HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ita Institute for Internet Technologies and Applications

MobileCloud

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2011

Autor(en):	Heinrich Muralt, Marcel Steiner
Betreuer:	Prof. Dr. Josef M. Joller
Projektpartner:	Institut für Internet-Technologien und -Anwendungen

AUFGABENSTELLUNG

1.1 BETREUER

1.1.1 Betreuer HSR:

- Prof. Dr. Josef M. Joller, Abteilung für Informatik HSR/FHO

1.2 Ausgangslage, PROBLEMBESCHREIBUNG

Mobile Devices (iPhones, iPads, Android basierte Geräte) sind sehr stark verbreitet und verfügen über häufig ungenutzte oder ungenügend genutzte Fähigkeiten.

Falls die mobilen Geräte miteinander / gemeinsam Aufgaben lösen würden / könnten, hätte man in fast jeder Gruppe von mobile Usern ein enormes Rechenpotential zur Verfügung. Heute stecken viele dieser Devices arbeitslos in irgendeiner Tasche.

Wichtige Aspekte einer mobilen Kooperation und Kollaboration sind:

- Privatsphäre: Teilnehmer müssen jederzeit wissen, was auf den mobilen Geräten zurzeit aktiv vor sich geht.

- Sicherheit: Teilnehmer müssen die Gewissheit haben, dass keine unerlaubten und unerwünschten Besucher auf den mobilen Geräten aktiv sind.

Ansätze, welche in dieselbe Richtung gehen und als Anschauungsbeispiele benutzt werden können:

- Viber (auf iPhone und Android): Diese Software stellt Basisdienste zur Verfügung.
  Unser Ziel ist weitergehend in Richtung "semantischer" Information (Topics, Social Networks, …)

Die Integration der mobilen Geräte steht in einer ersten Phase im Vordergrund.

1.3 Technologische Rahmenbedingungen

Als externe Datendarstellung kann XML verwendet werden. Damit lässt sich die Datendarstellung plattformneutral realisieren. Sicherheitsaspekte sollen fürs erste nicht berücksichtigt werden (keine Verschlüsselung der übermittelten Daten). Dies kann zu einemspäteren Zeitpunkt leicht nachgeholt werden.

Die Lösung soll sich auf eine mobile Plattform beschränken, konkret Android. iPhone wird ausgeklammert, da diese Geräte zu proprietär sind.

1.4 Ziele der Arbeit

Hauptziel dieser Arbeit ist es zu untersuchen, inwiefern mobile Geräte miteinander vernetzt werden können, unter Ausnutzung aller verfügbaren Netzwerk- Technologien (IP basierte Netzwerk- Technologien, mobile Telefonie- basierte Technologien).

Untergeordnetes Ziel ist der Aufbau eines mobilen Clusters aus lokal miteinander kommunizierenden mobilen Geräten. In einer späteren Phase würden wir versuchen die Einschränkung auf "lokale Cloud/ Cluster" fallen zu lassen und globale mobile Clouds ins Auge fassen.

Ein weiteres, längerfristiges Ziel ist der anschliessende Ausbau in Richtung "mobile Service Cloud", also die Anbietung unterschiedlichster Services. Diese können entweder noch entstehen; oder es kann sich um Services handeln, welche bereits irgendwo und irgendwie vorhanden sind und eingebunden/benutzt werden.

Ein mögliches Zusatzziel sind Discovery und Reconfigure Dienste, die es erlauben die mobile Cloud laufend der aktuellen Situation anzupassen (mobile Geräte treten bei, mobile Geräte verabschieden sich (oder fallen aus)

1.5 Zur Durchführung

Die erfolgreiche Bearbeitung dieser Aufgabe bedingt Grundkenntnisse in der Programmierung verteilter Systeme voraus (praktische Beispiele, nicht bloss Literaturkenntnisse) sowie die Bereitschaft sich in neue und neuste Ergebnisse einzuarbeiten (MobileAnwendungen, speziell ganz neue Möglichkeiten mit und Android).

Mit dem Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Das Sitzungsintervall ist flexibel: falls wenig oder keine Probleme auftreten können die Sitzungsintervalle vergrössert werden; falls Probleme auftauchen, werden die Intervalle verkleinert.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse sind in einem Protokoll zu dokumentieren, das dem Betreuer per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben.

1.6 Dokumentation und Abgabe

Wegen der beabsichtigten Verwendung der Ergebnisse in weiteren Projekten wird auf Vollständigkeit sowie (sprachliche und grafische) Qualität der Dokumentation erhöhter Wert gelegt.

Die Dokumentation zur Projekt- Planung und –Verfolgung ist gemäss den Richtlinien der Abteilung Informatik anzufertigen. Die Detailanforderungen an die Dokumentation der Recherche- und Entwicklungsergebnisse werden entsprechend dem konkreten Arbeitsplan festgelegt.

Die Dokumentation ist vollständig auf CD abzugeben oder Zugriff auf eine Ablage (SVN, GIT).

Neben der Dokumentation sind abzugeben:

- alle zum Nachvollziehen der Arbeit notwendigen Ergebnisse und Daten (Quellcode, Buildskripte, Testcode, Testdaten usw.)
- Material für eine Abschlusspräsentation (ca. 20')

1.7 Termine

Siehe Terminplan auf der HSR Seite.

HSR Terminplan	Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch die Betreuer
1. Semesterwoche	Kick-off Meeting
2. Semesterwoche	Abgabe der Projektplanung (Entwurf), einschliesslich ggf. zu beschaffender Hardware
4. Semesterwoche	Abgabe eines vorläufigen Technologierechercheberichts und eines detaillierten Vorschlags für einen Arbeitsplan. Festlegung des Projektplans mit dem Betreuer
6. Semesterwoche	Abgabe eines Umsetzungsvorschlags für die Experimentierumgebung (Funktionsumfang, Aufwandsabschätzung) Abstimmung und Festlegung mit dem Betreuer
7. Semesterwoche	Abgabe Anforderungs- und Domainanalyse für die Experimentierumgebung
10. Semesterwoche	Review-Meeting Softwaredesign für die Experimentierumgebung
13. Semesterwoche	Vorstellung der Implementierung (Arbeitsstand)
HSR Terminplan	Abgabe der Kurzbeschreibung für zur Kontrolle an den Betreuer
HSR Terminplan	Abgabe des Berichtes an den Betreuer (bis 12:00 Uhr)

1.8 Literaturhinweise

Generelle Referenzen:

Siehe Vorlesung über Verteilte Software Systeme .

Weitere Literatur wird steht bei Bedarf zur Verfügung!

1.9 Beurteilung

Eine erfolgreiche Studienarbeit erhält 8 ECTS-Punkten (1 ECTS Punkt entspricht einer Arbeitsleistung von ca. 25 bis 30 Stunden). Für die Modulbeschreibung der Studienarbeit siehe https://unterricht.hsr.ch/staticWeb/allModules/10938_M_SAI.html

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Mgmt Summary, techn. u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/5
3. Inhalt*)	3/5

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit mit dem Studenten vereinbart

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Studienarbeiten.

Rapperswil, den

Der verantwortliche Dozent

ERKLÄRUNG

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.

Ort, Datum: Rapperswil, 22.12.2011

Heinrich Muralt

Ort, Datum: Rapperswil, 22.12.2011

Marcel Steiner

Inhaltsverzeichnis

ABSTRACT

Laut einem Medienbericht des Schweizer Fernsehens vom Freitag, 29. Oktober 2010, ergab eine Umfrage bei über 1000 Jugendlichen zwischen 12 und 19 Jahren, dass 98% der Befragten ein eigenes Handy besitzen [1]. Davon nutzen über 40% das Internet über ein Smartphone [2].
Könnte man einige dieser mobilen Geräte in einem Netzwerk zusammenfügen, würde das entstandene Netzwerk ein enormes Rechenpotential darstellen. Es könnten Dienste, wie zum Beispiel Internettelefonie oder verteilter Speicher, angeboten werden – wie das im Bereich Cloud Computing bereits vorhanden ist.

Das primäre Ziel dieser Studienarbeit war es zu untersuchen, wieweit mobile Geräte untereinander vernetzt werden können.

Mit einem ersten Prototyp wurde eine zentralisierte Peer-to-Peer (P2P) Lösung untersucht. Dabei wurde der Fokus auf die Kommunikation über UDP/IP sowie das Multithreading gelegt.

Der zweite Prototyp wurde als loses P2P-Netzwerk, angelehnt an das von Microsoft entworfene Pastry-Netzwerk, umgesetzt. Als Routingtabelle wurde die Nachbarschaftsliste implementiert. Das hat für das Routing ein Laufzeitverhalten von $O(n)$ zur Folge.
Des Weiteren wurde für das Verwalten des Netzwerkes, sowie für die Kommunikation zwischen den einzelnen Clients ein Protokoll entworfen. Dieses wurde als zusätzliche Schicht auf UDP/IP umgesetzt.

Mit dem zweiten Prototyp konnte gezeigt werden, dass es möglich ist, mittels eines P2P-Netzwerkes mobile Geräte zu einem Cluster zu verbinden. Zusätzlich konnte mit einem einfachen Nachrichtendienst die Funktionsfähigkeit des Netzwerkes aufgezeigt werden.

Management Summary

Ausgangslage

Mobile Geräte wie Handys und Tablet-Computer sind aus dem Alltag nicht mehr wegzudenken. Deren Rechenleistung ist in den letzten Jahren stark gewachsen. Viel genutzte Dienste wie E-Mail- und Internet-Zugang wurden ausgebaut. Daher hat der Einsatz mobiler Geräte als Ersatz zu stationären Computer in den letzten Jahren markant zugenommen. Auch zukünftig ist mit einem Wachstum in diesem Bereich zu rechnen.

Das Potenzial dieser mobilen Kleincomputern wird noch sehr spärlich genutzt, im Gegensatz zu stationären Computern. Vor allem hinter ihrer weite Verbreitung und Anzahl verbirgt sich ein grosses Rechenpotenzial. Folglich liegt es nahe, diese Geräte miteinander zu vernetzen, um Möglichkeiten dieses Netzwerks zu nutzen. Dies könnten Dienste wie Internettelefonie oder verteilte Rechenoperationen sein.

In dieser Arbeit ging es darum, herauszufinden wie ein Netzwerk aus mobilen Geräten realisiert werden könnte. Für diesen Zweck musste ein Prototyp entwickelt werden. Er sollte als Grundlage für fortsetzungsarbeiten dienen können und den Aufbau und die Verwaltung des Netzwerks beinhalten.

Vorgehen, Technologien

Als erstes wurde zur Untersuchung der grundlegenden Kommunikationsmöglichkeiten zwischen den einzelnen mobilen Geräten einen zentralisierten Ansatz gewählt. Die Grundfunktionalitäten konnten in einem ersten Prototyp implementiert und getestet werden.
Aus den Erkenntnissen des ersten Prototyps entstand ein Zweiter. Dieser wurde, angelehnt an ein von Microsoft entwickeltes Netzwerk, vollkommen dezentralisiert umgesetzt. Dabei musste auf eine stabile und flexible Architektur geachtet, so dass der Prototyp bei einer späteren Arbeit leicht ausgebaut werden kann.

Die Prototypen wurden für mobile Geräte mit dem Betriebssystem Android entwickelt. Dafür wurde das Android Software Development Kit und Java als Programmiersprache verwendet. Mittels Android Emulatoren und 4 unterschiedlichen Smartphones, wurden die Tests durchgeführt. Mit Hilfe einer Computer-Applikation konnten mehrere Clients erstellt und somit die Skalierbarkeit des Netzwerks getestet werden.

Ergebnisse

Es wurde ein finaler Prototyp entwickelt, mit welchem erfolgreich mehrere Smartphones miteinander verbunden werden können. Dafür benötigt man lediglich Smartphones und ein Medium, über welches die Kommunikation stattfinden kann. Dieses Medium kann ein WLAN, das Internet, das Mobilfunknetz oder sogar Technologien, welche erst in Zukunft entwickelt werden, sein. Die einzige Voraussetzung ist, dass das Medium den UDP/IP Standard unterstützt.
Über das GUI des Prototyps können Textnachrichten untereinander ausgetauscht oder das Netzwerk überwacht werden.

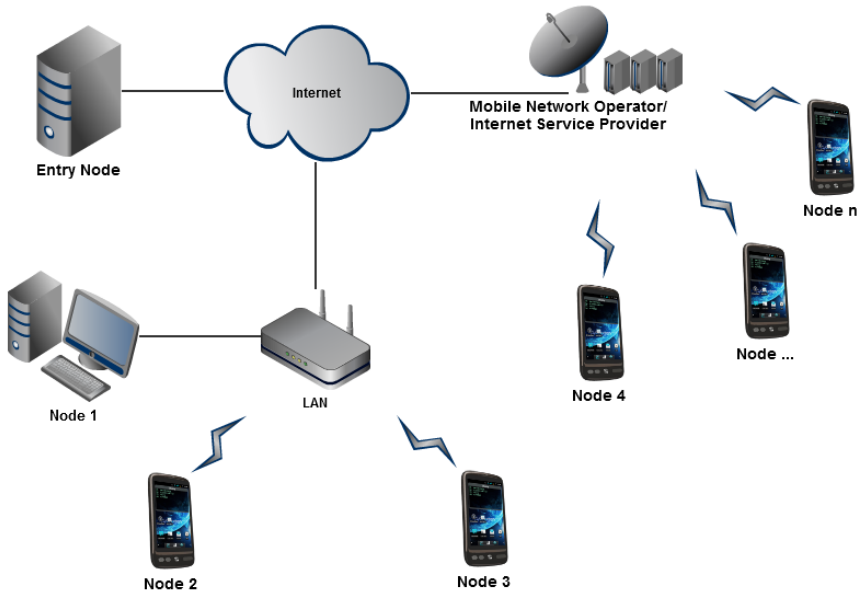


ABBILDUNG 1 ÜBER VERSCHIEDENE MEDIEN VERBUNDENE SMARTPHONES

AUSBLICK

Mit dem entwickelten Prototyp konnte aufgezeigt werden, dass es möglich ist mobile Geräte zu einem Netzwerk zu verbinden. Um MobileCloud jedoch produktiv einsetzen zu können, ist noch einige Arbeit nötig. Einerseits müsste eine sichere und anonyme Datenübertragung gewährleistet und das P2P-Netzwerk müsste automatisch rekonfigurierbar gemacht werden, um nur zwei zu Punkte nennen.

Würde in weiteren Projekten MobileCloud noch erweitert, könnte eine sehr fortschrittliche Software entstehen, die ebenfalls für kostenpflichtige Dienste verwendet werden könnte.

1 Einleitung

Mobile Geräte wie IPhone, Android oder Blackberry finden weltweit eine immer stärkere Verbreitung. Über die Jahre haben die Geräte um einiges an Funktionalität und Akkulaufzeit zugenommen und gleichzeitig an Grösse verloren. Das machte das frühere klobige und unhandliche Handy zum heutigen durchgestylten Modeaccessoir. Dabei änderte sich auch deren Einsatzgebiet. Vom einfachen Telefon wurde ein multifunktionales, mobiles Büro, auf welchem man immer erreichbar ist und sogar im Zug noch schnell kleine Arbeiten erledigen kann.

Durch diesen Prozess änderten sich auch die Anforderungen an die Mobilfunkbetreiber. Immer mehr Personen wollten ein Datenabo, damit sie auch unterwegs Zugang zum Internet haben. Dies führte zur Preissenkung der Abonnemente, was wiederum die Nachfrage steigen liess.
Heutzutage gibt es fast niemanden mehr, der kein mobiles internetfähiges Gerät besitzt. Und genau darauf baut diese Studienarbeit auf.

1.1 Einführung

Der Internetzugang auf mobilen Geräten, wird meistens nur für einen Blick in die Mailbox, das Überprüfen des Wetters, das schauen von YouTube Videos oder ähnlichem verwendet. Wenn man erweiterte Dienste wie zum Beispiel das Versenden von Textnachrichten oder das Führen von Telefongesprächen auch über das Internet nutzen möchte, um Telefonkosten zu sparen, ist das nur im beschränkten Rahmen möglich. Ausserdem müssen zusätzlichen Tools installiert werden, wie zum Beispiel Skype, WhatsApp oder Viber, was das ganze intransparent macht.
Auch ist die Rechenleistung der einzelnen Geräte gestiegen, diese bleibt allerding meist ungenutzt. Würde man diese Geräte zusammenfügen, hätte man einen Cluster mit einer enormen Rechenkapazität.

Das Ziel dieser Studienarbeit war es, zu untersuchen inwieweit es möglich ist, die einzelnen mobilen Geräte untereinander zu verbinden. Das Gerätenetzwerk sollte später die Möglichkeit bieten, Services wie etwa IP-Telefonie oder der Austausch von Daten anzubieten. Ein weiteres Anwendungsgebiet wäre die Parallelisierung gewisser Arbeiten. Dabei muss die Arbeit auf verschiedene Geräte verteilt, dort gelöst und anschliessend wieder zurückgeführt und ausgewertet werden.

In einem ersten Prototyp wurde die Kommunikation zwischen den einzelnen mobilen Geräten mit einem Android Betriebssystem überprüft. Dabei ging es hauptsächlich um den Datenaustausch zwischen zwei Geräten mithilfe eines Servers. Es wurde auch untersucht, wieweit Abläufe auf den mobilen Geräten parallelisiert werden können.
Gleichzeitig wurde ergänzend untersucht, welche theoretischen Möglichkeiten es gibt, die Geräte untereinander zu vernetzen und das entstandene Netzwerk ohne Server zu betreiben, also als P2P-Netzwerk.

Mit den Erkenntnissen des ersten Prototyps wurde anschliessend ein Zweiter entwickelt. Dabei wurde versucht eine flexible und stabile Grundarchitektur zu erstellen, auf welcher weitere Arbeiten aufbauen könnten.

Wichtig dabei war die Dokumentation der Erkenntnisse und des Codes, um Fortsetzungsarbeiten einen möglichst schnellen Einstieg zu ermöglichen.

In einem ersten Teil beschreibt dieses Dokument die Ergebnisse der ersten zwei Prototypen. Es wird dabei auf Funktionen eingegangen, welche implementiert wurden, noch implementiert werden könnten und einfach untersucht wurden.
Im zweiten Teil wird auf die Projektauswertung eingegangen und einen kurzen Ausblick in die Weiterentwicklung des Projektes MobileCloud gegeben.

1.2 Motivation

Die direkte Verbindung zwischen einzelnen mobilen Geräten über IP-basierte Netze würde es ermöglichen, miteinander zu kommunizieren, ohne dass dabei zusätzliche Telefonkosten entstehen. Allerdings nur, wenn eine Verbindung zu den anderen Geräten besteht, entweder lokal über Wireless oder global über das Internet. Da die meisten Besitzer eines mobilen Geräts ein Abonnement mit grossem oder sogar unlimitiertem Datenvolumen haben, ist es auch möglich diese Geräte über das Mobilfunknetz in das Netzwerk miteinzubeziehen.

Für uns war in erster Linie wichtig anzuschauen, mit welchen Mitteln der Aufbau eines solchen aus mobilen Geräten bestehenden Netzwerks möglich ist. Ein weiterer Wichtiger Punkt war die Umsetzbarkeit mit einem klassischen P2P Ansatzes zu testen. Als Herausforderung wurde dabei die Anpassung an die Gegebenheiten der mobilen Geräte angesehen.
Zudem bestand unserseits ein grosses Interesse in der Umsetzung eines P2P-Netzwerkes und dessen Nutzung als mobile *Cloud*, daher MobileCloud.

1.3 Ziel

Wesentliche Aspekte dieser Semesterarbeit waren die Machbarkeit und die Umsetzung in Form eines Prototyps. Natürlich mussten auch die Punkte wie Privatsphäre, Sicherheit und Transparenz beachtet werden, auch wenn diese bei diesen Prototypen nur ansatzweise implementiert wurden. Auf folgende Punkte wurde besonders geachtet.

Qualität: Die Qualität des Codes war ein sehr wichtiger Punkt, gerade weil auf dieses Projekt noch Folgeprojekte folgen könnten.

Effizienz: Bei jedem P2P-Netzwerk ist ein gewisser Verwaltungsaufwand nötig. Ein Ziel war es diesen mit geeigneten Protokollen und Techniken möglichst klein zu halten.

Performance: Die Software sollte in Zukunft möglichst unbemerkt im Hintergrund laufen. Deshalb sollte eine Software entstehen, welche nichtblockierende Aufrufe verwendet und eine Sinnvolle Anzahl Threads verwendet.

Redundanz: Es kann immer passieren, dass ein mobiles Gerät unverhofft ausfällt oder den Kontakt zum Netzwerk verliert, ohne sich korrekt abmelden zu können. Wichtig war aus diesem Grund Redundanz einzufügen, damit solche Probleme überbrückt werden können und sich das System möglichst schnell erholen kann.

Erweiterbarkeit: Eine Software wird immer erweitert, vor allem wenn sie in weiteren Projekten verwendet wird. Daher war es umso wichtiger, dass die Software einfach erweitert werden kann.

Punkte wie Reorganisation des Netzwerkes, eine hohe Fehlertoleranz oder auch die gesicherte Übertragung von Daten, wurde schon vor Beginn des Projektes als sekundär betrachtet und nicht in die Planung und Umsetzung miteinbezogen.

2 Begriffsdefinition

2.1.1 MobileCloud

Mit MobileCloud ist immer die gesamte *Cloud* bzw. das P2P-Netwerk gemeint.

Als MobileCloud Service werden die Dienste, welche von der MobileCloud zur Verfügung gestellt werden, bezeichnet. Zum Beispiel „Senden von Nachrichten“.

Android Service ist eine Android Applikation (oder einen Teil einer Applikation), welche im Hintergrund läuft und verschiedene Funktionen für andere Applikationen zur Verfügung stellt. [3]

MobileCloud Applikation ist die physische Applikation auf einem Gerät.

2.1.2 Nodes

Ein Node repräsentiert ein mobiles Gerät (Peer) in der MobileCloud. Ein Node hat eine Adresse, mit welcher er über das Netzwerk angesprochen werden kann. Jedes Gerät hat einen *Mainnode,* welcher das Gerät selbst repräsentiert und Nachbarnodes, welche andere Geräte in der MobileCloud repräsentieren.
Falls auf einem Gerät mehrere Instanzen von MobileCloud laufen kann es sein, dass dieses Gerät von mehreren Mainnodes repräsentiert wird.

2.1.3 Node-Adresse

Ein wichtiger Teil von MobileCloud sind die Node-Adressen. Diese beinhalten die IP, den Port sowie eine eineindeutige ID, welche das Gerät in der MobileCloud identifiziert. In der MobileCloud werden die Geräte nur über die ID angesprochen. Diese wird mit einer HASH-Funktion anhand der Telefonnummer berechnet. Die Adresse verbindet dann die ID mit der IP und dem Port. Will man also einem Node etwas senden, kann über die ID aus der Node-Adresse die IP und der Port abgefragt werden. Mit diesen Informationen ist das Gerät dann über das Netzwerk ansprechbar.
Die HASH-Funktion wird im Kapitel *3.3.8.1.1 Hash-Funktion* genauer beschrieben.

2.1.4 Begriffsdefinition

Für die einigen Beschreibungen, werden Beispiele anhand Grafiken beschrieben. In den Abbildungen werden folgende Symbole und Zeichen verwendet:

Symbol	Namen	Beschreibung
	***Mainnode***	Ein Node, welcher das aktuelle Gerät repräsentiert. Der Node hat die ID xx.
	**Node**	Ein Node, welcher nicht im *Leaf-Set* des *MainNodes* gespeichert ist. Der Node hat die ID zz.
	**Nachbarnode**	Ein Node, welcher im *Leaf-Set* des *MainNodes* gespeichert ist. Der Node hat die ID yy.
	**fehlerhafter Node**	Ein Node, welcher einen inkonsistenten Zustand auslöst, weil er noch in *Leaf-Sets* gespeichert jedoch nicht mehr in der MobileCloud ist oder weil er nicht mit den aktuellen Daten gespeichert wurde.
	**Daten**	Daten, welche versendet werden. Das können Verwaltungsdaten oder auch Bilder, Text, Sprache, usw. sein.
	Struktur	Grundlegende Struktur der zusammenhängenden Nodes.
	**Weg**	Route eines Nodes oder von Daten.

Tabelle 1 Symboltabelle

Solange nichts anderes angegeben wird, ist die maximale Anzahlt Nodes pro Seite drei.

3 Ergebnisse

3.1 Technologiewahl

An diese Stelle folg eine kurze Beschreibung der grundlegenden Technologien, welche für MobileCloud eingesetzt wurden.

3.1.1 Android

Android ist die Smartphone-Plattform, welche von der Open Handset Alliance entwickelt wird und auf mobilen Geräten wie Mobiletelefone oder Tablet-Computer zum Einsatz kommt. Android ist neben dem IOS von Apple und dem Blackberry OS eines der meistgenutzten Systeme auf dem Mobilemarkt [3]
Android steht unter der Apache 2.0 Lizenz und ist somit frei zugänglich. Deshalb sind auch viele Codebeispiele frei verfügbar. Dies kann beim Lösen von Problemen eine grosse Hilfe sein.

Aus folgenden Gründen wurde dieses Projekt auf Android realisiert:

- Grosse Community
  - Wie bereits oben erwähnt gibt es viele Codebeispiele und Tutorials
  - Gute Unterstützung in Foren
- Programmcode ist frei zugänglich
  - Für dieses Projekt ist eine offene Plattform von Vorteil, da alle Möglichkeiten des mobilen OS ausgenutzt werden können.
- Hardwarenahes Programmieren
  - Android bietet mit dem Native Development Kit (NDK) die Möglichkeit, hardwarenahe zu programmieren. Dies sollte allerdings nur nötig sein, wenn etwas dringend benötigt wird, vom Android SDK [1]jedoch nicht zu Verfügung gestellt wird.
- Beliebtes und sich schnelle verbreitendes Handy Betriebssystem
  - Dies ist für die Zukunft eines möglichen Produktes entscheidend.
- Implementierung in Java
  - Es ist kein Mehraufwand durch Lernen und Einarbeitung in eine Programmiersprache nötig. Einzig das Einarbeiten in den SDK von Android erfordert einige Einarbeitungszeit.

[1] Ein Software Development Kit (SDK) ist eine Sammlung von Anwendungen und Werkzeugen, die Hilfe bei der Softwareentwicklung bieten.

3.1.2 Cloud Computing

Cloud Computing bedeutet Abstraktion und Transparenz von IT-Infrastruktur. Dem Benutzer einer *Cloud* sollte es egal sein, was für eine Infrastruktur er schlussendlich benutzt und wo diese steht. Ihn sollte nur interessieren wie er den Service nutzen kann. Diese Abstraktion der Soft- von der Hardware ermöglicht auch eine sehr hohe Skalierbarkeit [4].
Im *Cloud Computing* gibt es grundsätzlich drei verschiedene Angebote:

- Infrastruktur als ein Service (IaaS): dem Benutzer wird Hardware-Ressourcen angeboten, welche er benutzen kann. Dies kann zum Beispiel ein virtueller Computer sein.
- Plattform als ein Service (PaaS): dem Benutzer werden vorinstallierte Computer-Plattformen wie z.B. Google App Engine zur Verfügung gestellt. Dieses Angebot wird hauptsächlich im Bereich Webanwendungen genutzt.
- Software als ein Service (SaaS): dem Benutzer werden vorinstallierte Programme angeboten. Er muss sich weder um die Software noch um die Hardware kümmern. Er benötigt nur ein internetfähiger PC. Ein Beispiel dafür ist Google Docs.

3.1.3 P2P Topologie

Um ein P2P-Netwerk zu realisieren, beziehungsweise die Virtuelle Netzwerktopologie aufzubauen, gibt es mehrere Möglichkeiten. Eines der weitverbreitetsten und performantesten P2P-Netzwerke ist *Pastry*. Es verwendet eine Ringtopologie für die logische Struktur und eine verteilte Hashtabelle für ein performantes Routing. Der Ring besteht aus verschiedenen Nodes, welche je ein physikalisches Gerät (*Peer*) darstellen und eine Adresse in Form eines Hashwertes haben. Zusätzlich hat jeder Node noch eine Routingtabelle, in welcher die Hashadresse und die IP einiger Nodes eingetragen sind. Durch diese Struktur ist ein Routing mit dem Erwartungswert $O(\log(n))$möglich [5].

Die meisten P2P-Netzwerke sind auf verteilte Dateiverwaltung, deren Indexierung und schnelle Suche im Netzwerk ausgelegt, so auch *Pastry*. Für MobileCloud wurde als Grundlage auch das *Pastry* Netzwerk verwendet. Da die Anforderungen von MobileCloud von denen einer verteilten Dateiverwaltung abweichen, wurde der *Pastry*-Ansatz angepasst.

3.2 ERSTER PROTOTYP

3.2.1 ZIEL UND FOKUS

Das Ziel bei diesem Prototyp war es, einen ersten Einblick in die Netzwerkprogrammierung und das Multithreading von Android zu bekommen. Es sollte eine einfache Umsetzung resultieren, damit der Fokus auf den Datenaustausch und nicht auf die *Cloud*-Verwaltung gelegt werden konnte. Dies sollte durch den Einsatz eines Servers realisiert werden.

Die wichtigsten Fragen, welche mit diesem Prototyp beantwortet werden sollten sind:

- Welches API von Android eignet sich für die Netzwerkprogrammierung am besten, java.net[2] oder java.nio[3]?
- Wie kann das Multithreading unter Android realisiert werden?
- Welches Protokoll eignet sich für die Verwaltung der Cloud am besten (TCP[4]/UDP[5])?
- Wie können Daten zwischen zwei Nodes einfach ausgetauscht werden?

3.2.2 ERKENNTNISSE

Android Netzwerk API: Für die Netzwerkkommunikation wurde das *Java NIO*[6] API verwendet. Mit *Java NIO* ist es möglich die Ein- und Ausgabe mit Hilfe von Puffern asynchron zu programmieren. Dies ermöglicht mit Multithreading eine einfachere Programmierung und somit eine nichtblockierende Implementierung. Gleichzeitig benutzt *Java NIO* die neusten Betriebssystemfunktionen, was auf den meisten Plattformen einen Geschwindigkeitsvorteil mit sich bringt. [6]

Android Multithreading: Um das Multithreading unter Android zu realisieren, gibt es wie in Java die zwei Möglichkeiten, entweder das Interface *Runnable* zu implementieren oder von *Thread* abzuleiten.
Das Ableiten von *Thread* ermöglicht das direkte Ausführen des Threads mit dem Aufruf der Methode *start*. Dabei wird der Overhead möglichst klein gehalten.

```
class MyClass extends Thread{
```

Muss man neben der Klasse Thread auch noch von einer weiteren Klasse ableiten, kann man anstelle von Thread das Interface *Runnable* verwenden. Dabei muss die Methode *run* überschrieben werden. Das Runnable-Objekt muss im Konstruktor eines Thread-Objektes mitgegeben werden, um es als eigenen Thread auszuführen.

```
class MyClass implements Runnable {
```

[2] Java.net ist das standard API von Java für die Netzwerkprogrammierung.
[3] Java.nio ist das neue API von Java für die Netzwerkprogrammierung und unterstützt asynchrone Umsetzungen.
[4] Das Transmission Control Protocol (TCP) ist ein ist ein zuverlässiges, verbindungsorientiertes und paketvermittelndes Transportprotokoll in Computernetzwerken.
[5] Das User Datagram Protocol (UDP), ist ein minimales, verbindungsloses Netzwerkprotokoll.
[6] Java New I/O ist ein Java API für die Verarbeitung von Ein- und Ausgaben

Um eine Operation auf einem separaten Thread auszuführen, können Android Handler verwendet werden. Diese erlauben es Message- und Runnable-Objekte an die *MessageQueue* eines Threads zu senden, welcher diese abarbeitet. Jeder Handler ist an einem Thread bzw. dessen *MessageQueue* gebunden und wird dem Thread, von welchem er instanziiert wurde, zugewiesen. [3]

Android GUI Thread: Beim Applikationsstart wird vom System der Main-Thread kreiert. Dieser ist für das GUI verantwortlich. Alle GUI-Änderungen sollten daher vom Main-Thread, auch UI-Thread genannt, ausgeführt werden. Um eine Operation auf dem UI-Thread auszuführen, können, wie oben erwähnt, Handler eingesetzt werden. Es gibt aber auch die Möglichkeit dem UI-Thread ein *Runnable* Objekt zu übergeben. Allerdings muss die aufrufende Klasse von *Activity* ableiten.

```
public final void runOnUiThread (Runnable myAction)
```

Leitet die aufrufende Klasse nicht von *Activity* sondern von *View* ab, kann dasselbe mit dem Aufruf *post* erreicht werden.

```
public boolean post (Runnable myAction)
```

Bei diesen drei Varianten muss beachtet werden, dass die Ausführungszeit einer Aktion auf dem Main-Thread möglichst kurz gehalten wird, um zu verhindern, dass das GUI „einfriert“ bzw. vom System eine *Application Not Responding (ARN)* Meldung erscheint.

Eine einfache und elegante Variante oben erwähnten Probleme gezielt aus den Weg zu gehen, ist von der generischen Klasse *AsyncTask* abzuleiten und die Methoden *doInBackground* und *onPostExecute* zu überschreiben. Zeitintensive Operationen werden mit *doInBackground* im Hintergrundthread erledigt. Wenn die Aufgabe erledigt ist, wird das Resultat der *onPostExecute* Methode übergeben. Die in dieser Methode definierten Operationen werden anschliessend auf dem UI Thread ausgeführt. [3]

```
private class MyDownloadFilesTask extends AsyncTask<URL, Integer, Long>
```

Zu verwendende Protokolle: Das meistverwendete Protokoll in P2P-Netwerken ist UDP. Dank seinem geringen Overhead ist es für den Austausch von kleinen Datenmengen sehr gut geeignet. Der grossen Nachteile ist, dass es keine Sicherheit gibt, dass die Daten beim Empfänger vollständig und in richtiger Reihenfolge angekommen.
Für die Semesterarbeit wird vorläufig UDP verwendet, es ist jedoch nicht ausgeschlossen, dass zu einem späteren Zeitpunkt gewisse Teile mit TCP realisiert werden.

Versenden von einfachen Daten zwischen Nodes: Das Versenden von kleinen Testdaten zwischen einzelnen mobilen Geräten hat über W-LAN ohne Probleme funktioniert. Es wurde ein Thread eingesetzt, welcher auf einem festgelegten UDP-Port horcht und sobald auf diesem Daten

hereinkommen, wurden diese an den GUI-Thread weitergeleitet. Dieser zeigt anschliessend die angekommen Daten dem Benutzer an. Dieses Vorgehen ist auch als Observer Pattern[7] bekannt.

3.2.3 Prototyp

Der Prototyp konnte mit dem geplanten Funktionsumfang in gegebener Zeit fertiggestellt werden. Da der Fokus nicht auf dem Endprodukt, sondern auf den Erkenntnissen lag, wurde auf ein Vorgehen mit Modellierung der Architektur sowie auf Unit Tests ganz verzichtet.
Dieser Prototyp ist im Source Verwaltungstool Git [8]mit dem Tag 0.1.0 gekennzeichnet und kann mittels des Befehles `git checkout 0.1.0` verfügbar gemacht werden.

Im Folgenden wird kurz der Entstandene Prototyp beschreiben.

[7] Pattern oder auf Deutsch Entwurfsmuster, sind bewährte Lösungsansätze für wiederkehrende Entwurfsprobleme.
[8] Git ist eine freie Software zur verteilten Versionsverwaltung von Daten.

3.2.3.1 Client

Der Client wurde mit einem minimalistischen GUI versehen, mit dem man auf einfach Art und Weise in der MobileCloud mit anderen Nodes interagieren kann.

Nachdem die Applikation gestartet ist, erscheint ein GUI. In diesem muss man zuerst die IP, Port und ID des Servers eingeben.

Solange man offline ist, zeigt die Wolke ein rotes Kreuz an  . Sobald durch drücken auf den Connect Button eine Verbindung zum Server aufgebaut wurde, ändert die Wolke ihr Aussehen  .

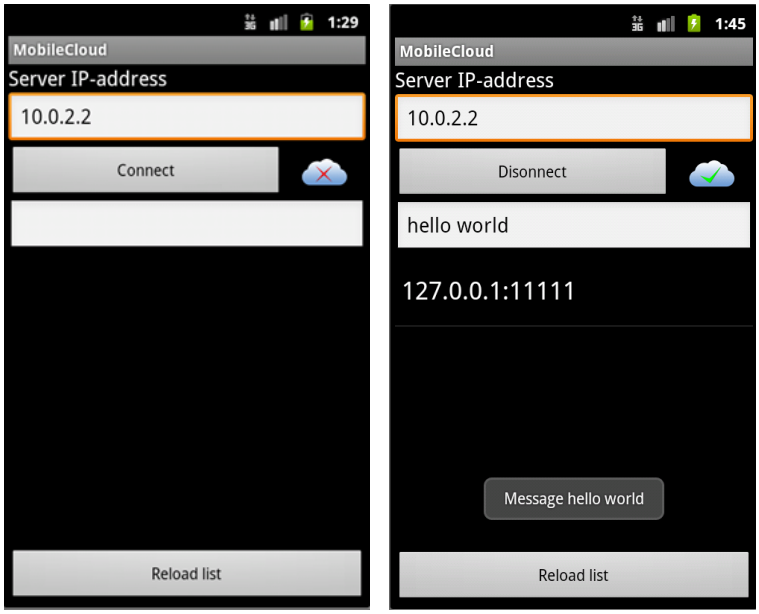


Abbildung 2 Prototyp 1

Sobald man eingeloggt ist, werden auf dem Bildschirm alle anderen Nodes aufgelistet, welche sich ebenfalls beim Server angemeldet haben. Durch Eingabe eines Textes in das leere Textfeld und anschliessendes Drücken auf die gewünschte Adresse, wird dem Node die Textnachricht gesendet. Dem Empfänger wird mittels Modaldialog die Nachricht angezeigt, wie das in der **Abbildung 2 Prototyp 1** zu sehen ist.

3.2.3.2 Server

Der Server wurde als Java Konsolenapplikation implementiert. Diese wartet bis sich ein Client meldet, speichert seine Adresse und sendet diesem alle gespeicherten Adressen zurück. Diese Aktivitäten werden anschliessend zur Übersicht auf der Konsole ausgegeben.

```
Tue Dec 13 13:13:33 CET 2011 Server: add /127.0.0.1:43425 as new Node
Tue Dec 13 13:13:34 CET 2011 Server: send Node /127.0.0.1:43425 the nodelist
```

3.3 Finaler Entwurf

3.3.1 Ziel und Fokus

Das Ziel bei diesem Prototyp war es, die ersten grundlegenden Funktionen eines P2P-Netwerkes zu implementieren. Die Implementation beschränkte sich dabei auf das *Leaf-Set* und dessen Aktualisierung beim Einfügen und Entfernen neuer Nodes. Als Eintrittspunkt sollte weiterhin ein Server dienen.

Ferner sollte eine flexible beziehungsweise einfach erweiterbare Grundarchitektur erstellt werden, damit weitere auf diesem Prototyp aufbauende Versionen entwickelt werden können. Auch sollte ein Protokoll festgelegt werden, um die Verwaltungs- und Testdaten auszutauschen.

Bei der Netzwerkprogrammierung sollte die Grundfunktionalität implementiert werden. Verbindungsprobleme durch Firewalls, NATs, Paketverluste und mögliche Lösungsansetze sollten beim Entwickeln der Architektur beachtet werden. Die Implementation dieser Lösungen gehörte aber nicht zu den Zielen.

Die wichtigsten Funktionalitäten, welche mit diesem Prototyp implementiert werden sollten, sind:

- Implementation des *Leaf-Sets* und dessen Grundfunktionalität
- Flexible Grundarchitektur entwerfen und implementieren
- Erstellen einer virtuellen Adresse
- Einfaches Routing einer Anfrage in der MobileCloud
- Erstellen eines Java Clients, um die *Cloud* für Testzwecke „künstlich“ zu vergrössern
- Stabile Netzwerkarchitektur

3.3.2 Erkenntnisse

Implementation des *Leaf-Sets*: Zuerst wurde als Grundlage ein Array verwendet, um die Nachbarschaft im *Leaf-Set* abzubilden. Durch die aufwändige Implementierung, wurde schlussendlich eine doppelt verkettete Liste verwendet. Dies ermöglicht ein einmaliges Einfügen und Entfernen der Nodes und es kann auf die wiederkehrenden Verschiebungen innerhalb des Array verzichtet werden. Durch die Kapselung der Node-Adressen in eigenen Objekten mit vorwärts- und Rückwärtszeiger, wird mehr Speicherplatz benötigt.

Flexible Grundarchitektur: Für die Architektur wurden zwei Schichten festgelegt. Diese Architektur macht es möglich, die Logik und Funktionsweise des P2P-Netzes unabhängig der darunterliegenden Netzwerk-Schicht zu erweitern. Somit können zum Beispiel auch andere Algorithmen für ein effizienteres Routing als Service implementiert werden.

Festlegung eines Protokolls: Es wurde ein möglichst einfaches und kleines Protokoll festgelegt. Das Simple Peer-to-Peer Management Protokoll (SPMP) ermöglicht es einkommende Pakete einem Service in richtiger Reihenfolge zuzuordnen und anzugeben, ob es eine Anfrage oder eine fehlerhafte- oder korrekte Antwort ist. Alles andere wird in der Service-Schicht von den Handlern selbst abgehandelt.

Erstellen einer virtuellen Adresse: Das Erstellen einer ID mittels einer Hash-Funktion wurde nur provisorisch implementiert. Die endgültige Implementation wird zu einem späteren Zeitpunkt erfolgen.
Es wurden jedoch die Anforderungen an diese Funktion genau spezifizier und im Kapitel *3.3.8.1.1 Hash-Funktion* festgehalten.

Einfaches Routing einer Anfrage in der Cloud: Das Routing wurde mit dem *Leaf-Set* auf einfachste Weise umgesetzt. Wenn die Adresse des Zielnodes nicht bekannt ist, wird die Nachricht einfach an den Nachbarnode ganz rechts im *Leaf-Set* weitergeleitet. Dies bedeutet eine Laufzeit von $O(n)$, was für Testzwecke vollkommen ausreicht.

Erstellen eines Java Clients, um die Cloud „künstlich“ zu vergrössern: Der vorhandene MobileCloud-Code wurde fast vollständig übernommen und auf die Java-Plattform angepasst. Der Java Client besitzt keine grafische Benutzeroberfläche und kann deshalb nur über die Konsole gestartet werden. Er verhält sich wie ein normaler Node, nur dass er auf einem PC mit Java läuft.
Es wurden zwei Konfigurationsmöglichkeiten implementiert. Bei der ersten wird der Java Client ohne zusätzliche Konsolenargumente gestartet. Die nötigen Parameter können anschliessend interaktiv eingegeben werden.
Bei der zweiten Möglichkeit werden gleich zu Beginn alle nötigen Parameter als Konsolenargumente angegeben. Dies wird hauptsächlich verwendet, um automatisiert eine grössere MobileCloud aufzubauen.
Der Java Client ist nur für Testzwecke ausgelegt und wurde nicht separat getestet.

Stabile Netzwerkarchitektur: Mit Hilfe von *Java NIO* konnte eine nicht blockierende Architektur programmiert werden. Es wird dasselbe Socket für die gesamte Kommunikation benutzt. Dieses ist durch *Java NIO* von Mehrfachzugriffe geschützt. Es kann daher über denselben Port gesendet sowie empfangen werden.

Es gibt ein Thread, welcher Nachrichten empfängt. Für die Abarbeitung der eingetroffenen Nachrichten werden separate Threads verwendet.

3.3.3 Design und Architektur

3.3.3.1 Architektonische Ziele und Einschränkungen

Aus den Zielen des Prototyps und aus der Anforderungsspezifikation konnten folgende Ziele für die Architektur entnommen werden.

Erweiterbarkeit

Ein wichtiges Ziel ist, dass neue Services ohne grossen Aufwand hinzugefügt werden können. Eine Erweiterung soll keine Anpassung an der bestehenden Architektur mit sich ziehen.

Ferner soll die Möglichkeiten der Parallelisierung genutzt werden. Die Anzahl an benutzten Threads soll änderbar sein.

Mobilität

Smartphones bieten zwar mehr Leistung als übliche Handys, haben jedoch dementsprechend einen grösserer Akkuverbrauch. Wenn möglich soll keine Rechenzeit verschwendet werden und mittels Threads eine möglichst performante Architektur entstehen.

3.3.3.2 Aufbau der Projektes

Die Semesterarbeit MobileCloud besteht aus drei Projekten.

MobileCloud
Ist das Hauptprojekt, sprich die Android Applikation. Dieses beinhaltet den gesamten Code, welcher für das Erstellen von MobileCloud nötig ist, inklusive des Android GUI's.

MobileCloudServer
Beinhaltet den MobileCloud Server. Es beinhaltet die Anbindung der Grundfunktionalitäten vom Projekt MobileCloud an die Konsole.
Dieses Projekt benötigt das MobileCloud-Projekt.

MobileCloudTest
Deses Projekt beinhaltet alle Unit Tests für das Projekt MobileCloud. Die Unit Test werden bei Android immer in einem separaten Projekt erstellt.
Dieses Projekt benötigt das MobileCloud- Projekt.

3.3.3.3 Einführung

Die MobileCloud Applikation wird mit einer multithreaded Peer-To-Peer-Architektur realisiert.

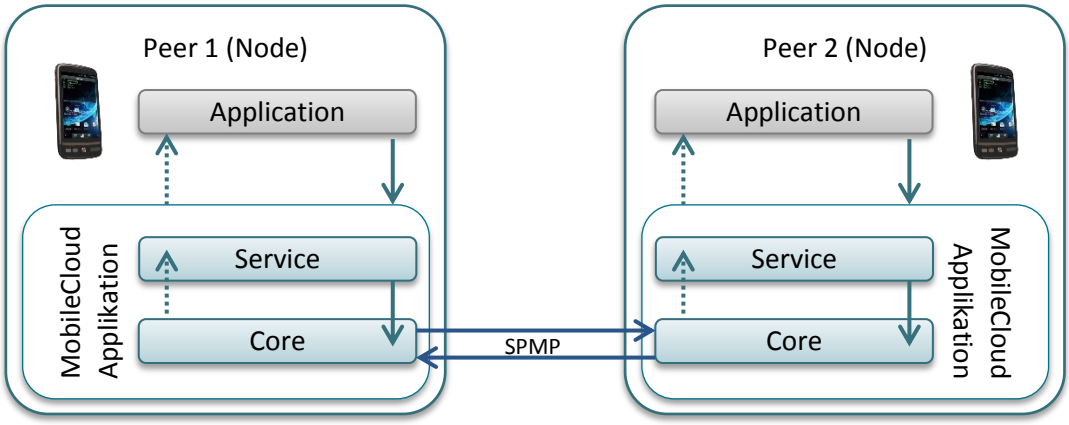


Abbildung 3 Architekturübersicht mit 2 Peers

Abbildung 3 Architekturübersicht mit 2 Peers gibt eine Übersicht über die Architektur. Es wird über das eigens dafür entwickelte Protokoll SPMP kommuniziert. Dieses wird im Kapitel *3.3.6 SPMP Protokoll* genau beschrieben. Die Schichten *Service* und *Core* gehören zur MobileCloud Applikation und die *Application*-Schicht soll eine separate Applikation darstellen, welche die MobileCloud nutzt. Die genaue Beschreibung zu den Schichten ist im Kapitel *3.3.3.5 Logische Sicht* zu finden.

Einfacher Ablauf mit einer Anfrage und Antwort als Beispiel:

1. Applikation initiiert eine Service-Funktionalität.
2. Der Service führt entsprechende Vorbereitungen aus und übergibt dem Netzwerk in der Core Schicht die zu versendende Daten.
3. Das Netzwerk erstellt eine SPMP-Anfrage-Nachricht mit den Daten und sendet diese zum Empfänger.
4. Die Nachricht wird beim Empfänger (Peer 2) beim Netzwerk empfangen und die Daten werden aus der Nachricht entnommen. Das Netzwerk meldet den Nachrichteneingang anschliessend per Callback an die Service-Schicht.
5. Die Service-Schicht bearbeitet die Daten und benachrichtigt gegebenenfalls die Applikation. Danach wird von der Service-Schicht das Resultat der abgearbeiteten Anfrage an die Core-Schicht übergeben.
6. Die Daten werden in einer SPMP-Antwort-Nachricht verpackt und zurückgeschickt.
7. Beim Sender (Peer 1) wird die Antwort-Nachricht beim Netzwerk empfangen und der Nachrichteneingang an die Service-Schicht mitgeteilt.
8. Die Service-Schicht leitet das Resultat der Service-Funktionalität an die Applikation weiter.

Es sollen mehrere Applikationen auf einem Geräte die MobileCloud nutzen können. Da es hier hauptsächlich, um die MobileCloud und nicht diese Applikationen geht, wird nur die Architektur der MobileCloud Applikation beschrieben. Die zu Testzwecke entwickelte externe Applikation beziehungsweise Benutzeroberfläche zur Nutzung der MobileCloud wird nicht beschrieben.

3.3.3.4 Architekturentscheid

Anhand der architektonische Ziele und Anforderungen der Arbeit wurde eine multithreaded P2P-Architektur festgelegt.

Mutlithreading ist nicht nur aus den oben erwähnten architektonische Ziele nötig, sondern auch aus Effizienzgründen. Dies ermöglicht, dass Services parallel abgehandelt werden können. Die asynchrone Kommunikation hilft dabei die Möglichkeiten der Parallelisierung noch weiter auszunutzen. Genauere Details zu Multithreading sind im Kapitel *3.3.4 Multithreading* zu finden.

3.3.3.5 Logische Sicht

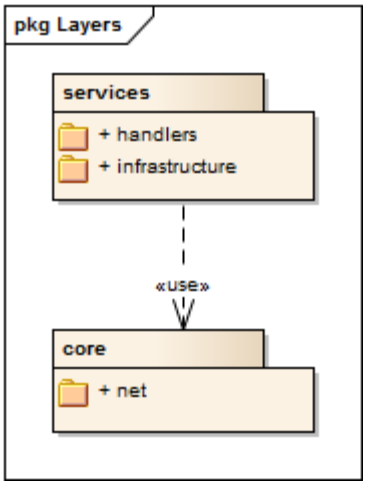


Abbildung 4 Layers von MobileCloud

Die MobileCloud Applikation ist in 2 Schichten aufgeteilt. Wie aus Abbildung 4 Layers von MobileCloud ersichtlich ist, verlaufen die Abhängigkeiten von oben nach unten. Die Kommunikation nach oben wird mittels Callbacks realisiert.

Services

Dieses Paket enthält die Logik zur Verwaltung der MobileCloud und die Services, welche darüber angeboten werden. Zudem befindet sich hier die Datenstruktur zur Abbildung des P2P-Netzwerks bzw. der verbundenen Peers.

Core

Dieses Paket enthält alle Netzwerkrelevante Informationen und ist für die Datenübertragung verantwortlich. Sie kapselt die verwendete Netzwerktechnologie sowie Serialisierung und Deserialisierung.

3.3.3.6 Architekturkonzepte

3.3.3.6.1 Exception Handling

Grundsätzlich soll die Abhandlung der Exceptions so tief wie möglich erfolgen. Das heisst, dass die erstmögliche Klasse, welche die Exception behandeln kann, sich darum kümmern soll. Wird eine Exception an die höhere Schicht geworfen, dann nur, wenn dies wirklich ein Ausnahmefehler ist und von der werfenden Schicht nicht behandelt werden kann.

Jede gefangene Exception wird mit dem Ort des Auftretens und einer Meldung geloggt.

Jede Schicht, welche eigene Exceptions verwendet, besitzt für diesen Zweck ein eigenes Paket *exceptions*.

3.3.3.6.2 Logging

Als Logging-Mechanismus wird der von Android zur Verfügung gestellten Logger verwendet. Mit diesem ist es möglich unterschiedliche Meldungen mit Tags und Logtyp abzuspeichern. Es stehen folgende Log-Typen zur Verfügung [3]:

- ASSERT
- DEBUG (Wird immer kompiliert, aber nur im Debuggingmodus aufgezeichnet)
- ERROR
- INFO
- VERBOSE (Wird nur im Debuggingmodus kompiliert)
- WARN

Alle gefangene Exceptions werden mit ERROR mitgeloggt.

3.3.3.7 Beschreibung der Pakete

In diesem Kapitel werden die Pakete genauer beschrieben. Die genauen Beschreibungen der Klassen und Methoden befinden sich im JavaDoc.

3.3.3.7.1 Paket core

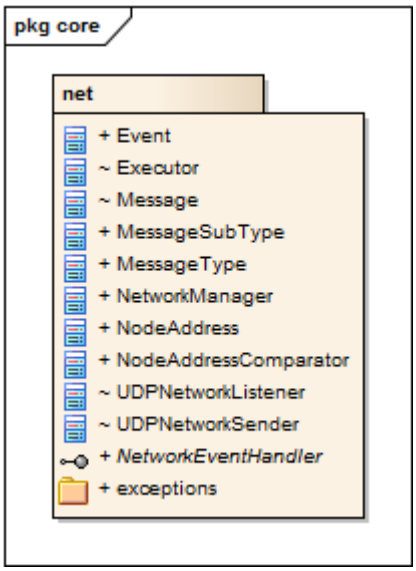


Abbildung 5 Übersicht core

core.net

Dieses Paket beinhaltet die gesamte Netzwerklogik. Es definiert ein Interface für die Implementation von NetzwerkHandler, welche die Daten einer empfangenen Nachricht erhalten und weiterverarbeiten. Es ermöglicht logischerweise ebenfalls das Versenden von Nachrichten.

KLASSENDIAGRAMM

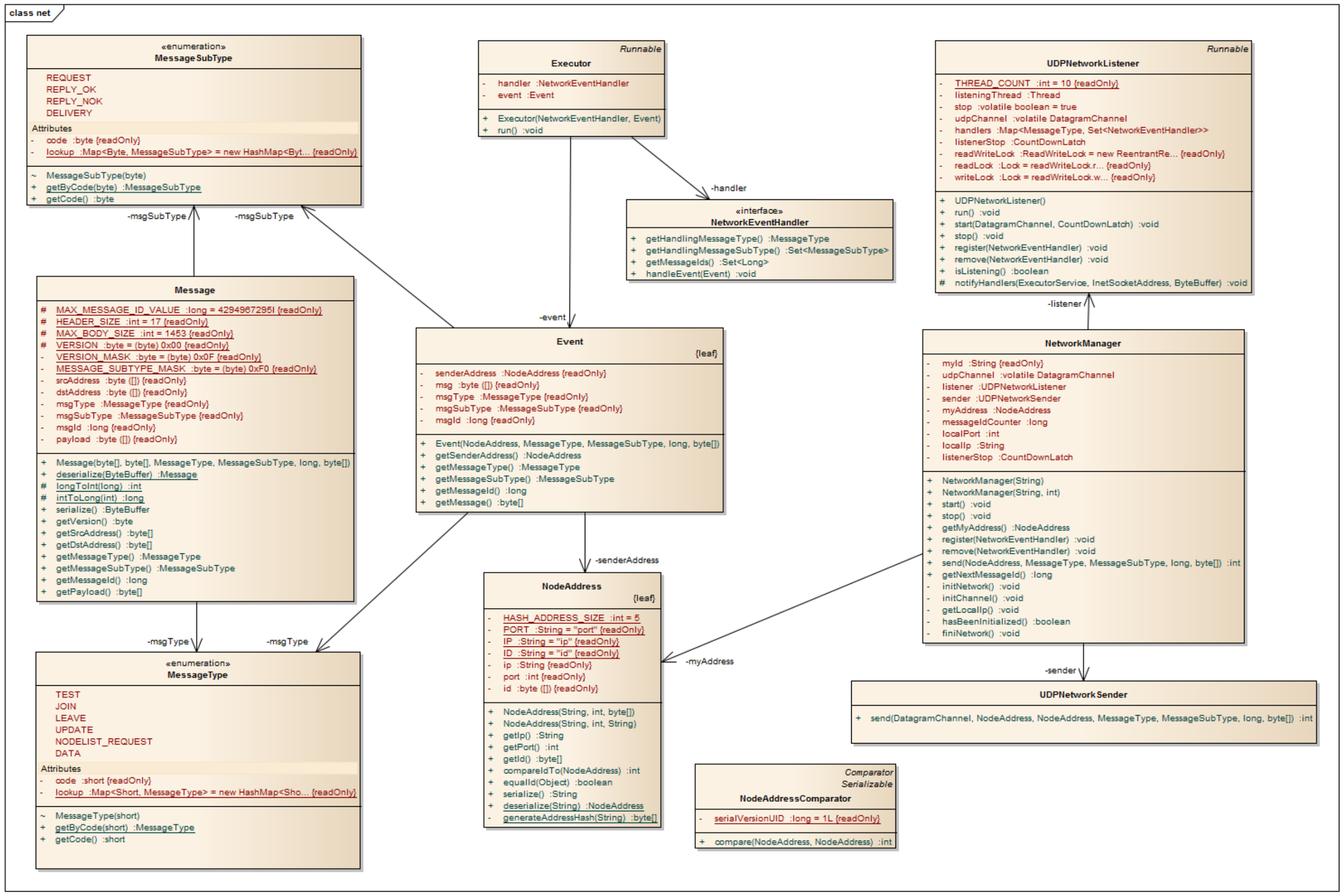


ABBILDUNG 6 KLASSENDIAGRAMM CORE.NET

Klassen

NetworkManager

Diese Klasse stellt der oberen Schicht alle Funktionalitäten des Netzwerks zur Verfügung. Sie kapselt die Klassen, welche für das Senden, Empfangen und Serialisieren/Deserialisieren der Nachrichten verantwortlich sind. Des Weiteren speichert sie Netzwerkdaten und die Node-Adresse des lokalen Nodes.

NetworkEventHandler

Durch Implementieren dieses Interface können Handler erstellt werden, welche nach Registrierung beim *NetworkManager* Events verarbeiten können. Ein Event ist dabei der Eingang einer Nachricht. Die *handleEvent*-Methode wird für die Verarbeitung des Events aufgerufen. Mittels der Methoden *getHandlingMessageTyp*, *getHandlingMessageSubTyp* und *getMessageId* kann angegeben werden, welche Nachrichten der Handler verarbeitet.

MessageType, MessageSubType

Diese Enum definieren den Typ der Nachricht und sind daher in jeder Nachricht enthalten. Sie werden für das Selektieren der Handler für die Event-Verarbeitung benötigt.

Event

Diese Klasse enthält die Daten einer Netzwerknachricht und wird den Handler übergeben.

NodeAddress

Diese Klasse wird für die Adressierung von Nodes verwendet.

UDPNetworkListener

Diese Klasse wird für das Empfangen und benachrichtigen der Handler verwendet.

UDPNetworkSender

Diese Klasse wird für das Senden von Nachrichten verwendet.

Executor

Diese Klasse implementiert das Interface *Runnable* und wird dem Thread-Pool übergeben, um die *Event*-Verarbeitungsmethode eines Handler auszuführen. Es ruft in der *run*-Methode die *handleEvent*-Methode des Handler auf.

Message

Diese Klasse wird für die Serialisierung und Deserialisierung der Nachrichten verwendet.

Exceptions

NetworkException

Diese Exception wird für Ausnahmefehler, welche im core.net Paket auftreten und an die obere Schicht weitergeleitet werden müssen, benutzt.

Anmekrungen

Asynchronous Completion Token Pattern

Es wird ein Asynchronous Completion Token (ACT) verwendet, damit beim Eingang einer Antwort-Nachricht der richtige Handler zum verarbeiten dieser Antwort aufgerufen wird.

Komponenten:

- Initiator = *NetworkManager* mit dem *UDPNetworkListener* für das Empfangen und demultiplexen und dem *UDPNetworkSender* für das Senden.
- Completion Handler = alle Klasse, welche das Interface *NetworkEventHandler* implementieren.
- ACT = Zusammensetzung des *MessageTypes*, *MessageSubTypes* und der *MessageId*.
- Service = Node, an welchem die Anfrage geschickt wird.

Facade

Damit das Netzwerk möglichst abgekapselt ist, dient der NetzwerkManager als Fassade des core.net Packet.

3.3.3.7.2 Paket services

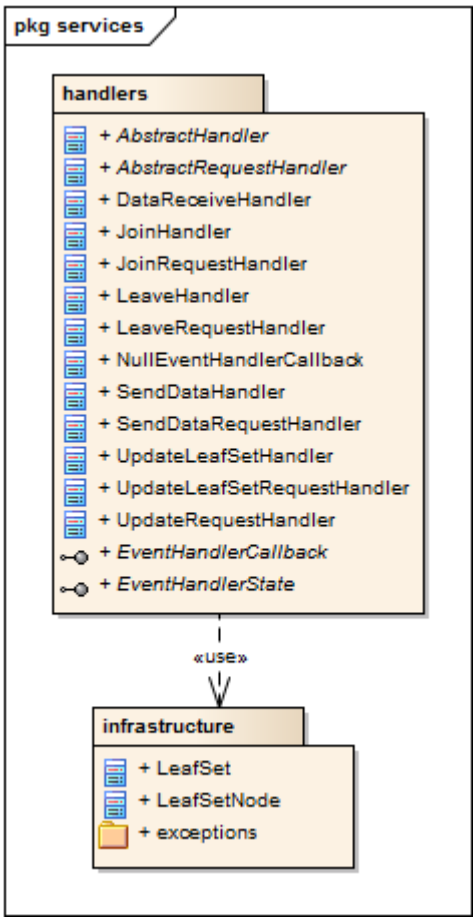
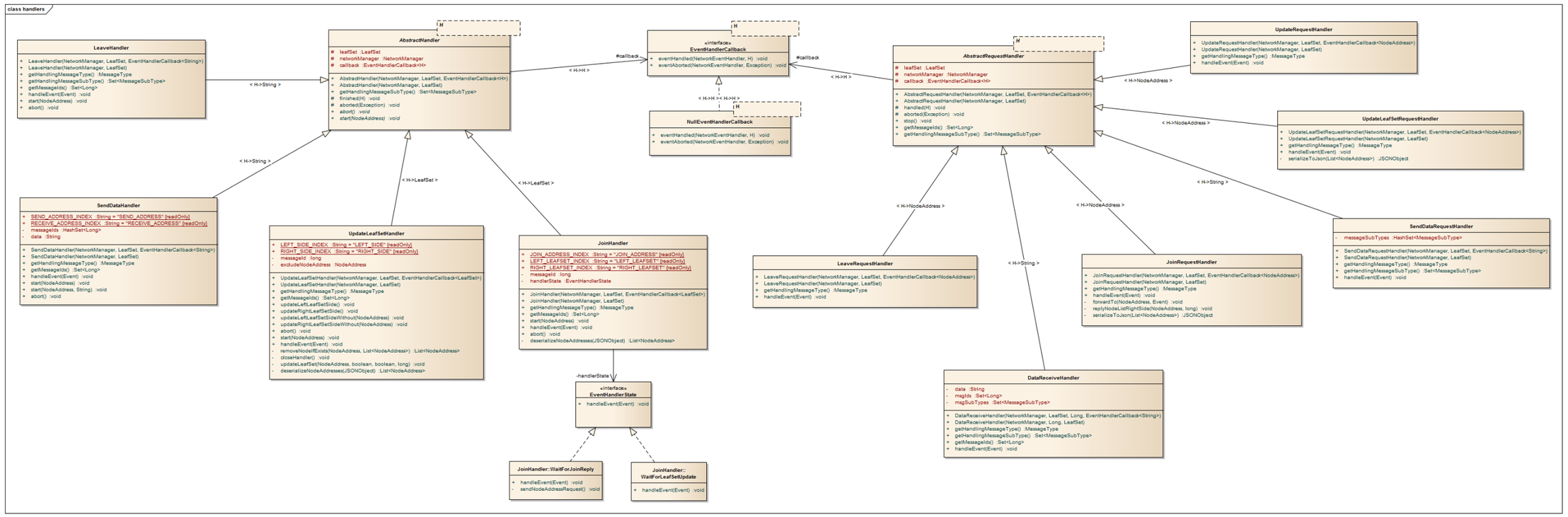


Abbildung 7 Übersicht services

Paket services.handlers

Dieses Paket beinhaltet alle Handler, welche die unterschiedlichen Services implementieren. Momentan gibt es nur Handler für die Verwaltung der MobileCloud und 3 Handler für das Senden von Textdaten an einen bestimmten Node. Für die genaue Beschreibung der Algorithmen und die Aufgaben der einzelnen Handler siehe *3.3.8 Algorithmen*.

KLASSENDIAGRAMM



Klassen

Alle vorhandene Handler haben Gemeinsamkeiten, welche in die abstrakten Klassen *AbstractHandler* und *AbstractRequestHandler* zusammengeführt wurden. Dazu gehört unter anderem die Möglichkeit einen Callback zu hinterlegen, um andere Software-Komponenten über bestimmte Ereignisse informieren zu können.

Da bestimmte Handler je nach Zustand ein anderes Verhalten bei der Verarbeitung eines Events aufweisen, wird zu diesem Zweck das State Pattern verwendet.

AbstractHandler

Diese abstrakte Klasse ist für Handler gedacht, welche ein Service initiieren und abhandeln.

AbstractRequestHandler

Diese abstrakte Klasse ist für Handler gedacht, welche auf Service-Anfragen reagieren.

EventHandlerCallback

Dieses Interface muss von den Klassen implementiert werden, welche als Callback bei den Handler hinterlegt werden.

EventHandlerState

Dieses Interface muss von den Klassen implementiert werden, welche das Verhalten für einen bestimmten Zustand eines Handler definieren.

NullEventHandlerCallback

Dies ist ein Null-Objekt und bei den Handler standardmässig als Callback hinterlegt. So entfallen allfällige Prüfungen auf *null*, wenn kein konkreter Callback hinterlegt wurde.

JoinHander, JoinRequestHandler

Diese Klassen kümmern sich um das Beitreten bzw. der Beitritt anderen Nodes zur MobileCloud.

LeaveHandler, LeaveRequestHandler

Diese Klassen verarbeiten das Verlassen der MobileCloud.

UpdateLeafSetHandler, UpdateLeafSetRequestHandler

Diese Klassen verarbeiten das Erneuern des *LeafSets*.

UpdateRequestHandler

Diese Klasse verarbeitet einen neuen angemeldeten Node.

SendDataHandler, SendDataRequestHandler, DataReceiveHandler

Diese Klassen sind für das Senden und Empfangen von Daten zuständig.

Anmerkungen

State Pattern

Einige der Handler haben ein statusabhängiges Verhalten. Diese wurde mit dem State Pattern realisiert. Jeder Status beschreibt seine Aktion und in welchen Status er nach der Bearbeitung wechseln wird.
Durch dieses Pattern konnte die Übersicht dadurch erhöht werden, dass eine geringere Kopplung erreicht wurde und die Klassennamen der verschiedenen States schon viel über dessen Funktion sagen.

Paket services.infrastructure

Dieses Paket beinhaltet die Ring-Datenstruktur-Klassen.

Klassendiagramm

Abbildung 8 Klassendiagramm services.infrastructure

KLASSEN

LEAFSET

Diese Klasse stellt das *LeafSet* dar, welches alle Nachbar-Nodes beinhaltet. Diese werden in einer *double linked-list* gespeichert. Die genaue Beschreibung dieser Datenstruktur ist im Kapitel *3.3.7.1 Leaf-Set* zu finden.

LEAFSETNODE

Diese Klasse stellt ein Node innerhalbe der *double linked-list* in der LeafSet-Klasse dar.

3.3.3.7.3 SCHNITTSTELLENBESCHREIBUNG

Um eine vollständige Beschreibung aller Interfaces und Klassen zu erhalten, würde ein JavaDoc erstellt.

3.3.4 Multithreading

3.3.4.1 Übersicht

Abbildung 9 Threads innerhalb der MobileCloud Applikation soll kurz eine Übersicht über vorhandene Threads geben.

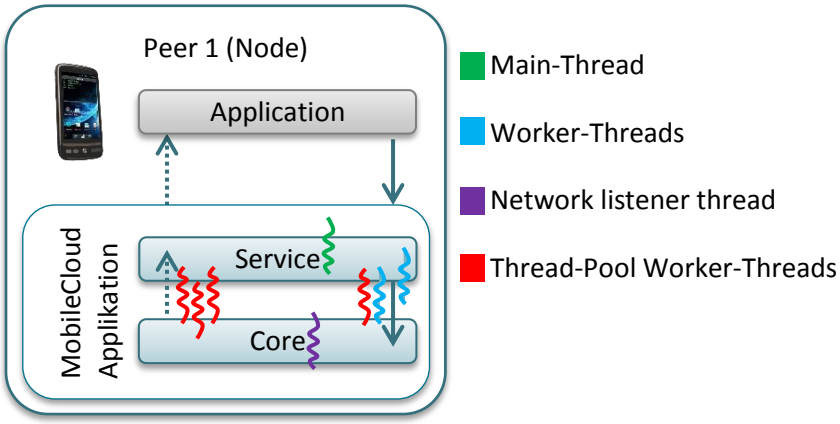


Abbildung 9 Threads innerhalb der MobileCloud Applikation

Die MobileCloud Applikation soll als Android Service in einem eigenständigen Prozess laufen. Aus Testing- und Zeitgründen wurde bisher darauf verzichtet. Nichtsdestotrotz kann die MobileCloud Applikation als eigenständiger Prozess betrachtet werden. Da sich der Main-Thread einer Android Applikation gleich verhält wie derjenige eines Android Service. Im folgenden Kapitel wird dies genauer erläutert.

3.3.4.1.1 Beschreibung der verschiedenen Thread-Typen

In diesem Kapitel folg eine Beschreibung der Thread-Typen, welche in der MobileCloud Applikation vorkommen.

Main-Thread (Grün)

In Android gibt es für jeden Prozess genau ein Main-Thread. Dieser ist bei einer Applikation für das GUI verantwortlich. Der Main-Thread darf in diesem Fall nie blockiert sein, damit das GUI stets auf Benutzerinteraktionen reagieren kann. Die rechenintensive Aufgabe müssen daher an Worker-Threads delegiert werden. Bei einem Service sieht das gleich aus. Der Main-Thread ist für die Interaktion mit den Applikationen verantwortlich und delegiert rechenintensive Arbeiten an Worker-Threads [3]. Der Main-Thread gelangt nie in die Core-Schicht.

Worker-Threads (Blau)

Wie bereits erwähnt, werden für rechenintensive Aufgabe separate Threads verwendet. Diese werden ausschliesslich vom Main-Thread erstellt und gestartet.

In der Regel besteht ihre Arbeit dadurch, einen MobileCloud Service zu initiieren. Dies könnte beispielsweise das Senden einer Nachricht sein. Da es sich um eine asynchrone Kommunikation handelt und in der Regel jeder Service mit dem Versand einer Nachricht startet, gelangt jeder Worker-Thread von der Service-Schicht bis in die Core-Schicht. Nach dem Versand der ersten Nachricht wird der Worker-Thread beendet. Der Service ist jedoch nicht fertig abgearbeitet. Darum

wird hier auch vom „Initiieren“ des Service gesprochen. Beim Auftreten von Fehlern kann es vorkommen, dass der Worker-Thread gar nicht bis in die Core-Schicht gelangt. Er kann aber auch über Callback von der Core-Schicht wieder in die Service-Schicht gelangen.

Es können mehrere Worker-Threads parallel vorkommen, wenn viele Interaktionen zwischen den Applikationen und dem Android-MobileCloud-Service vorkommen.

NETWORK LISTENER THREAD (VIOLETT)

Dieser Thread wird beim initialisieren des Netzwerks automatisch gestartet. Dies geschieht beim ersten MobileCloud Service-Aufruf. Er läuft solange bis er durch ein Interrupt beim Schliessen des Netzwerks beendet wird. Dies geschieht, wenn die MobileCloud explizit verlassen wird. Er beendet auch durch Fehler auf Netzwerk-Ebene wie beispielsweise das abrupte Schliessen des Netzwerk-Channels. Dieser Thread kommt genau einmal vor und gelangt nie ausserhalb der Core-Schicht.

THREAD-POOL WORKER-THREADS (ROT)

Diese Threads werden von einem Thread-Pool verwaltet. Dieser wird wiederum von Network Listener Thread gesteuert. Sobald eine Netzwerk-Nachricht eintrifft, übergibt der Netzwerk-Listener dem Thread-Pool pro Handler, welcher sich für diese Nachricht interessiert, ein *Executor*-Objekt. Die Thread-Pool Worker-Threads führen anschliessend die *run*-Methode des zugewiesenen Executor-Objekts aus. Die Anzahl an Threads im Thread-Pool ist änderbar.

Diese Threads führen die *handleEvent*-Methoden der Handler aus und je nach dessen Implementation noch Operationen auf dem Netzwerk aus. Somit sind diese Threads auf beide Schichten, Service und Core, verteilt. Je nach maximale Anzahl an Threads und eintreffen von Nachrichten können mehrere Threads parallel vorkommen.

3.3.4.2 THREAD-SICHERHEIT

3.3.4.2.1 STANDARD-SZENARIO

Folgendes Standard-Szenario soll grob aufzeigen, wie 2 Worker-Threads von der Service- in die Core-Schicht gelangen und 2 Thread-Pool Worker-Threads von der Core- in die Service-Schicht.

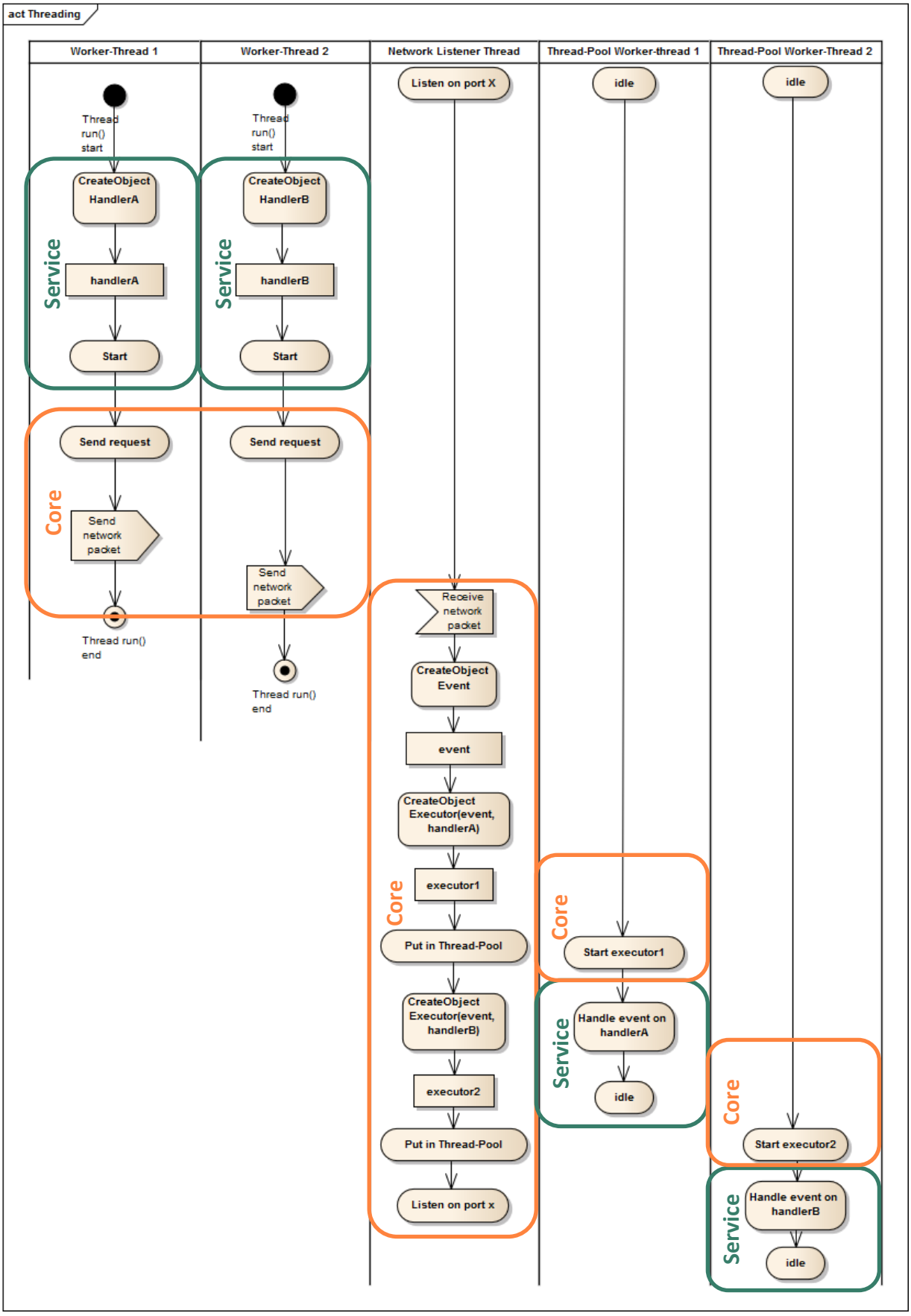


Abbildung 10 Threading Beispiel

Wie dieses Szenario zeigt, arbeiten die Worker-Threads am Anfang auf Service-Ebene mit eigenen Objekten und brauchen anschliessend gemeinsame Objekte der Core-Schicht. Diese sind threadsicher. Deswegen erfolgt das effektive Senden der Nachricht nacheinander und nicht gleichzeitig. Der Network Listener-Thread arbeitet ausschliesslich in der Core-Schicht. Die Thread-Pool Worker-Threads arbeiten mit einem Executor-Objekt (Core-Schicht) und anschliessend mit einem Handler-Objekt. Es ist durchaus möglich, dass bei der Event-Verarbeitungsmethode der Handler wieder gemeinsame Objekte der Core-Schicht verwendet werden, um weitere Nachrichten zu senden.

Ein weiteres Szenario kann sein, dass die Thread-Pool Worker-Threads gleichzeitig mit demselben Handler-Objekt arbeiten. Wie damit umgegangen wird, ist im folgenden Kapitel beschrieben.

3.3.4.2.2 Von Threads gemeinsam genutzte Objekte

Alle in den folgenden Unterkapiteln aufgeführten Klassen sind vom Multithreading betroffen und müssen threadsicher sein. Alle anderen sind davon nicht betroffen.

Service-Schicht

LeafSet

Es gibt ein einziges Objekt dieser Klasse innerhalb der gesamten MobileCloud Applikation. Dieses wird von verschiedenen Handler, welche in unterschiedlichen Threads laufen, bearbeitet. Zustandsänderungen dieser Klassen werden mittels Locks gesichert. Um das Threading zu optimieren werden beim Lesen Read- und beim Schreiben Write-Locks verwendet.

Handler

Wie das Standard-Szenario zeigt, werden Handler-Objekte von unterschiedlichen Threads bearbeitet. Folgende Abbildung soll noch genauer aufzeigen, wie die Events von den unterschiedlichen Handler mittels mehreren Threads parallel verarbeitet werden.

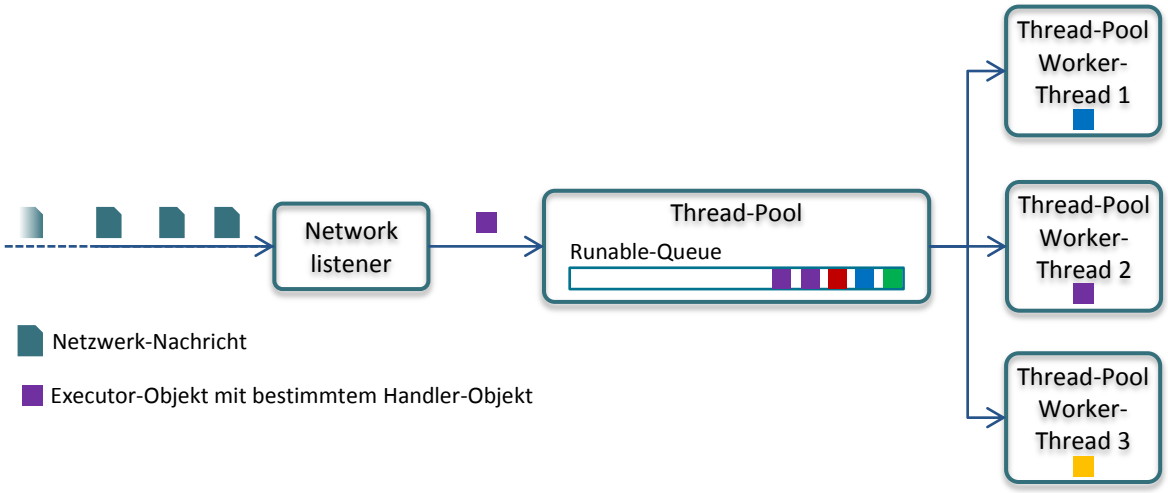


Abbildung 11 Abarbeiten von Events

Die kleinen farbigen Quadrate stellen Executor-Objekte dar, welche alle je ein Handler-Objekt beinhalten. Die Farben geben an, um welches Handler-Objekt es sich handelt.

Wenn mehrere Netzwerk-Nachrichten für ein bestimmtes Handler-Objekt nacheinander eintreffen, werden demzufolge mehrere Thread-Pool Worker-Thread die Event-Verarbeitungsmethode dieses Objekt ausführen wollen.

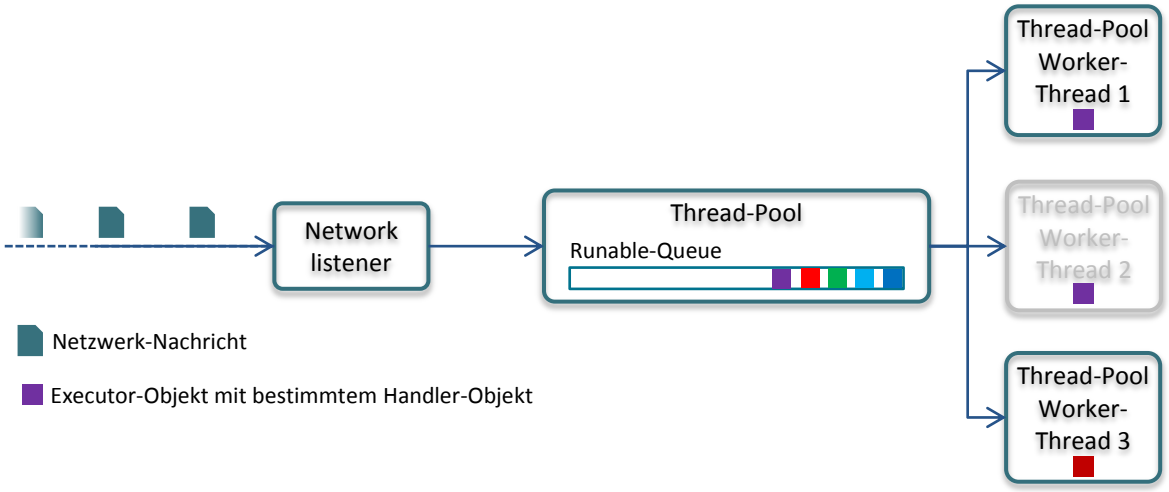


Abbildung 12 Paralleles Ausführen desselben Handler-Objekts

Damit keine Dateninkonsistenz innerhalb der Handler-Objekte entsteht, wurde entschieden, dass ein Handler nicht mehrere Events gleichzeitig verarbeiten können. Sprich die *handleEvent*-Methode desselben Handler-Objekts kann nicht gleichzeitig von mehreren Thread-Pool Worker-Thread ausgeführt werden. Dies veranschaulicht **Abbildung 12 Paralleles Ausführen desselben Handler-Objekts.** Dies wurde mit *Locking* auf dem Handler-Objekt realisiert. Somit sind alle Handler, was diese Threads angeht, threadsicher.

Was die Worker-Threads angeht, sieht es je nach Handlertyp unterschiedlich aus. RequestHandlers sind threadsicher, da diese grundsätzlich Zustandslos sind und nach dem Start nicht mehr von diesen Threads benutzt werden. Dann führen nur noch Thread-Pool Worker-Threads Operationen auf diesen Objekten aus. Falls ein Zustand gespeichert werden muss, werden bei den RequestHandlers separate Handler-Objekte erstellt, welche sich um diese fortführende Arbeit kümmern. Diese werden wiederum ausschliesslich von den Thread-Pool Worker-Thread benutzt und sind somit threadsicher.

Bei allen anderen Handler, in diesem Fall *Join*- und *SendDataHandler*, muss separat darauf geachtet werden, dass diese nicht von den Worker-Threads und Thread-Pool Worker-Threads parallel bearbeitet werden.

Core-Schicht

Um Synchronisationsmechanismen für Objekte zu vermeiden, wurde unteranderem das *Immutable-Object* Pattern verwendet. Dabei sind Objekte einer Klasse unveränderbar. Man kann von der Klasse nicht erben und es gibt keine Methode, welche den Zustand des Objekts ändert. Gibt es veränderbare Objekte als Klassen bzw. Instanz-Variablen werden deren Referenzen nie rausgegeben oder nur Referenzen zu Kopien dieser Objekte. Dadurch sind die unveränderbaren Objekte threadsicher und die Synchronisation entfällt. [7]

EVENT

Für jede eingehende Nachricht wird ein Event-Objekt erstellt und anschliessend von den Handler-Objekten benutzt. Diese sind auf verschiedenen Threads verteilt. Die Objekte dieser Klasse sind *Immutable-Object*s und somit threadsicher.

MESSAGETYPE

Als Aufzählungstyp (*enum*) ist *MessageType* threadsicher.

MESSAGESUBTYPE

Als Aufzählungstyp (*enum*) ist *MessageSubType* threadsicher.

NETWORKMANAGER

Alle Handler verwenden das gleiche NetworkManager-Objekt. Da die Klasse unterschiedliche Klassen des Netzwerks kapselt, wurden die Synchronisationsmechanismen unterschiedlich implementiert.

Die *send*- und *stop*-Methoden verändern die Zustände des NetworkManager-Objekts und der gekapselten Objekte. Daher sind diese Methoden synchronisiert.

Die *getMyAddress*-Methode wird sehr oft aufgerufen, benötigt aber keine Synchronisation, weil auf ein Immutable-Object zugegriffen wird. Das NetworkManager-Objekt kann dadurch nicht geändert werden.

Die Methoden *register* und *remove* haben keinen Einfluss auf das NetworkManager-Objekt. Die jeweilige Operation wird an das UDPNetworkListener-Objekt delegiert. Dieses kümmert sich selber, um die Synchronisation.

Die *send*-Methode hat keinen Einfluss auf das NetworkManager-Objekt und muss nicht synchronisiert werden.

Die *getNextMessageId-Methode* ändert den Zustand des NetworkManager-Objekts. Daher wird diese Methode synchronisiert.

Durch den NetworkManager sind Objekte folgender Klassen den Threads ausgesetzt:

- DatagramChannel
  Diese Klasse ist threadsicher. [8]
- UDPNetworkSender
  Diese Klasse ist Zustandslos und somit threadsicher.
- UDPNetworkListener
  Diese Klasse verwaltet den Listener-Thread und wird von Worker-Threads instanziiert und gesteuert. Der NetworkManager adaptiert ein Teil dieser Klasse und schützt bereits teilweise vor Mehrfachzugriffe. Ein vom NetworkManager nicht geschützter Teil ist die Liste der registrierten Handler, auf welche von allen möglichen Threads aus zugegriffen wird. Diese Liste wird innerhalb der UDPNetworkListener-Klasse mit Read- und Write-Locks gesichert.

Zusammenfassend sieht das so aus, dass das Sichern von Mehrfachzugriff wie folgt aufgeteilt ist:

- Start und Stopp werden über das Monitor-Objekt des NetworkManager gelockt.
- Registrieren und entfernen der Handler werden über Read- und Write-Locks im UPDNetworkListener gesichert.

NODEADDRESS

Diese Klasse wurde so implementiert, dass deren Instanzen *Immutable-Object*s sind.

3.3.5 Netzwerk

Diese Kapitel soll eine Übersicht über die wichtigsten Netzwerkkomponenten und die gewählte Netzwerktechnologie geben. Es beschreibt den Aufbau des Netzwerkes, sowie die Auswertung der Untersuchung verschiedener NAT's[9].

3.3.5.1 Netzwerkkomponenten

Die wichtigsten Netzwerkkomponenten sind die Nodes[10]. Diese bilden zusammen das P2P-Netzwerk.

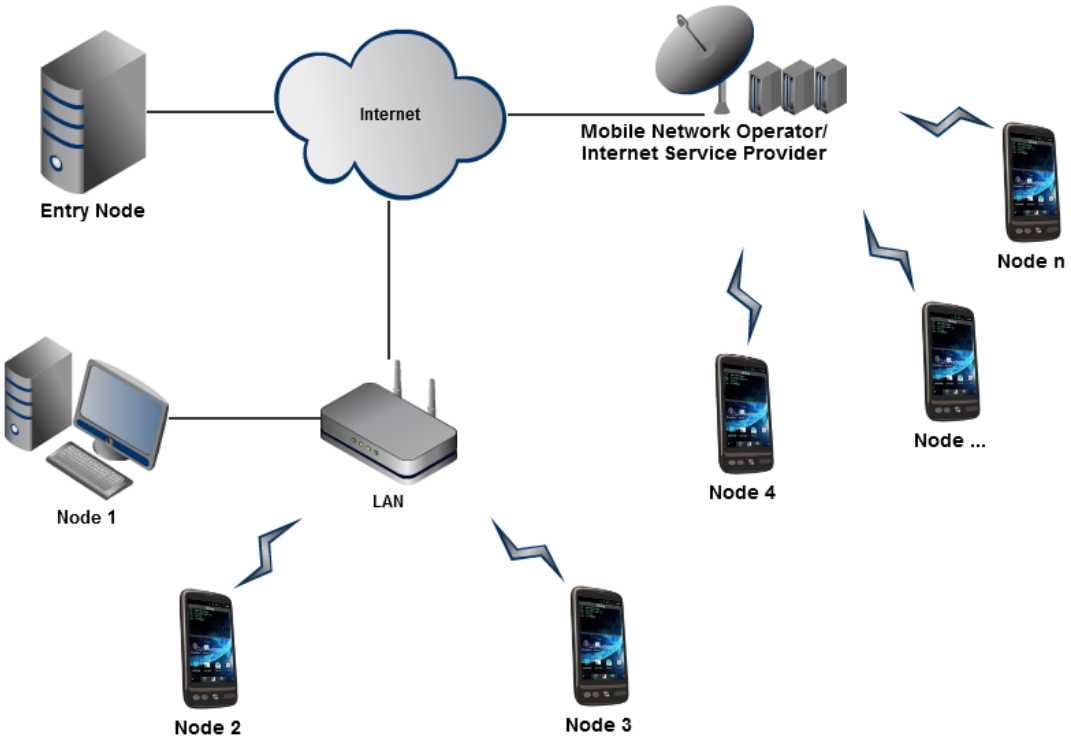


Abbildung 13 Netzwerkkomponenten

Abbildung 13 Netzwerkkomponenten zeigt einen möglichen Zustand vernetzter Nodes. Da das Ziel dieser Arbeit die Vernetzung von mobilen Geräten ist, treten vorwiegend Android Smartphone als Nodes auf. Es ist jedoch möglich, dass auch PCs als Nodes fungieren.

3.3.5.2 Netzwerktechnologien

Die gesamte Kommunikation basiert auf dem IP-Protokoll. Das heisst, dass die Verbindungen zwischen den Nodes über unterschiedliche IP-basierten Netzwerktechnologien erfolgen können. Die Medien, über welche die Kommunikation erfolgen könnte, wurden in dieser Arbeit nicht genauer betrachtet.

[9] Network Address Translation (NAT) ist ein Verfahren, um einem Netzwerk mit mehreren Geräten eine einzige IP-Adresse zuweisen zu können.
[10] Node ist ein Knoten (Peer) im Peer-to-Peer-Netzwerk. Siehe Datenstrukturen und Algorithmen für die genaue Beschreibung.

3.3.5.3 Network Address Translation

Es wird davon ausgegangen, dass ein- und ausgehende Verbindungen von und zu den Nodes möglich sind. Innerhalb eines NATs ist eine Verbindung zwischen den Nodes möglich. Eine eingehende Verbindung von ausserhalb in das NAT ist jedoch nicht gewährleistet. Das heisst, dass je nach NAT-Typ die Verbindungen blockiert werden.
Grundsätzlich gibt es folgende 4 NAT-Verhalten [9]:

Full Cone NAT

Bei diesem NAT-Typ werden alle ausgehende Pakete derselben internen IP-Adresse und Port auf dieselbe externe IP-Adresse und Port abgebildet. Jedes externe Gerät kann ein Paket an das interne Geräte senden, indem es dieses an die externe Adresse (IP und Port) des internen Gerät gesendet wird.

➔ Erlaubt eingehende Verbindungen. Verbindungsaufbau zu einem Node innerhalb eines NATs ist ohne weiteres möglich.

Restricted Cone NAT

Bei diesem NAT-Typ werden alle ausgehende Pakete derselben internen IP-Adresse und Port auf dieselbe externe IP-Adresse und Port abgebildet. Im Gegensatz zum *Full Cone* kann ein externes Geräte nur dann ein Paket an das interne Gerät senden, wenn das Interne zuvor ein Paket an die IP-Adresse des externen Geräts gesendet hat. Der NAT-Gateway merkt sich in diesem Fall diese IP-Adresse und erlaubt dem externen Gerät, Pakete an den internen Node zu senden.

➔ Erlaubt nur eingehende Verbindungen, wenn zuvor eine ausgehende Verbindung zur IP-Adresse des externen Nodes erfolgt ist. Verbindungsaufbau zu einem Node innerhalb des NATs ist in dem Fall nur mit dem Eingriff des internen Nodes möglich.

Port Restricted Cone NAT

Dieser NAT-Typ entspricht dem Restricted Cone NAT, erweitert jedoch die Verbindungseinschränkung auf den Port. Das heisst, das interne Gerät muss zuerst ein Paket an die IP-Adresse und Port-Nummer X des externen Geräts senden. Erst anschliessend kann das externe Gerät von seinem Port X aus ein Paket ans interne Gerät senden. Der NAT-Gateway merkt sich also die IP-Adresse und Port-Nummer des externen Nodes.

➔ Erlaubt nur eingehende Verbindungen, wenn zuvor eine ausgehende Verbindung zur IP-Adresse und Port-Nummer des externen Nodes erfolgt ist. Verbindungsaufbau zu einem Node innerhalb des NATs ist in dem Fall nur mit dem Eingriff des internen Nodes möglich.

Symmetric NAT

Bei diesem NAT-Typ werden alle Pakete einer internen Adresse (IP und Port) zu einer bestimmten Zieladresse auf eine einmalige externe Adresse abgebildet. Das heisst, wenn das gleiche interne Gerät ein Paket an unterschiedliche Zieladressen sendet, wird jedes Mal ein anderes Mapping verwendet. Die Zuordnung der internen mit der externen Adresse ist nicht mehr fix.

➔ Erlaubt nur eingehende Verbindungen, wenn zuvor eine ausgehende Verbindung zur IP-Adresse und Port-Nummer des externen Nodes erfolgt ist. Verbindungsaufbau zu einem Node innerhalb des NATs ist in dem Fall nur mit dem Eingriff des internen Nodes möglich. Zusätzlich können die externen Nodes die Adresse des internen Nodes nicht untereinander mitteilen, da sie alle mit einer unterschiedlichen Adresse eine Verbindung zum internen Node aufbauen müssen.

Ohne bestimmte Massnahmen ist es wie bereits erwähnt, nicht immer möglich eine Verbindung zum internen Node aufzubauen. Es gibt verschieden Ansätze und Techniken zur NAT-Traversierung. Einige davon sind [10]:

- STUN - Session Traversal Utilities for NAT
- TURN - Traversal Using Relay NAT
- UDP hole punching
- TCP hole punching
- ICMP hole punching
- ICE - Interactive Connectivity Establishment (verschiedene Techniken zusammen)

Keine dieser Techniken wurden eingebaut, da es nicht zum Umfang des Projektes gehört. Die MobileCloud Architektur ermöglicht es jedoch, einfache Techniken wie *hole punching* einzubauen. In einem kleinen Test, wurde *UDP hole punching* erfolgreich getestet. Mit der NAT-Traversierung war es möglich durch *Restricted Cone* NATs durchzukommen. Da derselbe Port für das Senden wie auch Empfangen benutzt wird, war das traversieren durch Port *Restricted Cone* NAT ebenfalls ohne weiteres möglich.

3.3.5.4 Eintrittspunkt zur MobileCloud

Wie aus **Abbildung 13 Netzwerkkomponenten** ersichtlich ist, übernimmt immer mindestens einer der Nodes in der MobileCloud die Rolle des sogenannten Eintrittspunktes (Entry Node). Dieser ist nötig, um der *Cloud* beitreten zu können und muss daher bekannt sein. Weiter ist es wegen den im Kapitel *3.3.5.3 Network Address Translation* beschriebenen NAT-Probleme nötig, dass er über eine öffentliche IP-Adresse erreichbar ist.

In der MobileCloud unterscheidet sich diese Nodes nicht von anderen und können auch nicht selber bestimmen, ob sie als solche fungieren möchten oder nicht. Es ist weder zusätzliche Funktionalität nötig, noch muss eine bestimmte Hardware diesen Job übernehmen. Somit kann jeder Node in der MobileCloud diesen Job übernehmen. Dies geschieht sobald sich ein neuer Node ein ihm bekannter Node als Einstiegspunkt aussucht und der MobileCloud über diesen beitritt.
Wenn viele Nodes der MobileCloud über den gleichen Eintrittspunktes beitreten möchten, ist es von Vorteil einen Leistungsfähigen Rechner als Entry Node zu benutzen.

3.3.5.5 Dynamischer Eintrittspunkt

Wie Im Kapitel *3.3.5.4 Eintrittspunkt zur MobileCloud* beschreiben, ist immer ein Eintrittspunkt für das Beitreten in die MobileCloud nötig. Das Problem ist allerdings, dass nicht immer die gleichen Nodes zu Verfügung stehen. Somit müsste der Benutzer immer zuerst einen aktuellen Node suchen und dessen ganze Adresse manuell eintragen. Dies ist allerdings alles andere als benutzerfreundlich. Um das zu verhindern, kann ein dynamisches binden der Adresse eines aktuellen Nodes an eine URL[11] geschehen. Somit müsste nur einmalig eine URL eingegeben werden und anschliessend kann dynamisch das Binden der Adressen an diese geschehen. Um dies zu realisieren gibt es mehrere kostenlose Dienste auf dem Internet wie zum Beispiel:

- http://www.dyndns.org
- http://www.asuscomm.com

Für Testzwecke wurde im Projekt MobileCloud die dynamische URL mobilecloud.asuscomm.com verwendet.

Um die Aufgabe des Eintrittspunkt unter mehreren Nodes aufzuteilen und trotzdem dynamisch zu halten, können mehrere solche dynamischen URLs parallel verwendet werden. Über ein vorgegebenes API, kann die Adressbindung dynamisch geändert werden.

[11] Die Uniform Resource Locators (URL) identifizieren und lokalisieren Ressourcen in Computernetzwerken.

3.3.6 SPMP Protokoll

SPMP bedeutet Simple Peer-to-Peer Management Protocol und wurde speziell für das Projekt MobileCloud ausgearbeitet. Das Protokoll befindet sich noch im Anfangsstadion, wurde aber bereits erfolgreich integriert und getestet.

3.3.6.1 Protokoll Stack

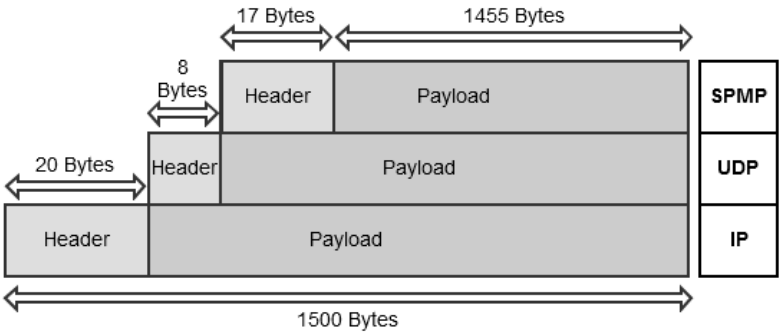


Abbildung 14 Protokollstack mit UDP

Für die Verwaltung von MobileCloud wird UDP als Transport-Protokoll eingesetzt. Somit stehen dem SPMP bei 1500 Bytes MTU[12] 1472 Bytes zur Verfügung. Davon werden 17 Bytes für den Header verwendet. Die detaillierte Beschreibung des SPMP-Aufbaus ist im Kapitel *3.3.6.2 Nachrichtenaufbau* zu finden.

UDP eignet sich gut, weil durch einen kleinen *Header* der Datenoverhead möglichst klein gehalten wird und die Geschwindigkeit und Skalierbarkeit hoch sind [11]. Zudem besteht die Kommunikation für die Verwaltung der MobileCloud ausschliesslich aus Anfrage- Antwort-Nachrichten, was sich mit UDP im Vergleich zu TCP pragmatischer und ohne grossen Overhead realisieren lässt. Einerseits können die Antwort-Nachrichten gleich als *Acknowledgement*[13] dienen. Anderseits ist der Overhead durch den TCP-Header grösser. Dies veranschaulicht die **Abbildung 15 Protokollstack rotokollstackmit TCP**.

Bei UDP sind jedoch folgende mögliche Fehler zu beachten [11]:

- Pakete können verloren gehen
- Reihenfolge der ankommenden Pakete ist nicht definiert

Des Weiteren gibt es bei UDP keine Datenflusskontrolle, was beim Übertragen von grossen Datenmengen negative Auswirkungen haben kann. Der Empfänger könnte beim Überlauf seines Empfangsspeichers Datenpakete verwerfen, was zu deren Verlust führt.
Da es keinen Sinn macht das Rad neu zu erfinden, ist zu einem späteren Zeitpunkt für die Übertragung grosser Datenmengen TCP geplant. Für TCP könnte wie auch für UDP, SPMP verwendet werden.

[12] Die Maximum Transmission Unit (MTU) beschreibt die maximale Paketgröße eines Protokolls.
[13] Bestätigung des Empfangs der Anfrage

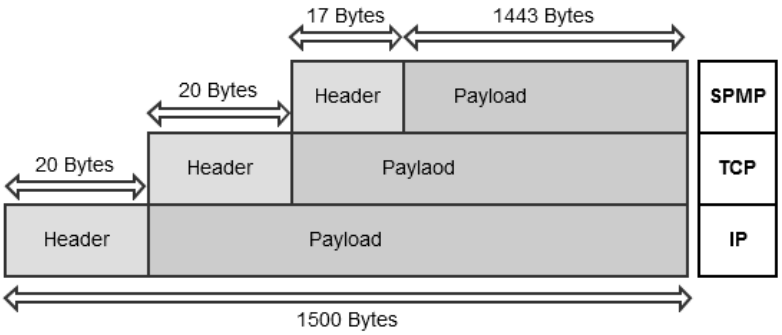


ABBILDUNG 15 PROTOKOLLSTACK MIT TCP

Die **ABBILDUNG 15 PROTOKOLLSTACK ROTOKOLLSTACKMIT TCP** zeigt den Aufbau des Protokollstacks mit TCP. In diesem Fall stehen dem SPMP im Standardfall, sprich wenn das Options-Feld im TCP Header nicht genutzt wird, 1460 Bytes zur Verfügung. Davon werden wiederum 17 Bytes für den SPMP Header verwendet.
Diese Variante mit TCP wurde nicht implementiert. Eine Erweiterung ist jedoch möglich. Eine genaue Beschreibung der Netzwerkschicht ist im Architektur-Dokument zu finden.

3.3.6.2 Nachrichtenaufbau

Bits								
8	7	6	5	4	3	2	1	
Message sub-type				Version				Octet 1
Source address								Octet 2
								Octet 3
								Octet 4
								Octet 5
								Octet 6
Destination address								Octet 7
								Octet 8
								Octet 9
								Octet 10
								Octet 11
Message type								Octet 12
								Octet 13
Message ID								Octet 14
								Octet 15
								Octet 16
								Octet 17
Payload								Octet 18 – Octet 1472

Abbildung 16 Nachrichtenaufbau

Version

Die Versionsnummer des Protokolls wird in den ersten vier Bits beschrieben. Sie startet mit dem Wert 1 und wird bei jeder Änderung um eins inkrementiert. Dies ergibt eine maximale Versionsnummer von 15. Um eine Änderung im Protokoll vorzunehmen, muss dies daher gut überlegt werden, damit die Versionsnummer nicht zu schnell steigt. Es darf nicht sein, dass jede Softwareänderung auch eine Änderung des Protokolls mit sich zieht.

Solange es kein aktuelles Release von MobileCloud gibt, ist die Versionsnummer 0. Auch führen Änderungen am Protokoll, vor einem offiziellen Release, zu keiner Erhöhung der Versionsnummer.

Source-, Destination Address

Die Absender- und Empfängeradressen sind je fünf Bytes lang und identifizieren die einzelnen Clients in der Cloud. Die Adressen werden für die Adressierung von Mitteilungen verwendet.
Die genaue Bedeutung und Generierung der Adressen wird im Kapitel *3.3.8.1.1 Hash-Funktion* beschrieben.

Message type

Der Mitteilungstyp besteht aus zwei Bytes und beschreibt der Typ der Nachricht. Dieser wird benötigt, damit der Empfänger weiss, wie er die empfangene Mitteilung behandeln muss. Es gibt Mitteilungstypen, welche für die Verwaltung der MobileCloud benötigt werden und solche die für die einzelnen Dienste benötigt werden.

Die verschiedenen Typen werden im Kapitel *3.3.6.3 Festgelegte Mitteilungstypen* genauer beschrieben.

Message sub-type

Der Mitteilungssubtyp besteht aus vier Bits. Er dient dazu den Mitteilungstyp noch genauer zu beschreiben. Dies ermöglicht dem Dienst, welcher einem bestimmten Mitteilungstyp entspricht, weitere Subtypen zu benutzen. Es können bis zu 15 unterschiedlichen Subtypen dargestellt werden.

Die verschiedenen Subtypen werden im Kapitel *3.3.6.3 Festgelegte Mitteilungstypen* genauer beschrieben.

Message ID

Die vier Bytes für die Mitteilungsnummer, weisen eine Mitteilung einer laufenden Sitzung zu. Dies ist nötig damit von einem Mitteilungstypen parallel mehrere Sitzungen geöffnet sein können.

Eine Mitteilung wird durch den Absender und den Empfänger, den Mitteilungstypen und der Mitteilungsnummer Identifiziert.

Payload

Dieses Feld beinhaltet die eigentlichen Daten und hat daher eine variable Grösse. Das Format der Daten ist nicht festgelegt.

3.3.6.3 Festgelegte Mitteilungstypen

Bei den Mitteilungstypen gibt es zwei Sorten. Die Einen werden für die Verwaltung der MobileCloud verwendet, die Anderen für die Dienste, welche über die MobileCloud genutzt werden können. An dieser Stelle wird nur auf die Typen für die MobileCloud-Verwaltung eingegangen. Diese liegen im Bereich von 0 bis 100. Die Anderen 65435 Typen können für Services verwendet werden.
Folgende Typen sind festgelegt.

Wert	Kürzel	Beschreibung
0	TEST	Wird für Testzwecke während der Entwicklung verwendet.
1	JOIN	Wird für die Abhandlung eines Node-Beitritts zur MobileCloud verwendet.
2	LEAVE	Wird beim Verlassen eines Nodes verwendet.
3	UPDATE	Ein neuer Client ist vorhanden, oder hat seine Adresse geändert und kann in die Adressliste aufgenommen werden.
4	NODELIST_REQUEST	Ein Adressbereich wird angefordert.
5	DATA	Wird für das Senden von Daten an einem bestimmten Client verwendet.

Tabelle 2 Festgelegte Mitteilungstypen

Folgende Mitteilungssubtypen sind festgelegt.

Wert	Kürzel	Beschreibung
0	REQUEST	Wird für Anfragen verwendet. Darauf muss zwingend eine Antwort folgen.
1	REPLY_OK	Wird für Antworten verwendet, wenn die Anfrage erfolgreich abgearbeitet werden konnte. Die Payload der Nachricht enthält dabei das Resultat der abgearbeiteten Anfrage.
2	REPLY_NOK	Wird für Antworten verwendet, wenn die Anfrage nicht erfolgreich Abgearbeitet werden könnte. Die Payload der Nachricht enthält dabei eine Fehlermeldung.
3	DELIVERY	Wird für Nachrichten verwendet, auf welche keine Antwort erfolgen muss. Es ist also weder eine Anfrage noch eine Antwort, sondern eine einfach „Zustellung“.

Tabelle 3 Festgelegte Mitteilungssubtypen

Jeder Mitteilungstyp kann mit jedem Subtyp kombiniert werden. Damit ergibt sich z.B. die Möglichkeit zwischen Anfragen und Antworten innerhalb eines Mitteilungstyp zu unterscheiden. Beispielswiese JOIN REQUEST für die Anfragen und JOIN REPLY_OK für die Antworten beim Abhandeln eines Node-Beitritts.

3.3.6.4 Fehler bei UDP

Wie bereits erwähnt, können bei UDP Fehler vorkommen. Auf dessen genauen Ursachen wird hier nicht näher eingegangen. Diese Fehler können zu folgende Probleme führen:

- Ein Node wartet ewig auf eine Antwort, wenn die Anfrage- oder Antwort-Nachricht verloren geht.
- Werden alle Nachrichten übertragen und empfangen, kann nicht ohne weiteres davon ausgegangen werden, dass eine Antwort zur der kurz zuvor gesendete Anfrage gehört.

Letzteres Problem wird gelöst, indem mit Hilfe der Message ID, welche in diesem Fall als Korrelationsnummer dient, eine Antwort einer zuvor gesendeten Anfrage zuordnen kann. Zu jeder Anfrage eines Dienstes, wird eine andere Message ID verwendet. In der Antwort-Nachricht, welche gleich die Aufgabe als *Acknowledgement* übernimmt, wird die Message ID der Anfrage benutzt.

Das erste Problem kann mit einem Timeout gelöst werden. Dies wurde aber bisher nicht implementiert. Beim Timeout ist zu beachten, dass dieser nicht zu kurz sein darf. Der Empfänger muss genug Zeit haben, um die Anfrage zu bearbeiten und eine Antwort zurückzusenden. Ist der Timeout abgelaufen, kann man nicht sicher sein, ob der Empfänger die Anfrage nicht erhalten hat oder die Antwort verloren ging. Ferner kann man bei einer asynchronen Kommunikation, wie sie bei der MobileCloud vorkommt, nicht sicher sein, ob der Empfänger abgestürzt ist und nicht mehr läuft oder ein Byzantinisches Verhalten[14] an den Tag legt. [11]

[14] Verhalten, bei dem ein Node z.B. sich nicht mehr an das Protokoll hält und nur teilweise Antworten zurücksendet. Man kann diesem Node nicht mehr vertrauen.

3.3.7 Datenstrukturen

Das P2P-Netzwerk *Pastry* verwenden für die Verwaltung des Netzwerkes drei verschiedene Datenstrukturen [5]:

- Die Routing-Tabelle: Eine verteilte Hash-Tabelle, welche für das effiziente Routing von Anfragen nötig ist.
- Das *Leaf-Set*: Eine Menge von Nodes, welche für die Ringförmige Struktur des P2P-Netzwerkes zuständig ist.
- Die Nachbarschaftsmenge: Menge von Nodes welche die kürzeste Latenzzeit zum *Mainnode* haben.

NodeID: 2310			
Leaf-Set			
2303	2308	2312	2320
Routing-Tablelle			
0331	1300	2	3101
2010	2130	2203	3
2301	1	2323	2330
0	-	2312	-
Nachbarschaftsmenge			
1300	1002	1133	1330

Tabelle 4 Die drei Datenstrukturen von Pastry [5]

Grundsätzlich funktioniert das P2P-Netzwerk auch ohne Routing-Tabelle, da das Routing auch mit dem *Leaf-Set* durchgeführt werden kann. Dadurch sinkt allerdings die Routing-Geschwindigkeit vom Erwartungswert $O(log_2(n))$ [5]auf $O(n)$ (n = Anzahl aller Nodes im P2P-Netzwerk). Dabei werden die Daten einfach an den rechten oder linken äussersten bekannten Node weitergeleitet.

Bei MobileCloud wurde auf die Nachbarschaftsmenge ganz verzichtet. Dafür wurden folgende Annahmen getroffen:
Die meisten mobilen Geräte mit dem gleichen Telefonnummern-Präfix …

- … befinden sich im gleichen Land.
- … sind beim gleichen Mobilefunkanbieter.
- … haben bei der Kommunikation mit mobilen Geräten vom gleichen Mobilfunkanbieter eine kürzere Latenzzeit als wenn diese bei einem anderen Anbieter sind.

Da die ID grundsätzlich der Telefonnummer entspricht, haben Nodes mit ähnlichen ID's kurze und Nodes mit unterschiedlichen ID's höhere Latenzzeiten. Dadurch, dass im *Leaf-Set* die Nodes der Grösse ihrer ID's nach angeordnet sind, übernimmt das *Leaf-Set* gleichzeitig die Funktion der Nachbarschaftsliste.

In einem ersten Schritt wurde daher nur das *Leaf-Set* implementiert. Auf die verteile Hash-Tabelle wurde aus zeitlichen Gründen verzichtet. Diese könnte zu einem späteren Zeitpunkt noch implementiert werden.

3.3.7.1 Leaf-Set

Das *Leaf-Set* beinhaltet die Grundstruktur von MobileCloud. Es bildet alle Nodes in der MobileCloud auf einen Ring ab. Jeder *Mainnode* hat ein eigenes *Leaf-Set* mit linken und rechten Nachbarnodes. Die linken Nachbarnodes haben eine tiefere, die Rechten eine höhere ID.
Dabei gibt es eine Ausnahme. Weil der Node mit der grössten ID keinen rechten und der Node mit der kleinesten ID keinen linken Nachbar haben, werden diese gegenseitig zusammengehängt. Somit wären der rechte Nachbar des Nodes mit der höchsten ID der Node mit der kleinsten ID und der linke Nachbar der Nodes mit der kleinsten ID der Node mit der grössten ID. In der **Abbildung 17 Leaf-Set mit einigen Nodes** wären das die Nodes mit den ID's 99 und 01. Dadurch werden die Nodes auf eine Art Ring abgebildet.
In der folgenden

Abbildung **17** ist ein komplettes *Leaf-Set* mit einem *Mainnode* mit der ID 50 zu sehen.

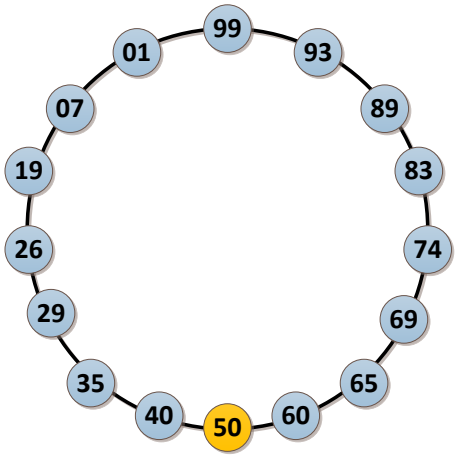


Abbildung 17 Leaf-Set mit einigen Nodes

Es macht kein Sinn, wenn jeder Node alle anderen Nodes in seinem *Leaf-Set* speichern würde. Das bräuchte sehr viel Speicherplatz und der Aufwand, um alle *Leaf-Sets* aktuell zu halten wäre riesig. Aus diesem Grund werden nur eine fixe Anzahl von rechten und linken Nodes gespeichert, wie das in der

Abbildung 18 mit je drei Nodes zu sehen ist. Somit hat jeder einzelne Node eine Teilinformation über die MobileCloud. Es gibt keinen Node, welcher alle Informationen hat und auch verwalten muss. Diese ist auf alle Nodes verteilt. Möchte nun ein Node eine Nachricht an einen anderen Node senden, welcher nicht in seinem *Leaf-Set* ist, muss zuerst dessen Node-Adresse in der MobileCloud abgefragt werden. Wie dies genau funktioniert ist im Kapitel *3.3.8.2 MobileCloud Services Algorithmen* beschrieben.

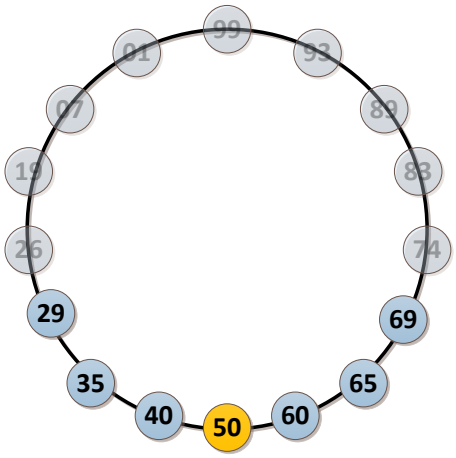


Abbildung 18 Leaf-Set eines MainNodes mit beschränkter Anzahl Nodes

3.3.7.1.1 Umsetzung

In MobileCloud wurde das *Leaf-Set* mit einer doppelt verketteten Liste realisiert. Jeder Node hat ein nächsten, grösseren Node und einen Vorherigen, kleineren Node. Durch diese Implementierung werde im Gegensatz zu einem Array, einige Operationen, wie zum Beispiel das Hinzufügen neuer Nodes, vereinfacht.

Die *Node-List* hat eine maximale Anzahl von Nodes n pro Seite. Daraus folgt die maximale Anzahl Nodes $m = 2 * n$, also immer eine gerade Anzahl von Nodes.
Daraus ergeben sich folgende Mengen:

Nodes auf der linken Seite $N_l = \{p_{-n} \dots p_{-1}\}$
Nodes auf der rechten Seite $N_r = \{p_1 \dots p_n\}$
Alle gespeicherten Nodes $M = N_l + N_r$
Alle Nodes in MobileCloud $Z = \{p_0 \dots p_z\}$

Solange die aktuelle Anzahl von gespeicherten Nodes $a \leq n$ ist gilt:

$$M = N_l = N_r$$

Wenn $a > n,\ a < m$ gilt:

$$A \subset N_l,\ A \subset N_r,\ A \subset M \text{ und}$$

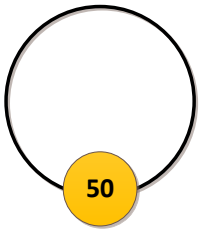
$$M = Z$$

Ansonsten gilt:

$$N_l \not\subseteq N_r, \qquad N_l \subseteq M, \qquad N_r \subseteq M, \qquad M \subseteq Z$$

3.3.7.1.2 ADD

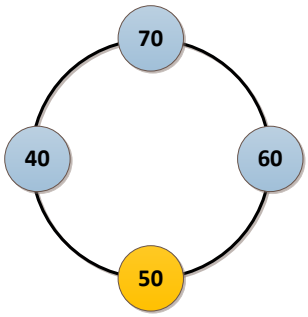
Wenn die *Node-Liste* leer ist, zeigen der nächste, sowie der vorherige Nodezeiger auf den *Mainnode* selbst.



$$N_l = \{\square\}$$
$$N_r = \{\square\}$$
$$M = \{\square\}$$

ABBILDUNG 19 leeres Leaf-Set

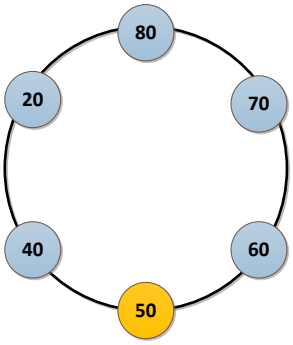
Sobald die ersten Nodes eingefügt werden, ändert sich dies. Die neuen Nodes werden so eingefügt, dass sie in aufsteigender Reihenfolge im Ring platziert werden.
Bei diesem Beispiel werden die Nodes mit den ID's 40, 60 und 70 eingefügt.



$$N_l = \{40, 60, 70\}$$
$$N_r = \{40, 60, 70\}$$
$$M = \{40, 60, 70\}$$

ABBILDUNG 20 Leaf-Set mit drei Nodes

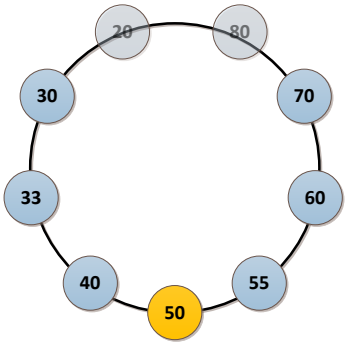
Nun werden noch die Nodes mit den ID's 20 und 80 hinzugefügt.



$$N_l = \{20, 40, 80\}$$
$$N_r = \{60, 70, 80\}$$
$$M = \{20, 40, 60, 70, 80\}$$

ABBILDUNG 21 Leaf-Set mit fünf Nodes

Nach weiterem Hinzufügen der Nodes 30, 55, 27, würden die Nodes 80 und 20 wieder aus dem *Leaf-Set* gelöscht, da wie bereits oben erwähnt nur die sechs Nodes mit den nächsten ID's gespeichert werden.



$$N_l = \{30, 33, 40\}$$
$$N_r = \{55, 60, 70\}$$
$$M = \{30, 33, 40, 55, 60, 70\}$$

Abbildung 22 Gefülltes Leaf-Set

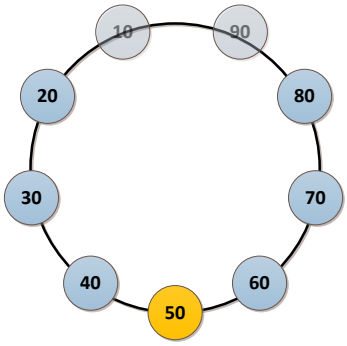
3.3.7.1.3 REMOVE

Das Löschen von Nodes aus dem *Leaf-Set* ist nicht direkt möglich, da es zu inkonsistenten Zuständen kommen kann. Das Entfernen von Nodes ist nur über das Aktualisieren des *Leaf-Sets* möglich. Selbst dadurch könnte ein inkonsistenter Zustand entstehen, jedoch wird die Wahrscheinlichkeit dass dieser unabsichtlich erzeugt wird stark verringert.

Das folgende Beispiel sollte dies aufzeigen.

$$Z = \{10, 20, 30, 40, 60, 70, 80, 90\}$$

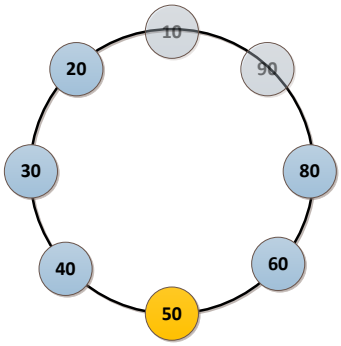
Daraus ergibt sich folgendes *Leaf-Set:*



$$N_l = \{20, 30, 40\}$$
$$N_r = \{60, 70, 80\}$$
$$M = \{20, 30, 40, 60, 70, 80\}$$

Abbildung 23 Gefülltes Leaf-Set vor dem Löschen

Nun wird der Node mit der ID 70 entfernt.



$$Z = \{10, 20, 30, 40, 60, 80, 90\}$$
$$N_l = \{20, 30, 40\}$$
$$N_r = \{20, 50, 60\}$$

Abbildung 24 Leaf-Set im inkonsistenten Zustand nach dem Löschen

Daraus ergibt sich wie oben definiert:

$$a > n, \qquad a < m$$

Was heissen würde dass:

$$M = Z$$

Was allerdings nicht sein kann und somit das *Lea-Set* in einen inkonsistenten Zustand bringt. Damit dieser Fehler wieder behoben werden kann, muss das *Leaf-Set* aktualisiert und somit ein zusätzlicher Node aufgenommen werden. Dies darf nicht vergessen gehen, sonst bleibt das *Leaf-Set* in diesem Zustand.
Um dies zu vermeiden, wird auf das separate Löschen verzichtet und es nur indirekt über das Aktualisieren ermöglicht. Somit kann es nicht passieren, dass das Aktualisieren vergessen geht.

3.3.7.1.4 Aktualisieren

Das Aktualisieren des *Leaf-Sets* ist eine der wichtigsten Operation. Diese wird verwendet, um das *Leaf-Set* von einem inkonsistenten Zustand in einen Konsistenten zu überführen, um Nodes zu löschen, welche die MobileCloud verlassen haben oder einfach um das *Leaf-Set* auf den aktuellsten Stand zu bringen.

Das Aktualisieren ist in zwei Operationen aufgeteilt. Eine um die rechte Seite und eine um die linke Seite zu aktualisieren. Die beiden Seiten können nur separat aktualisiert werden.

Das Aktualisieren erfolgt in zwei Schritten:

1. Alle Nodes der zu aktualisierenden Seite löschen. Das sind auf der rechten oder linken Seite:

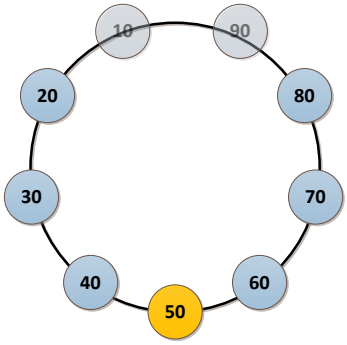
   Alle Nodes wenn: $a > n$

   Nur die n rechten oder linken Nodes wenn: $a < n$
2. Die aktuellen Nodes einfügen. Dies geschieht wie im Kapitel *3.3.7.1.2 Add* beschrieben.

Aktualisieren bedeutet:

- wenn $a \leq n$, führt das linke und das rechte Aktualisieren zum gleichen Ergebnis, da $M = N_l = N_r$.
- wenn $a > n$, $a < m$, werden beim linken Aktualisieren auch einige Rechte Nodes aktualisiert und auch umgekehrt, da $A \subset N_l$, $A \subset N_r$, $A \subset M$.
- wenn $a = m$, hat das Aktualisieren einer Seite keinen Einfluss auf die Andere, da $N_l \nsubseteq N_l$.

Beim folgenden Beispiel wird die rechte Seite aktualisiert.



$$Z = \{10, 20, 30, 40, 60, 70, 80, 90\}$$
$$N_l = \{20, 30, 40\}$$
$$N_r = \{60, 70, 80\}$$

Abbildung 25 Leaf-Set vor dem Aktualisieren

Und zwar ist der Node mit der ID 70 ausgetreten, das heisst das Aktualisieren wird ohne dem Node 70, dafür mit dem Node 90 durchgeführt. Dazu muss ein Update auf der rechten Seite mit den Nodes $U = \{60, 80, 90\}$ gemacht werden.

Als erstes werden alle Nodes auf der rechten Seite aus dem *Leaf-Set* entfernt.

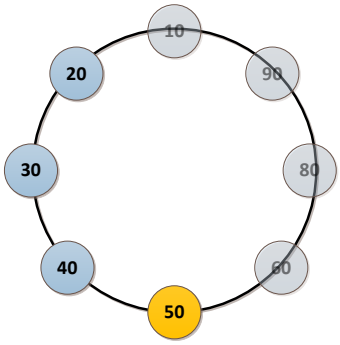
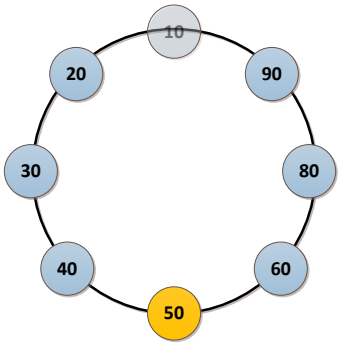


Abbildung 26 Leaf-Set mit gelöschter rechten Seite

Anschliessend werden die aktuellen Nodes U eingefügt.



$$Z = \{10, 20, 30, 40, 50, 60, 80\}$$
$$N_l = \{20, 30, 40\}$$
$$N_r = \{50, 60, 80\}$$

Abbildung 27 Aktualisierte Leaf-Set

Nach diesen zwei Schritten, hat das *Leaf-Set* wieder einen aktuellen Zustand.
Wichtig ist, dass diese zwei Operationen atomar verlaufen, also kein paralleler Zugriff auf das *Leaf-Set* erfolgt. In MobileCloud wird das mit einen *read-write-lock* gelöst.

3.3.8 Algorithmen

3.3.8.1 Allgemeine Algorithmen

In diesem Kapitel werden die Algorithmen beschrieben, welche nichts mit der Verwaltung und den Services von MobileCloud zu tun haben.

3.3.8.1.1 Hash-Funktion

Die Hash-Funktion ist für die Generierung der ID der Nodes zuständig. Diese muss aus der Telefonnummer berechnet werden, jedoch dürfen die Informationen über den Mobilfunkanbieter nicht verloren gehen. Weil ähnliche ID im *Leaf-Set* nahe beieinander liegen, was im Kapitel *3.3.7 Datenstrukturen* beschrieben ist, führt dies zu einem schnellen Routing und eine Nachbarschaftsliste wird unnötig.

Durch die HASH-Funktion ist es für jeden Teilnehmer möglich, für bekannte Telefonnummern die ID zu berechnen und somit mit diesem Kommunizieren.

Theoretisch könnte für die ID auch gleich die Telefonnummer genommen werde. Das Problem darin liegt jedoch bei der Sicherheit. Wenn ein Datenpacket durch die MobileCloud geroutet wird, könnte jeder Node, der dieses Packet zwischenspeichert bzw. weiterleitet, die Telefonnummer des Senders und Empfängers auslesen.
Dies wäre kein Problem, wenn nur Bekannte Nodes Kontakt zu dem Packet hätten. Für die Verwaltung von MobileCloud werden im Hintergrund jedoch Pakete an Nodes versendet, von denen nur die ID und nicht die Telefonnummer bekannt ist. Durch den Hash, kann so nur die ID und nicht die Telefonnummer ausgelesen werden.
Eine andere Möglichkeit wäre als ID eine beliebige Zeichenfolge zu nehmen, was allerding zu Problemen führt wie:

- mehrfache ID's
- Zusätzliche Kenntnis der ID, obwohl die Telefonnummer evtl. schon bekannt ist.
- Authentifizierung ist nicht gegeben, mit der Telefonnummer grösstenteils schon.

Diese Funktion ist nur provisorisch implementiert und muss zu einem späteren Zeitpunkt noch vollständig implementiert werden.
Momentan werden nur fünf ASCII-Zeichen in Bytes abgebildet. Im Normalfall die fünf letzten Zahlen der Telefonnummer oder fünf beliebigen Zeichen, falls keine Telefonnummer vorhanden ist.
Dies ist vor allem für Testzwecke sinnvoll, da so die Richtigkeit der Reihenfolge der Nodes besser überprüft werden kann.

3.3.8.2 MobileCloud Services Algorithmen

In MobileCloud werden alle Funktionen als Services angesehen. Somit sind auch Verwaltungsfunktionen, wie etwa das Betreten oder das Verlassen der MobileCloud, Services. Dies erlaubt, dass nachträglich mit geringstem Aufwand zusätzliche Services oder schnellere Algorithmen implementiert oder erweitert werden können. Es ermöglicht auch, dass spezifische Routingmechanismen für einzelne Services implementiert werden können.

Ein Service wird immer von einem Handler repräsentiert. Ein Handler kann allerdings weitere Handler nutzen oder genutzt werden. Mehr Informationen über Handler und Services sind im Kapitel *3.3.3 Design und Architektur* zu finden.

MobileCloud verfügt über folgende Services und Handler:

Name	Kurzbeschreibung	Verwendete Handler
join	Der MobileCloud beitreten	JoinHandler
leave	Die MobileCloud verlassen	LeaveHandler
sendData	Daten an einen Node senden	SendDataHandler

Tabelle 5 Services

Name	Kurzbeschreibung	Verwendete Handler
JoinHandler	Bearbeitet Service join	UpdateHandler, UpdateLeafSetHandler
UpdateHandler	Fügt ein neuen Node dem *Leaf-Set hinzu*	
UpdateLeafSetHandler	Aktualisiert eine Seite des *Leaf-Sets*	
LeaveHandler	Bearbeitet Service leave	UpdateLeafSetHandler
SendDataHandler	Bearbeitet Service sendData	DataReceiveHandler
DataReceiveHandler	Empfängt Daten	

Tabelle 6 Handler

3.3.8.2.1 Externe Darstellung

Das Format der zu übermittelnden Daten ist den einzelnen Handler überlassen. Somit kann jeder Handler die Darstellung wählen, die für seine Daten am effizientesten ist.
Für die Übermittlung der Verwaltungsdaten wird JSON[15] verwendet. Der grösste Vorteil ist die kompakte, übersichtliche Darstellung den Daten. Im Vergleich zu XML wird viel weniger Speicherplatz benötigt und es können so mehr Daten auf einmal übertragen werden.
JSON wird standardmässig von Android unterstützt und kann ohne grosse Einarbeitungszeit verwendet werden.

Weitere Informationen zu JSON und Android sind unter den unten aufgelisteten URL zu finden.

http://developer.android.com/reference/org/json/package-summary.html

[15] Die JavaScript Object Notation (JSON) ist ein kompaktes und für Mensch und Maschine einfach lesbares Datenformat für den Datenaustauschs zwischen Anwendungen.

3.3.8.2.2 JOIN

Dieser Service behandelt das Eintreten in die MobileCloud und das dafür notwendige Erstellen und Aktualisieren der Infrastruktur.

Sequenzdiagramm

	JoinHandler
	UpdateLeafSetHandler
	UpdateHandler

TABELLE 7 LEGENDE JOIN

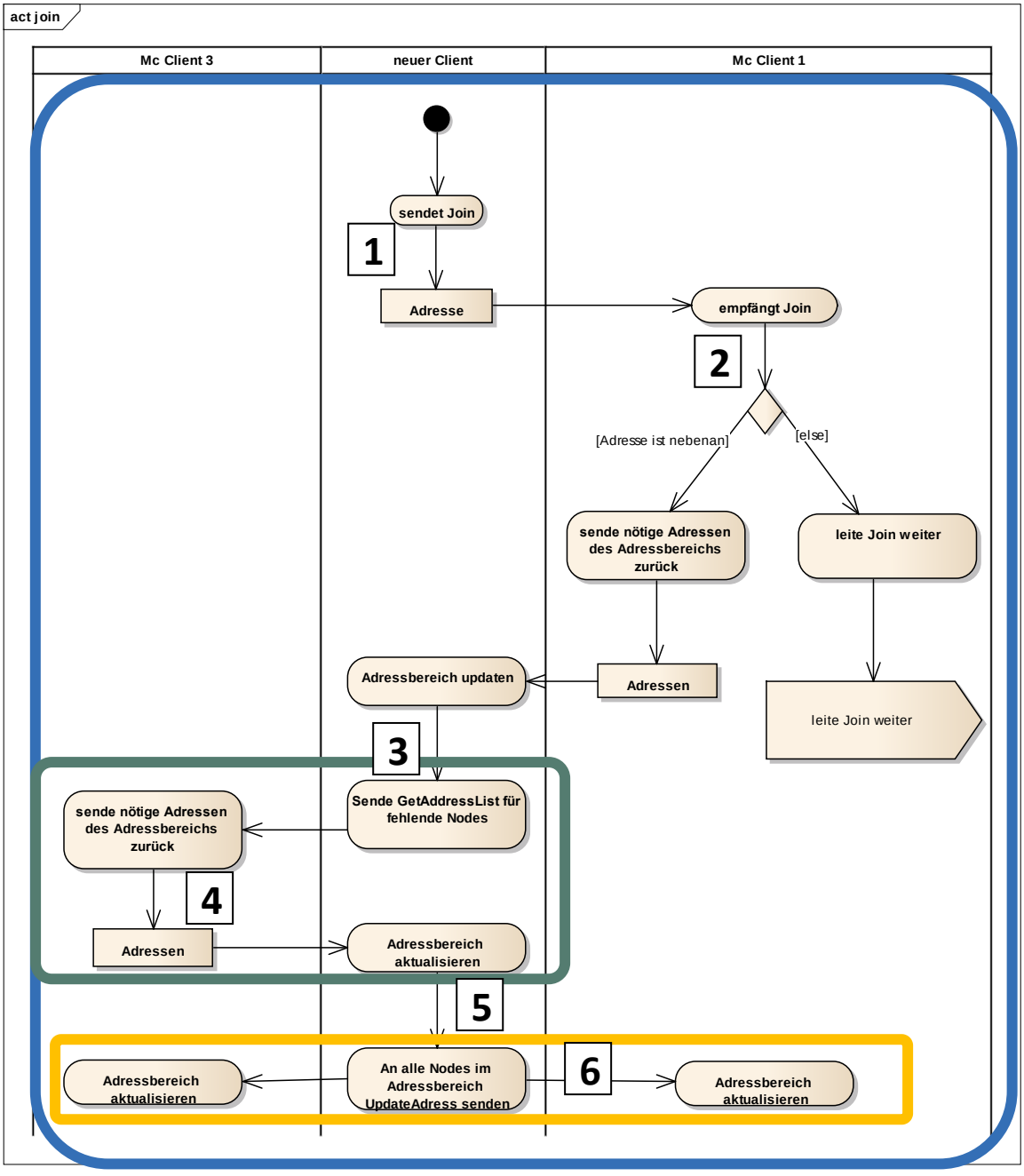


ABBILDUNG 28 Aktivitätsdiagramm join

Ablauf

Der Ablauf von *join* wird anhand des Sequenzdiagrammes beschrieben.

1. Der neue Node sendet an einen bekannten Node, welcher schon in der MobileCloud ist, eine *join* Nachricht.

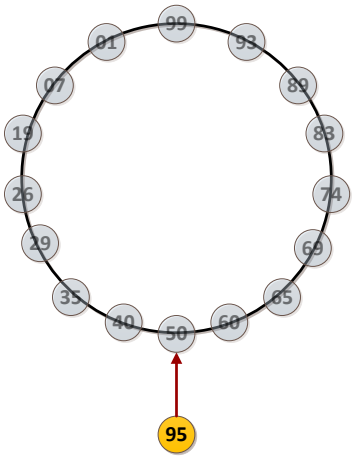


Abbildung 29 Senden der join Nachricht

2. Der Empfänger überprüft, ob der neue Node sein linker Nachbar wäre. Ist das der Fall, sendet er dem neuen Node die Adressen seiner linken und rechten Nachbarnodes aus dem *Leaf-Set* zurück.
   Wenn das nicht der Fall ist, sucht er in seinem *Leaf-Set* nach dem zukünftigen rechten Nachbarn des neuen Nodes. Findet er diesen, leitet er die Nachricht an diesen weiter. Ansonsten leitet er die Nachricht an den rechts äussersten Node im *Leaf-Set* weiter.

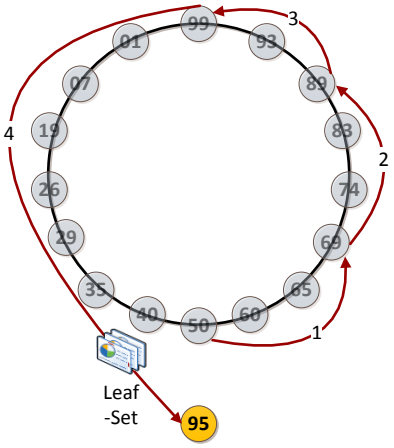


Abbildung 30 Ersten Leaf-Set Anfrage

3. Der neue Node erhält alle Node-Adressen aus dem *Leaf-Set* seines zukünftigen rechten Nachbarn. Diese fügt er inklusive der Node-Adresse des rechten Nachbarn in sein eigenes *Leaf-Set* ein.
   Die rechte Seite seines *Leaf-Sets* ist nun aktuell, auf der Linken fehlt jedoch noch eine Node-Adressen. Dafür fragt er seinen direkten linken Nachbarn nach dessen linken *Leaf-Set*-Seite an.

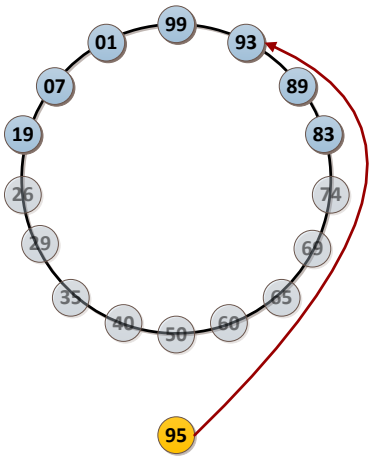


Abbildung 31 Anfrage des linken Leaf-Sets

4. Der linke Nachbar sendet die Node-Adressen aus seiner linken *Leaf-Set*-Seite zurück.

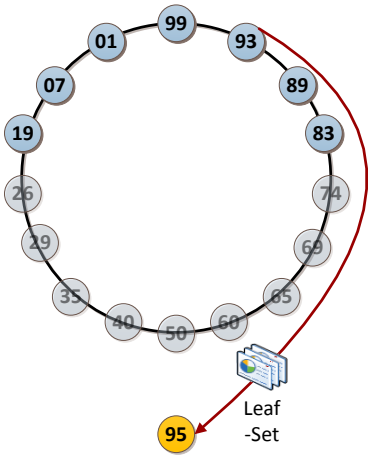


Abbildung 32 Aktualisieren des Leaf-Sets

5. Der neue Node fügt der erhaltenen Node-Adressliste noch die Adresse des linken Nachbars hinzu. Dies ist nötig, weil der Sender einer Adressliste nie selber in der gesendeten Liste enthalten ist. Anschliessend aktualisiert der neue Node mit dieser Adressliste die linke Seite seines *Leaf-Sets*.

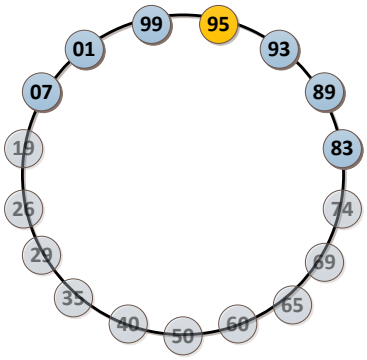


Abbildung 33 Platzierung in der MobileCloud

6. Nun hat der neue Node ein aktuelles *Leaf-Set*, ist jedoch bei den anderen Nodes noch nicht registriert und daher noch nicht in der MobileCloud. Deshalb sendet er allen Node-Adressen im *Leaf-Set* seine eigene Adresse, damit sie ihr *Leaf-Set* aktualisieren können.

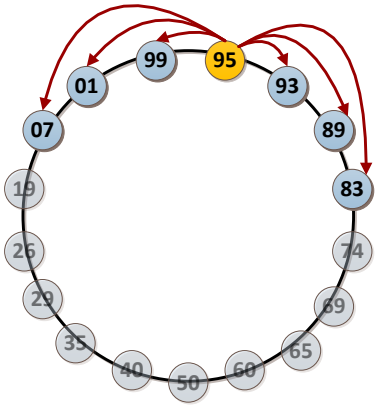


Abbildung 34 Update senden

3.3.8.2.3 LEAVE

Dieser Service behandelt das Verlassen der MobileCloud. Das beinhaltet das Informieren aller Nachbarn sowie das Aktualisieren dessen *Leaf-Sets*.

Sequenzdiagramm

	LeaveHandler
	UpdateLeafSetHandler

TABELLE 8 LEGENDE LEAVE

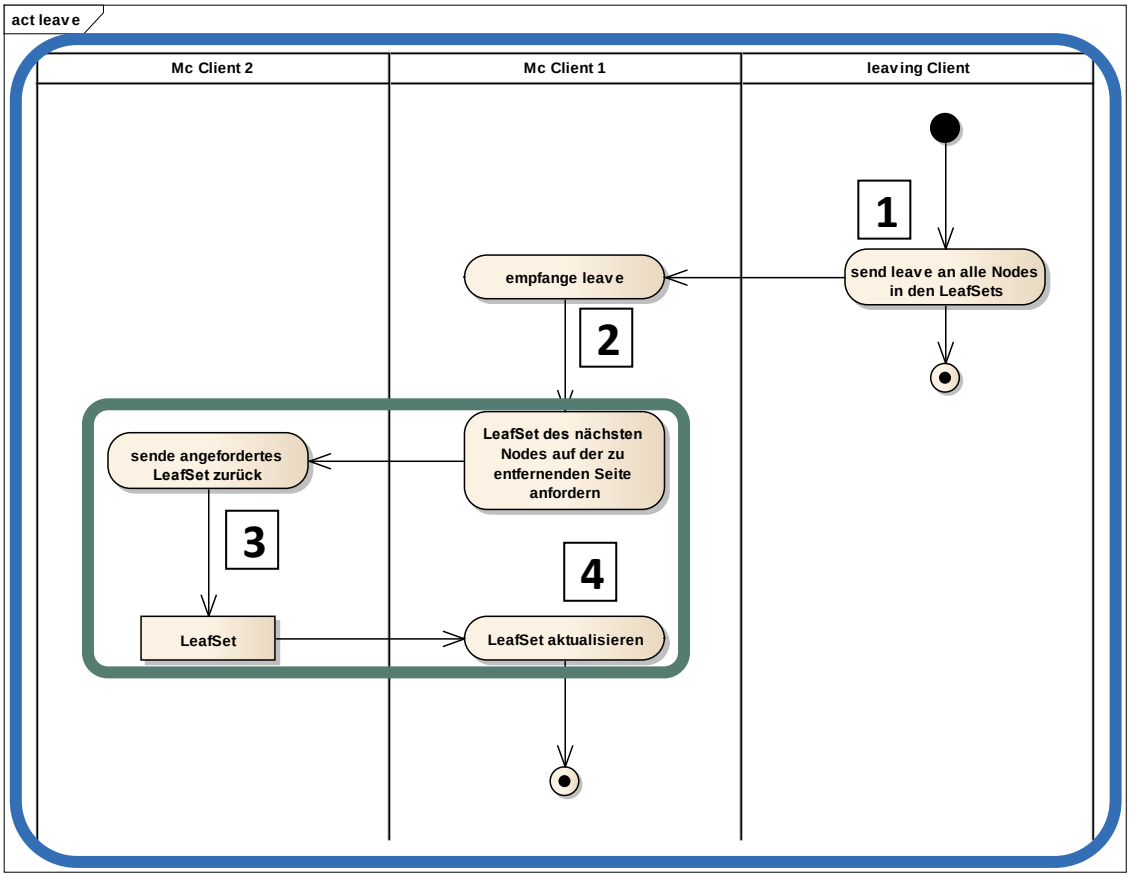


ABBILDUNG 35 AKTIVITÄTSDIAGRAMM LEAVE

Ablauf

Der Ablauf von *leave* wird anhand des Sequenzdiagrammes beschrieben.

1. Der Node, welcher die MobileCloud verlässt, sendet allen Nodes in seinem *Leaf-Set* eine *leave* Nachricht, um ihnen mitzuteilen, dass er die MobileCloud verlässt. Anschliessend ist das Verlassen der MobileCloud für ihn erledigt.

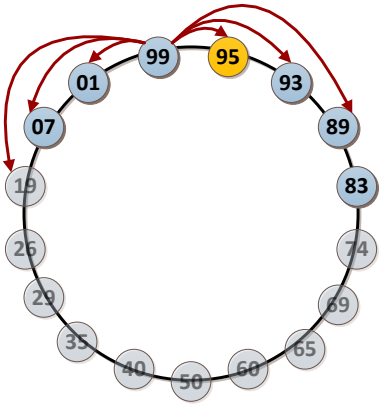


Abbildung 36 Node verlässt die MobileCloud

2. Die Nodes, welche diese Nachricht erhalten, aktualisieren die Seite ihres *Leaf-Sets*, in welcher der ausgeloggte Node gespeichert war. Dafür fordern sie die linke Seite des *Leaf-Sets* des direkten linken, oder falls der ausgeloggte Node auf der rechten Seite war, die rechte Seite des rechten Nachbarnodes an.

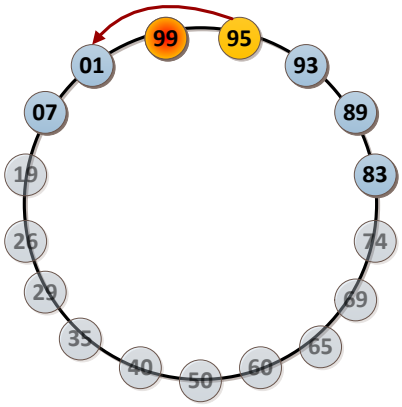


Abbildung 37 Leaf-Set Anfrage

3. Der Nachbar sendet die Node-Adressen aus der angeforderten *Leaf-Set*-Seite zurück.

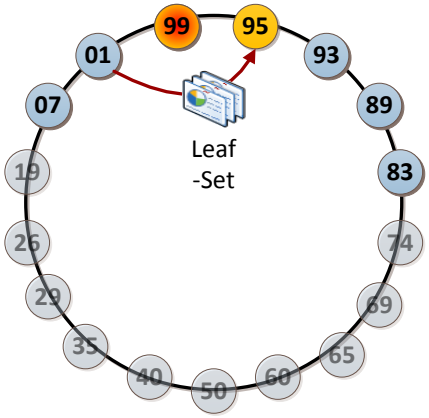


Abbildung 38 Leaf-Set wird zurückgesendet

4. Der erhaltenen Node-Adressliste fügen die Nodes noch die Absenderadresse hinzu und entfernt die Adresse des ausgeloggten Nodes. Mit diesen Adressen kann nun das *Leaf-Set* wieder auf einen gültigen Zustand aktualisiert werden.

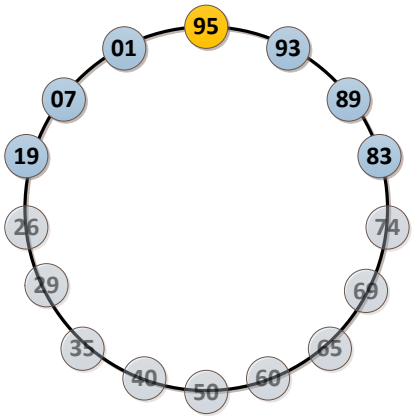


Abbildung 39 aktualisierte MobileCloud

3.3.8.2.4 SendData

Dieser Service behandelt das Versenden von Daten innerhalb der MobileCloud. Momentan beschränkt sich dies auf Text.

Sequenzdiagramm

	SendDataHandler
	DataReceiveHandler

Tabelle 9 Legende sendData

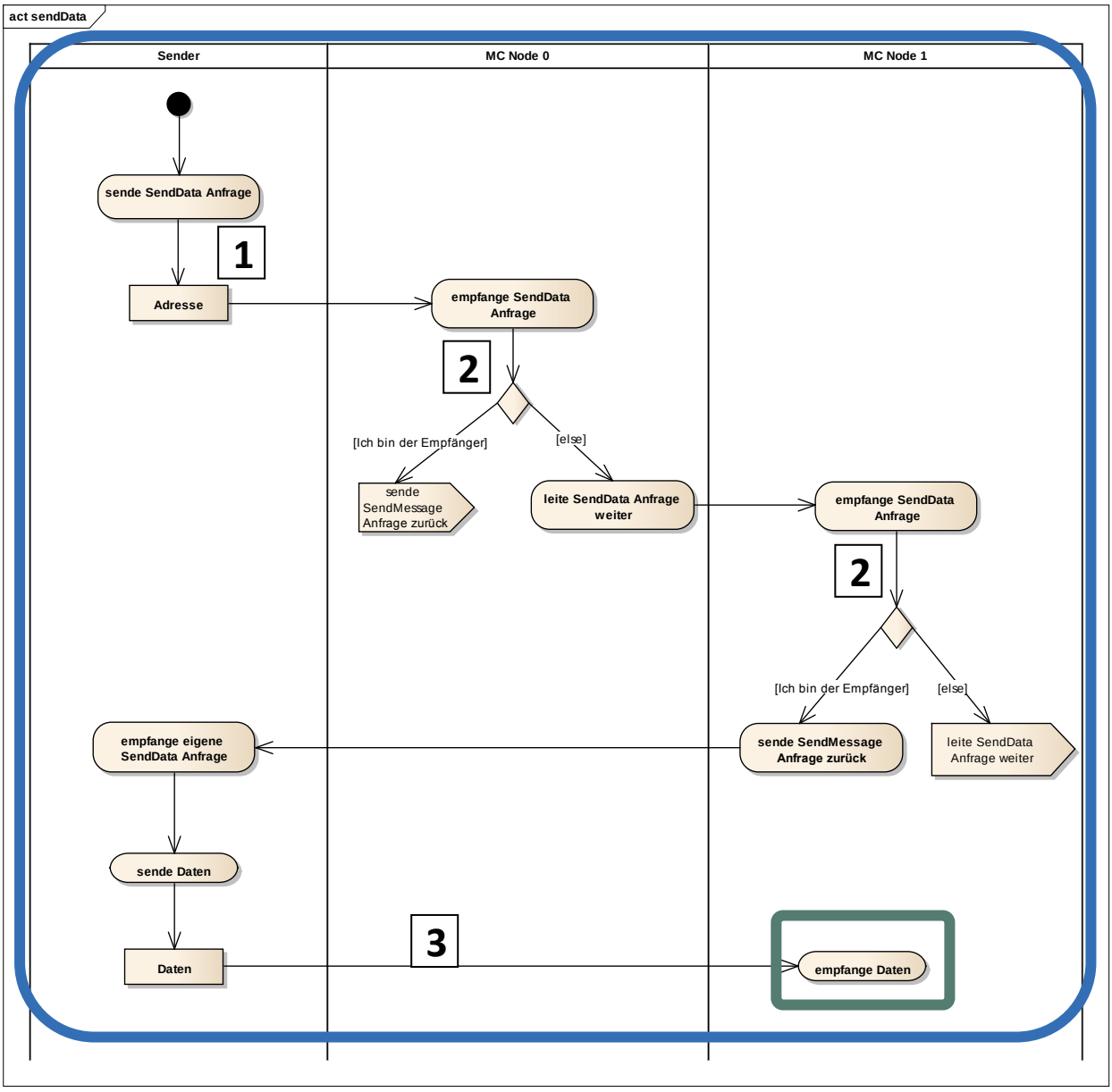


Abbildung 40 Aktivitätsdiagramm SendData

Ablauf

Der Ablauf von *sendData* wird anhand des Sequenzdiagrammes beschrieben.

1. Ein Node möchte an einen anderen Node Daten senden. Dazu muss er eine Anfrage an den Empfänger senden.
   Ist die Node-Adresse nicht in seinem *Leaf-Set*, sendet er die Anfrage an den Node, welcher ganz rechts aussen im *Leaf-Set* ist. Ansonsten direkt an den eigentlichen Node.

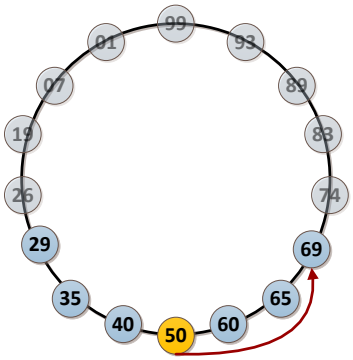


Abbildung 41 Senden einer Anfrage

2. Der Empfänger der Anfrage überprüft, ob er den gewünschten Empfänger der Daten ist. Wenn nicht, leitet er die Anfrage an den eigentlichen Empfänger weiter. Ist dieser nicht in seinem *Leaf-Set* enthalten, leitet er die Anfrage an den rechts äussersten Node seines *Leaf-Sets* weiter.
   Wenn er der eigentliche Empfänger ist, initialisiert er einen Handler für den Empfang von Daten und sendet eine Bestätigung an den Sender-Node.

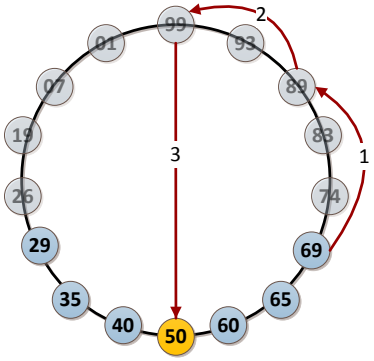


Abbildung 42 Beantworten der Anfrage

3. Nach dem Erhalten der Bestätigung, werden die Daten versendet.

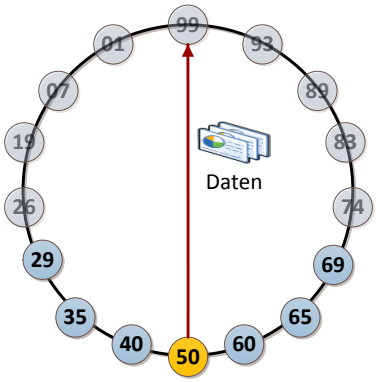


ABBILDUNG 43 SENDEN DER DATEN

Fehlerbehandlung

Node ist nicht in der MobileCloud:

Ist der Node, an welchen die Daten gesendet werde möchten, nicht in der MobileCloud, wäre ab Punkt zwei der folgender Ablauf:

2. Der Empfänger der Anfrage überprüft, ob er den gewünschten Empfänger der Daten ist. Wenn nicht, überprüft er, ob der eigentliche Node in seinem *Leaf-Set* sein müsste. Ist das nicht der Fall, leitet er die Anfrage wie oben bei Punkt zwei beschrieben weiter.
   Falls der Node jedoch in seinem *Leaf-Set* sein müsste, bedeutet das, dass dieser nicht existiert. Deshalb sendet er dem Sender-Node eine Fehlernachricht zurück.

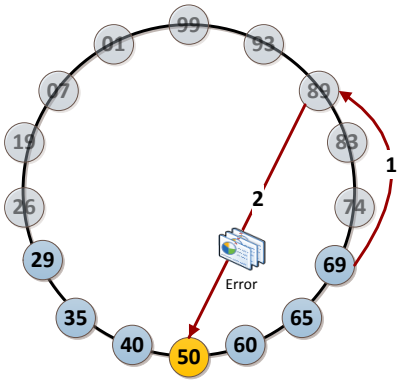


ABBILDUNG 44 FEHLER ALS ANTWORT

3.3.9 Prototyp

Bei der Implementierung sind einige unvorhergesehene Schwierigkeiten aufgetreten. Daher verzögerte sich die Fertigstellung um zwei Wochen. Allerdings konnten dafür einigen Teile implementiert werden, welche erst in einer späteren Version geplant waren.

Dieser Prototyp ist im Source Verwaltungstool Git mit dem Tag 0.2.0 gekennzeichnet und kann mittels des Befehles `git checkout 0.2.0` verfügbar gemacht werden.

Im Folgenden wird kurz der Entstandene Prototyp beschreiben.

3.3.9.1 Client

Auf dem Client sollte MobileCloud in einer finalen Version als Service auf Android laufen. Weil das einige Schwierigkeiten beim Testen und Debuggen mit sich bringt, wurde ein einfaches GUI für den Client erstellt. Der Softwareteil, welcher die Visualisierung macht, spricht die unterliegenden Schichten nur über die Klasse “ClientNode“ an. Dies ermöglicht ein Einfaches umstellen auf einen Service. Auch ist es dadurch möglich ohne grosse Anpassungen die Funktionalität auf den Server zu portieren.

Für die Visualisierung wurden Reiter mit folgenden Funktionen verwendet:

- Admin: Hier kann man der MobileCloud beitreten oder verlassen. Dafür muss man alle nötigen Angaben machen und auf Join/Leave drücken.
  Solange man nicht in der MobileCloud angemeldet ist, ist nur dieser Reiter sichtbar.
- Send/Receive: Hier kann man einem anderen Node eine Textnachricht senden. Auch werden hier einkommende Textnachrichten angezeigt.
- Nodes: Hier werden die Verwaltungsdaten des Nodes angezeigt. Dies wird hauptsächlich für Testzwecke verwendet.

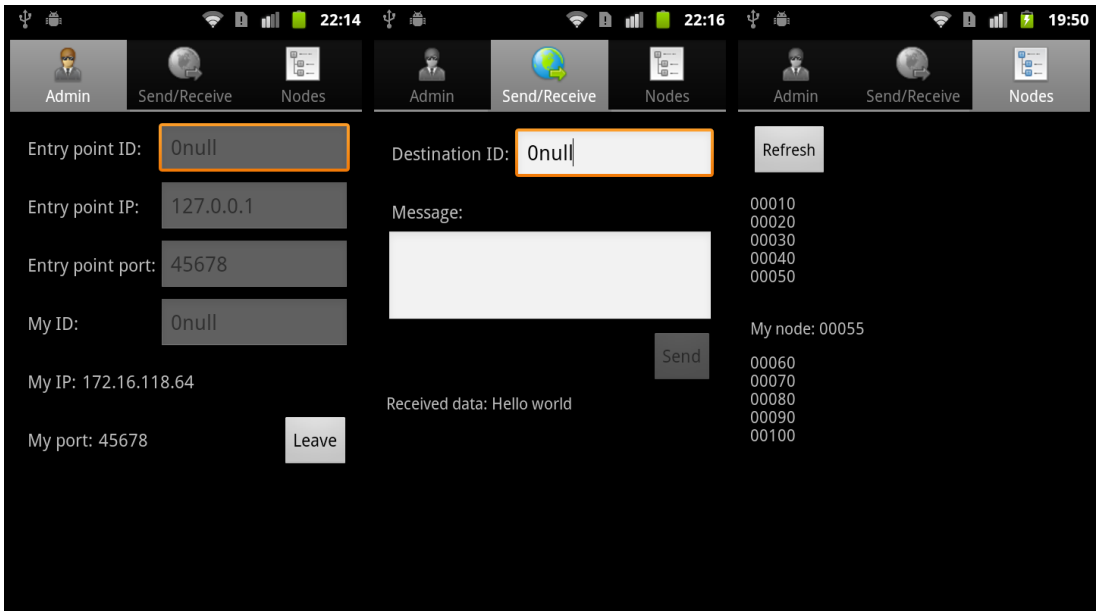


Abbildung 45 GUI des zweiten Prototyps

3.3.9.2 Server

Der Server wurde wie in der Version 0.1.0 als Java-Konsolenapplikation implementiert. Grundsätzlich gibt es keinen funktionalen Unterschied zwischen dem Android Client und dem Server bis auf die Laufzeitumgebung. Der Server könnte auch als Java-Client bezeichnet werden.

Der Server wurde nicht für einen aktiven Einsatz entwickelt, sondern nur, um ohne mobile Geräten eine MobileCloud aufzubauen. Dies ist hauptsächlich für Testzwecke notwendig.

Der Server kann über die Konsole auf zwei verschiedene Arten gestartet werden.

- Mit allen parameter als Konsolenargumente

```
java -jar ./MCServer.jar [eigene ID] [Eintrittspunkt ID] [Eintrittspunkt IP]
[Eintrittspunkt Port]
```

- Interaktiv, ohne Konsolenargumente

```
java -jar ./MCServer.jar
Entry point ID: [Eintrittspunkt ID]
Entry point IP: [Eintrittspunkt IP]
Entry point port: [Eintrittspunkt Port]
```

Nachdem sich der Server angemeldet hat, können folgende Befehle benutz werden:

Befehl	Beschreibung
getIp	Gibt die eigenen IP auf der Konsole aus.
getPort	Gibt den Port, auf den der Server hört auf der Konsole aus.
getId	Gibt die ID des Servers auf der Konsole aus.
listNodes	Listet alle linken und rechten Nodes aus dem *Leaf-Set* auf der Konsole auf.
quit	Meldet den Server von der MobileCloud ab und beendet das Programm.

Tabelle 10 Befehle für den Server

3.3.11 Entwicklungsinfrastruktur

Ein grundlegender und wichtiger Punkt bei der Studienarbeit war es, eine stabile und nützliche Infrastruktur aufzubauen. Dazu gehört ein Versionsverwaltungssystem, ein Continuous Integration Server [16] und ein Projektmanagement-Tool. Diese Produkte wurden bereits im Voraus getestet und konnten zu Projektbeginn auf einem von der HSR zu Verfügung gestellten Server installiert werden. Der Server ist über http://www.mobilecloud.ch.vu erreichbar.

3.3.11.1 Versionsverwaltungssystem

Als Versionsverwaltungssystem wurde Git mit Gitosis zur Benutzerverwaltung verwendet. Git ist ein verteiltes Versionsverwaltungssystem und wird oft als Nachfolger von Subversion[17] gehandelt. Es ermöglicht ohne direkte Serververbindung *Commits* zu tätigen, oder frühere Versionen zu laden. Die Benutzerverwaltung wurde über Gitosis mit SSH-Keys realisiert, was auf Windows-Clients zu gewissen Problemen führte.

Für die Entwicklung wurde ein Zweig namens *dev* erstellt. Auf diesem findet die aktuelle Entwicklung statt. Die aktuelle Version kann mit dem Befehl `git checkout dev` verfügbar gemacht werden.

Für jede Version wird ein Tag erstellt, damit zu jeder Zeit auf diese zurückgegriffen werden kann. Diese können mit dem Befehl `git checkout [Versionsnummer]` verfügbar gemacht werden.

Weil der entwickelte Code nicht aller frei zugänglich gemacht werden wollte, wurde vorerst auf eine Lösung mit Gitorious[18] oder GitHub verzichtet.

3.3.11.2 Continuous Integration Server

Als CI-Server wurde Jenkins verwendet. Jenkins stellt nur die Grundfunktionalitäten zur Verfügung. Durch Hinzufügen von Add-ons kann diese erweitert werden. So konnte auch mit einigem Aufwand, Android Applikationen verwaltet werden. Für die Verwaltung und Überwachung wird ein Webinterface zur Verfügung gestellt.

Das konfigurieren der einzelnen Schritte wurde in einem ANT-File gemacht. Dies ermöglicht die Build- und Testvorgänge über Git zu verwalten. Android stellte ein rudimentäres *Build Script* zur Verfügung. Bei diesem mussten noch einige Anpassungen und Erweiterungen gemacht werden. Das *Build Script* "build.xml" befindet sich im Projekt MobileCloudTest.

Der CI-Server übernimmt folgende Funktionen:

- Vorbereiten und starten es Android Emulators
- Erstellen der MobileCloud Applikation
- Testen der MobileCloud Applikation
- Testabdeckung berechnen und auswerten
- Javadoc[19] erstellen und publizieren

[16] Ein Continuous Integration Server wird in der Informatik für die Automatisierung von Builden, Testen, Erstellen, usw. der Software.
[17] Apache Subversion (SVN) ist eine Server-Client Software zur Versionsverwaltung.
[18] Gitorious oder GitHub sind Webbasierte Plattformen, welche für Open source Programme kostenloses Git Hosting anbietet

3.3.11.3 Projektmanagement-Tool

Für das Planen und das Reporting des Projektes, wurde das freie Tool Redmine verwendet. Redmine ist webbasiert und ermöglicht so wie Jenkins einen Zugriff über den Browser.

Durch den Einsatz von Redmine konnte jederzeit den Projektstand überwacht werden. So wurde auch schnell sichtbar, wenn der Zeitplan nicht mehr stimmte und es konnten Massnahmen eingeleitet werden.

Redmine wurde für folgende Aktivitäten verwendet:

- Zeitplanung
- Zeiterfassung
- Projektübersicht

3.3.11.4 Erfahrung

Nachdem das Einrichten des Servers einige Zeit in Anspruch genommen hatte, erleichterte dieser die Arbeit und erhöhte die Softwarequalität.
Das einzige Problem war das Android Update auf die Version 4.0. Obwohl die Version von MobileCloud auf 2.3.3 blieb, änderten die Buildfiles und es mussten einige Änderungen gemacht werden, damit der Server wieder sauber lief.

[19] Javadoc ist ein Dokumentationswerkzeug, das aus Java-Quelltexten HTML-Dokumentationsdateien erstellt

3.3.12 Tests

Das Testen in einem verteilten System ist bekannter Weise ziemlich schwierig und aufwändig. Da es jedoch unumgänglich ist, dass Software getestet wird, wurden zwei Ansätze gewählt. Als erste mit Unit Test und als zweites mittels Visualisierung der Informationen der Nodes in der MobileCloud.

Unsere Tests führten wir stets mit mobilen Geräten durch, somit wurde sichergestellt, dass die Software auch auf verschiedener Hardware läuft. Mit folgenden Modellen wurde die Software getestet:

- HTC Desire Z
- Google Samsung Nexus S
- HTC Desire
- HTC Desire HD

3.3.12.1 Unit Test

Für das Testen der einzelnen Klassen wurden Unit Test geschrieben. Diese sind in einem separaten Projekt "MobileCloudTest" abgelegt.
Das Ziel war es nicht eine hundertprozentige Testabdeckung zu erhalten, sondern nur die wichtigsten und komplexesten Funktionalitäten zu testen.

Die Auswertung der Tests am Ende der Studienarbeit ergab:
Anzahl Tests: 150

Was	Abdeckung in Prozent	Abdeckung absolut	Abdeckung maximal
Klassen	87%	33	38
Methoden	81%	185	229
Blocks	75%	3542	4737
Linien	79%	779	991

Tabelle 11 Testabdeckung

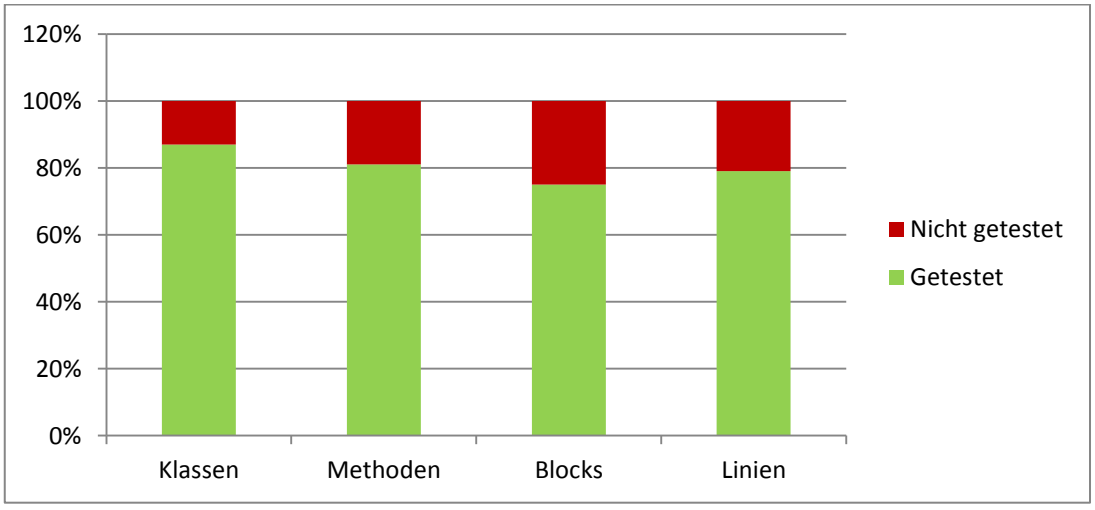


Abbildung 46 Testabdeckung

3.3.12.2 Visualisierung von Informationen

Weil das Testen der MobileCloud Applikation über mehrere Nodes automatisch sehr aufwändig und fehleranfällig wäre, wurde eine einfache Möglichkeit erstellt, um den Zustand der einzelnen Nodes zu überprüfen. Sobald ein mobiles Gerät in der MobileCloud angemeldet ist, gibt es die Möglichkeit das *Leaf-Set* zu visualisieren. Dies ist unter dem Reiter Nodes zu sehen, wie das in der **Abbildung 47 Visualisierung Leaf-Set auf Client** zu sehen ist.

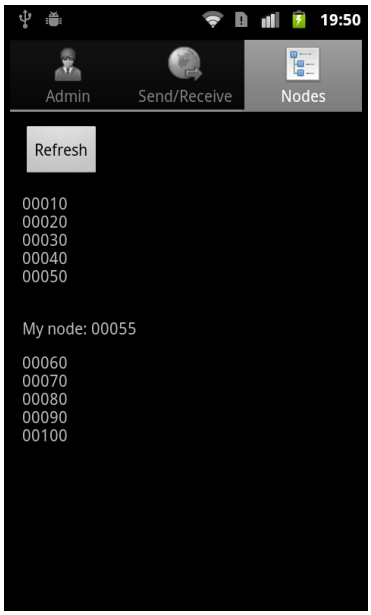


Abbildung 47 Visualisierung Leaf-Set auf Client

Auch für den Server wurde eine solche Möglichkeit implementiert. Weil es eine Konsolenapplikation ist, gibt es kein schönes GUI wie auf dem Client. Nachdem sich der Server angemeldet hat, kann durch eingaben von `listNodes` und anschliessendes Bestätigen mit der Entertaste, das *Leaf-Set* auf der Konsole ausgegeben werden, wie das in der **Abbildung 48 Visualisierung Leaf-Set auf Server** zu sehen ist.

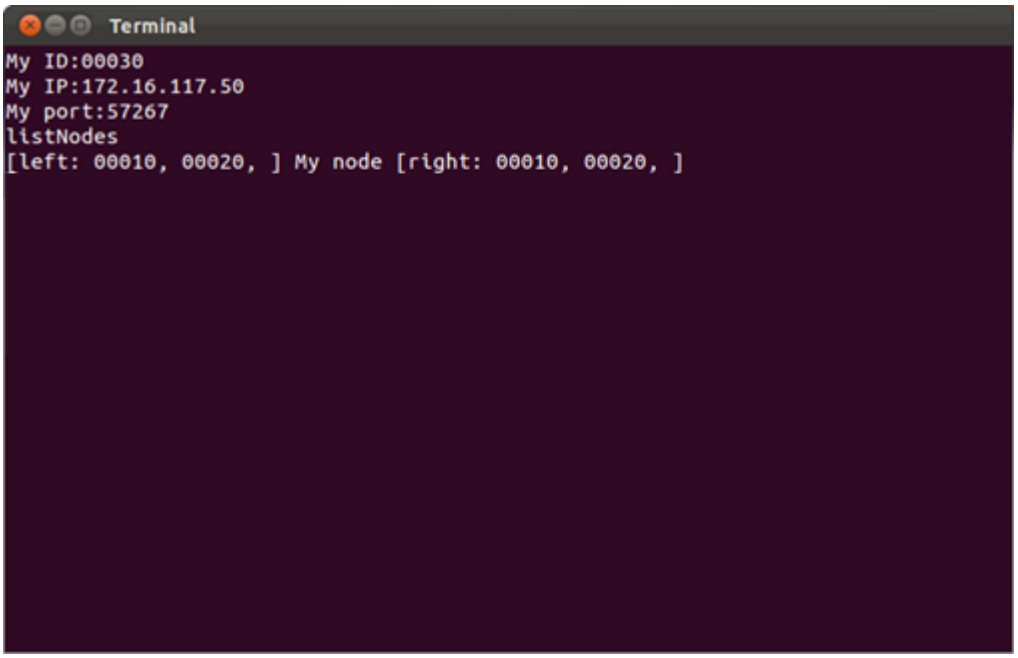


Abbildung 48 Visualisierung Leaf-Set auf Server

Durch diese Visualisierung ergibt sich die Möglichkeit, nach dem Aufbau der MobileCloud zu überprüfen, ob alle Nodes in der MobileCloud sind und ob sie auf dem Ring richtig eingeordnet wurden.

4 Schlussfolgerung

4.1 Zusammenfassung

Als Erstes wurde die Infrastruktur eingerichtet. Das beinhaltete das Einrichten des Servers sowie der Entwicklungsumgebungen auf den Computern. Das Einrichten der Server nahm mehr Zeit in Anspruch als zuerst gedacht. Es war nicht ganz einfach das Jenkins auch für Android Projekte sauber genutzt werde konnte. Der Server wurde während des Projektes nach und nach erweitert, zum Beispiel mit der Anzeige der Testabdeckung oder dem automatischen generieren von Javadoc.

Zu Beginn des Projektes, war die Aufgabenstellung relativ ungenau definiert, da zuerst noch untersucht werden musste, was überhaupt möglich ist. Nachdem der erste Prototyp fertiggestellt werden, konnte anhand dessen Auswertung der nächste Schritt festgelegt werden.
Weil beim ersten Prototyp relativ wenige Probleme aufgetreten sind, wurde beim Planen angenommen, dass noch ein zweiter und dritter Prototyp erstellt werden könnte. Schlussendlich musste der dritte Prototyp gestrichen werden. Dafür wurde einige Funktionalität des dritten in den zweiten Prototypen verschoben.

Das primäre Ziel dieser Arbeit wurde erreicht indem gezeigt werden konnte, wie mobile Geräte miteinander vernetzt werden können. Mit dem zweiten Prototyp konnten sogar zwei sekundäre Ziele erreicht werden. Einerseits konnte ein mobiler Cluster aufgebaut und ein Nachrichte-Service implementiert werden.
Mit dem abschliessenden Prototyp wurde eine stabile Grundlage geschaffen, auf welcher aufgebaut werden kann.

Was genau erreicht wurde und was nicht, ist im Kapitel des jeweiligen Prototypen zu finden.

4.2 Projektauswertung

4.2.1 Codeauswertung

Folgende Code Metriken wurden mit dem Tool Structure 101 ermittelt.

Beschreibung	Anzahl in MobileCloud	Anzahl gesamt
Jars	1	3
Packeges	6	10
Klassen	38	83
Zeilen mit Bytecode Instruktionen (NI)	6‘173	22‘000
Zeilen mit Code (LOC)	2‘654	9‘000

Tabelle 12 Code Metriken

4.2.2 Zeitauswertung

Die folgende Zeitauswertung bezieht sich auf die aufgewendete Zeit beider. Als Vergleich wurde der Durchschnittswert nach ECCS-Punkten angenommen. Zu Beginn wurde eher weniger Zeit aufgewendet, was jedoch ab Mitte des Projektes nachgeholt wurde.

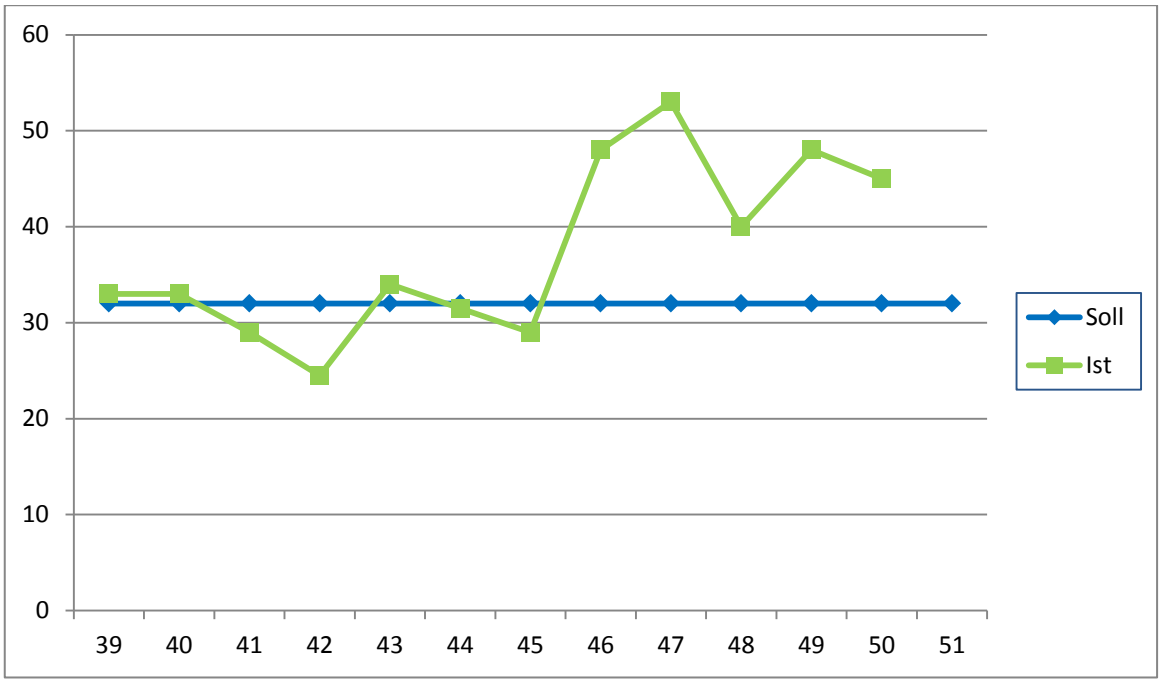
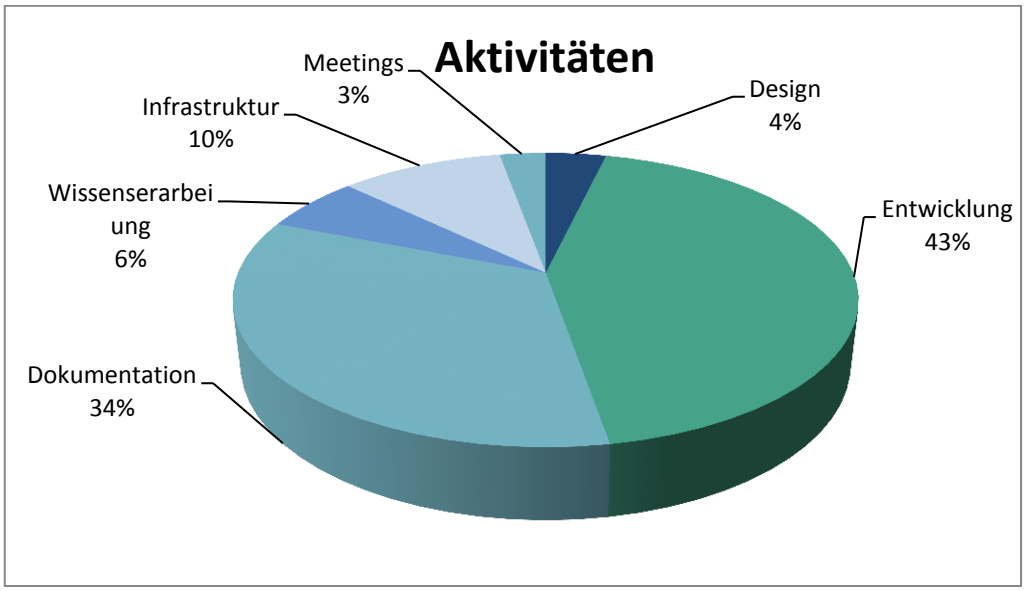


Abbildung 49 Aufgewendete Zeit pro Woche

Die folgende Grafik gibt Auskunft über die Aufteilung der aufgewendeten Zeit. Die grössten Teile nehmen die Entwicklung und die Dokumentation in Anspruch, zusammen über drei Viertel der gesamten Zeit. Für das Design wurde eher weniger Zeit aufgewendet, da viel Design während der Entwicklung entstand und daher in diesen Bereich fällt.



4.3 Ausblick

Nicht alle Funktionen, welche untersucht wurden konnten im zweiten Prototyp implementiert werden. Wichtige Funktionen oder Anpassungen, welche in einem Fortsetzungsprojekt angeschaut werden müssen sind:

- Implementation einer gescheiten NAT-Traversierung
- Datenschutz und Konsistenz
- Vollständige UDP Fehlerbehandlungen
- Ausarbeitung einer sinnvollen, verteilten Fehlerbehandlung
- Fehlertoleranz erhöhen
- Funktionsfluss der Handler nochmals überarbeiten
- Implementation als Android Service
- Erhöhen auf Android Version 4.0

5 Persönliche Berichte

5.1 Heinrich Muralt

5.1.1 Thema

Für dieses Projekt war und bin ich immer noch sehr motiviert. Seit Studienbeginn haben mich Themen wie Mutlithreading, asynchrone Kommunikation und P2P-Netzwerke sehr interessiert. Es war mir schon früh klar, dass ich gerne eine Arbeit rund um diese Themen schreiben möchte. Die Module parallele Netzwerkprogrammierung (PnProg) und verteilte Software Systeme (VSS) fand ich daher sehr spannend und gaben für diese Arbeit eine gute Wissensgrundlage.

Als Besitzer eines Smartphones war ich von der Idee das Ganze für mobile Geräte zu entwickeln begeistert. Dies vor allem wegen den denkbaren Möglichkeiten und Nutzen eines P2P-Netwerk für mobile Geräte. Diese Idee wurde kurz von Herrn Prof. Dr. Josef M. Joller während einer VSS-Vorlesung erwähnt. Für mich war damit klar, bei welchem Dozent bzw. Betreuer eine Arbeit in den oben erwähnten Themen zu machen ist. Glücklicherweise war mein Mitstudent und Freund Marcel Steiner ebenfalls sehr interessiert.

5.1.2 Projekt

Zu Beginn hatten wir viel Zeit für die Vorbereitung und das Einrichten des Projekts aufgewendet. Marcel Steiner und ich waren uns einig für die Entwicklung einen CI-Server[20] einsetzen zu wollen. Vor allem für Git und Jenkins musste mein Partner viel Zeit aufwenden. Nichtsdestotrotz hat sich dieser Aufwand gelohnt. Nicht nur wegen dem Nutzen, sondern weil ich dabei auch viel gelernt habe.

Wir hatten viele Freiheiten und konnten alle unsere Ideen in das Projekt einfliessen lassen. Dies fand ich wiederum sehr motivierend und toll. Es hatte jedoch auch Nachteile. Wir mussten uns regelmässig die Ziele wieder in Erinnerung rufen, um nicht zu sehr von den Hauptzielen abzuschweifen. Durch die Interesse und Begeisterung an Lösungen für viele Probleme, die im Bereich dieser Arbeit auftreten, habe ich viel nebenbei gelesen und gelernt. Doch konnte ich dieses Wissen wegen den definierten Ziele und der Zeit nicht immer in die Arbeit einfliessen lassen. Deswegen kam es vor, dass weniger fertiggestellt als angeschaut und angefangen wurde.

Bei der Planung lief anfangs alles gut. Der Aufwand des ersten Prototyps liess sich gut schätzen. Wir hatten eine gesunde Anzahl an Ziele definiert. Diese konnten auch in absehbarer Zeit erreicht werden. Beim zweiten Prototyp war dies jedoch anders. Einerseits gab es Ziele, welche mehr Zeit als gedacht in Anspruch genommen hatten. Anderseits waren auch mehr Ziele für diesen Prototyp definiert worden. Deswegen war die Planung nicht mehr genau und wir mussten Arbeitspakete verschieben.

[20] Continuous integration Server dient als Unterstützung beim Entwickeln. Der ganze Stand des Projekts wird darauf gespeichert und versioniert. Es werden unter anderem automatisch Builds erstellt und Tests durchgeführt.

5.1.3 Fazit

Diese Arbeit hat mir viel Spass bereitet und ich konnte dabei vieles im Bereich Android, P2P und allgemein Netzwerkprogrammierung lernen. Mir hat die Zeit bzw. Möglichkeit, um Fach- und Programmierwissen mit Marcel Steiner auszutauschen, ein wenig gefehlt. Wir konnten z war uns viel gegenseitig erklären und über das Projekt sprechen, doch in der Planung ist dieser Punkt ein wenig untergegangen. Eine feste Zeit für Pair-Programming hätte beispielsweise weitergeholfen.

Bei der Planung und Zielsetzung ist es bei einem Projekt wie dieses wichtig kleinere und dafür besser abschätzbare Ziele zu definieren. Prototypen mit einer übersichtlichen Anzahl an Funktionalitäten sind dabei eine gute Unterstützung und können gut in iterative Entwicklungsprozesse eingeplant werden. Auch wichtig ist, dass ich für spätere Projekte den Fokus viel stärker setzen und mich an diesen halten muss.

Die Zusammenarbeit mit Marcel Steiner empfand ich als angenehm und fördernd. Die Arbeitsaufteilung war einfach und unkompliziert. Wir konnten uns immer einigen und es gab daher keine Unstimmigkeiten.

Wie bereits erwähnt, bin ich immer noch sehr motiviert an diesem Projekt weiterzuarbeiten. Daher freue ich mich bereits sehr auf die Fortsetzungsarbeit.

5.2 Marcel Steiner

5.2.1 Thema

In den letzten Jahren häuften sich die Berichte über illegale Downloads von Software. Oft wurden dabei auch P2P-Netzwerke angesprochen. Das verleite dieser Technologie einen negativen Touch, obwohl es auch Software gibt, welche diese Technologie legal einsetzt, wie zum Beispiel Skype. Diese Tatsache weckte das Interesse weitere Einsatzgebiete für P2P-Netwerke zu finden.
Nachdem in der Vorlesung Verteilte Software Systeme an der HSR erwähnt wurde, dass ein Versuch mobile Geräte zu verbinden nicht sehr erfolgreich war, weckte dies mein Interesse. Nach einem Gespräch mit Prof. Dr. Josef M. Joller machte er den Vorschlag, dies nochmals in einer Semesterarbeit zu versuchen. Durch die grosse Freiheit in der Umsetzung, war für uns ziemlich schnell klar, dass wir dieses Projekt gerne in der Studienarbeit umsetzen würden.

5.2.2 Projekt

Zu Beginn des Projektes mussten wird als erstes uns selber klar werden, welche Richtung wir einschlagen möchten und welche Technologien wir verwenden. Durch die Ähnliche Vorstellung und die bereits im Voraus geführten Diskussion über die Arbeit, war das im Team schnell geklärt und wir konnten mit der Arbeit beginnen.

Für mich war nebst der Arbeit wichtig einen CI-Server einzusetzen. Erstens für eine bessere Softwarequalität und zweitens, um Erfahrung zu sammeln. Die Arbeit für das Einrichten und den Unterhalt habe ich jedoch unterschätzt. Bis alles sauber lief dauerte es länger als gedacht und es mussten zwischendurch immer wieder Anpassungen gemacht werden. Allgemein stellte sich dieser jedoch als sehr positiv heraus, da die Software gründlich geprüft wurde, bevor man einen neuen Commit machte.

Unser Ziel war von Anfang an die Umsetzung dieses Projektes als P2P-Netzwerk. Nach dem ersten Prototyp wurde auch klar, dass sich dieses hervorragend dafür eignete. Schon während des ersten Prototyps befassten wir uns mit den verschiedenen Möglichkeiten, einer P2P-Umsetzung. Dabei wurde uns auch klar, dass wir als Vorlage Pastry verwenden werden. Dieses ermöglichte ein schnelles Routing, was für unser Projekt essentiell war. Uns war auch klar, dass wir Pastry nicht eins zu eins umsetzen konnten, da die Anforderung an unser Netzwerk von dem von Pastry abwich.

Nachdem wir beim ersten Prototyp nur wenige Probleme hatten, änderte sich das beim zweiten. Es gab ein paar Punkte, welche wir mit dem Betreuer besprechen mussten. Es wurde jedoch immer eine Lösung gefunden.
Durch die Verzögerung um ca. zwei Wochen, entschieden wir uns mehr Gewicht auf die Stabilität und weniger auf die Erweiterung der Funktionalität zu legen. Das führte dann dazu, dass wir nicht unseren ursprünglich erarbeiteten und sicherlich sehr optimistisch geplanten Funktionsumfang implementieren konnten. Dafür haben wird eine gute Basis für eine Aufbauarbeit gelegt.

Während der Entwicklungsphase wurde die Dokumentation ein wenig vernachlässigt, was wir gegen Schluss des Projektes nachholen mussten. Wie vielen anderen Informatikern, fiel auch mir das Schreiben von Dokumentationen schwerer als das Schreiben von Programmcode. Das führte dazu, dass ich mich gegen Ende des Projektes mehr motivieren musste als das noch zu Beginn der Fall war.

5.2.3 Fazit

Wir durften ein sehr interessantes Projekt machen, bei welchem wir grosse Freiheiten hatten. Der Lerneffekt war sehr hoch, da viel Gelerntes angewendet und einiges neu gelernt werden konnte. Ein ganz wichtiger Punkt war die sehr angenehme Zusammenarbeit mit Heinrich. Wir hatten meistens die gleichen Vorstellungen und konnten so sehr speditiv arbeiten, ohne grosse Diskussionen führen zu müssen. Auch erinnerte er mich zwischendurch daran, was unsere eigentlichen Ziele sind, wenn ich mal wieder Patternphilie hatte.

Ich freue mich schon auf die Fortsetzung in der Bachelor Arbeit.

6 Abbildungsverzeichnis

7 TABELLENVERZEICHNIS

8 LITERATURVERZEICHNISS

[1] Schweizer Fernesehen, «Tagesschau,» Schweizer Fernesehen, 29 Oktober 2010. [Online]. Available: http://www.tagesschau.sf.tv/Nachrichten/Archiv/2010/10/29/Schweiz/98-der-Jugendlichen-haben-ein-Handy. [Zugriff am 17 Dezember 2011].

[2] F. B. Fink, «Die Zukunft der Mediennutzung ist digital und mobil,» Medienkontakt bei der Young & Rubicam Gruppe, CH-8037 Zürich, 2010.

[3] Android, «developer.android,» Android, 20 Dezember 2011. [Online]. Available: http://developer.android.com. [Zugriff am 20 Dezember 2011].

[4] C. Baun, M. Kunze, J. Nimis und S. Tai, Cloud Computing - Web-basierte dynamische IT-Services, Heidelberg: © Springer-Verlag Berlin, 2010.

[5] P. Mahlmann und C. Schindelhauer, Peer-to-Peer-Netzwerke - Algorithmen und Methoden, Heidelberg: © Springer-Verlag Berlin, 2007.

[6] Oracle, «Java NIO,» Oracle, [Online]. Available: http://docs.oracle.com/javase/6/docs/api/java/nio/package-summary.html. [Zugriff am 19 12 2011].

[7] Oracle, «A Strategy for Defining Immutable Objects,» Oracle, 20 Dezember 2011. [Online]. Available: http://docs.oracle.com/javase. [Zugriff am 20 Dezember 2011].

[8] Oracle, «Class DatagramChannel,» 20 Dezember 2011. [Online]. Available: http://docs.oracle.com/javase/6/docs/api/java/nio/channels/DatagramChannel.html. [Zugriff am 20 Dezember 2011].

[9] J. Rosenberg, J. Weinberger, C. Huitema und R. Mahy, «http://www.ietf.org/rfc/rfc3489.txt,» 2003. [Online]. [Zugriff am 20 Dezember 2011].

[10] Wikipedia, «http://en.wikipedia.org,» [Online]. Available: http://en.wikipedia.org/wiki/NAT_traversal. [Zugriff am 20 Dezember 2011].

[11] G. Coulouris, J. Dollimore und T. Kindberg, The API for the Internet Protocols, 2005.

9 ANHANG

Dokument 1 Projektplan

Dokument 2 Anforderungsspezifikation

Dokument 3 Codereview

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

MobileCloud

MobileCloud

PROJEKTPLAN

VERANTWORTLICH

Name: Heinrich Muralt

ÄNDERUNGEN

Autor	Beschreibung	Datum	Version
hmu	Dokument erstellt	26.09.2011	0.1.0
hmu	Erste Texte in Kapitel 1 bis 3 geschrieben	26.09.2011	0.1.1
hmu	Texte in Kapitel 4-8 geschrieben	30.09.2011	0.1.2
mst	Review und Kapitel 7, 8 ergänzt.	02.10.2011	0.1.3

MITGELTENDE/ÜBERGEORDNETE DOKUMENTE

Dokumentname	Beschreibung
Aufgabenstellung_MobileCloud.docx	Enthält die Aufgabenstellung dieses Projekts
Vorgehensplan.docx	Enthält der genaue Vorgehensplan

INHALT

TABELLEN

1 Einführung

1.1 Zweck

Dieses Dokument gibt eine Projektübersicht der Studienarbeit MobileCloud an der Hochschule für Technik Rapperswil. Es beschreibt die Ziele der Arbeit sowie der Ablauf, die Organisation und weitere Projektmanagement spezifische Informationen.

1.2 Gültigkeit

Die Gültigkeit dieses Dokumentes bezieht sich auf die Dauer der Studienarbeit MobileCloud während dem Herbstsemester 2011/2012. In dieser Zeit können jederzeit Änderung vorgenommen werden.
Das Dokument kann basierend auf dem aktuellen Projektstand als gültig betrachtet werden.

1.3 Definitionen und Abkürzungen

Alle verwendeten Definitionen und Abkürzungen werden im globalen Glossar festgehalten.

2 Projektübersicht

2.1 Ausgangslage

Mobile Geräte (iPhones, iPads, Android basierte Geräte) sind sehr stark verbreitet und verfügen über häufig ungenutzte oder ungenügend genutzte Fähigkeiten.

Falls die mobilen Geräte miteinander/gemeinsam Aufgaben lösen würden/könnten, hätte man in fast jeder Gruppe von mobile Usern ein enormes Rechenpotential zur Verfügung. Heute stecken viele dieser Devices arbeitslos in irgendeiner Tasche.

Wichtige Aspekte einer mobilen Kooperation und Kollaboration sind:

- Privatsphäre: Teilnehmer müssen jederzeit wissen, was auf den mobilen Geräten zurzeit aktiv vor sich geht.
- Sicherheit: Teilnehmer müssen die Gewissheit haben, dass keine unerlaubten und unerwünschten Besucher auf den mobilen Geräten aktiv sind.

Ansätze, welche in dieselbe Richtung gehen und als Anschauungsbeispiele benutzt werden können:

- Viber (auf iPhone und Android): Diese Software stellt Basisdienste zur Verfügung.

Unser Ziel ist weitergehend in Richtung "semantischer" Information (Topics, Social Networks, …)

Die Integration der mobilen Geräte steht in einer ersten Phase im Vordergrund.

2.2 Ziele

Hauptziel dieser Arbeit ist es zu untersuchen, inwiefern mobile Geräte miteinander vernetzt werden können, unter Ausnutzung aller verfügbaren Netzwerk-Technologien (IP basierte Netzwerk-Technologien, mobile Telefonie-basierte Technologien).

Untergeordnetes Ziel ist der Aufbau eines mobilen Clusters aus lokal miteinander kommunizierenden mobilen Geräten. In einer späteren Phase würden wir versuchen die Einschränkung auf "lokale Cloud/Cluster" fallen zu lassen und globale mobile Clouds ins Auge fassen.

Ein weiteres, längerfristiges Ziel ist der anschliessende Ausbau in Richtung "mobile Service Cloud", also die Anbietung unterschiedlichster Services. Diese können entweder noch entstehen; oder es kann sich um Services handeln, welche bereits irgendwo und irgendwie vorhanden sind und eingebunden/benutzt werden.

Ein mögliches Zusatzziel sind Discovery und Reconfigure Dienste, die es erlauben die mobile Cloud laufend der aktuellen Situation anzupassen (mobile Geräte treten bei, mobile Geräte verabschieden sich (oder fallen aus)

2.3 Hauptfeatures

Diese Features sollten am Ende des Projekts untersucht und gegebenenfalls implementiert sein:

- Vom System gemanagte Features:
  - Herstellung von Verbindungen zwischen lokale und untereinander erreichbare mobile Geräte.
  - Bildung und Verwaltung der lokalen Cloud.
  - Herstellung von Verbindung zu anderen Clouds ausserhalb des lokalen Netzes.
  - Datenübermittlung zwischen mobile Geräte unabhängig vom Standort (Cloud).
- Vom Benutzer genutzte Features (über GUI):
  - Aktivierung/Deaktivierung des Diensts
  - Anzeige des Verbindungsstatus
  - Senden und empfangen von Daten

2.4 Erweiterte Features

Diese Features sind Ideen, welche bei genügend Zeit noch untersucht und gegebenenfalls implementiert werden:

- Vom System gemanagte Features:
  - Discovery in der Cloud
  - Reconfiguration der Cloud
  - Verwaltung von Dienste
    - SMS
    - Telefonie
- Vom Benutzer genutzte Features (über GUI):

- Nutzen von Dienste über die Cloud:
  - SMS (Textnachrichten versenden)
  - Tefefonie

3 PROJEKTORGANISATION

3.1 DURCHFÜHRUNG

Das Projektteam besteht aus 2 Mitgliedern. Alle Entscheidungen beruhen auf gemeinsame Beschlüsse und wenn für nötig empfunden, nach Absprache mit dem Betreuer. Das Projekt wird von Herrn Prof. Dr. Josef M. Joller betreut.

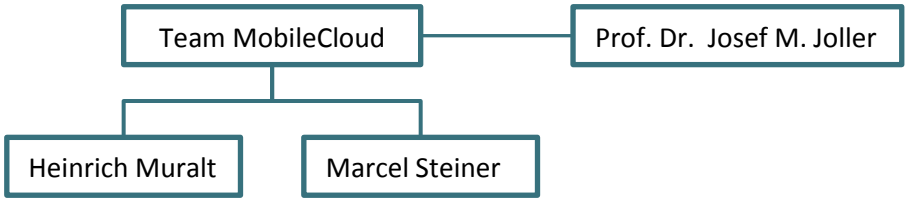
Mit dem Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Das Sitzungsintervall ist flexibel: falls wenig oder keine Probleme auftreten, können die Sitzungsintervalle vergrössert werden. Falls Probleme auftauchen, werden die Intervalle verkleinert.

Des Weiteren sind fest geplante Arbeitsstunden in der Woche für die Arbeit am Projekt geplant:

Freitag: 08^{00}-17^{00}

Durchschnittlich wird pro Woche 16 Stunden am Projekt gearbeitet. Wie die Zeit eingeteilt wird ist jedem selbst überlassen, bis auf die oben erwähnten festen Arbeitsstunden.

3.2 ORGANIGRAMM



3.3 EXTERNE SCHITTSTELLEN

Name	E-Mail Adresse	Rolle
Prof. Dr. Josef M. Joller	jjoller@hsr.ch	Betreuer: Für Beratung und Benotung zuständig

TABELLE 1 EXTERNE SCHNITTSTELLE

4 Management Abläufe

4.1 Projektplan

4.1.1 Iterationsplanung

Da im Software Engineering 2 Projekt RUP umfassend angeschaut und angewendet wurde und dies das am Projektteam bestbekannteste Vorgehensmodel ist, wurde der Projektplan nach RUP erstellt. Es ist jedoch zu beachten, dass sich RUP für dieses Projekt nur beschränkt eignet. Darum wurden gewisse Anpassungen gemacht, welche vom RUP abweichen.

Phase	Start	Ende	Inhalt
Inception 1	19.09.2011	01.10.2011	• Beschaffen der Hardware • Einrichtung Infrastruktur • Projektplan • Literatur beschaffen
Inception 2	03.10.2011	15.10.2011	• Technologierecherchen • Umsetzungsvorschlag • Tests auf Android
Elaboration 1	17.10.2011	29.10.2011	• Anforderungsanalyse • Domainanalyse • Prototyp (mit grundlegendsten Funktionen)
Construction 1	31.10.2011	12.11.2011	• MobileCloud mit Server
Construction 2	14.11.2011	26.11.2011	• MobileCloud Peer to Peer (lokal)
Construction 3	28.11.2011	10.12.2011	• MobileCloud Peer to Peer (global) • Feature freeze
Transition	12.12.2011	23.12.2011	• Code freeze • Kurzfassung • A0-Poster • Abgabe Projekt

Tabelle 2 Iterationsplan

Zu Beginn dieses Projekts muss viel recherchiert werden, um mögliche Ansätze zu evaluiert. Daher umfasst die Inception im Verhältnis zu den anderen Phasen mehr Zeit als üblich. In der Elaboration soll ein erster schlanker Prototyp entwickelt werden, um Erfahrungen zu sammeln und Umsetzungsmöglichkeiten zu testen. In der Construction 1 wird als ein weiterer Prototyp eine erste Version mit Server entwickelt. In der Construction 2 wird darauf aufbauend eine Version ohne Server und Peer to Peer erarbeitet. In Construction 3 soll die Lösung global, d.h. über das Internet, funktionieren. Des Weiteren wird Ende dieser Iteration unabhängig vom Stand der Arbeit keine Features mehr eingebaut. Der genaue Vorgehensplan befindet sich im Dokument Vorgehensplan.docx.

In der Construction wird am Ende jeder Iteration eine neue Softwareversion erstellt; Ende Construction 3 die Release Version. Die Versionsnummer besteht aus drei Teilen:

Versionsnummer	Bedeutung
0.0.*	*= optional, z.B. für Bugfix innerhalb von Iterationsversionen.

0.*.0	*= Iterationsversion: Wird nach einer Iteration um eins erhöht. Steigt die Release Version, wird diese wieder auf 0 gesetzt.
*.0.0	*= Release Version: Wird bei jedem Release um eins erhöht.

4.1.2 Arbeitspakete

Die genaue Planung wird im Redmine erstellt. Diese beinhaltet alle nötigen Arbeitspakete in Form von Issues für die jeweiligen Iterationen. Am Ende jeder Iteration werden die neuen Arbeitspakete für die nächste Iteration eingetragen, diese können jedoch im Verlaufe der Iteration ergänzt oder abgeändert werden. Für jede Iteration gibt es daher einen detaillierten Iterationsplan.

4.1.3 Meilensteine

Die Meilensteine werden im Redmine bei der entsprechenden Iteration eingetragen. Diese sehen wie folgt aus:

Meilenstein	Inhalt	Zeitpunkt	Iteration
MS1	Projektplan erstellt	01.10.2011	Inception 1
MS2	Umsetzungsvorschlag erstellt	08.10.2011	Inception 2
MS3	Anforderungen spezifiziert, Prototyp erstellt	29.10.2011	Elaboration 1
MS4	Lokale Cloud funktionsfähig	12.11.2011	Construction 2
MS5	Globale Cloud funktionsfähig	26.11.2011	Construction 3
MS6	System läuft stabil, Kurzfassung und A0-Poster erstellt, Schlusspräsentation ist vorbereitet	15.12.2011	Transition

Tabelle 3 Meilensteine

5 Risiko Management

Es können unterschiedliche Risiken im Verlauf des Projekts auftreten. Diese wurden in zwei Gruppen unterteilt:

1. Generelle Risiken, die sich nicht spezifisch aufs Projekt beziehen jedoch vorkommen können.
2. Projektspezifische Risiken mit deren Auswirkung und Vermeidung bzw. Lösung.

5.1 Generelle Risiken

- Ausfall einer Arbeitsstation
  - Es sind 2 Arbeitsstationen und 2 Laptops mit der nötigen Entwicklungsumgebung vorhanden. Beim Ausfall einer Arbeitsstation kann auf einen Laptop ausgewichen werden.
- Ausfall des Git-Servers
  - Die Repositories können lokal ausgetauscht werden.
- Ausfall der Netzwerkinfrastruktur
  - Alternative Austauschmöglichkeiten bereithalten (z.B. externe Festplatte).Projektspezifische Risiken

5.2 Projektspezifische Risiken

Risk ID	Risiko	Auswirkung	Massnahme	Kosten der Massnahmen in Stunden	Max. Schaden in Stunden	Wahrschein-lichkeit des Eintreffens	Gewichteter Schaden in Stunden	Priorität
R01	Personelle Defizite	Es entsteht ein zu grosser Zeitaufwand für die Lösung bestimmter Aufgaben.	Möglichst viele Informationen im Voraus beschaffen. Defizite mit lernen/recherchieren beseitigen. Kleine Funktionalitäten mit Prototypen realisieren.	**30**	50	60%	**30**	Hoch
R02	Ziele mit gewählter Richtung/Technologie nicht erreichbar	Projekt kann nicht wie geplant fortgesetzt/beendet werden. Es entsteht ein Mehraufwand durch das Wechseln/ändern der Richtung/Technologie.	Wöchentliche Besprechungen mit dem Betreuer (Expert). Kleine Funktionalitäten mit Prototypen realisieren.	**15**	90	10%	**9**	Hoch
R03	Nicht klare/ungenaue Anforderungen (Experimentier-projekt)	Anforderungen werden nicht erfüllt. Es entsteht ein Mehraufwand durch Änderungen.	Iterative Entwicklung. Rücksprachen mit dem Betreuer.	**15**	60	10%	**6**	Mittel
R04	Zu kleiner Zeitrahmen für die Implementation aller Ziele (Haupt- und Unterziele)	Nicht alle Ziele können implementiert werden. Es entsteht ein Mehraufwand durch das Nachtragen der Ziele.	Ziele zusammen mit Betreuer priorisieren. Bestimmte Ziele, wenn möglich, für diese Projekt wegfallen lassen.	**5**	90	15%	**14**	Hoch
R05	Probleme mit CIS (Continuous Integration Server)	Mehraufwand für das Testen, Bilden, etc .... Qualität sinkt.	VM Image des Servers erstellen.	**10**	30	5%	**2**	Mittel

R06	Zu späte Lieferung der Testgeräte (Android Handys)	Es können nicht alle nötigen Tests ausgeführt werden. Arbeit kann nur eingeschränkt weitergeführt werden.	Fixer Termin mit Betreuer abmachen.	**5**	70	15%	**11**	Hoch
R07	Datenverlust	Mühsame Datenwiederherstellung.	Regelmässiges Backup der Daten, alle Daten unter Versionskontrolle stellen. Backup Recovery testen.	**6**	480	1%	**5**	Mittel
R08	Probleme mit Android Framework	Projekt kann nicht wie geplant fortgesetzt/beendet werden.	Mit Expert Lösung finden.	**10**	40	2%	**1**	Niedrig
R09	Ausfall des HSR WLAN	Bestimmte Funktionalitäten können nicht mehr weiterentwickelt werden.	Eigener Access Point bereithalten.	**2**	30	1%	**0**	Niedrig
	Total Kosten in Arbeitspaketen enthalten			**98**	940			
	Total Rückstellungen						**76**	

6 INFRASTRUKTUR

6.1 ENTWICKLUNGSTOOLS

6.1.1 HARDWARE

Hardware	Beschreibung
Arbeitsstationen	Für die Entwicklung werden die Arbeitsstationen im Raum 1.262 Nr. 11 und 12 verwendet.
Laptop	Unterwegs oder privat werden die eigenen Laptops verwendet.
Server	Als Server für das Redmine, Jenkins und Git sowie die Serverversion wird von der HSR ein virtueller Linux Server zur Verfügung gestellt.
Drucker	Für das Drucken der Dokumente werden die Drucker an der HSR verwendet.

TABELLE 4 HARDWARE- ENTWICKLUNGSTOOLS

6.1.2 SOFTWARE

Software	Einsatzbereich	Link
Git	Wird für die Versionskontrolle aller digitalen Daten verwendet. Gleichzeitig wird die Versionierung der Software mittels Tags in Git festgehalten.	http://git-scm.com
Jenkins	Wird als Integration Server verwendet.	http://jenkins-ci.org/
Redmine	Wird für die Zeiterfassung, die Iterationsplanung sowie für das Bugtracking verwendet.	http://www.redmine.org/
Enterprise Architekt	Enterprise Architekt wird für das Erstellen von UML-Diagrammen verwendet.	http://www.sparxsystems.com/
Ubuntu 11.04	Wird als Betriebsystem für die Entwicklung und den Server verwendet.	http://www.ubuntu.com/
Eclipse mit Android SDK	Wird als Entwicklungsumgebung verwendet.	http://www.eclipse.org http://developer.android.com/sdk/index.html
Apache Server	Wird für den http Zugriff für Jenkins und Redmine verwendet.	http://httpd.apache.org/
Microsoft Skydrive	Wird für die Versionierung und parallele Bearbeitung von Microsoft Office Dokumenten verwendet. Ende jeder Iteration werden diese im Git aktualisiert.	https://skydrive.live.com
FindBugs	Wird für die Fehlersuche im Code verwendet.	http://findbugs.sourceforge.net

TABELLE 5 SOFTWARE- ENTWICKLUNGSTOOLS

7 QUALITÄTSMASSNAHMEN

7.1 ALLGEMEIN

7.1.1 VERSIONSVERWALTUNGSSYSTEM

Für die Versionierung und Verwaltung aller Projekt-relevanten Dateien wird Git eingesetzt. Die Entwicklung findet auf dem dev(Develope) Zweig statt. Nach jedem Synchronisieren mit dem Server testet der Jenkins-Server den neuen Code und veröffentlicht diesen, falls die Tests erfolgreich waren, auf dem master-Zweig. Somit ist auf dem master-Zweig immer eine stabile und getestete Version.

Jede erstellte Version, wird mit einem Tag gekennzeichnet. Tritt bei dieser Version ein Fehler auf, wird vom Tag aus ein neuer Zwei erzeug um diesen zu beheben. Nach dem Beheben wird der HEAD mit der neuen Version getagt und der Zweig wieder gelöscht.

Für die Verwaltung von Office Dokumenten wird in erster Linie MS Windows Live Skydrive eingesetzt. Damit ist gleichzeitiges Bearbeiten desselben Dokuments möglich. Git spielt hier nur eine sekundäre Rolle.

7.1.2 PROJEKTMANAGEMENT-TOOL UND BUGTRACKER

Die gesamte Verwaltung von Arbeitspaketen und das Bugtracking sowie die Zeiterfassung laufen über das webbasierten Projektmanagement-Tool Redmine. Die Pakete (Issues) können leicht verteilt und für die Erfassung der Arbeitszeiten benutzt werden. Entdeckte Bugs können ebenfalls eingetragen und dem Verantwortlichen zugewiesen werden. Die Übersichten über die erledigte Arbeit und das Bugtracking ergeben stets einen guten Einblick auf den Stand des Projektes.

7.1.3 PROJEKTAUTOMATION

Die Projektautomation beinhaltet automatische Builds, Unit-Tests und Code testabdeckungs-Analysen, welche ausgeführt werden, sobald eine neue Version auf dem Git-Server verfügbar ist. Waren die Builds und Tests erfolgreich, werden die Daten auf den master-Zweig synchronisiert und auf den Server geladen.

7.1.4 DOKUMENTATION

Für die Dokumentation werden eigene Formatvorlagen verwendet. Die Dokumentation wird fortlaufend weitergeführt, damit sie möglichst aktuell ist. Bei jeder Änderung führt der Bearbeitende im geänderten Dokument die Versionsnummer nach und schreibt eine kurze Änderungsbeschreibung mit seinem Kürzel. Alle Dokumente werden von beiden Projektteam-Mitgliedern durchgelesen und auf Fehler überprüft.

7.1.5 TEAMSITZUNGEN

Unter den Projektteam-Mitgliedern finden fortlaufend Besprechungen über den Stand der Arbeit, Reviews und die weitere Planung statt. Diese werden nicht protokollier.

8 Software-Qualität

8.1 Code-Qualität

8.1.1 Allgemein

Der Code wird von den 2 Teammitgliedern nach jeder Iterationsversion in der Construction-Phase überprüft. Die Resultate der Reviews werden im Dokument CodeReview.docx festgehalten.

Die Code-Guidelines sind im separaten Dokument CodeGuidelines.docx zu finden.

Für die Automation der Test und Überprüfung der Testabdeckung soll ein CI-Server zum Einsatz kommen.

8.2 Testing

8.2.1 Unit-Tests

Die Module, Klassen und Methoden werden mit Hilfe der Unit-Tests getestet. Diese Tests werden vor dem Ausprogrammieren der Klassen definiert. Sie werden von den Projektteilnehmern während der Entwicklung manuell und nach einem Commit auf dem Server automatisch ausgeführt. Diese Tests zeigen an, ob der Code lauffähig ist und ins Hauptrepository abgelegt werden kann. Es soll eine Mindesttestabdeckung von 80% erreicht werden.

MobileCloud

ANFORDERUNGSSPEZIFIKATION

VERANTWORTLICH

Name: Heinrich Muralt

ÄNDERUNGEN

Autor	Beschreibung	Datum	Version
hmu	Dokument erstellt	17.10.2011	0.1.0

MITGELTENDE/ÜBERGEORDNETE DOKUMENTE

Dokumentname	Beschreibung

INHALT

ABBILDUNGEN

1 Einführung

1.1 Zweck

Die Anforderungsspezifikation bezieht sich auf das Projekt MobileCloud. Das Dokument dient dazu, die Anforderungen an die MobileCloud aufzuzeigen, um sie später überprüfen zu können.

1.2 Gültigkeit

Die Gültigkeit dieses Dokumentes bezieht sich auf die Dauer der Studienarbeit "MobileCloud“ während dem Herbstsemester 2011/2012. In dieser Zeit können jederzeit Änderung vorgenommen werden.
Das Dokument kann basierend auf dem aktuellen Projektstand als gültig betrachtet werden.

1.3 Definitionen und Abkürzungen

Alle verwendeten Definitionen und Abkürzungen werden im globalen Glossar festgehalten.

2 Allgemeine Beschreibung

2.1 Produkt

Momentan gibt es noch keine allgemeine P2P Lösung für mobile Geräte. Das Produkt, welches mit diesem Projekt entwickelt wird, soll als Service auf mobile Geräte im Hintergrund laufen. D.h. es muss von verschiedenen auf den Geräte installierten Applikationen benutzt werden können, um die Dienste der mobilen Cloud zu benutzen.

Eine Applikation zur Ansteuerung des Service ist nicht teil der Anforderungen.

2.2 Einschränkungen

Das Produkt soll für Android 2.2.3 entwickelt werden. Dadurch wird es auf allen Android (ab Version 2.2.3) Geräte lauffähig sein.

2.3 Annahmen

Es wird davon ausgegangen, dass die Verbindungen zu den IP- bzw. mobile Telefonie-basierte Netzwerke von den Android Geräte selbst übernommen verwaltet werden.

3 Funktionale Anforderungen

3.1 Use Case Diagramm

Folgende Use Cases geben eine Übersicht über die funktionalen Anforderungen an die MobileCloud. Der Primary Actor ist dabei der Android Benutzer, welcher eine App zur direkten Ansteuerung des MobileCloud Service bedient. Auf diese Weise ist ersichtlich, welche funktionalen Anforderungen an den Service nötig sind.

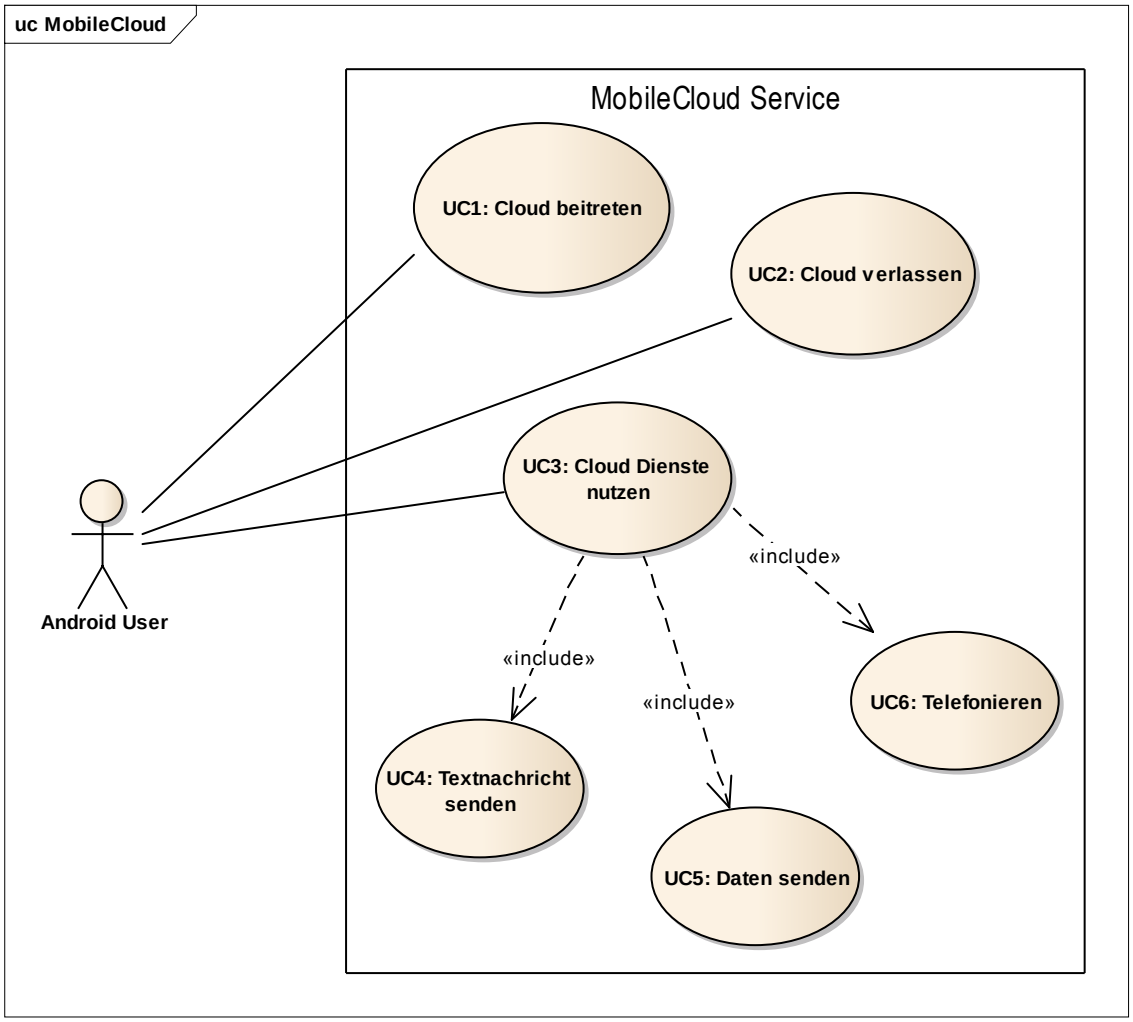


ABBILDUNG 1 USE CASE DIAGRAMM

3.2 USE CASE BRIEF

3.2.1 UC1: CLOUD BEITRETEN

Primary Actor	Android Benutzer
Beschreibung	Der Benutzer tritt der Cloud bei, welches die Nutzung der in der Cloud vorhandenen Dienste aktiviert.

3.2.2 UC2: CLOUD VERLASSEN

Primary Actor	Android Benutzer
Beschreibung	Der Benutzer verlässt die Cloud. Die Nutzung der Dienste wird dabei deaktiviert.

3.2.3 UC3: DIENSTE NUTZEN

Primary Actor	Android Benutzer
Beschreibung	Der Benutzer kann angebotene/aufgelistete Cloud Dienste benutzen. D.h. der MobileCloud Service gibt dem Benutzer auf Wunsch eine Liste der vorhandenen Dienste zurück.

3.2.4 UC4: TEXTNACHRICHT SENDEN

Primary Actor	Android Benutzer
Beschreibung	Der Benutzer sendet eine Nachricht an ein anderes mobiles Gerät. Er übergibt dafür dem Service der gewünschte Dienst, die Nachricht und Telefonnummer des Empfängers.

3.2.5 UC5: Daten senden

Primary Actor	Android Benutzer
Beschreibung	Der Benutzer sendet Daten an ein anderes mobiles Gerät. Er übergibt dafür dem Service der gewünschte Dienst, die Daten und Telefonnummer des Empfängers.

3.2.6 UC6: Telefonieren

Primary Actor	Android Benutzer
Beschreibung	Der Benutzer startet eine Sprachübermittlungssession und beendete diese nach dem Gespräch. Dabei übergibt er für die Initialisierung der Session der gewünschte Dienst und die Telefonnummer des Empfängers.

Dieser Use Case dient als Beispiel für die Verwendung eines Beispiel-Dienst und ist nicht teil der Anforderungen.

3.3 Nichtfunktionale Anforderungen

3.3.1 Funktionalität

3.3.1.1 Nachrichtenübermittlung

Alle übermittelten Nachrichten werden beim korrekten Empfänger mit den originalen Daten empfangen, falls dieser online ist. Bei Abwesenheit des Empfängers wird die Nachricht verworfen.

Folgende Funktionale-Ausnahmen gelten für die Nachrichtenübermittlung:

- Sicherheitsaspekte wie beispielsweise Man-in-the-middle Attacken
- Netzwerkverbindung wird unterbrochen
    - Diese Ausnahme zählt nicht zu den Fehlern, sollte aber korrekt abgehandelt werden. D.h. es darf kein Systemabsturz oder ähnliches daraus resultieren.

3.3.1.2 Identifikation und Datenschutz

Daten zu Identifikation und Ansteuerung der mobilen Geräte im Netzwerk müssen, wenn in den übermittelten Nachrichten enthalten, geschützt werden. D.h. sie dürfen nicht als Klartext aus der Nachricht ausgelesen werden können.

3.3.1.3 Schnittstelle zum Service

Der Service muss für mehrere Applikationen auf dem mobile Geräte zur Verfügung stehen. Es soll jedoch möglich sein, dass eine Applikation einen bestimmten Dienst während der Verwendung für sich allein beanspruchen kann.

3.3.2 Zuverlässigkeit

3.3.2.1 Netzwerk und Reaktionsverhalten

Der Service soll bei folgenden Netzwerkproblemen Reaktionsfähig bleiben:

- Netzwerkverbindung nicht vorhanden
- Firewalls (lokal oder global) blockieren Verbindung
- Falsche Netzwerkkonfigurationen (z.B. IP Adresse)
- Unerwarteter Unterbruch der Netzwerkverbindung

3.3.3 Wartbarkeit / Änderbarkeit

MobileCloud muss ohne grossen Aufwand mit Service erweiterbar sein. Eine Erweiterung soll ohne Anpassung der bestehenden Architektur eingebaut werden können.

Die Änderung eines Service soll keinen Einfluss auf andere Service haben.

3.3.4 Übertragbarkeit

In erster Linie wird für Android SmartPhones entwickelt. Für das Testing muss es möglich sein die Cloud künstlich aufzublasen.

MobileCloud

CodeReview

Verantwortlich

Name: Heinrich Muralt

Änderungen

Autor	Beschreibung	Datum	Version
hmu	Dokument erstellt	11.10.2011	0.1.0
hmu	Review	01.12.2011	0.2.0

INHALT

1 Einführung

1.1 Zweck

Dieses Dokument soll Code-Reviews festhalten.

1.2 Gültigkeit

Die Gültigkeit dieses Dokumentes bezieht sich auf die Dauer der Studienarbeit "MobileCloud“ während dem Herbstsemester 2011/2012. In dieser Zeit können jederzeit Änderung vorgenommen werden.
Das Dokument kann basierend auf dem aktuellen Projektstand als gültig betrachtet werden.

2 Überschrift 1

2.1 Review 1

Datum:	Donnerstag, 22. Dezember 2011
Zeit:	13:10Donnerstag, 22. Dezember 2011
Ort:	Rapperswil
Teilnehmer:	4

2.1.1 Geprüfte Dokumente

Dokument	Filename
LeafSet	LeafSet.java (Package: ch.hsr.mobilecloud.services)

2.1.2 Review Team

Funktion im Review	Name	Datum
Sitzungsleiter	Heinrich Muralt	01.12.2011
Sekretär	Heinrich Muralt	01.12.2011
Präsentator	Marcel Steiner	01.12.2011
Advocat Diaboli	Franziska Schäpper	01.12.2011
Anwenderin (IT)	Simone Schneider	01.12.2011

2.1.3 Liste der Beanstandungen

Nr	Code-Zeile	Beanstandung	Wichtigkeit max = 10	Korrigiert=✓ Pendent=X
1	106	Redundante Abfrage in der „if“-Bedingung.	6	✓
2	101	Methode gibt nicht an, dass eine Exception geworfen werden könnte. („throws“-Deklaration fehlt)	9	✓
3	604-606	Math.min verwenden.	5	✓
4	619-621	Math.max verwenden.	5	✓
5	502	Methodenname mit Rechtschreibefehler. Funktionalität der Methode daher nicht ganz klar.	2	✓
6	397	Methodenname mit Rechtsschreibefehler.	2	✓
7	471	Synchronisationsmechanismus fehlt.	10	✓
8	485	Synchronisationsmechanismus fehlt.	10	✓

2.1.4 ZUSAMMENFASSUNG

Ergebnis

✓Akzeptiert

Nicht akzeptiert

Dokument bleibt so

Kleinere Revisionen notwendig

Grössere Revisionen notwendig

✓ Dokument nochmals überarbeiten

Die Review ist unvollständig

Rapperswil, 22. Dezember 2011

Der Protokollführer:

Heinrich Muralt