnittfläche der beiden Kreise oberhalb der X-Achse. Um auf die Gesamte Schnittfläche zu kommen muss man diesen Wert nur noch verdoppeln und wir erhalten $1.22\,\mathrm{cm}^2$.

Da es sich in der Implementation jedoch um einen Radius von $0.1\,\mathrm{km}$ handelt multiplizieren wir die berechnete Schnittfläche mit dem Faktor 10^8, damit wäre die Schnittfläche

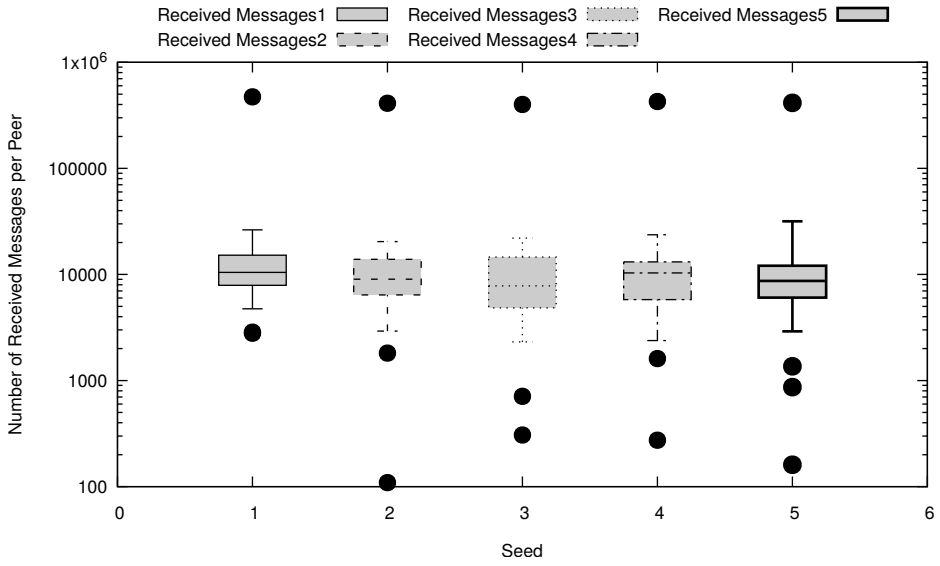

Abbildung 5.6: Durchschnittliche Anzahl empfangenen Nachrichten Log. Skala Sz 2

in unserem Fall $0.0122\,\mathrm{km}^2$ groß. Somit erhalten wir bei einer perfekten Kette von Neun Knoten die folgende Fläche

$$\begin{aligned} A &= 9 \times \text{Kreisfläche} - 8 \times \text{Schnittfläche} \\ &= 9 \times 0.0314\,\mathrm{km}^2 - 8 \times 0.0122\,\mathrm{km}^2 = 0.185\,\mathrm{km}^2 (92{,}5\%)\,. \end{aligned} \tag{5.1}$$

Mit den vorhandenen weiteren elf Knoten ist es somit sehr wahrscheinlich, dass Nachrichten weitergeleitet werden und auch am Ziel ankommen. Es sollten somit nur noch wenige Nachrichten verloren gehen.

Bei der Auswertung betrachten wir zunächst die Anzahl der erstellten Nachrichten pro Peer, diese können wir der Abbildung 5.9 entnehmen. Man sieht, dass ein Knoten stets sehr viele Nachrichten erstellt hat und zwar zwischen 500 000 und 600 000 Stück. Beim näheren betrachten der Boxplots ohne den Ausreißer in der Grafik 5.10 sehen wir, dass jeder Host zwischen 12 000 und 13 000 Nachrichten in das Netzwerk eingespeist hat.

$$n = 550000 + 19 \times 12500 = 787500$$

Insgesamt wurde also bei jedem Durchgang mit verschiedenen Seeds versucht rund

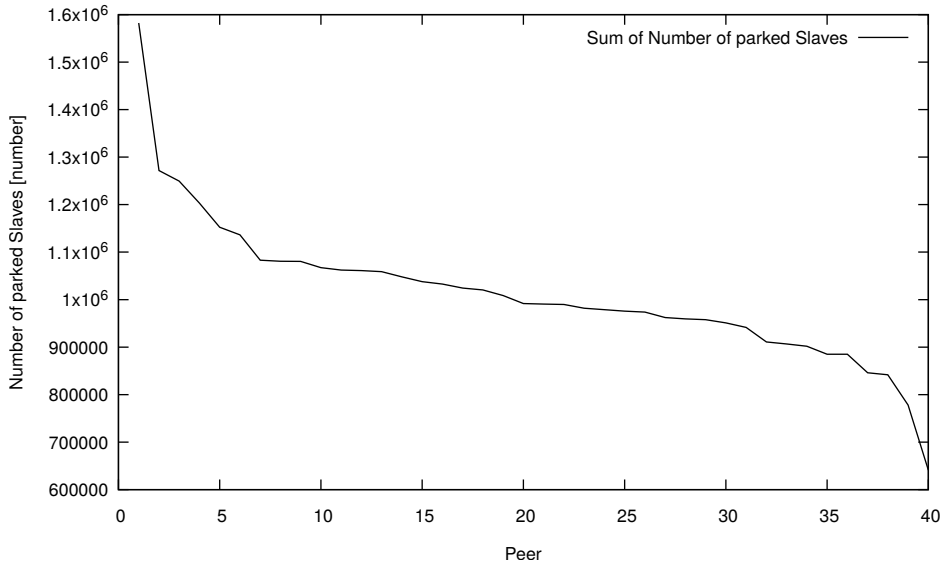

Abbildung 5.7: Fünfter Durchlauf Szenario 2 Anzahl von Park Aktionen

800 000 Nachrichten zu verschicken.

Sieht man sich dazu das Diagramm 5.13 über die empfangenen Nachrichten an, erkennt man, dass ein Knoten stets ca. 170 000 Nachrichten und die restlichen Knoten im Durchschnitt ca. 20 000 Nachrichten empfangen haben. Die Summe dieser beträgt somit ca. $170000 + 19 \times 20000 = 550000$ Nachrichten. Es sind also $\frac{550000}{800000} = 0,6875$ ungefähr 70% der Nachrichten am Zielort angekommen.

Zusätzlich dazu könnte man sich als Kontrolle die Abbildungen 5.12 und 5.11 anschauen. Hier sieht man die immens hohe Anzahl an versendeten beziehungsweise weitergeleiteten Nachrichten und im Gegensatz dazu die geringe Anzahl an gedroppten Nachrichten, was ein Zeichen dafür ist, dass die Knoten oft mindestens einen Nachbar in Ihrer nähe hatten.

Zusammenfassend lässt sich aufgrund der drei simulierten Szenarien sagen, dass die Implementation wie erwartet funktioniert. Wie man beim ersten Szenario erkennt, kommen bei einer sehr großen Welt mit 50 Knoten so gut wie keine Nachrichten an, denn dabei haben die Knoten sehr viel Platz sich zu verteilen und es kommt nur selten vor, dass Knoten in Reichweite sind. Bei einer sehr kleinen Welt hingegen, wie im dritten Szenario,

erwartet man, dass die Knoten viel mit einander kommunizieren können, da diese kaum Möglichkeiten haben sich zu verteilen. Das Ergebnis des dritten Szenarios war meinen Erwartungen recht nah. Doch das wirklich ca. 70% der Nachrichten den Empfänger erreichen ist schon überraschend.

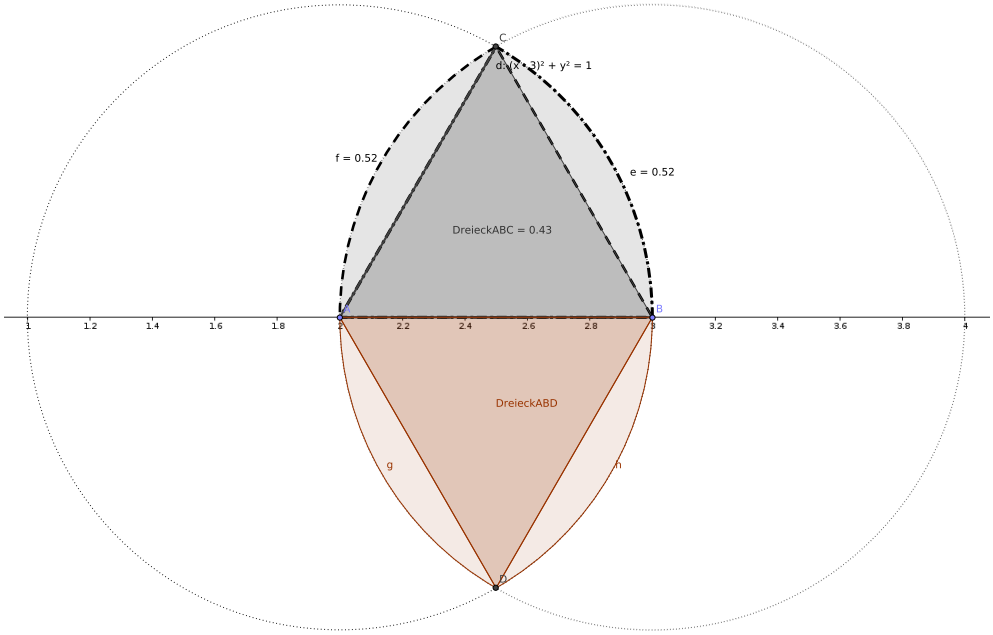

Abbildung 5.8: Schnittfläche zweier Kreise mit Abstand=Radius

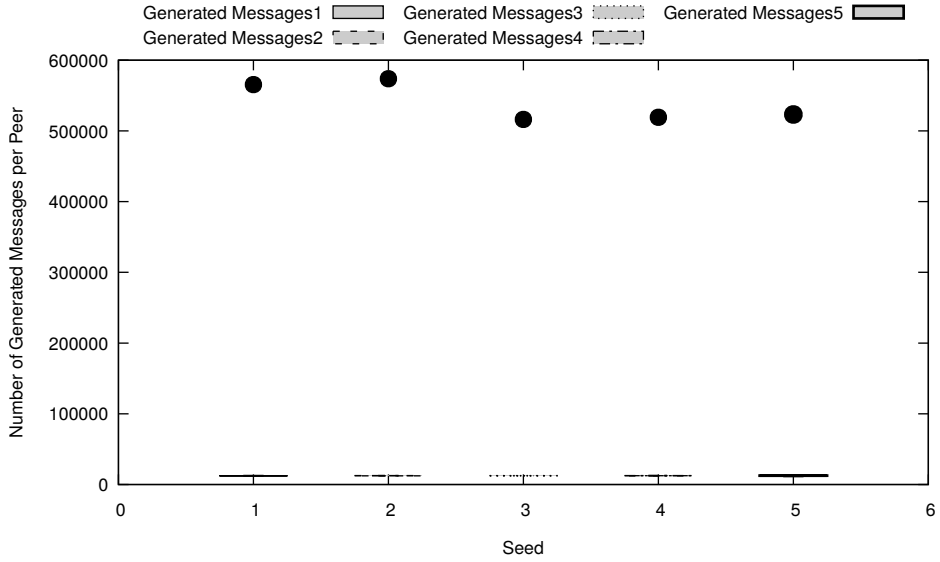

Abbildung 5.9: Durchschnittliche Anzahl erstellter Nachrichten Sz 3

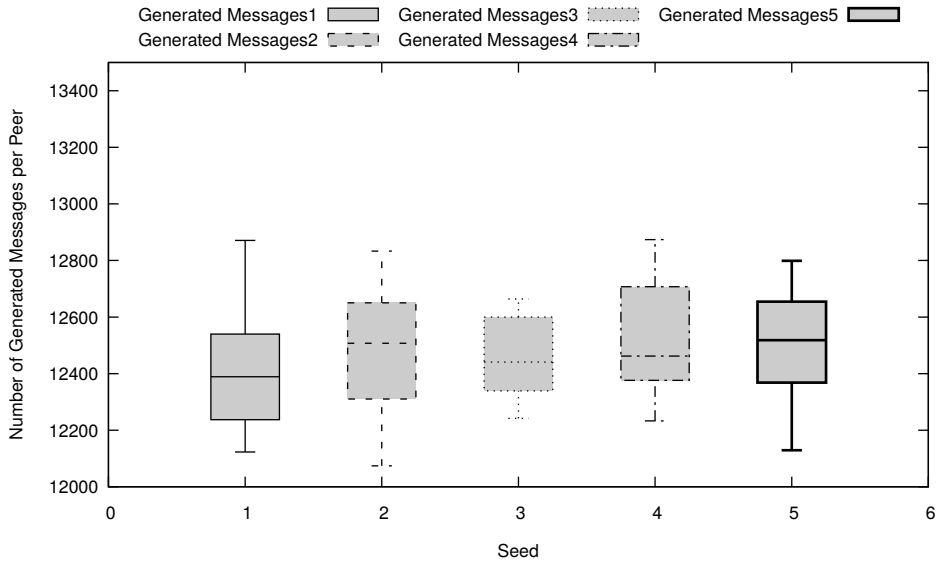

Abbildung 5.10: Durchschnittliche Anzahl erstellter Nachrichten ohne Ausreißer Sz 3

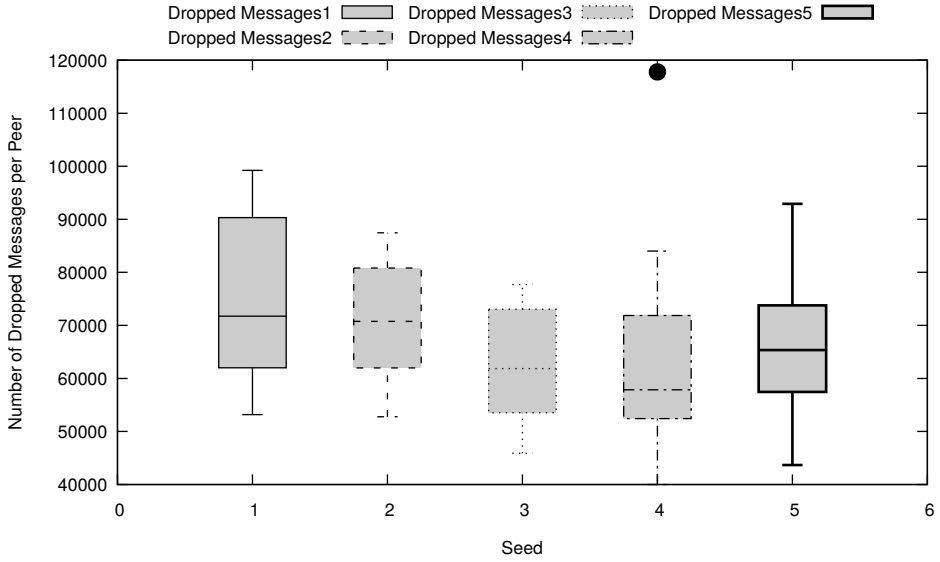

Abbildung 5.11: Durchschnittliche Anzahl gedroppter Nachrichten Sz 3

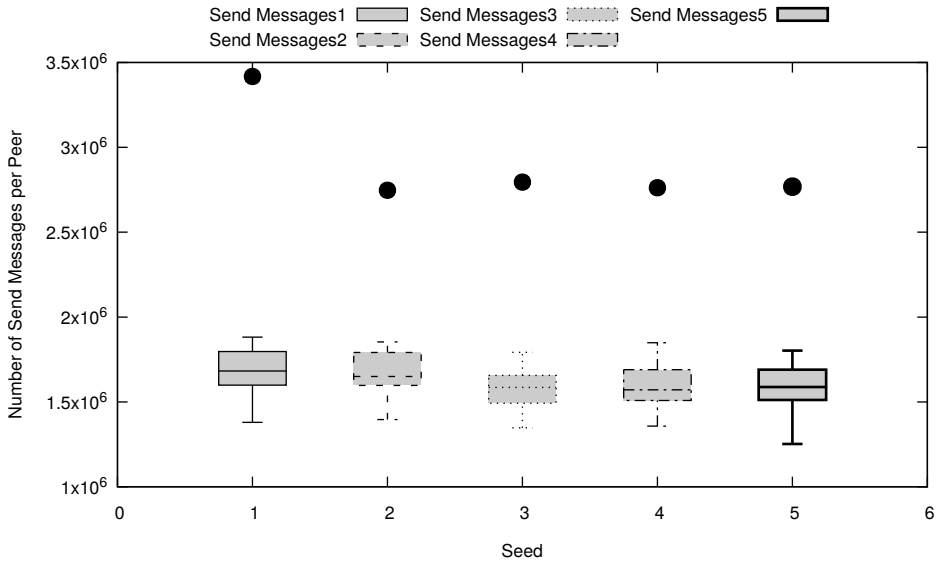

Abbildung 5.12: Durchschnittliche Anzahl gesendeter Nachrichten Sz 3

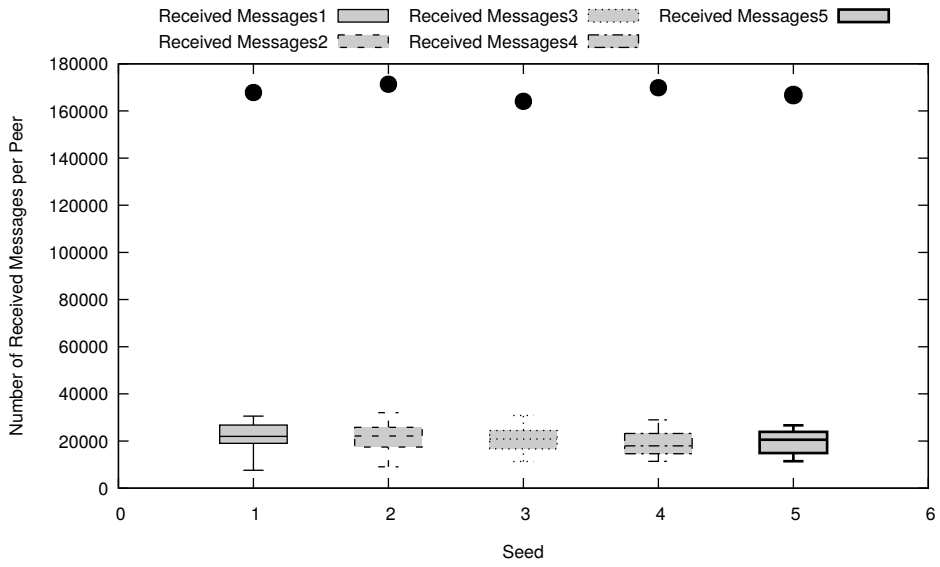

Abbildung 5.13: Durchschnittliche Anzahl empfangenen Nachrichten Sz 3

Kapitel 6

Fazit

In diesem Kapitel erstelle ich anhand der Auswertungen im vorherigen Kapitel ein Fazit zu meiner Implementation der Bluetooth-Kommunikation und generell zu der Bachelorarbeit. Dazu Fasse ich zuerst die Simulationsergebnisse zusammen und gehe auf diese ein.

Aufgrund der Ergebnisse aller drei Szenarien, die ich im Kapitel Simulation betrachtet habe, bin ich der Meinung, dass meine Implementation den Erwartungen und somit der reellen Bluetooth-Technologie ziemlich nah kommt. Dadurch, dass im dritten Szenario, das in einer $0.2\,\mathrm{km}^2$ kleinen Welt mit 20 Hosts simuliert wurde, zwei Drittel der Nachrichten von den Hosts gut weitergeleitet werden und ihren Weg vom Sender bis zum Empfänger finden, scheint der Code angebracht zu funktionieren. Denn im Gegensatz dazu kommen im ersten Szenario, wie man sich auch gut vorstellen kann nur wenige Nachrichten an und es werden beinahe alle Nachrichten gedroppt. Die Simulationen erscheinen also plausibel und es gibt keine offensichtliche Diskrepanz.

Die geplante Analyse der Anzahl von Hops pro Nachricht wäre mit sehr vielen Problemen verbunden. Da man hierbei für jede Nachricht alle Zurückgelegte Schritte der Nachrichten dokumentieren und verwalten müsste. Das ganze wäre dann unüberschaubar, da zum Beispiel im dritten Szenario ca. 800 000 Nachrichten verschickt wurden. Und selbst wenn jeder Ursprungsknoten einer Nachricht nur maximal zwei Nachbarn gleichzeitig

hätte, würde es bereits doppelt so viele Einträge nach nur einem Hop liefern.

6.1 Future Work

Als Future Work lässt sich die Implementation der verschiedenen Versionen von Bluetooth einrichten, die den Rahmen dieser Arbeit leider gesprengt hätte. Dazu könnte man über die Konfigurationsdatei mithilfe von zwei Variablen beispielsweise einen Bereich bestimmen, in dem die Bluetooth-Version der Hosts liegen darf (zum Beispiel zwischen 1.0 und 3.0) und lässt dafür durch den Zufallsgenerator vom Peerfact für jeden Host einzeln entscheiden, welche Version dieser erhält. Dies ließe sich auch ähnlich für die Bluetooth Klassen implementieren.

Die Anzahl an Hops, die eine Nachricht gebraucht hat bis Sie den Empfänger erreicht hat, kann man zurzeit leider nicht ausrechnen. Dies könnte man eventuell noch implementieren, indem man eine Struktur einbaut, in der für alle Nachrichten, die den Empfänger erreichen zurückverfolgt wird, welche einzelnen Zwischenstationen erreicht wurden und einen Pfad daraus konstruiert. Zusätzlich dazu müsste eine geeignete Metrik und Ausgabe für den Monitor implementiert werden.

Ein letzter Punkt, den man noch als Future Work einbauen könnte, wäre die Definition und Berücksichtigung verschiedener Zustände für die Hosts. Diese könnten dann beispielsweise in den Standby Zustand wechseln und würden dann länger brauchen für eine Verbindung mit einem anderen Knoten.

Literaturverzeichnis

[Aky04] AKYILDIZ, F. Cuomo; T. Melodie; I.: *Distributed self-healing and variable topology optimization algorithms for qos provisioning in scatternets,*. Vol. 22, No. 7. 2004.

[Blu99] *Specification of the Bluetooth System.* : *Specification of the Bluetooth System.* v1.0 B, 1999. 1–1000 S.

[BS] BLUETOOTH SIG, Inc: *Origin of the Bluetooth Name.* `https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-origin`, . abgerufen am 6. Oktober 2017.

[Che13] CHEN, C.-M. Yu; S.-K. Hung; Y.-C.: *Forming mesh topology for bluetooth ad hoc networks.* 2013.

[Chl01] CHLAMTAC, Gergely V. ZBruba; Stefan0 Basagni; I.: *Bluetrees-Scatternet Formation to Enable Bluetooth-Based Ad Hoc Networks.* 2001.

[Chl04] CHLAMTAC, C. Petrioli; S. Basagni; I.: *Bluemesh: degree-constrained multi-hop scatternet formation for bluetooth networks.* Vol. 9, No. 1. 2004.

[Com] COMPUTERBILD: *Geschwindigkeit und Reichweite von Bluetooth.* `http://www.computerbild.de/artikel/`

cb-Ratgeber-Handy-Alles-ueber-Bluetooth-3177119.html, . abgerufen am 20. Oktober 2017.

[Con06] CONTI, Luciana Pelusi; Andrea Passarella; M.: *Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad hoc Networks*. Vol. 44, No. 11. 2006.

[Dar] DARMSTADT, Technische U.: *Technische Universität Darmstadt Website*. https://www.tu-darmstadt.de/universitaet/index.de.jsp, . abgerufen am 20. Oktober 2017.

[Dr.14] DR.B.SANKARAGOMATHI, K.R.Kanagavalli;: *A BLUETOOTH MOBILE AD HOC NETWORK COMMUNICATION TOPOLOGIES*. Vol. 5, No. 1. 2014.

[Dü] DÜSSELDORF, Heinrich-Heine-Universität: *Heinrich-Heine-Universität Düsseldorf Website*. https://www.uni-duesseldorf.de/home/startseite.html, . abgerufen am 23. Oktober 2017.

[Gnu] GNUPLOT: *Command-line driven graphing unitlity*. http://www.gnuplot.info/, . abgerufen am 23. Oktober 2017.

[Gnu17] *Gnuplot 5.0, An Interactive Plotting Program.* : *Gnuplot 5.0, An Interactive Plotting Program*. v5.0.7, 2017. 48 S.

[HC] HOUSE CONTROLLERS, Wimoto: *Wimoto Bluetooth Sensoren für Hausüberwachung*. https://www.housecontrollers.de/hausueberwachung/wimoto-bluetooth-sensoren-sollen-haus-ueberwachen/, . abgerufen am 24. Oktober 2017.

[Kha17] KHAN, Khiza Asaf; Muhammad Umer Sarwar; Muhammad Kashif Ha-

nif; Ramzan Talib; I.: *A review of Bluetooth based Scatternet for Mobile Ad hoc Networks*. Vol. 8, No. 6. 2017.

[Kri05] KRISHNAMURTHY, Prasant Mohaparta; Srikanth V.: *Ad Hoc Networks, Technologies and protocols*. Springer, 2005. ISBN 0387226893

[Kro14] KRONTHALER, Franz: *Statistik angewandt. Datenanalyse ist (k)eine Kunst.* Springer Verlag, 2014. 38 S. ISBN 9783642537394

[NYCYSJP99] NI, Sze-Yao; YU-CHEE, Tseng; YUH-SHYAN, Chen; JANG-PING, Sheu: The Broadcast Storm Problem in a Mobile Ad Hoc Network. In: *Proceedings of the fifth annual ACM/IEEE International Conference on Mobile computing and networking (MobiCom '99)*. Seattle, Washington, August 1999, S. 151–162.

[Olu15] OLUWARANTI, A.: *Evaluation of tree scatternet formation algorithm for enhanced bluetooth scatternet*. Vol. 2, No. 2. 2015.

[PS] PEERFACTSIM.KOM-SIMULATOR: *A Large Scale Simulation Framework for Peer-to-Peer System.* `http://peerfact.kom.e-technik.tu-darmstadt.de/de/index.html`, . abgerufen am 19. Oktober 2017.

[Pus11] PUSSEP, Dominik Stingl; Christian Gross; Julius Rückert; Leonhard Nobach; Sebastian Kaune; K.: *PeerfactSim.KOM - Dokumentation 2011*, 2011

[Tan03] TANNENBAUM, Andrew S.: *Computernetzwerke*. 4.Auflage. Pearson Studium, 2003. ISBN 9783827370464

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 25.Oktober 2017 Maxim Specianov

Diese CD enthält:

- eine *pdf*-Version der vorliegenden Bachelorarbeit
- die LaTeX- und Grafik-Quelldateien der vorliegenden Bachelorarbeit samt aller verwendeten Skripte
- die Quelldateien der im Rahmen der Bachelorarbeit erstellten Implementation für PeerfactSim.KOM
- den zur Auswertung verwendeten Datensatz

- die Websites der verwendeten Internetquellen