Ende-zu-Ende-Messungen von Mobilfunkparametern mit Smartphones

Bachelorarbeit

von

Malte Olfen

aus

Kleve

vorgelegt am

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

Juni 2013

Betreuer:
Norbert Goebel, M. Sc.

Zusammenfassung

In dieser Bachelorarbeit wird die Portierung eines vorhandenen Messframeworks für Mobilfunknetze auf Android-Endgeräte durchgeführt. Das zentrale Ziel dabei ist es zu zeigen, dass Ende-zu-Ende-Messungen von Mobilfunkparametern unter Verwendung dieses Messframeworks mit Smartphones möglich sind. Die aktuelle Messumgebung besteht aus speziell angefertigten Hardwareclients (Alix), die gegen einen Server messen, gegen den auch in der vorliegenden Arbeit gemessen wurde. Mithilfe des Android Native Development Kit (NDK) wurde der bestehende C++ Code des Messframeworks in eine Android-Anwendung eingebettet. Dies wurde durch eine Verknüpfung von Java und C++ realisiert.

Das Ausführen eines bestehenden, komplexen C++ Programms in einer Androidumgebung ist eine relativ junge Disziplin. Eine Herausforderung ist dabei die notwendige Implementierung von in Android-C++ noch nicht vorhandenen Funktionen, die allerdings teilweise durch Open-Source-Android-Communities zur Verfügung gestellt werden. Für alle übrigen Funktionen ist eine Neuimplementierung notwendig.

Zum Vergleich der durch Android gewonnenen Messdaten wurden ebenfalls Messungen mit der Alix durchgeführt. Dabei zeigten sich große Diskrepanzen in der Konstanz der mittleren Latenzen beider Messclients. War bei den Alix-Messungen die Latenz relativ gleichbleibend, so schwankte sie in den Android-Messungen stark. Außerdem machten sich Einflüsse einer ungenauen Zeitsynchronisierung auf dem Android bemerkbar. Auffällig dabei waren vor allem die negativen Latenzen in den Android-Download-Messungen, welche in den Vergleichsmessungen mit der Alix nicht auftraten.

Insgesamt lassen die Analysen der Testmessungen aber auf Android als nutzbare Messumgebung schließen.

Danksagung

Bei meinem Betreuer Norbert Goebel möchte ich mich für die Geduld während der Umsetzung und die fachlich anregenden Diskussionen bedanken.

Besonders möchte ich mich bei Johanna für die Rücksicht und Unterstützung während der Arbeit danken.

Zuletzt noch ein großer Dank an alle Studenten im Rechenlabor und unserem Systemadministrator Thomas Spitzlei, welche mir immer mit Rat und Tat zur Seite standen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Durch den ständigen Fortschritt der Technik und neue Technologien werden Mobilfunknetze ständig weiterentwickelt und der Zugang zu diesen Netzen immer erschwinglicher. Mit sinkender Netzwerklatenz, einer hohen Netzabdeckung und steigender Datenrate ist das Mobilfunknetz ein wichtiges Medium für den privaten und auch wirtschaftlichen Sektor. Während noch vor einigen Jahren die Grundidee der mobilen Sprachübertragung im Vordergrund stand, ist der Schwerpunkt auf die Datenübertragung übergegangen. Das früher leitungsvermittelnde Mobilfunknetz ist heute zu einem paketvermittelnden Netz geworden. Die in drahtgebundenen Netzen schon gut erforschten Parameter, verfügbare Datenrate und Latenz, sind im Mobilfunkbereich noch ein junges Forschungsgebiet. Diese Parameter stehen im Fokus der vorliegenden Arbeit und werden anhand eines speziell an der Heinrich-Heine-Universität Düsseldorf (HHU) entwickelten Messframeworks erhoben. Als Startpunkt wurden die Messverfahren für drahtgebundene Netze verwendet und für Mobilfunknetze optimiert [Goe10, Wil12]. Dieses Framework arbeitet mit speziell für diesen Zweck zusammengestellter und konfigurierter Hardware [Lan13] und ist somit weder kostengünstig noch leicht zu vervielfachen.

In der heutigen Zeit besitzt der Großteil der Bevölkerung ein Smartphone und die Rechenleistung dieser Geräte ist im Laufe der letzten Jahre stark gestiegen. Somit steht der Nutzung von Smartphones als Clientseite nur noch eine fehlende Messsoftware im Wege. Die vorliegende Arbeit befasst sich mit der Aufgabe ein bestehendes C++ Messframework auf einem Smartphone umzusetzen.

1.1 Aufbau der Arbeit

In Kapitel 2 wird das vorhandene Messframework, die Plattformentscheidung, sowie die Android-Systemarchitektur, Android-SDK, Android-NDK und NTP dargelegt. Es folgt in Kapitel 3 die Erläuterung möglicher Umsetzungsformen des Messframeworks in Android. Hier wird erörtert, ob eine javabasierte Neuimplementierung oder eine native Portierung stattfinden soll. Außerdem wird der Implementierungsprozess mit dem NDK dargelegt. Anschließend werden in Kapitel 4 die Herausforderungen im Implementierungsprozess und der Prozess der realen Portierung dargestellt. Die Realisierung der Datenübertragung zwischen Java und C++, sowie das Einfügen von fehlenden Funktionen in C++ bilden hier den Schwerpunkt. Außerdem wird eine nicht verzichtbare C++ Funktion als Assemblerfragment realisiert. Eine Auswertung und Interpretation der Android-Messdaten findet in Kapitel 5 statt. In Kapitel 6 werden eine kurze Zusammenfassung der Arbeit und weitere Optimierungsmöglichkeiten der Umsetzung vorgestellt.

Kapitel 2

Grundlagen

Im foldenden Kapitel werden rudimentäre Grundlagen über das verwendete Messframework und ein Netzwerkzeitsynchronisierungsprotokoll erläutert. Es wird eine Entscheidung über das zu verwendende Smartphone-Betriebssystem getroffen. Abschließend werden die Android-Systemarchitektur, die Android-Entwicklungsumgebungen und die Android Open-Source-Communities dargestellt.

2.1 Messframework für Datenraten und Latenz

Das Messframework für Datenraten und Latenz *Rate Measurment Framework* (RMF) wurde am Lehrstuhl für Rechnernetze und Kommunikationssysteme der HHU entwickelt [Goe10, Wil12, Lan13]. Die Aufgabe des RMF ist es, in zeitlich kurzen Intervallen die Datenrate, die Verzögerungszeit und den Packetverlust mithilfe von Ende-zu-Ende Messungen zu ermitteln. Entwickelt wurde das RMF, um Messungen in Mobilfunknetzen unter Bewegung für die Fahrzeug-zu-Fahrzeug-Kommunikation durchzuführen. Das RMF ist in mehrere Threads aufgeteilt, welche Testpakete beim Absenden mit Zeitstempeln versehen. Beim Empfang von einem Messpaket wird die Empfangszeit notiert und diese mit dem Absendezeitpunkt verglichen. Hieraus lässt sich, wenn die Uhren des Senders und Empfängers synchronisiert sind, die Latenz bestimmen. Eine Uhrensynchronisation wie im RMF per Global Positioning System [Nat13] konnte wegen fehlender

Pulse Per Second [MMB+00] Unterstützung im Smartphone nicht genutzt werden. Aus den angekommenen Paketen wird der Paketverlust und auch die Datenrate berechnet. All diese Aufgaben werden von dem RMF, in der Programmiersprache C++, mit verschiedenen Threads realisiert. Die Synchronisierung dieser Threads ist abhängig von sehr genau messenenden Zeitfunktionen.

2.2 Network Time Protocol

Laufgenauigkeiten von Computeruhren sind abhängig von einem verwendeten Zeitgeber. In heutigen Computersystemen wird ein elektronischer Quarzoszillator eingesetzt. Auf Basis eines schwingenden Quarzes, welcher durch seine bestimmte Frequenz als Taktgeber verwendet wird, kann eine genaue Zeitmessung erfolgen. Allerding ist die Schwingfrequenz eines Quarzes nicht konstant, wodurch Gangabweichungen z.B. durch Temperaturveränderungen entstehen. Somit ist es notwendig, dass sich die Uhren von Computersystemen, mit denen Messungen mit Sende- und Empfangszeitstempeln durchgeführt werden sollen, vor dem Messbeginn und während der Messungen synchron halten. Das *Network Time Protocol* (NTP) [MMBK10] ist solch ein Zeitsynchronisationsprotokoll. Mit dem NTP ist es möglich die Zeiten zwischen zwei Computern zu synchronisieren. In der vorliegenden Arbeit wurde die Zeitsynchronisierung mithilfe des *Simple Network Time Protocol* (SNTP) [Mil06] umgesetzt. Dieses synchonisiert nicht, wie im Server und der Alix verwendet, kontinuierlich die Zeit. Solch eine Synchronisierung muss manuell gestartet werden und kann daher nicht kontinuierlich erfolgen. Diese war als Android-Anwendung verfügbar [Ser12], benötigt jedoch Root-Rechte, welche auf dem verwendeten Smartphone Samsung Nexus [Gro13] problemlos eingerichtet werden konnten [And13]. Von der Neuimplementierung von NTP auf der Androidplattform wurde abgesehen, da dies den Zeitrahmen der Arbeit überschreiten würde.

2.3 Smartphone-Betriebssysteme

Mit der fortschreitenden Entwicklung der Technik wurde es möglich auf kleinstem Raum in ein Mobiltelefon eine Computerfunktionalität einzubauen. Schnelle Internetanbindung per WLAN und Mobilfunk wurden realisiert und sogar GPS und HD-Kameras sind in den neuesten Smartphones integriert. Dieser Trend schlägt sich auch in Mobilfunkbesitzerstatistiken in Deutschland nieder: Es zeigte sich, dass sich der Anteil der Smartphonebesitzer aller Mobilfunknutzer von 01/2010 bis 12/2012 von ca. 17% auf über 50% anstieg [Com13] .
Die Rechenleistung der Smartphones ist beachtlich. So besitzt z.B. das in dieser Arbeit verwendete Smartphone einen Dual Core Prozessor mit 1,2 GHz. Somit stellen Smartphones heutzutage eine hinreichende Leistung zum Ausführen von komplexen Programmen zur Verfügung. Es sollte daher möglich sein das vorhandene Datenraten Messframework (s. Kapitel 2.1) auf einem aktuellen Smartphone ohne leistungsbeschränkende Faktoren auszuführen.

Da es verschiedene Betriebssysteme für Smartphones gibt, muss eine Entscheidung für ein zu verwendendes Betriebssystem in der vorliegenden Arbeit getroffen werden. Zur Auswahl standen dabei Google Android OS, Apple iOS, BlackBerry OS und Microsoft Windows Phone . Laut Marktanteilstatistik [Gar13] sind die Anteile im weltweiten Smartphone-Absatz in den letzten 2 Jahren deutlich auf der Seite von Android (ca. 75%) im Vergleich zu iOS (ca. 19%). BlackBerry OS und Microsoft Windows Phone (jeweils unter 5 %) werden, aufgrund ihres sehr geringen Marktanteils, in der Auswahl als Entwicklungsplattform ausgeschlossen.

Für die Entwicklung mit dem Betriebssystem iOS von Apple fallen Kosten für einen Entwickleraccount und weitere Kosten für ein Endgerät an. Auch die Kompatibilität mit dem schon vorhandenen C++ Messframework kann nicht abgeschätzt werden, da in iOS mit der Programmiersprache Objective-C entwickelt wird. Hier wäre ein Neuschreiben des Messframeworks in Objective-C nötig, was den zeitlichen Rahmen dieser Arbeit deutlich überschreiten würde. Der Zugang zu unbeschränkten Rechten in dem ausgewählten Betriebssytem ist ein weiterer wichtiger Faktor, da vor dem Messen mit dem RMF (s. Kapitel 2.1) die Uhrzeiten mit dem Server abgeglichen werden müssen. Ohne uneinge-

schränkte Rechte kann man die Systemzeit eines Smartphones nicht automatisch angleichen. Dieser Root-Zugang zur Systemverwaltung bei iOS ist nur schwierig zu erhalten und führt bei Durchführung zum Verlust der Garantieansprüche des Kunden.

Die Open Handset Alliance [Ope13] hat die mobile Entwicklungsplattform Android [Inc13c] erarbeitet. Sie basiert auf dem Linux-Kernel [TLKO13] und wird als Open-Source-Software weiterentwickelt. Für die Entwicklung mit dem Betriebssytem Android fallen daher keine Kosten für die Software an. Man benötigt ein *Java-Software Development Kit* (JDK) [Cor13] und ein *Android-Software-Development-Kit* (SDK) [Inc13d]. Der Zugang zu unbeschränkten Rechten in der Systemverwaltung kann mit den meisten Android-Smartphones per Root hergestellt werden [And13]. Anwendungen für Android werden in der Regel in Java geschrieben und können auf in C oder C++ (im Folgenden: C/C++) geschriebene native Funktionen zurückgreifen, um einen Geschwindigkeitszuwachs zu erreichen. Das Messframework wurde unter Linux in C++ entwickelt. Diese Funktionalität ermöglicht eine Implementierung des Messframeworks mit Übernahme des bestehenden C++ Kerns. Aus den oben diskutierten Gründen wurde in der vorliegenden Arbeit auf der Androidplattform entwickelt.

2.4 Android-Systemarchitektur

Die Systemarchitektur von Android [Inc13e] teilt sich in vier Schichten auf (s. Abbildung 2.1). Die unterste Schicht ist der Linux Kernel. Diese wird von Android für die Gerätetreiber, Speicherverwaltung, Prozessmanagement und den Netzwerkbetrieb genutzt. In der darauffolgenden Schicht sind die nativen Java-Android Klassenbibliotheken [Inc13a] enthalten, die alle in C/C++ geschrieben sind. Allerdings sind Java-Interfaces vorhanden, durch die man die enthaltenen Funktionen aufruft. Sie umfasst geschwindigkeitskritische Funktionen für verschiedene Grafikverwaltungen (2/3D, Oberfläche und Videocodecs), die Verwendung von SQL und das WebKit. Hier befindet sich auch die Android-Laufzeitumgebung, welche mit der *Dalvik Virtual Machine* (DVM) arbeitet. Diese führt eine effiziente Umwandlung des Quellcodes in Maschinensprache für verschiedene Prozessorarchitekturen aus. Ebenso sind hier die in Java geschriebenen Java-Kernprogrammbibliotheken enthalten. Die vorletzte Schicht ist das Anwendungs-

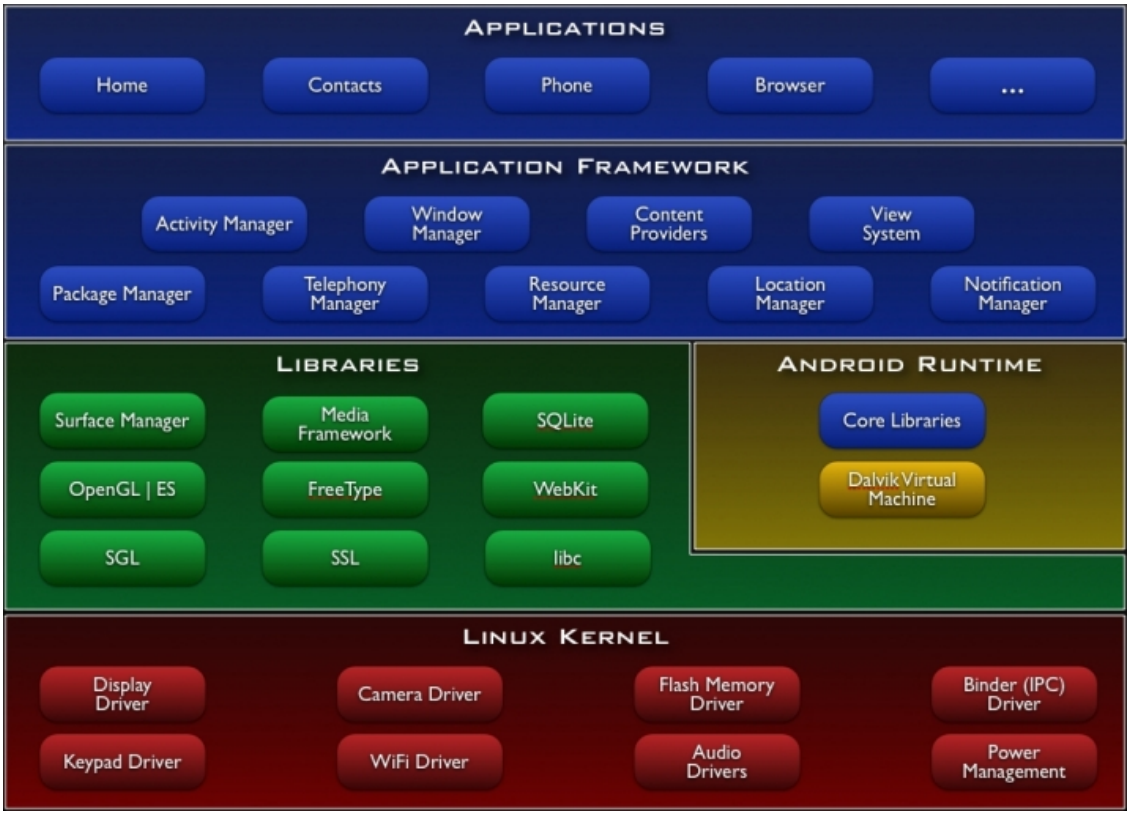

Abbildung 2.1: Android Application Layer.

Programmiergerüst. Hier befinden sich die Verwaltungsprogramme für die Grundfunktionen des Smartphones, wie das Ressourcenmanagement, die Anrufverwaltung und weitere Komponenten, mit denen eine Anwendung interagieren kann. Ein Beispiel ist der Location Manager, welcher mit nur einem Funktionsaufruf im Java Code, per GPS oder Mobilfunkzelle, den Standort an eine Anwendung zurückgeben kann. Die oberste Schicht ist die Anwendungsschicht. Der Quellcode der meisten für die Androidplattform entwickelten Anwendungen ist hier zu finden. Man kann durch Funktionsaufrufe vorhandene Grundfunktionen verwenden oder diese bei Bedarf neu entwickeln. So kann z.B. die Kontaktverwaltung, welche dann an Stelle der ursprünglichen Kontaktverwaltung benutzt wird, mit einem eigenen Anforderungsprofil neu entwickelt werden. Es ist somit möglich, auf der obersten Schicht eigene Android-Anwendungen in Java zu entwickeln und diese dann auf vom Anwendungs-Programmiergerüst bereitgestellte Funktionen zugreifen zu lassen.

2.5 Android-SDK und Android-NDK

Das Android-Open-Source-Project stellt das SDK [Inc13d] für die Anwendungsentwicklung zur Verfügung. Dieses beinhaltet die plattformunabhängige Open-Source *integrierte Entwicklungsumgebung Eclipse* (IDE), das von der Eclipse Foundation seit 2004 als Open-Source-Project entwickelt wird. Im SDK ist die Schnittstelle zur Anwendungsprogrammierung (API) für Android-Endgeräte enthalten. Mit der API lassen sich einfach grafische Benutzeroberflächen erstellen oder anwendungsbasierte Voraussetzungen setzen. Das SDK beinhaltet die *Android Developer Tools* (ADT). In den ADTs sind ein virtueller Gerätemanager, ein Projektassistent, der *Dalvik Debug Monitor Server* (DDMS) und Android Lint enthalten. Mit dem virtuellen Gerätemanager lassen sich verschiedenste Endgeräte und Android-Versionen simulieren und testen. Der DDMS dient dem Debugging, also dem Auffinden von Fehlern, von Programmen und wurde z.B. in der Analyse der Timingproblematik der nativen Threads eingesetzt. Android Lint ist ein Code-Analyzer, der den geschriebenen Android-Code prüft und potentielle Fehler und Optimierungsmöglichkeiten aufzeigt. Hiermit lässt sich eine komplett auf Java basierende Android-Anwendung entwickeln.

Sollen jedoch noch Codefragmente oder sogar ganze Programme, die in C/C++ geschrieben wurden, im Android-Programm genutzt werden, so muss auf das Android-NDK [Knu12, Inc13b] zurückgegriffen werden. Zeitkritische Java-Codefragmente durch C/C++ Code zu ersetzen kann eine höhere Performanz der Android-Anwendung erzielen, jedoch warnt selbst Google auf der Android-NDK Homepage, dass die Komplexitätszunahme der Anwendung dadurch immer gegeben ist und eine erhöhte Performanz nicht zwangsläufig eintritt [Inc13b]. Das Kernstück des Android-NDK ist das Shell-Skript ndk-build. Es ermöglicht die Verbindung der nativen C/C++ Seite mit der Java-Seite, um diese anschließend für verschiedene CPU-Architekturen zu kompilieren.

2.6 Android Open-Source-Communities

In der Entwicklung mit dem NDK nehmen die Open-Source Communities eine besondere Rolle ein. Sie entwickeln eigene Lösungen für nicht implementierte C/C++ Funktionen in eigenen Bibliotheken, die für jeden Nutzer frei verfügbar sind. Ganz besonders aktiv in dieser Entwicklung ist das CrystaX-Team [.NE13], welches eine eigene Version des ndk-build Tools mit vielen erweiterten C++ Funktionen entwickelt. Auch die Weiterentwicklung des Internet-Browsers Mozilla Firefox für Smartphones [Fou13a] oder die Bionic-Bibliothek [Inc13f] sind solche Open-Source-Community-Projects.

Kapitel 3

Portierung des RMF nach Android

Im Folgenden werden die möglichen Alternativen der Portierung abgewogen und eine Übersicht des geplanten Prozesses dargestellt. Dabei sollte der Fokus, einen direkten Vergleich der Messungen von Alix und Android-Smartphones zu ermöglichen, stets im Vordergrund stehen.

3.1 Vorbereiten der Portierung

Zwei Wege der Portierung des RMF in das Android-System waren möglich und mussten gegeneinander abgewogen werden: eine Neuimplementierung auf reiner Java-Basis oder eine Übernahme des bestehenden C++ Quellcodes mit dem NDK.

3.1.1 Implementierung in Java

Da Android in dem SDK eine Entwicklungsumgebung in Java propagiert, ist eine Refaktorierung des RMF in Java eine Möglichkeit. Das momentane RMF, entwickelt in C++, ist sehr hardwarenah. Durch eine komplette Abbildung des RMF in Java könnten viele Funktionen einfacher entwickelt werden, was den Implementierungsprozess beschleunigen würde. Eine komplette Refaktorierung des RMF würde jedoch nur eine

Momentaufnahme der aktuellen Version bedeuten. Da eine Weiterentwicklung des RMF auf C++ und nicht auf Java geplant ist, wäre sehr viel Aufwand damit verbunden, kleine Veränderungen von Methoden und Funktionen in das Java-RMF einzupflegen. Eine fehlerfreie Umsetzung des RMF in einer neuen Programmiersprache ist im zeitlichen Rahmen dieser Arbeit ein weiterer Risikofaktor.

3.1.2 Implementierung mit dem NDK

Mithilfe des NDK von Android ist es möglich Funktionen und sogar ganze Programme über Android auszuführen. Diese Methode stellt einen großen Vorteil dar, da so der C++ Quellcode des RMF in eine aus Java bestehenden Android-Umgebung eingebettet werden kann. Hiermit wird die Ähnlichkeit der Messungen im Vergleich zur kompletten Java-Umsetzung erhöht. Einsparungen in der Entwicklung sind auch gegeben, da nun nicht mehr auf zwei unterschiedlichen Plattformen weiterentwickelt werden muss, sondern ein Update des C++ Kerns bei einer neuen RMF Version einfach durchgeführt werden kann. Ein hohes Risiko besteht allerdings in den Punkten Komplexität der Implementierung und der im NDK realisierten C++ Bibliotheken. Mit dem NDK native Programme auf Android auszuführen kann zu den verschiedensten Fehlern führen, welche in der Alix nicht aufgetreten sind. Besonders die nur sehr rudimentär implementierten C++ Bibliotheken bergen ein sehr hohes Risiko, da bei fehlenden Funktionen die gesamte Implementierung scheitern könnte.

3.1.3 Entscheidung für die Implementierung mit dem NDK

Um möglicherweise in einem späteren Schritt die Messungen von Alix und der hier entwickelten Portierung zu vergleichen, scheint die Entwicklung mit dem NDK von Vorteil zu sein, da in beiden Messframeworks derselbe C++ Kern verwendet wird. Aus diesem Grund wurde diese Methode für die vorliegende Arbeit favorisiert und wird in den folgenden Abschnitten ausführlich dargestellt.

3.2 Grundprinzipien der Implementierung des RMF mit dem Android-NDK

Die native Implementierung eines bestehenden C++ Programms in Android wird in sechs Schritten durchgeführt.

1. Deklaration der nativen Methode in Java,
2. Erstellung des *Java Native Interface* (JNI),
3. Anpassen des Android Makefiles,
4. Implementieren der nativen Methoden in C++,
5. Kompilieren der neuen gemeinsamen Bibliotheken,
6. Linken der neuen gemeinsamen Bibliothek.

Zunächst wurde jedoch eine Java-„Hülle“, in der die Startbefehle, eventuelle Parametereingaben und, sofern gewollt, auch eine Parameterrückgabe erzeugt, entwickelt. Hiermit war es möglich z.B. Startparameter eines Programmaufrufs in Java zu verarbeiten und eventuell mit der Rückgabe Reaktionen von dem nativen Programm anzuzeigen oder über das Debugging auszugeben. In Java wurde die Eingabeparameter als ein String-Array bereitgestellt und dieses wird im nächsten Schritt über das JNI so verarbeitet, dass mit den übergebenen Parametern dann das native C++ Programm gestartet werden konnte. Um ein erfolgreiches Debugging der C++ Schicht auf der Android-Hardware durchführen zu können, war es unerlässlich auch eine Übergabe von C++ nach Java zu implementieren, da sonst eventuelle Fehler beim Kompilieren oder Linken nicht korrekt interpretiert werden könnten. Zusätzlich musste neben der Deklaration der nativen Methoden in Java auch die gemeinsame Bibliothek in die Javaseite eingebunden werden, um die nativen Methoden anzusprechen.

Kapitel 4

Herausforderungen im Implementierungsprozess

Nach der Entscheidung für die native Android-Implementierung und der Erklärung des prinzipiellen Ablaufs dieser Implementierung auf Android wird in diesem Kapitel der reale Implementierungsprozess dargestellt. Die Android-Implementierung wird im folgenden mit ETEMS *(End-to-End-Measurement-System)* abgekürzt. Hier werden besonders die aufgetretenen Herausforderungen sowie die Bewältigung dieser dargestellt.

4.1 Java Implementierung

Um das ETEMS von einem Android-Smartphone ausführen zu können, muss eine Androidanwendung erstellt werden. Diese besteht aus einer einfachen GUI, mit der das RMF auf dem Smartphone gestartet werden kann. Die fest eingestellten Startparameter für das RMF werden hier als String eingebunden. Wichtig ist es in der Java-Seite der Anwendung alle Energiesparfunktionen auszuschalten. Ansonsten wäre es möglich, dass die Prozessoren in der Messung nicht mit maximaler Geschwindigkeit messen. Die auszuschaltenden Energiesparfunktionen umfassten außerdem den Schlafmodus und die WiFi-Einstellung.

4.2 C/C++ Logging-API Pantheios

Die im Messframework verwendete Logging-API Pantheios [Ltd13] ist eine performante C/C++ Logging-Bibliothek. Die Aufgabe von Pantheios ist es, bei reibungslosem Ablauf den Messprozess mit wenigen ausgegebenen Nachrichten darzustellen. Sollte es jedoch zu einem Fehler kommen, kann die Loggingtiefe von Pantheios individuell konfiguriert werden. So kann ein optimales Debugging erfolgen, sodass Fehler schnell gefunden, analysiert und behoben werden können. Vor der Portierung des C++ Messframeworks musste das Pantheios-Logging-Framework jedoch aus dem kompletten Programm entfernt werden. Pantheios besteht aus vielen verschiedenen Klassen und dies würde einen deutlichen Portierungsaufwand alleine für das Logging bedeuten. Falls Fehler im Pantheios-Framework, durch unzureichende C/C++ Bibliotheken im Android-NDK, auftreten, müssten diese zusätzlich entdeckt und behoben werden. In dieser Arbeit liegt der Fokus auf der Portierung eines entwickelten Datenraten-Messframeworks. Somit sind die Statusmeldungen bei der Laufzeit des Programms von besonderem Interesse. Sollte ein Update des Messframeworks erfolgen, so muss lediglich das Logging Framework Pantheios entfernt werden, um dieses Update auch im ETEMS durchführen zu können. Hierfür wurde in dieser Arbeit ein Bash-Skript [Fou13g, Fou13b] erstellt, welches alle notwendigen automatisierbaren Änderungen des RMF-Frameworks anwendet. Das so bereinigte C++ Framework wurde anschließend auf einem Linux Desktop Rechner gegen den Messserver gemessen. Darauf folgende Tests zeigten, dass das RMF uneingeschränkt Messungen durchführte. Der Portierungsaufwand wurde deutlich gesenkt.

4.3 Datentransfer zwischen Java und C++

Um einen Datentransfer zwischen dem Java und dem C++ Teil des ETEMS ermöglichen zu können, waren mehrere Schritte notwendig, auf die im Folgenden eingegangen wird. Besonders der Datenaustausch zwischen C++ und Java wird detailliert erklärt, da dort viele komplexe Aufrufe beachtet werden müssen.

4.3.1 Datentransfer von Java nach C++

Zunächst wird hier der Aufruf von Daten aus Java in C++ beschrieben. Im Anschluss wird die Rückrichtung, von Daten aus C++ in Java, erläutert. Der Aufruf des RMF erfolgt mithilfe von Kommandozeilenparametern, in welchen Server-IP, Server-Port, Client-IP, Client-Port, Messmethode, sowie die zu messende Zeit übergeben werden. Um diese Parameter nun von Java nach C++ zu übertragen, muss in der aufrufenden Java-Datei die gemeinsame Bibliothek geladen sowie eine native Funktion in Java eingebunden und aufgerufen werden. Durch die aus einer Java-Klasse erzeugte Headerdatei *jni_Natives* wird die Funktion für beide Programmiersprachen durch das JNI sichtbar. Der nächste Schritt ist der Zugriff auf die nun erreichbaren Java-Daten von der C++ Seite. Dies wird im Folgenden am Beispiel der Übergabe des String-Arrays mit den Startparametern erläutert. Hierbei handelt es sich um einen Aufruf aus der Java-Klasse Natives, der den Übergabeprozess startet.

public static native int SendInputToCpp(String[] argv)

Diese native Funktion erwartet einen String-Array als Eingabe und gibt einen Integer zurück. Mit diesem Aufruf startet nun die Übergabefunktion auf der C++ Seite.

```
JNIEXPORT jint JNICALL Java_jni_Natives_SendInputToCpp
(JNIEnv *env, jclass clazz, jobjectArray jargv)
{
        // Pointer zur DVM erstellen
        env->GetJavaVM(&g_VM);
        // Länge des Java-Arrays bei der DVM abfragen
        jsize clen = getArrayLen(env, jargv);
        // Char-Array anlegen für die einzelnen Strings aus Java
        char * args[(int)clen];
        //Variablendeklaration
        int i;
        jstring jrow;
        // Auslesen der Einzelstrings starten
        for (i = 0; i < clen; i++)
        {
           // Java-String auslesen
           jrow = (jstring) env->GetObjectArrayElement(jargv, i);
           // Umwandlung des Java-Strings in einen UTF-8 String
```

```
            const char  *row = env->GetStringUTFChars(jrow, 0);

            //...Verarbeitung des ausgelesenen Strings

            // Informiert die VM, dass der Pointer auf
            // den String[i] nicht mehr benötigt wird.
            env->ReleaseStringUTFChars(jrow, row);
        }
        // Aufruf des RMF mit den umgewandelten Parametern.
        main(clen, args);
        // Rückgabewert für die Javaseite
        return 0;
}
```

Um aus C++ Werte an Java zu übergeben, müssen in C++ Datentypen mit einem vorangestellten j bezeichnet werden. Auch die aufgerufenen Klassen müssen als jclass bezeichnet werden, damit diese korrekt interpretiert werden. Der Aufruf der Funktion *SendInputToCpp* ist eine Zusammensetzung, die wie folgt aufgebaut ist : Programmiersprache_Paketname_Funktionsname (JNI Parameter). In den ersten beiden Zeilen befinden sich der genaue Bezeichner der Funktion, der Rückgabewert für Java und die JNI-Parameter. Die JNI-Parameter erfüllen folgende Funktionen:

*JNIEnv *env*

ist ein Pointer zur DVM (s. Kapitel 2.4). Durch diesen Pointer wird es möglich Funktionen anzusprechen, mit denen man Zugriff auf die DVM bekommt, über die mit Java Objekten interagiert werden kann.

jclass clazz

liefert eine Referenz zu der Javaklasse, welche die native Funktion aufruft.

JobjectArray jargv

ist die Referenz zu dem javaseitigen Programmaufruf in Form eines Arrays aus Strings. Dieser wird in ein JNI jobjectArray eingepackt. Zur Übergabe der Informationen an die C++ Seite des RMF ist es notwendig, dass diese noch korrekt referenziert, formatiert,

zwischengespeichert und dereferenziert werden. Nun ist es möglich eine Funktion mit den Eingabeparametern der Javaseite auf der C++ Seite aufzurufen, ohne die im RMF verwendete Hauptmethode verändern zu müssen.

4.3.2 Datentransfer von C++ nach Java

Nachdem das Projekt nun aufgerufen werden kann, ist es unerlässlich eine Möglichkeit der Ausgabe von C++ Nachrichten zu schaffen, um ein Debugging des C++ Originalcodes zu ermöglichen. Der minimale C++ Header-Support des NDK und die veränderte Hardware, auf der das RMF ausgeführt werden soll, können zu Problemen im Programmablauf führen. Zu diesem Zweck wurde eine Funktion in C++ geschrieben, die direkt aus C++ eine in Java vorhandene Funktion mit Parametern aus C++ aufruft.

```
static void jni_send_str(const char * text) {
  // Pointer zur DVM erstellen
  JNIEnv *env;
  // jni_send_str bei der DVM registrieren.
  g_VM->AttachCurrentThread (&env, NULL);
    if (!mSendStr) {
      // Abfrage der DVM-ID für die Methode RecieveMessageFromCpp
      mSendStr = env->GetStaticMethodID
      (jNativesCls, "RecieveMessageFromCpp", (Ljava/lang/String;)V);
     }
  // Aufruf der Java-Methode mit der korrekten Klasse, DVM-ID
  // und des durch C++ in einen String umgewandelte Char-Arrays
  env->CallStaticVoidMethod
  (jNativesCls, mSendStr, env->NewStringUTF(text));
}
```

Die Funktion RecieveMessageFromCpp(String text) führt in Java ein einfaches Syste.out.printl(text) zur Standardausgabe (stdout) aus.

4.3.3 C++ Implementierung

In dem folgenden Abschnitt wird auf alle Herausforderungen in der Portierung des C++ Codes des RMF in den für Android ausführbaren C++ Code im ETEMS eingegangen. Diese waren besonders gekennzeichnet durch einen hohen Aufwand der Fehleranalyse und Entwicklung sowie der Einbindung von in der nativen Android C++ Bibliothek fehlenden Funktionen. Da die Fehlermeldungen des NDK oft viel Interpretationsspielraum ließen, war das Identifizieren der folgenden fehlenden Funktionen ein langwieriger Prozess.

1. **getifaddrs ()**

   Als erste nicht im NDK implementierte Funktion wurde die Funktion getifaddrs [Fou13e] identifiziert. Sie liefert eine Übersicht der Netzwerkschnittstellen des lokalen Systems. Entwickler nutzen das Java-Android-Interface, welches diese Funktionalität enthält. Dieses Problem ist bei der Entwicklung des Web-Browsers Mozilla Firefox für Android entdeckt und gelöst worden. Die Lösung ist es, die nicht vorhandene Funktion in die C++ Seite des Android manuell einzupflegen.

2. **CPU_SET ()**

   Die zweite fehlende Funktion war CPU_SET [Fou13d]. Diese Funktion ermöglicht die manuelle Verteilung von Threads in Mehrkernsystemen. Man erhält also die Kontrolle darüber Threads den Prozessorkernen zuzuweisen. Ein Thread ist eine Abarbeitung von Instruktionen innerhalb einer Anwendung. Die oben erwähnte Zuweisung kann die Ausführung einer Anwendung verbessern. Zeitkritische oder rechenintensive Threads werden je nach Anzahl der verfügbaren Prozessorkernen systematisch auf die einzelnen Prozessorkerne im RMF zugewiesen. Dies dient der optimierten Ausführung des RMF in unterschiedlichen Mehrkernsystemen. Auch diese Funktion wurde schon für die Android Entwicklung nachimplementiert und hier übernommen [Net13].

3. **clock_nanosleep**

   Die dritte fehlende Funktion war clock_nanosleep [Fou13c]. Diese Funktion dient dazu einen Thread bis zu einer gewissen Uhrzeit schlafen zu lassen. Gibt es in einer Anwendung mehrere Threads, können diese parallel arbeiten und zu einem massiven Geschwindigkeitsanstieg in der Ausführung der Anwendung führen. Ein großes Problem ist jedoch die zeitliche Synchronisierung solcher Threads. Im RMF werden Threads zu diesem Zweck bis zu bestimmten Zeiten schlafen gelassen. Da die Funktion clock_nanosleep nicht in der C++ Bibliothek von Android vorhanden war, wurde die in der CrystaX-Bibliothek (s. Kapitel 2.6) vorhandene Funktion nanosleep [Fou13f] verwendet. Diese lässt einen Thread für eine gewisse Zeitspanne schlafen. Hier gibt es keine definierte Uhrzeit, bei welcher der Thread aufwachen soll.
   Nun war es möglich das ETEMS zu kompilieren. Ein Test zeigte zwar eine Kontaktaufnahme und Initialisierung von Client und Server, es wurden jedoch keine Messdaten versendet.

   Da aus der ETEMS-Seite jede Fehlernachricht durch Entfernen von Pantheios neu eingepflegt werden müsste, wurde in einem ersten Schritt der Server im Debuggingmodus gestartet, um die Fehlermeldungen analysieren zu können. Durch die Debuggingnachrichten am Server wurde deutlich, dass nach Erstaustausch von Initialparametern für 5 Sekunden keine Kommunikation vom ETEMS zum Server stattfand. Nun wurde mithilfe des DDMS [Inc13g] ein Verlaufsabbild der Android Threads erstellt. Die nativen Threads (C++ Threads) wurden im ETEMS korrekt initialisiert. Allerdings war vor Abbruch nur noch ein Thread aktiv, welcher immer schlief, kurz aufwacht und dann wieder schlief. Dieses Verhalten lässt auf eine Fehlfunktion der Threadsynchronisierung schließen. Die einzige Veränderung im Quellcode des RMF war das ersetzte clock_nanosleep. Um zu überprüfen, ob der beobachtete Fehler durch das Ersetzen von clocknanosleep durch nanosleep entstanden war, wurde diese Ersetzung in der ursprünglichen Entwicklungsumgebung des RMF erneut durchgeführt. Zusätzlich lieferte dieser Test die Möglichkeit auf der Clientseite mit Pantheios (s. Kapitel 4.2) eine detailliertere Fehleranalyse durchzuführen. Es wurden zwischen einem Desktop-Client und dem Server Testmessungen initiiert und das gleiche Verhalten wie bei den ETEMS-Messungen be-

obachtet. Hier stellte sich heraus, dass der mit der DDMS erkannte Fehler erneut auftritt. Nach dem initialen Austausch der Messparameter startet auf der Clientseite ein Thread eine andauernde Wiederholung von nanosleep.

Somit wurden als zwei Lösungsansätze die Möglichkeit der nativen Implementierung von clock_nanosleep für Android oder das Adaptieren der Threadsynchronisierung des RMF auf die Verwendung von nanosleep identifiziert.

Die Funktion clock_nanosleep ist eine hochauflösende Funktion, da sie es Threads ermöglicht mit einer Genauigkeit im Nanosekundenbereich zu schlafen. Es ist unter Linux möglich mit der Bionic-Bibliothek und dem integrierten Tool gensyscalls.py [Inc12] systemkritische Funktionen in Assembler zu erzeugen. Hiermit war es möglich, eine an die ARM-Hardware des Android angepasste Assemblerfunktion zu generieren. Diese wurde dann anschließend in die Android.mk Datei eingebunden. Ein darauf folgener Test des ETEMS zeigte, dass der vorher beobachtete Threadsynchronisierungsfehler nun nicht mehr auftrat.

Zuletzt traten noch Fehler im Dateizugriff von C++ im Android-Dateisystem auf. Es war nicht möglich eine Datei von C++ ausgehend in dem Android Dateiensystem offen zu halten, um die gemessenen Werte dort festzuhalten. Nach genauer Untersuchung der Kompilierungsreihenfolge der in CrystaX-NDK vorhandenen C++ Bibliotheken konnte dieser Fehler in der Kompilierungsreihenfolge der Standardbibliothek und der CrystaX-Fremdbibliothek gefunden und durch Neuanordnung behoben werden.

Eine Darstellung der einzelnen Komponenten des ETEMS findet sich in Abbildung 4.1.

Erste Messungen zeigten, dass nun auch Messpakete ausgetauscht und Messdaten erfasst wurden. Diese Messdaten werden in Kapitel 5.2 genau beschrieben und ausgewertet.

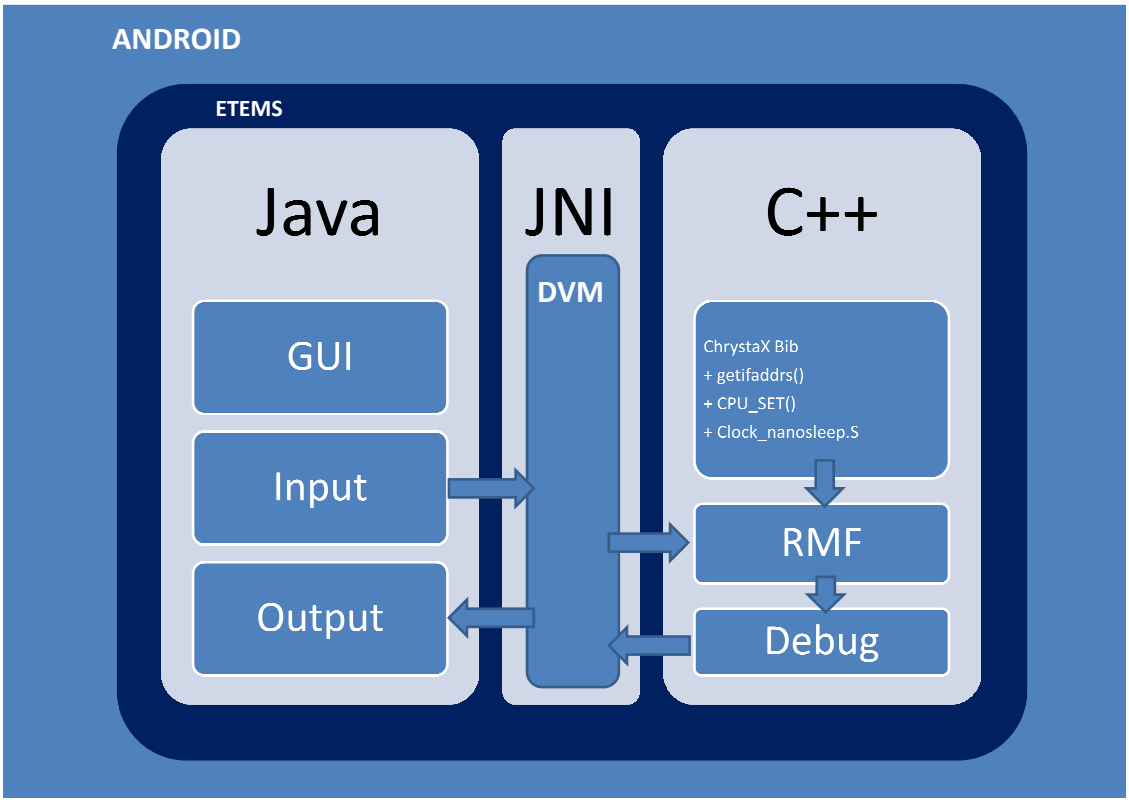

Abbildung 4.1: Aufbau des ETEMS.

Kapitel 5

Messungen und Auswertungen

In diesem Kapitel werden der Messaufbau, die Messdurchführung und die Auswertungen der Messungen, die mit dem ETEMS auf dem Android-Smartphone durchgeführt wurden, beschrieben. Es wurden auch Messungen mit der Alix vorgenommen, sodass ein Vergleich der Messdaten des ETEMS mit denen der Alix erfolgen konnte. Als Darstellungsformen der Datenrate bzw. Latenz wurden zum Einen die Mittelwerte mit Standardabweichung und zum Anderen die kumulative Verteilungsfunktion (Cumulative Distribution Function, CDF) gewählt.

5.1 Aufbau der Messungen

Pro Einzelmessung wurde gleichzeitig in beide Richtungen gemessen. Es wurden also Messpakete sowohl vom Messclient ausgehend an den Server gesendet als auch zeitgleich vom Server an den jeweiligen Client. Für die Messungen wurde ein PC als Messserver benutzt. Die technischen Daten der verwendeten Messhardware können Tabelle 5.1 entnommen werden. Es wurden jeweils fünf Messungen mit dem Android-ETEMS und dem Alix-RMF von jeweils fünf Minuten im Wechsel durchgeführt. Als Messzeiten wurden Zeiten nach 20 Uhr an der Universität verwendet, da das Mobilfunknetz dann als sehr wenig belastet angenommen wurde. Die genauen Zeitabläufe der alternierenden Messungen pro Messclient sind in den Tabellen 5.2 und 5.3 aufgeführt.

	Messserver	Android	Alix
Hersteller	keine Angabe	Samsung	PC Engines
Bezeichnung	keine Angabe	Galaxy Nexus	alix6f2
Betriebssystem	Debian 6.0	Android 4.2.2	Linux
Netzwerk Andbindung	Kabel	Mobilfunk	Mobilfunk
Mobilfunk-Upload		HSUPA max. 5,7 MBit/s	max. 5,7 MBit/s
Mobilfunk-Download		HSPA+ max. 21 MBit/s	max. 7,2 MBit/s
Ethernet Anbindung	1000 Mb/s	nicht vorhanden	10 MBit/s
CPU	1 x 2,8 GHz	2 x 1,2 GHz	1 x 500 MHz
RAM	3 GB	1 GB	256 MB

Tabelle 5.1: Technische Daten der verwendeten Messhardware.

Messung	Messclient	Startzeit	Endzeit
1	Android	20:08	20:13
2	Android	20:20	20:25
3	Android	20:32	20:37
4	Android	20:44	20:49
5	Android	20:56	21:01

Tabelle 5.2: Zeitablauf der alternierenden Messungen mit ETEMS (Android).

Messung	Messclient	Startzeit	Endzeit
1	Alix	20:14	20:19
2	Alix	20:26	20:31
3	Alix	20:38	20:43
4	Alix	20:50	20:55
5	Alix	21:02	21:07

Tabelle 5.3: Zeitablauf der alternierenden Messungen mit Alix.

5.2 Auswertung und Interpretation der Ergebnisse

5.2.1 Datenrate

Abbildung 5.1 stellt die mittlere gemessene, verfügbare Datenrate jeder Messung dar. Als Maß für die Streuung wurde die Standardabweichung angegeben. Die Upload-Datenraten von Android und Alix waren fast identisch. Die Download-Datenraten zeigten allerdings einen Unterschied von Alix zu Android auf. Hier lag die durchschnittliche Differenz der Downloadrate bei 287 kB/s. Ausgehend von den Werten für die Datenrate im Download kann angenommen werden, dass die Differenz in der Datenrate von Alix und Android an dem leistungsfähigeren Mobilfunkmodem des verwendeten Android-Smartphones gegenüber dem der Alix-Messhardware liegt. Vergleicht man den Wert für die maximal mögliche Downloadrate in Tabelle 5.1, liegt der Wert des Android-Smartphones bei bis zu 21 MBit/s im Vergleich von 7,2 MBit/s bei der Alix.

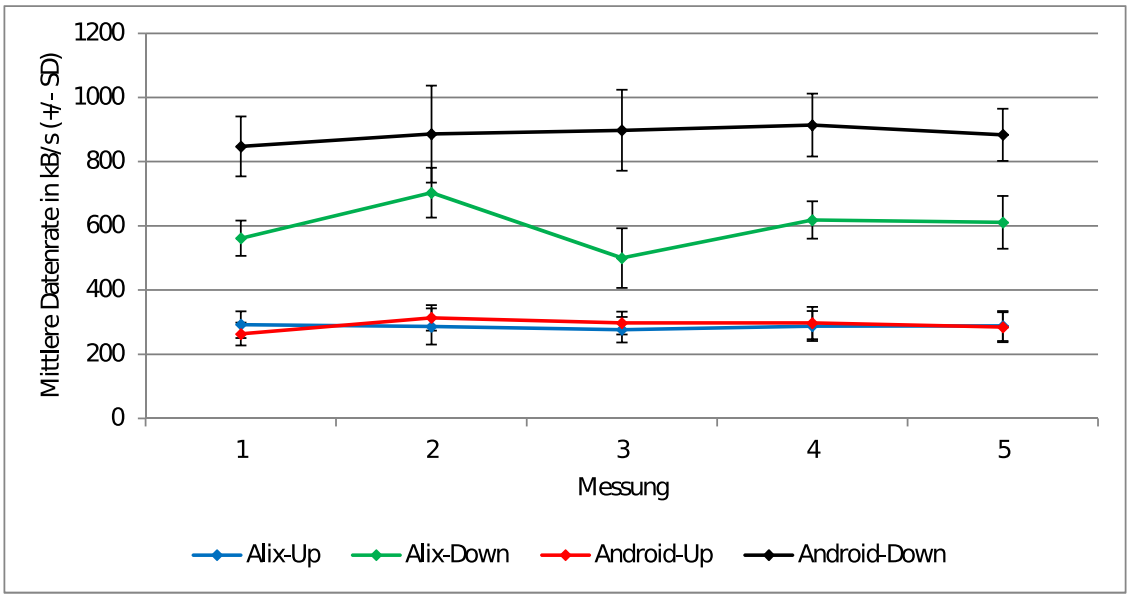

Abbildung 5.1: Mittlere Datenrate in kB/s für Alix und Android.

5.2.2 CDF-Datenraten in Downloadrichtung

Der Vergleich der Häufigkeit der aufgetretenen Messwerte bestätigte die oben beschriebene durchschnittliche Differenz in der Download-Datenrate. Ebenfalls fiel auf der Alix-Seite eine deutliche Stufenbildung auf. Exemplarisch betrachtet befanden sich in der zweiten Alix-Messung 70 Prozent der Messwerte zwischen 702 und 750 kB/s. Der nächste niedrigere Messwert war 662 kB/s und der nächsthöhere 801 kB/s. Es ist möglich, dass solch eine ausgeprägte Stufenbildung, wie sie hier im Download der Alix beobachtet wurde, durch die Programmstruktur des RMF, welche mit mehreren Threads gleichzeitig arbeitet, induziert wurde. Arbeiten mehrere Threads im RMF konkurrierend und wird von jedem Messteilnehmer gleichzeitig gesendet und empfangen, so kann es durch fehlende Rechenleistung zu einer Verzögerung in der Bearbeitung von eingegangen Paketen kommen. Verzögert sich die Bearbeitung, so kommt es auch zu einer verzögerten Empfangszeit. Da hier nur wenige mögliche Werte für die Latenzen auftreten, liegt es nahe davon auszugehen, dass in der Threadpriorisierung des RMF Effekte auftreten, welche solche Verzögerungen verursachen. Zusätzlich bestätigt würde diese Vermutung, wenn in der Upload-Datenrate der Alix keine solchen Verzögerungen auftreten. Dies würde deutlich zeigen, dass der Messserver mit seiner im Vergleich zur Alix höheren Rechenleistung in der Lage wäre die Paketverarbeitung ohne Verzögerung zu bewältigen.

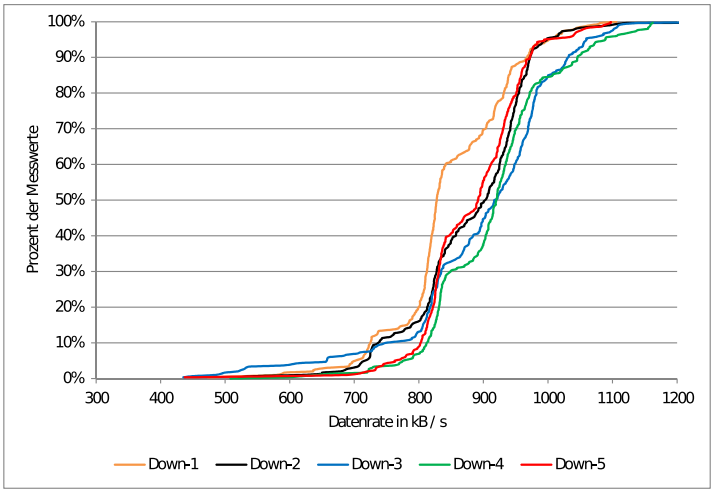

Abbildung 5.2: CDF-Datenrate-Android-Download.

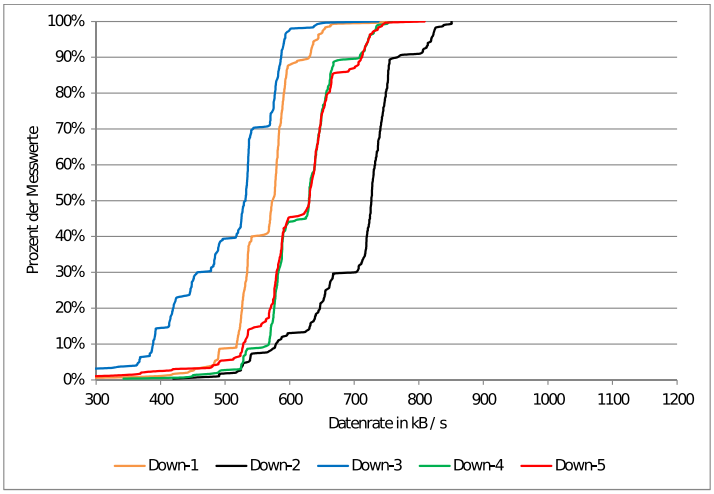

Abbildung 5.3: CDF-Datenrate-Alix-Download.

5.2.3 CDF-Datenraten in Uploadrichtung

In der Darstellung der Upload-Datenrate in Abbildung 5.4 findet sich keine Stufenbildung. Hier war die Verteilung der gemessenen, verfügbaren Datenrate in großen Bereichen gleichmäßig verteilt. Der Vergleich mit den Datenraten des Android in Abbildung 5.5 lieferte ein anderes Bild als Abbildung 5.1. Der Verlauf war nicht so homogen wie bei der Alix. Die Heterogenität der Upload-Datenrate bei den Android-Messungen kann möglicherweise durch die veränderte Lage des Smartphones bei den Messungen erklärt werden. Während der Messungen war die Alix konstant an einem Ort und wurde durch eine Remote-Verbindung gesteuert. So war die Ausrichtung der Antenne im Verlauf aller Messungen gleich ausgerichtet. Dies war beim Android nicht der Fall, da zwischen den Messungen die SNTP-Anwendung manuell eingestellt wurde und auch der Start des ETEMS führte zu Bewegungen des Messgeräts und somit zu einer Änderung der Ausrichtung. Im Verlauf der Upload-Datenrate der Alix war keine Stufenbildung zu erkennen. Wie in Kapitel 5.2.2 diskutiert zeigte sich, dass der Messserver in der Lage war den Upload ohne Verzögerung zu bewältigen. Dies bestärkt die Vermutung der Verzögerung in der Bearbeitung von eingegangenen Paketen durch zu niedrige Rechenleistung der Alix.

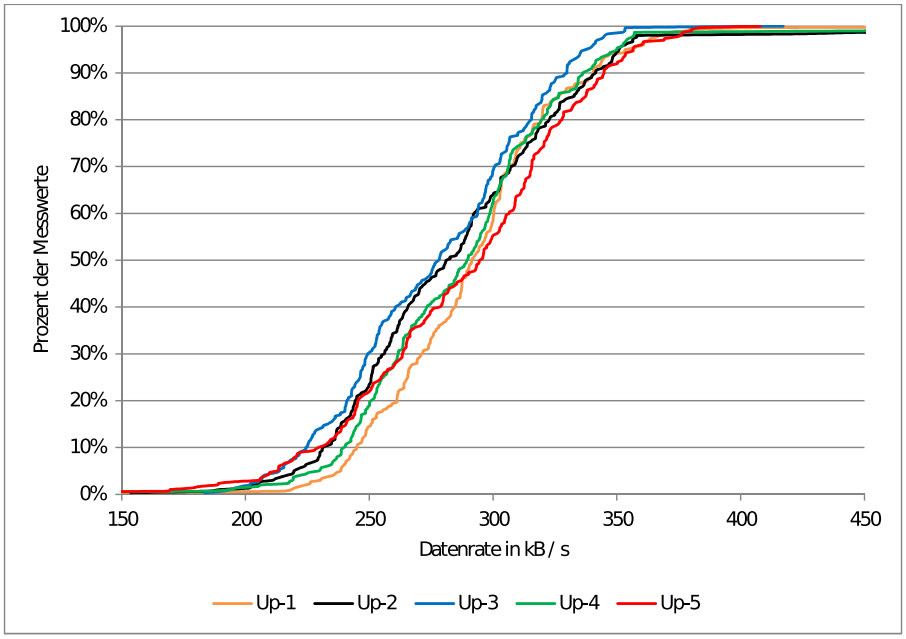

Abbildung 5.4: CDF-Datenrate-Alix-Upload.

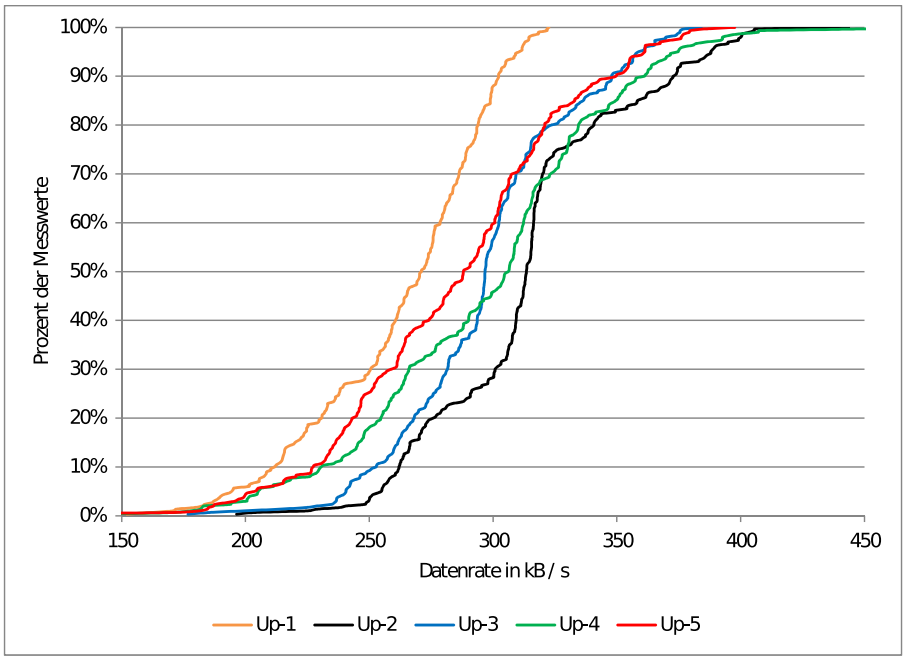

Abbildung 5.5: CDF-Datenrate-Android-Upload.

5.2.4 Latenz

Die Latenz wird pro Messpaket durch die Differenz der Sende- zur Empfangszeit bestimmt. Der Sender fügt seine aktuelle Uhrzeit beim Versand des Pakets als Sendezeit hinzu. Bei Empfang des Pakets fügt dann der Empfänger seine aktuelle Uhrzeit als Empfangszeit hinzu. Nach Abschluss der Messungen werden die gesammelten Pakete ausgewertet und die Latenz bestimmt. Bezüglich der mittleren Latenzen der Messungen, welche in Abbildung 5.6 dargestellt werden, ließ sich eine deutliche Differenz in den verwendeten Clients feststellen. War bei den Alix-Messungen die Latenz sehr ausgeglichen, so schwankte sie in den Android-Messungen stark. Auffällig waren die negativen Latenzen in den Android-Download-Messungen und die große Diskrepanz in den Android-Up- und Download-Mittelwerten. Laufen die Uhren von Sender und Empfänger nicht synchron kann es vorkommen, dass der Empfangszeitpunkt vor dem Sendezeitpunkt liegt. Dies führt dann zu einer negativen Latenz bei der Auswertung der Download-Messdaten des Android. Zu beobachten ist, dass sich die mittleren Up- und Download-Latenzen des Android mit fortlaufenden Messungen immer stärker aneinander und auch an die relativ konstanten mittleren Up- und Download-Latenzen der Alix annähern. Dieser Verlauf ist nicht erklärbar, da vor jeder Messung mit dem Android-Smartphone die Uhrzeit manuell synchronisiert wurde.

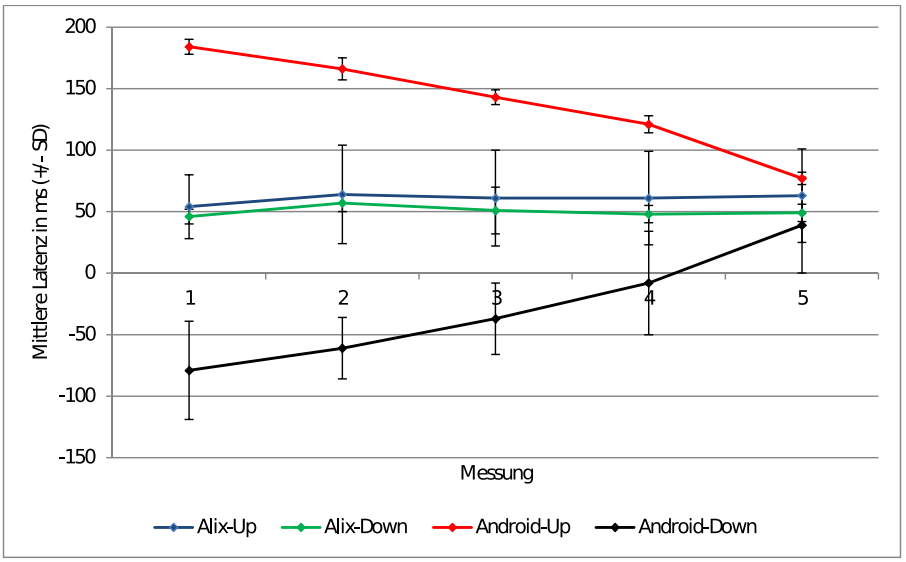

Abbildung 5.6: Mittlere Latenz in ms für Alix und Android.

5.2.5 CDF-Latenzen von Android

Auch in Abbildung 5.7 und Abbildung 5.8 zeigte sich die in den mittleren Latenzen beobachtete Streuung der gemessen Latenzen erneut. In Downloadrichtung besaß die Latenz eine hohe Konstanz mit über 90 Prozent der Messwerte in einem Intervall von 30 ms. In Uploadrichtung ließ sich eine nicht so hohe Konstanz wie beim Download feststellen, jedoch lagen 70 Prozent der Messwerte in einem Intervall von 30 ms. Der Latenzwert, an welchem die Häufungen auftraten, entstand durch die in Kapitel 5.2.4 erwähnte ungenaue Zeitsynchronisierung. Hier war im Upload des Androids eine leichte Stufenbildung erkennbar.

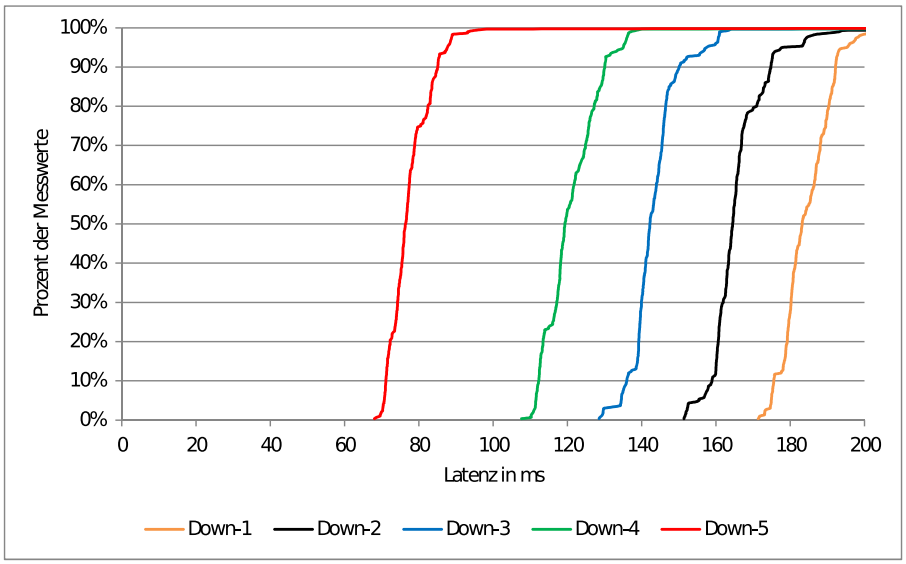

Abbildung 5.7: CDF-Latenz-Android-Download.

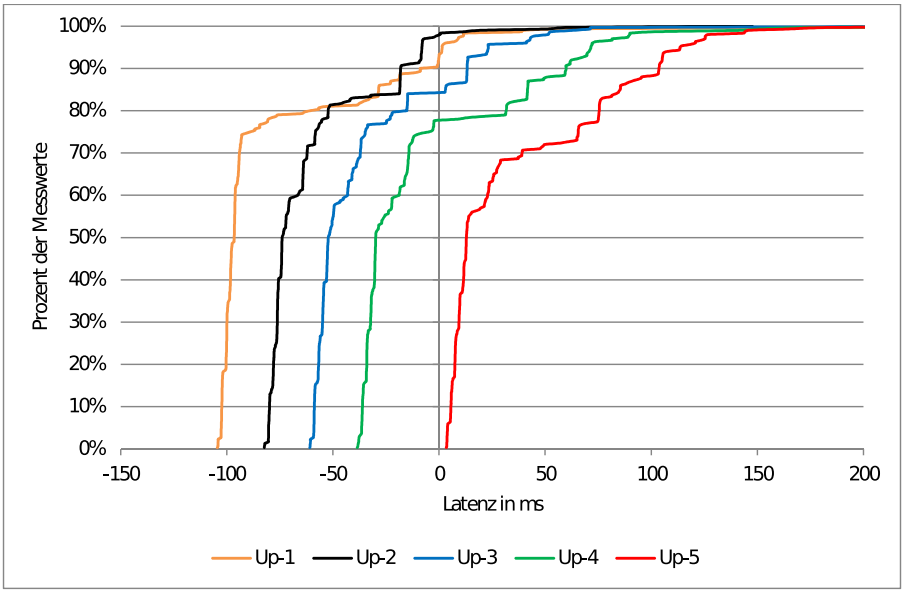

Abbildung 5.8: CDF-Latenz-Android-Upload.

5.2.6 CDF-Latenzen von Alix

Abbildung 5.9 und Abbildung 5.10 zeigen, dass sich im Download ca. 90 Prozent der Latenzen ebenfalls in einem Intervall von 30 ms befanden. Im Unterschied zu den Android-Latenzen war jedoch hier am Startpunkt der Latenzwerte die Streuung minimal. Im Upload war diese hohe Konstanz aus dem Download nicht mehr zu beobachten. Zwar waren ebenfalls etwa 50 Prozent der Messwerte in einem Bereich von 10 ms, jedoch trat hier eine breitere Streuung auf. Sehr interessant sind die zu beobachtenden verschiedenen Möglichkeiten der Werte bei der Latenzmessung im Up- und Download der Alix. Hier war eine starke Stufenbildung erkennbar. So bildeten in Messung 2 der Alix im Download nur drei verschiedene Werte der Latenz (43, 54 und 64 ms) ca. 95 Prozent aller gemessenen Werte. Im Upload war diese Stufenbildung ebenfalls erkennbar und es gab drei (40, 50 und 135 ms) Latenzwerte, welche zusammen etwa 80 Prozent aller Messwerte bildeten. Die Entstehung der Stufenbildung wurde in Kapitel 5.2.2 genauer erläutert. Auffällig war hier allerdings eine Stufenbildung in beide gemessene Richtungen. Diese wurde in den Datenraten der Alix nur in eine Richtung beobachtet. Erklären lässt sich diese Beobachtung durch die Verzögerung in der Alix. Wenn die Alix nicht ausreichend Rechenleistung besitzt, um die Pakete ohne Verzögerung zu verarbeiten, so

überträgt sie diese Verzögerung auch beim Senden der Pakete. Ein vergleichbares Verhalten zeigte sich in den durchschnittlichen Latenzen der Android-Messungen (s. Kapitel 5.2.4), daher wird an dieser Stelle nicht weiter darauf eingegangen. Es wird hier zusammenfassend davon ausgegangen, dass die Stufenbildungen einerseits durch eine ungenaue Synchronisierung der Uhren von Client und Server und andererseits durch die verzögerte Bearbeitung von Paketen aufgrund von nicht ausreichender Rechenleistung verursacht wurden.

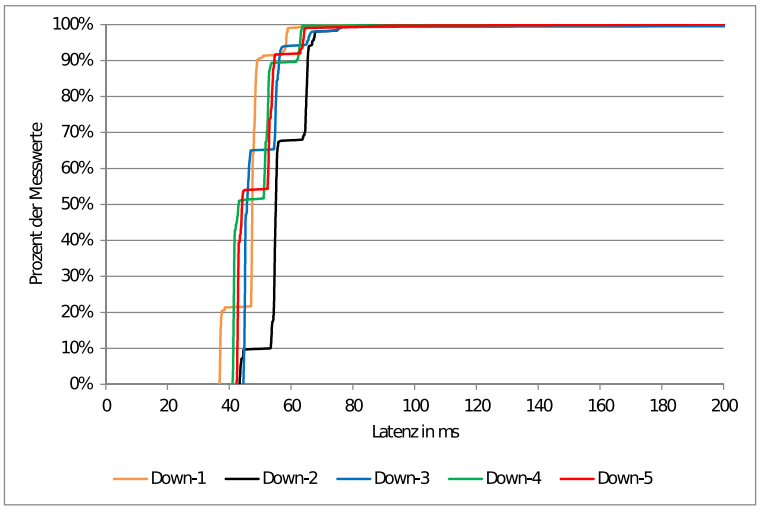

Abbildung 5.9: CDF-Latenz-Alix-Download.

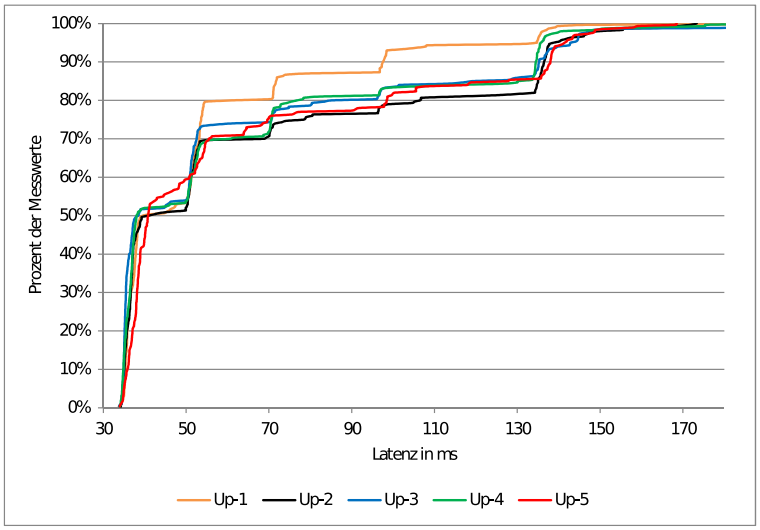

Abbildung 5.10: CDF-Latenz-Alix-Upload.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde ein am Lehrstuhl für Rechnernetze und Kommunikationssysteme entwickeltes Datenraten-Messframework erfolgreich auf die Android Plattform portiert. Nach Entwicklung einer Android-Anwendung mithilfe des NDK wurde das vorhandene RMF auf einem Android-Smartphone ausführbar gemacht. In der Umsetzung kam es zu vielen Herausforderungen durch eine sehr rudimentär vorhandene C++ Bibliothek für Android. Vor allem die Übergabe von Daten zwischen Java und C++ auf der Android-Plattform und die Implementierung von nicht vorhandenen, benötigten C++ Funktionen begleiteten diese Arbeit.
Analysen der Testmessungen lassen auf Android als nutzbare Messumgebung schließen. Einflüsse einer ungenauen Zeitsynchronisierung auf dem Android machten sich in den Messungen bemerkbar. Ebenfalls zeigte sich, dass die Mobilfunk-Messungen der Datenraten mit der Alix im Download eine Verzögerung der Bearbeitung von Paketen aufwies. Als möglichen Grund kann die im Vergleich zu den anderen Messkomponenten geringe Rechenleistung vermutet werden.

6.2 Ausblick

Nach Abschluss der Bachelorarbeit bleiben noch weitere Optimierungsmöglichkeiten des ETEMS. Die wichtigste Verbesserungsmöglichkeit ist eine Realisierung einer NTP-Zeitsynchronisierung für das Android-Smartphone. So könnte es möglich sein den Audioport als Schnittstelle für einen zusätzlichen Datentransfer zu nutzen. Diese Schnittstelle kann möglicherweise als per GPS gesteuerter Zeitgeber eingesetzt werden. Ein solcher GPS gesteuerte Zeitempfänger wird auch aktuell in der Alix und dem Messserver eingesetzt. Hierfür müsste eine Erweiterung des ETEMS mit NTP-Funktionalitäten erfolgen. Die beobachtete Stufenbildung in den Messungen mit der Alix müsste in weiteren Messungen genauer untersucht und mit den hier gewonnenen Messdaten verglichen werden. Durch ständige Weiterentwicklung des RMF [Lan13] ist allerdings nicht auszuschließen, dass in der aktuellen Version das Zusammenspiel der Threads optimiert und somit die Stufenbildung aufgehoben wurde.

Literaturverzeichnis

[And13] ANDROID GEEKS: *Samsung 9250 root*. Website, Juni 2013. Online verfügbar unter http://www.android.gs/root-galaxy-nexus-i9250-android-4-2-jelly-bean-os/

[Com13] COMSCORE: *Anteil der Smartphone-Nutzer an allen Mobilfunkbesitzern in Deutschland von Januar 2010 bis Dezember 2012*. Website, März 2013. Online verfügbar unter http://de.statista.com/statistik/daten/studie/237079/umfrage/anteil-der-smartphone-nutzer-an-allen-mobilfunknutzern-in-deutschland/

[Cor13] CORPORATION, Oracle: *Java SDK*. Website, Juni 2013. Online verfügbar unter http://www.oracle.com/technetwork/java/javase/downloads/index.html

[Fou13a] FOUNDATION, Mozilla: *Mozilla Mobile-Browser*. Website, Juni 2013. Online verfügbar unter http://hg.mozilla.org/mobile-browser/summary

[Fou13b] FOUNDATION, The L.: *bash man*. Website, Juni 2013. Online verfügbar unter http://linux.die.net/man/1/bash

[Fou13c] FOUNDATION, The L.: *Clock Nanosleep*. Website, Juni 2013. Online verfügbar unter http://linux.die.net/man/2/clock_nanosleep

[Fou13d] FOUNDATION, The L.: *CPU_SET man*. Website, Juni 2013. Online verfügbar unter http://linux.die.net/man/3/cpu_set

[Fou13e] FOUNDATION, The L.: *getifaddrs man*. Website, Juni 2013. Online verfügbar unter http://linux.die.net/man/3/getifaddrs

[Fou13f] FOUNDATION, The L.: *Nanosleep man*. Website, Juni 2013. Online verfügbar unter http://linux.die.net/man/2/nanosleep

[Fou13g] FOUNDATION, The L.: *sed man*. Website, Juni 2013. Online verfügbar unter http://linux.die.net/man/1/sed

[Gar13] GARTNER: *Marktanteile der Bertriebssysteme am weltweiten Smartphone-Absatz 2010 - 2013 (in %)*. Website, März 2013. Online verfügbar unter http://de.statista.com/themen/581/smartphones/infografik/1097/marktanteile-der-smartphone-betriebssysteme/

[Goe10] GOEBEL, Norbert: *Trace-basierte Simulation von Mobilfunkcharakteristiken für die Fahrzeug-zu-Fahrzeug Kommunikation*, Department of Computer Science, Heinrich Heine University Düsseldorf, Diplomarbeit, August 2010

[Gro13] GROUP, Samsung: *Samsung Galayx Nexus 9250 Datenblatt*. Website, Juni 2013. Online verfügbar unter http://www.samsung.com/hk_en/consumer/mobile/mobile-phones/smartphone/GT-I9250TSATGY-spec

[Inc12] INC., Google: *genesyscalls.py*. Website, Januar 2012. Online verfügbar unter https://android.googlesource.com/platform/bionic/+/master/libc/tools/gensyscalls.py

[Inc13a] INC., Google: *Android-Java Bibliothek*. Website, Juni 2013. Online verfügbar unter http://developer.android.com/reference/packages.html

[Inc13b] INC., Google: *Android NDK*. Website, Juni 2013. Online verfügbar unter http://developer.android.com/tools/sdk/ndk/index.html

[Inc13c] INC., Google: *Android SDK*. Website, Juni 2013. Online verfügbar unter http://www.android.com

[Inc13d] INC., Google: *Android SDK*. Website, Juni 2013. Online verfügbar unter http://developer.android.com/sdk/index.html

[Inc13e] INC., Google: *Android Systemarchitecture*. Website, Juni 2013. Online verfügbar unter http://developer.android.com/guide/basics/what-is-android.html

[Inc13f] INC., Google: *The Bionic libstc*. Website, Juni 2013. Online verfügbar unter http://elinux.org/Android_Notes#C_Library_.28bionic.29_info

[Inc13g] INC., Google: *Dalvik Debugging Monitor Server*. Website, Juni 2013. Online verfügbar unter http://developer.android.com/tools/debugging/ddms.html

[Knu12] KNUTH, Donald E.: *Android NDK*. Packat Publishing, 2012

[Lan13] LANGE, Christian: *Untersuchung der Auswirkungen paralleler Datenratenmessungen in einer Mobilfunkzelle*, Department of Computer Science, Heinrich Heine University Düsseldorf, Diplomarbeit, Juni 2013

[Ltd13] LTD, Synesis Software P.: *Pantheios*. Website, Juni 2013. Online verfügbar unter http://www.pantheios.org/

[Mil06] MILLS, D.: *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*. RFC 4330 (Informational). `http://www.ietf.org/rfc/rfc4330.txt`. Version: Januar 2006 (Request for Comments). Obsoleted by RFC 5905

[MMB+00] MOGUL, J.; MILLS, D.; BRITTENSON, J.; STONE, J.; WINDL, U.: *Pulse-Per-Second API for UNIX-like Operating Systems, Version 1.0*. RFC 2783 (Informational). `http://www.ietf.org/rfc/rfc2783.txt`. Version: März 2000 (Request for Comments).

[MMBK10] MILLS, D.; MARTIN, J.; BURBANK, J.; KASCH, W.: *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905 (Proposed Standard). `http://www.ietf.org/rfc/rfc5905.txt`. Version: Juni 2010 (Request for Comments).

[Nat13] NATIONAL COORDINATION OFFICE FOR SPACE-BASED POSITIONING AND NAVIGATION AND TIMING: *GPS*. Website, Juni 2013. Online verfügbar unter http://www.gps.gov/

[.NE13] .NET, CrystaX: *CrystaX NDK*. Website, Juni 2013. Online verfügbar unter http://www.crystax.net/en/android/ndk

[Net13] NETWORK, Stack E.: *Android set thread affinity*. Website, Mai 2013. Online verfügbar unter http://stackoverflow.com/questions/16319725/android-set-thread-affinity

[Ope13] OPENHANDSETALLIANCE, The: *Open Handset Alliance*. Website, Juni 2013. Online verfügbar unter http://www.openhandsetalliance.com/

[Ser12] SERGEY BARANOV: *ClockSync*. Google Play Store, März 2012. Online verfügbar unter http://amip.tools-for.net/wiki/android/clocksync

[TLKO13] THE LINUX KERNEL ORGANIZATION, Inc.: *Linux Kernel*. Website, Juni 2013. Online verfügbar unter https://www.kernel.org/

[Wil12] WILKEN, Sebastian: *Verfahren zur Datenratenmessung in Mobilkunknetzwerken*, Department of Computer Science, Heinrich Heine University Düsseldorf, Diplomarbeit, Juni 2012

Ehrenwörtliche Erklärung

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

Düsseldorf, den 24. Juni 2013 Malte Olfen

Diese CD enthält:

- eine *pdf*-Version der vorliegenden Bachelorarbeit
- die LaTeX- und Grafik-Quelldateien der vorliegenden Bachelorarbeit samt aller verwendeten Skripte
- die **ETEMS** Quelldateien, die Messdaten in Rohform, sowie deren Auswertungen
- die Websites der verwendeten Internetquellen
- die Datenblätter der verwendeten Hardware

Die genaue Struktur aller vorhandenen Dateien ist in der Datei inhalt.txt auf der CD zu finden.