Untersuchung der Auswirkungen paralleler Datenratenmessungen in einer Mobilfunkzelle

Masterarbeit

von

Christian Simon Lange

aus
Düsseldorf

vorgelegt am

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

Juni 2013

Betreuer:
Norbert Goebel, M. Sc.

Zusammenfassung

Das Ziel der Arbeit war die Untersuchung der Auswirkungen paralleler Datenratenmessungen in einer Mobilfunkzelle. Um Messungen zu ermöglichen, wurde eine Hard- und Softwarebasis benötigt, welche das Messgerät bilden sollte.

Die Kombination von Hardware und Betriebssystem war bereits vorgegeben. Im Rahmen dieser Arbeit wurden Verbesserungen vorgenommen, welche die Stabilität des Systems sowie die Genauigkeit der Zeitsynchronisation für spätere Latenzmessungen erhöht hat. Des Weiteren wurde die Installation der Messsoftware auf den Messgeräten durch die Bereitstellung eines angepassten Images vereinfacht.

Als Messsoftware wurde eine Kombination aus einem Framework zur Entwicklung von Datenratenmessverfahren (Rate Measurement Framework), sowie ein Experimentenkontrollsystem (Stateful Experiment Control) gewählt. Im Laufe der Arbeit wurden immer wieder Fehler der Messsoftware gefunden, welche im Rahmen der Arbeit korrigiert wurden um Messungen zu ermöglichen. Entsprechend viel Zeit wurde für die Entwicklung eines funktionierenden Systems verwendet.

Um der Herausforderung schwankender Datenraten im Mobilfunknetz gerecht zu werden, wurde ein dynamisches Messverfahren entwickelt. Mit diesem Messverfahren wurden zum Ende der Arbeit parallele Messungen im Mobilfunk durchgeführt und analysiert.

Danksagung

Ich möchte mich an dieser Stelle bei all jenen bedanken, welche mich im Rahmen meines Studiums und bei der Anfertigung dieser Masterarbeit unterstützt haben.

Meinem Betreuer Norbert Goebel danke ich für die anregenden Diskussionen und die moralische Unterstüzung, insbesondere in den letzten Wochen der Masterarbeit. Bei den zahlreichen Herausforderungen, hat es sehr gut getan hin und wieder auch mal lachen zu können.

Des Weiteren möchte ich mich ausdrücklich bei allen Kommilitonen bedanken, welche ein offenes Ohr für meine Probleme hatten. Danke, dass ich bei euch hin und wieder mal etwas Dampf ablassen konnte.

Meiner Familie möchte ich dafür danken, dass sie es in den letzten Wochen und Monaten weitgehend akzeptiert haben, dass ich keine Zeit für sie hatte, auch wenn dies manchmal nötig gewesen wäre. Das gleiche gilt für meine Freunde, welche sich trotzdem gerne für mich Zeit genommen haben, wenn ich mal eine Ablenkung brauchte.

Bei meiner Freundin Ute möchte ich mich für die Unterstüzung, nicht nur während meiner Zeit an der HHU, sondern auch darüber hinaus bedanken. Du hast immer an mich geglaubt und mich auch nach dem Rückschlag in Aachen nicht im Stich gelassen. Wir haben zusammen schon viel durchgemacht und ich hoffe, dass für uns jetzt auch mal längere Zeit die Sonne scheint.

Zu guter Letzt möchte ich mich bei all jenen bedanken, die mich auf meinem Weg begleitet haben und diesen Moment nicht mehr mit mir teilen können. Ohne euch wäre ich nicht derjenige, der ich heute bin. Ich werde euch nie vergessen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1.

Einleitung

Die Entwicklung moderner Kommunikationstechniken hat die Gesellschaft in den letzten Jahren maßgeblich beeinflusst. Dienste wie Facebook, Youtube oder Twitter wären ohne die Entwicklung immer schnellerer Internetzugänge nicht realisierbar.

Gerade der mobile Zugang zum Internet mittels Mobilfunk und Wireless LAN hat das Leben vieler Menschen nachhaltig geändert. Informationen können inzwischen praktisch überall konsumiert, erstellt und verbreitet werden.

Eine Technologie, welche im letzten Jahrhundert ähnlich tiefgreifende Auswirkungen hatte war die Entwicklung des Automobils. Ohne Autos bzw. Lastkraftwagen wäre zum Beispiel eine Logistik wie sie heutzutage existiert schwer vorstellbar und damit auch ein Großteil der existierenden Wirtschaft.

Inzwischen beschäftigen sich Wissenschaft und Industrie mit der Frage, wie man diese beiden Technologien verbinden kann. Zum Beispiel ob und wie die moderne drahtlose Kommunikation genutzt werden kann, um die Sicherheit und den Komfort im Straßenverkehr zu erhöhen. Der Forschungskomplex, welcher sich mit dieser Thematik beschäftigt wird als *Car-to-X Communication (C2X)* bezeichnet.

Der Begriff Car-to-X umfasst dabei sowohl die direkte Kommunikation von Fahrzeugen, auch als *Car-to-Car Communication (C2C)* bezeichnet, als auch die Kommunikation von Fahrzeugen mit einer vorhandenen Infrastruktur in Form von sogenannten *Road Side Units (RSU)*. Dieser Bereich wird *Car-to-Infrastructure Communication (C2I)* genannt.

Auch der Lehrstuhl für Rechnernetze der Heinrich-Heine-Universität Düsseldorf beschäftigt sich mit dieser Thematik. Beispiele sind die Projekte *Next Generation Car-2-X Communication (NGC2X)* sowie *Sichere Intelligente Mobilität - Testfeld Deutschland (SIM-TD)*, in deren Rahmen Beiträge zur Entwicklung der C2X Kommunikation geleistet werden.

Die Erforschung und Entwicklung neuer Anwendungen mit Feldtests ist sehr kostenintensiv. Daher werden Simulationen genutzt, um Fehler und Probleme schon frühzeitig erkennen und beheben zu können. Die Simulation von C2X Kommunikation basierend auf Wireless LAN wird bereits durch mehrere Simulatoren unterstützt.

Anders sieht dies bei der Simulation basierend auf Mobilfunk aus. Aktuell vorhandene, frei verfügbare Simulatoren basieren auf einem probabilistischen Ansatz. Die Güte der Simulation hängt dabei von der Genauigkeit des Modells ab. Es gibt auch kommerziell vertriebene Simulatoren, welche jeden einzelnen Schritt im Mobilfunknetz simulieren. Allerdings ist davon auszugehen, dass diese mit einer steigenden Anzahl von Entitäten schlecht skalieren. Außerdem unterliegen sie meist einem Geheimhaltungsvertrag (*Non-Disclosure Agreement (NDA)*), da für die korrekte Simulation Einblicke in die Netze von Mobilfunkprovidern nötig sind. Damit sind diese Simulatoren für die wissenschaftliche Anwendung nicht interessant.

Daher hat Norbert Goebel in seiner Arbeit [Goe10] einen weiteren Ansatz verfolgt. Das von ihm entwickelte Simulationsmodell basiert auf der Verwendung realer Messdaten. Es wird erwartet, dass dieses Simulationsmodell genauer mit der realen Welt übereinstimmt als probabilistische Modelle und gut skaliert. Des Weiteren werden keine Informationen über das Providernetz benötigt, welche einer Verschwiegenheitserklärung unterliegen würden.

Um die für das Simulationsmodell benötigten Messdaten zu erfassen, werden an die Eigenschaften des Mobilfunks angepasste Messverfahren benötigt. Wie in [Goe10] beschrieben, sind die bekannten Datenratenmessverfahren jedoch nicht geeignet.

Ziel dieser Arbeit war es, mit Hilfe von am Lehrstuhl in Rahmen von Abschlussarbeiten entwickelten Programmen, eine Messplattform zu schaffen, mit deren Hilfe im Anschluss ein angepasstes Messverfahren entwickelt werden sollte. Des Weiteren sollte der Einfluss von parallelen Messungen innerhalb einer Mobilfunkzelle auf die Messverfahren untersucht werden.

1.1. Aufbau der Arbeit

Im folgenden Kapitel 2 werden zunächst die für die vorliegende Masterarbeit relevanten Arbeiten und Programme, welche zu diesem Themenkomplex am Lehrstuhl erstellt wurden, vorgestellt. Zusätzlich werden Einführungen zu den Themen Zeitsynchronisation, Datenratenmessverfahren und Mobilfunk gegeben.

In den Kapiteln 3 und 4 werden die im Rahmen der Masterarbeit durchgeführten Änderungen am Messgerät sowie an der Messsoftware vorgestellt.

Im Anschluß werden in Kapitel 5 der Versuchsaufbau sowie die durchgeführten Experimente beschrieben und analysiert. Kapitel 6 bietet zum Ende einen Ausblick bezüglich möglicher, sich anschließender Arbeiten.

Kapitel 2.

Verwandte Arbeiten

Im Folgenden werden die für die vorliegende Masterarbeit relevanten Arbeiten bzw. die im Rahmen der Arbeiten implementierten Programme zum Thema am Lehrstuhl vorgestellt. Des Weiteren werden Einführungen zu den Themen Zeitsynchronisation, Datenratenmessverfahren und Mobilfunk gegeben.

2.1. TBUSLogger und TBUSAnalyser

Die Programme *TBUSLogger (Trace-based UMTS Simulation Logger)* und *TBUSAnalyser (Trace-based UMTS Simulation Analyser)* wurden von Norbert Goebel während seiner Masterarbeit mit dem Titel *Trace-basierte Simulation von Mobilfunkcharakteristiken für die Fahrzeug-zu-Fahrzeug Kommunikation* [Goe10] entwickelt. Das Ziel der Arbeit war die prototypische Entwicklung eines Messframeworks, sowie eines Simulationsmodells für die Fahrzeug-zu-Fahrzeug Kommunikation in Mobilfunknetzen.

Das entwickelte Modell basiert auf der Verwendung von Daten, welche im Vorfeld durch Messfahrten gewonnen werden müssen. Zu diesem Zweck wurden die Java basierten Programme TBUSLogger und TBUSAnalyser entwickelt. Die Programme wurden für sogenannte Black-Box Messungen in Mobilfunknetzen verwendet. Dabei werden Messpakete von einem Client, welcher über ein Mobilfunknetz mit dem Internet verbunden ist, an einen über das Internet erreichbaren Server gesendet. Informationen über den internen Aufbau des Netzwerks beim Provider werden nicht verwendet.

Ein Ergebnis der Arbeit war, dass die bekannten Datenratenmessverfahren, unter der Berücksichtigung, das parallel zusätzliche Werte wie z.B. Latenzen ermittelt werden sollen, nicht für mobile Messungen im Mobilfunk geeignet sind.

2.2. Rate Measurement Framework

Die Entwicklung geeigneter Datenratenmessverfahren war das Ziel der Masterarbeit von Sebastian Wilken mit dem Titel *Verfahren zur Datenratenmessung in Mobilfunknetzwerken* [Wil12]. Um die Entwicklung zu vereinfachen, wurde zunächst ein modulares Messframework entworfen, welches TBUSLogger und TBUSAnalyser ersetzen sollte.

Das neue Framework wurde in C++ implementiert, um den Programmablauf zu beschleunigen und einen direkteren Zugriff auf die Hardware zu erhalten. Beim Java basierten Framework erfolgten Zugriffe auf die Hardware über die *Java Virtual Machine (JVM)*, was für eine zusätzliche Verzögerung zur Folge hatte. Des Weiteren sollte die Entwicklung neuer Messmethoden, sowie der Vergleich zwischen diesen durch die Abstraktion der Netzkommunikation erleichtert werden.

Im Folgenden werden zunächst die implementierten Kommunikationskanäle vorgestellt. Anschließend wird die Organisation des Programms in Threads, sowie die zu implementierenden Schnittstellenfunktionen im Falle der Entwicklung einer neuen Messmethode vorgestellt.

2.2.1. Voraussetzungen für den Verbindungsaufbau

Die Messungen, welche mit dem *Rate Measurement Framework (RMF)* durchgeführt werden, haben immer zwei Endpunkte. Gemessen wird gegen einen Server, welcher über eine bekannte Kombination aus IP Adresse und Portnummer ohne Einschränkung erreichbar sein muss. Wird der Server hinter einer Firewall oder einem *Network Address Translation (NAT)* System betrieben werden, so muss deren Konfiguration entsprechend angepasst werden. Bei einer Firewall genügt es die entsprechenden Portnummern für UDP und TCP Pakete freizugeben. Die Adressumsetzung eines NAT Systems muss so konfiguriert werden, dass eingehende Verbindung am NAT System zum RMF Server weitergeleitet werden. In diesem Fall ist die öffentliche IP Adresse durch die Adresse des NAT Systems bestimmt.

Der zweite Messpunkt ist das Client System, welches über einen Zugang zum gleichen Netzwerk wie der Server verfügen muss. Der Betrieb des Clientsystems hinter einem NAT System ist möglich, da das RMF einen eingebauten Mechanismus zum überbrücken bietet (siehe Abschnitt 4.1.2.1). Theoretisch funktioniert dieser auch auf Serverseite, allerdings muss mindestens ein System über eine öffentlich zugängliche IP:Port Kombination verfügen. Wie in [Goe10] beschrieben, haben Tests gezeigt, dass die Betreiber von Mobilfunknetzen NAT in ihrem Kernnetz einsetzen. Somit ist davon auszugehen, dass das Clientsystem nicht öffentlich erreichbar ist, was die Anforderungen an den Server erklärt.

2.2.2. Kommunikationskanäle

Für die Kommunikation der Messsysteme stehen zwei bis drei reale und ein virtueller Kanel zur Verfügung, welche in Abbildung 2.1 dargestellt sind. Der sogenannte *Control Channel* dient dem Verbindungsaufbau und dem Austausch von Statusnachrichten. Vor dem Beginn einer Messung werden über diesen Kanal Nachrichten zur Synchronisation der RMF Instanzen und der Messmethoden ausgetauscht.

Bei Konfigurationskonflikten zwischen Client- und Serverinstanz hat der Client grundsätzlich die höhere Priorität und entscheidet, welche Parameter verwendet werden. Das ist nötig, da die Serverinstanz darauf ausgelegt ist, ohne Unterbrechung betrieben zu werden. Somit können Kommandozeilenparameter zur Programm- und Methodenkonfiguration nur einmal zum Start der Serverinstanz übergeben werden.

Beim Control Channel handelt es sich um eine TCP Verbindung. Solange die Verbindung nicht abbricht, kann davon ausgegangen werden, dass gesendete Pakete die Gegenstelle in der richtigen Reihenfolge erreichen. Wird ein Verbindungsabbruch bemerkt, bricht die Client Instanz eine eventuell laufende Messung ab und beendet sich. Auch die Serverinstanz unterbricht in diesem Fall die Messung und wartet auf Anfragen neuer Client Instanzen.

Für die Messungen der Datenrate steht ein zweiter Kanal zur Verfügung, der sogenannte *Measurement Channel*. Dieser ist als UDP Verbindung implementiert, da eine TCP Verbindung die Messung durch das erneute Senden verloren gegangener Pakete beeinflussen würde. Durch die Verwendung von UDP können Pakete verloren gehen und ihre Reihenfolge ändern. Dies muss bei der Entwicklung eigener Messmethoden berücksichtigt werden.

Der Measurement Channel dient zusätzlich als Basis für einen virtuellen Kanal, den sogenannten *Backchannel*. Die Informationen des Backchannels werden als Teil des Payloads der Messpakete im Measurement Channel gesendet. Der Payload wird dazu in einen dynamischen und einen statischen Teil getrennt. Während der statische Teil bereits bei der Übergabe des Pakets an den Sender Thread fixiert ist, wird der dynamische Payload erst kurz vor dem Senden eingetragen. Somit ist es möglich, aktuelle Informationen an die Gegenstelle zu übertragen.

Zu den möglicherweise relevanten Informationen zählt zum Beispiel die empfangene Messdatenrate, sofern ein dynamisches Verfahren implementiert werden soll. Der Sender der Messpakete hat keine Möglichkeit, die Empfangsdatenrate der Messpakete beim Empfänger zu bestimmen. Das kann dazu führen, dass mehr Messpakete gesendet werden als der Empfänger verarbeiten kann. Der dadurch entstehende Paketstau, auch *Self Induced Congestion* genannt, behindert Latenzmessungen. Um dies zu vermeiden, kann der Empfänger über den Backchannel den Sender über die aktuell empfangene

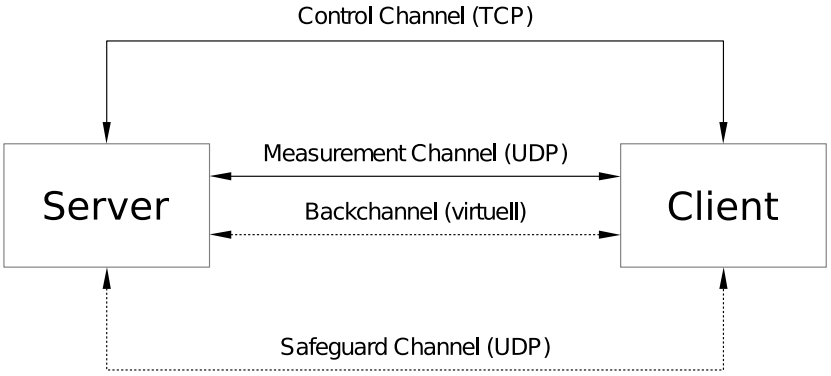

Abbildung 2.1.: Kommunikationskanäle des RMF

Messdatenrate informieren, so dass der Sender sein Verhalten entsprechend anpassen kann.

Da der Backchannel als Teil des Measurement Channels anfällig für einen Paketstau ist, wurde ein weiterer Kanal implementiert. Der optionale *Safeguard Channel* ist eine UDP Verbindung, über die beliebige Informationen gesendet werden können. Im Falle eines Paketstaus im Backchannel, kann der Safeguard Channel dazu genutzt werden, diesen Stau zu umgehen, indem eine andere physische Verbindung genutzt wird. Bei mobilen Messung könnte dies zum Beispiel das Mobilfunknetz eines anderen Betreibers sein, welches nicht durch die Messungen belastet wird.

2.2.3. Threads

Um das RMF zu strukturieren und die Leistungsfähigkeit des Programms zu erhöhen, wurde der Prozess in mehrere nebenläufige Threads unterteilt. Die Performance des RMF kann durch die Verwendung von Threads insbesondere bei Systemen mit Mehrkernprozessoren gesteigert werden, da dann mehrere Programmteile parallel ablaufen können. Zusätzlich unterstützt die Aufteilung in Threads die Beschleunigung einzelner Programmteile durch den in Abschnitt 2.2.5 beschriebenen Echtzeit Scheduler. Die im folgenden beschriebene Aufteilung der Threads kann anhand von Abbildung 2.2 nachvollzogen werden.

Für das Senden und Empfangen von Paketen, sowie die Ermittlung der entsprechenden Zeitstempel sind im RMF zwei Threads vorgesehen. Die Aufgabe des *Threaded Measurement SEnder (TMSE)* besteht darin, Pakete aus der Sendequeue zu entnehmen und zu prüfen, wann diese gesendet werden sollen. Ist der Sendezeitpunkt erreicht oder überschritten, wird sofort der aktuelle Zeitstempel bestimmt und im Paket eingetragen. Dann wird das Paket zum Senden an den Kernelspace mittels Systemaufruf weitergereicht. Anschließend wird eine Kopie des Pakets inklusive Sendezeitpunkt in einer Loggingqueue eingefügt.

Das Füllen der Sendequeue mit Paketen ist Aufgabe der Messmethode. Diese Funktion wird im Kontext des Hauptthreads bearbeitet. Es muss also sichergestellt werden, dass der Hauptthread ausreichend Prozessorzeit zugewiesen bekommt.

Der *Threaded Measurement REceiver (TMRE)* wartet auf eingehende Pakete aus dem Kernelspace. Wird ein Paket empfangen, wird zunächst der aktuelle Zeitstempel bestimmt. Anschließend wird der Empfangszeitpunkt zusammen mit einer Kopie des empfangenen Pakets in die Empfangsqueue eingefügt.

Die Minimierung der Aufgaben von TMSE und TMRE auf das Senden und Empfangen von Paketen, sowie das entnehmen bzw. einfügen von Paketen in Queues, soll einen möglichst hohen Durchsatz garantieren. Auch die Queues zum Austausch der Pakete wurden entsprechend implementiert [Wil12].

Um die Daten, welche durch den TMSE bzw. TMRE Thread in die Queues eingefügt werden zu verarbeiten, wurden zwei weitere Threads implementiert. Der *Threaded Sender Data Distributor (TSDD)* entnimmt die Informationen zu den gesendeten Paketen aus der Logginqueue und ruft die Funktion dataFileWriter_measureSender() auf, welche durch die Messmethode zur Verfügung gestellt werden muss. In den bereits implementierten Messverfahren wird diese Funktion dazu genutzt, die Informationen zu den gesendeten Paketen in eine Datei zu schreiben.

Der *Threaded Receiver Data Distributor (TRDD)* entnimmt die Informationen zu empfangenen Paketen aus der Empfangsqueue. Als Schnittstelle zum Messverfahren dienen in diesem Thread die beiden Funktionen dataFileWriter_measureReceiver(), sowie fastDataCalculator(). Die Funktion fastDataCalculator() wird in den bereits implementierten Messverfahren genutzt, um eine schnelle Schätzung der aktuellen Datenrate zu bestimmen und diese der Gegenstelle über den Backchannel und gegebenenfalls über den optionalen Safeguard Channel zur Verfügung zu stellen. Die Funktion dataFileWriter_measureReceiver() wird genutzt, um die Informationen über das empfangene Paket in eine Datei zu schreiben.

Sofern der Safeguard Channel aktiviert ist, sind zwei weitere Threads aktiv. Der *Threaded Safeguard Sender (TSSE)* entnimmt Pakete aus einer Queue und sendet diese direkt, ohne einen Sendezeitpunkt zu prüfen. Es werden auch keine Logging Informationen zum Paket gesichert. Der *Threaded Safeguard Receiver (TSRE)* wartet auf eingehende Pakete über den Safeguard Channel. Wird ein Paket empfangen, so wird direkt die Funktion ProcessSafeguardPacket() aufgerufen. Der direkte Aufruf ist nötig, da eine Safeguard Nachricht wichtige Informationen enthält, die sofort bearbeitet werden sollten. Eine Optimierung bezüglich des Durchsatzes ist nicht nötig.

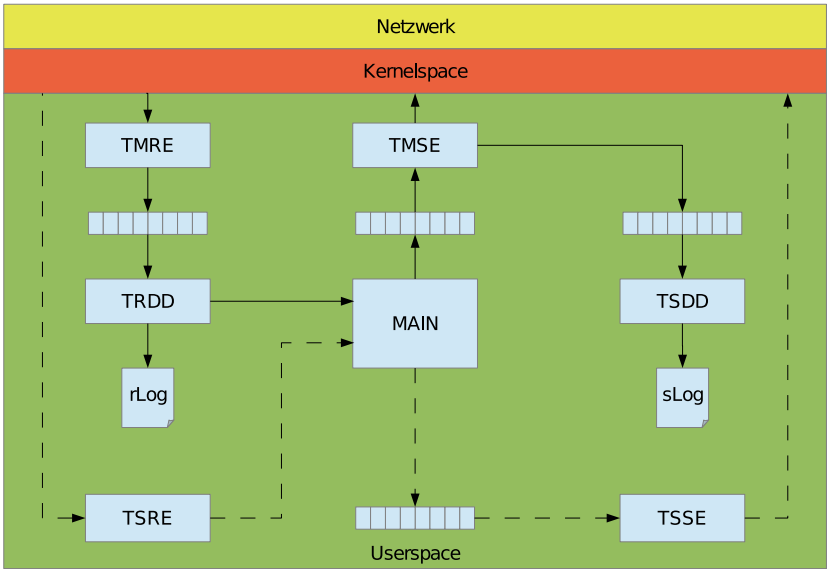

Abbildung 2.2.: Threads und Queues

2.2.4. Schnittstellen

Um eine eigene Messmethode zu implementieren, müssen die Schnittstellenfunktionen des RMF in einer Methodenklasse implementiert werden. Diese Methodenklassen sind im Ordner „measurement-Methods“ im Quelltextverzeichnis gespeichert. Die Klassen mBasic, mAssolo und mNew können als Basis für die Entwicklung eigener Messverfahren verwendet werden. Die benötigten Funktionen sind in Tabelle 2.1 aufgelistet.

Des Weiteren müssen in der aktuellen Version des Programms an diversen Stellen die genutzten Switch-Case Konstrukte erweitert werden. Auch die Synchronisierung der Parameter des RMF muss entsprechend erweitert werden. Hier bietet es sich zukünftig an, Methoden der Objektorientierung zu nutzen, um nur noch an einer Stelle im Quellcode Änderungen durchführen zu müssen.

2.2.5. Echtzeitfähigkeit

In den Arbeiten [GRT09] und [NSGR05] wird jeweils betont, dass die Verwendung von Echtzeit-Erweiterungen für den Linux Kernel das Senden bzw. Empfangen von Paketen beschleunigen kann. Ein großer Teil der Änderungen, welche durch die Echtzeit-Erweiterungen im Linux Kernel vorgenommen wurden, sind inzwischen im regulären Kernel enthalten.

Dazu gehören unter anderem die Echtzeit-Scheduler SCHED_FIFO und SCHED_RR [man12c]. Im Gegensatz zum regulären Scheduler erlauben die Echtzeit Scheduler das Verdrängen laufender Threads

Funktion	Beschreibung
initMeasurement()	Steuert den Ablauf der Messmethode. Insbesondere wird definiert, wie die Sendequeue gefüllt wird.
dataFileWriter_measureSender()	Definiert die Verarbeitung der Daten aus der Logging-Queue.
fastDataCalculator()	Berechnung der Datenrate mit Hilfe des aktuellen Paketes.
dataFileWriter_measureReceiver()	Definiert die Verarbeitung der Daten aus der Receive-Queue.
set_loggingDone()	Leitet das Beenden einer Messung ein.
ProcessSafeguardPacket	Bearbeitet ein empfangenes Safeguard Paket.

Tabelle 2.1.: RMF Schnittstellen

durch andere Threads mit einer höheren Priorität. Die Prioritäten können durch den Programmierer vorgegeben werden und damit der Ablauf des Programms steuern.

Der reguläre Scheduler erlaubt dem Benutzer über den Nice-Wert einzelnen Programmen einen höheren Anteil an der CPU Zeit zuzuweisen. Ein höherer Nice-Wert bedeutet jedoch nicht, das ein Programm die Möglichkeit hat andere Programme zu verdrängen.

Das RMF kann mit Hilfe eines Echtzeit-Scheduler das Senden und Empfangen in Form der Threads TMSE und TMRE gegenüber den anderen Threads des Programms priorisieren. Dies kann die Genauigkeit der Zeitstempel erhöhen, da TMSE und TMRE sofort CPU Zeit zugeteilt wird, sofern ein Paket empfangen wurde oder gesendet werden muss. Dabei werden auch alle anderen Prozesse des Systems verdrängt, sofern diese nicht die gleiche oder eine höhere Priorität aufweisen.

2.3. Stateful Experiment Control

Im Rahmen seiner Bachelorabeit entwickelte Tobias Amft das Programm *Stateful Experiment Control (SEC)* [Amf12]. Die Aufgabe des Programms ist die Steuerung und Überwachung eines Experiments über einen fehleranfälligen Kommunikationskanal. Um dies zu ermöglichen, werden die Experimente in sogenannte Runs eingeteilt, in denen Teilexperimente durchgeführt werden. Eine Kommunikation zwischen dem Server und den Clients findet nur zwischen den einzelnen Runs statt.

Diese Eigenschaft macht das Programm für Datenratenmessungen mit einem mobilen Messgerät im Mobilfunknetz interessant, da in diesem Fall meist nur ein Kommunikationskanal existiert, der parallel den Messkanal darstellt. Um zu gewährleisten, dass die teilnehmenden Systeme ihre Aufgaben

zum richtigen Zeitpunkt ausführen, wird der Startzeitpunkt durch den Server an alle Teilnehmer übermittelt. Während des eigentlichen Experiments werden relative Zeiten zum Startzeitpunkt genutzt. Zwischen den einzelnen Runs senden alle Teilnehmer periodisch Nachrichten an den Server, damit dieser eventuelle Ausfälle bemerken und entsprechend durch die Aktivierung von Reserveinstanzen oder den Abbruch des Experiments reagieren kann.

Die Beschreibung eines Experiments erfolgt durch zwei XML Dateien. Dabei enthält die Konfigurationsdatei neben weiteren Informationen den angestrebten Startzeitpunkt des Experiments, sowie den Namen der Experimentdatei. Diese beschreibt die einzelnen Runs eines Experiments. Innerhalb der Runs können die Aufgaben fest einzelnen Clients zugeordnet werden. Alternativ besteht die Möglichkeit eine automatische Auswahl zu nutzen.

2.4. Projektarbeit

Vor dem Beginn der vorliegenden Masterarbeit wurde durch den Autor eine Projektarbeit mit dem Titel *Entwicklung und Evaluierung einer mobilen Einheit zur Erstellung von UMTS Traces* bearbeitet. Ziel der Projektarbeit war die Zusammenstellung und Inbetriebnahme eines mobilen Messgeräts. Dieses Messgerät sollte den in der Masterarbeit von Herrn Goebel [Goe10] verwendeten Laptop für Messfahrten ersetzen. Insbesondere sollte die Möglichkeit berücksichtigt werden, das Messgerät später in ein Fahrzeug einzubauen.

Im Folgenden werden die Anforderungen an, sowie die Zusammenstellung des Messgeräts beschrieben. Die Anpassungen, welche in der vorliegenden Masterarbeit vorgenommen wurden, werden in Kapitel 3 erklärt.

2.4.1. Anforderungen

Im Verlauf der Projektarbeit wurden, basierend auf den Erfahrungen aus [Goe10], Anforderungen an die Hardware eines Messgeräts für mobile Datenratenmessungen in Mobilfunknetzen definiert.

Am Lehrstuhl waren zu diesem Zeitpunkt bereits *Global Positioning System (GPS)* Empfänger vorhanden, welche zur Positionsbestimmung während der Messfahrten genutzt werden können. Des Weiteren kann der Empfänger wie in Abschnitt 2.5.2 beschrieben, zur Synchronisation der lokalen Uhr eines Computersystems auf das Zeitsignal des GPS genutzt werden. Zum Betrieb dieser Empfänger wird ein serieller Anschluss zur Datenübertragung, sowie ein USB Anschluss zur Stromversorgung benötigt.

Anforderung	Grund
serielle Schnittstelle USB Schnittstelle	GPS Empfänger
Mini PCI Express Schnittstelle SIM Kartenhalterung	3G-Modem
12V Eingangsspannung geringer Stromverbrauch keine beweglichen Teile	Mobilität
Linux Kompatibel	Modifikation

Tabelle 2.2.: Anforderungen

Um die Unterstützung eines 3G-Modems sicherzustellen, wurde festgehalten, dass sowohl eine Mini PCI Express Schnittstelle, als auch eine SIM Kartenhalterung notwendig sind. Des Weiteren sollte der Einsatz des Messgeräts im Auto durch einen möglichst geringen Stromverbrauch, sowie die Unterstützung von 12V Eingangsspannung erleichtert werden. Die komplette Hardware sollte kompatibel zum Linux Kernel sein. Da der Quellcode des Linux Kernels frei verfügbar ist, sind gegebenenfalls eigene Anpassungen und Optimierungen möglich.

2.4.2. Hardware

Ausgehend von den in Tabelle 2.2 definierten Anforderungen wurde die im folgenden beschriebene Hardware angeschafft.

2.4.2.1. Single Board Computer

Als Basis für das Messgerät dient ein *Single Board Computer (SBC)* der Firma PCEngines. Das Modell Alix6F2 bietet einen 500Mhz AMD Geode LX800 Prozessor, sowie 256MB Arbeitsspeicher. Als nichtflüchtiges Speichermedium ist eine CompactFlash Karte vorgesehen. Eine vollständige Auflistung der vorhandenen internen und externen Schnittstellen liefert Tabelle 2.3.

2.4.2.2. 3G-Modem

Zur Kommunikation mit Mobilfunknetzen wird ein Modem der Firma Sierra Wireless verwendet. Das Modell MC8790 bietet Geschwindigkeiten bis zu 2Mbps im Uplink (HSUPA Cat.5), sowie 7,2Mbps im Downlink (HSDPA Cat.8). Die Anbindung an den SBC erfolgt über die Mini PCI Express Schnittstelle. Das Modem wird durch einen im Linux Kernel vorhandenen Treiber unterstützt.

Schnittstelle	Bemerkungen
Mini PCI	intern
Mini PCI Express	intern, nur USB 2.0 Beschaltung
2 x Ethernet	extern, 10/100Mbps
DB9 seriell	1 intern, 1 extern, nur Rx/Tx
2 x SIM	1 intern, 1 extern
LPC Bus	

Tabelle 2.3.: Alix6F2 Schnittstellen

2.4.2.3. Serieller Anschluss

Die serienmäßig vorhandenen seriellen Schnittstellen des Modells Alix6F2 unterstützen nur zwei Signalleitungen zum Senden und Empfangen (Rx/Tx). Für eine möglichst genaue Zeitsynchronisation auf das Signal des verwendeten GPS Empfängers wird jedoch noch die Signalleitung Data Carrier Detect (DCD) benötigt. Über diese Leitung wird das *Pulse-Per-Seconds (PPS)* Signal des GPS Empfängers gesendet, welches für eine sehr genaue Synchronisation benötigt wird (siehe Abschnitt 2.5.2.1).

Die Verwendung eines USB nach Seriell Adapters ist nicht möglich, da dieser eine nicht konstante Latenz zum Signal hinzufügen würde. Diese ist dadurch bedingt, dass das Polling Intervall von USB mindestens 125μs beträgt [CHPI$^+$00] und nicht bekannt ist, wann innerhalb dieses Intervalls das PPS Signal empfangen wird. Somit kann dieser Fehler nicht durch einen statischen Offset korrigiert werden. Als Alternative wurde die Erweiterungskarte Commell MP-954 erworben, welche bis zu vier weitere serielle Schnittstellen an einer Mini PCI Schnittstelle anbietet.

2.4.2.4. GPS Empfänger

Im Rahmen der Arbeit [Goe10] wurden GPS Empfänger der Firma Garmin angeschafft. Das Modell Garmin GPS 18x LVC ermöglicht den Empfang von GPS Signalen und ermittiert zusätzlich ein Pulse-Per-Second Signal, mit dessen Hilfe eine Zeitsynchronisation im einstelligen Mikrosekundenbereich möglich ist. Die verwendeten Empfänger sind wie in [Goe10] beschrieben mit einem USB Anschluss zur Stromversorgung, sowie mit einem seriellen Anschluss zur Datenübertragung ausgestattet.

2.4.3. Betriebssystem

Bei eingebetteten Systemen wird häufig auf Linux Kernel basierende Betriebssysteme zurückgegriffen. Diese ermöglichen auf Grund der Quelloffenheit Anpassungen an die meist begrenzte Leistungs-

fähigkeit dieser Systeme. Für das Messgerät wurde die Distribution Voyage Linux ausgewählt, welche ein Derivat von Debian Linux ist.

2.4.3.1. Debian

Die Debian Distribution ist ein gemeinschaftlich entwickeltes, freies Betriebssystem [Deba]. Üblicherweise nutzt das System einen angepassten Linux Kernel und ist sehr stabil. Neben dem Basissystem bietet Debian ein Paketverwaltungssystem, dass die einfache Installation und Aktualisierung zusätzlicher Software ermöglicht. Wie bei Open Source Projekten üblich, exisitieren basierend auf Debian diverse weitere Projekte. Diese modifizieren das Debian System, um es den Bedürfnissen bestimmter Anwendungsfälle oder Hardware anzupassen.

2.4.3.2. Debian Live

Das Debian Live Projekt stellt Werkzeuge bereit, um die Entwicklung von Debian basierten Live Systemen zu erleichtern [Debb]. Ein Live System ist ein Betriebssystem, welches nicht installiert werden muss, sondern direkt von einem Medium wie einem USB Stick oder einer CD gestartet werden kann. Die verwendeten Werkzeuge werden durch Konfigurationsdateien gesteuert, welche es ermöglichen das Zielsystem bezüglich der Softwareauswahl und der Konfiguration anzupassen.

2.4.3.3. Voyage Linux

Voyage Linux nutzt die Werkzeuge des Debian Live Projekts, um eine an geringen, nicht beliebig beschreibbaren Speicherplatz angepasstes Debian Derivat zu erstellen [Voya]. Dazu wird das gesamte System so konfiguriert, dass es im Read-Only Modus laufen kann. Des Weiteren werden Anpassungen am Linux Kernel vorgenommen, falls die unterstützten Hardware Plattformen diese benötigen. Außerdem werden alle, für ein eingebettetes System nicht benötigten Bestandteile eines Debian Systems entfernt. Dazu zählen zum Beispiel die Handbuchseiten sowie die Unterstützung verschiedener Tastaturlayouts.

2.5. Zeitsynchronisation

Für die Messung von Latenzen bei der Übertragung von Nachrichten zwischen zwei Systemen ist es notwendig, dass die Uhren beider Systeme synchronisiert sind. In Computern basieren die vom Be-

triebssystem zur Verfügung gestellten Uhren meist aus einer Hardware- und einer Softwareuhr. Die Hardwareuhr ist dabei ein Taktgeber mit einer bestimmten Frequenz, welcher den Wert eines Registers erhöht um somit den Zeitverlauf anzuzeigen. Dieses Register wird vom Betriebssystem genutzt, um eine Softwareuhr zu betreiben. Das Betriebssystem nutzt dabei aus, dass die nominale Taktfrequenz bekannt ist und erhöht entsprechend nach einer bestimmten Anzahl von Takten der Hardwareuhr den Wert der Softwareuhr. Die Taktgeber weichen jedoch bedingt durch Produktions- und Umwelteinflüsse meist minimal von der nominalen Frequenz ab, so dass die Genauigkeit der Uhr mit der Zeit abnimmt.

Die Synchronisation von Uhren über ein Netzwerk wird durch das *Network Time Protocol (NTP)* gelöst, welches in [MMBK10] beschrieben wird. Das Network Time Protocol Project [ntp] stellt eine Referenzimplementierung des Prokolls zur Verfügung, welches zur Synchronisation gegen Zeitserver im Internet genutzt werden kann. Es können auch lokale Zeitquellen, wie zum Beispiel durch das Programm *gpsd* angebundene GPS Empfänger angebunden werden. Auch der Betrieb eines eigenen Zeitservers wird durch die Referenzimplementierung ermöglicht.

Im folgenden Kapitel wird eine Einführung in die Mechanismen des Network Time Protocol gegeben. Anschließend wird in Kapitel 2.5.2 beschrieben, warum das GPS eine gute Zeitquelle darstellt und wie die Daten des GPS mit Hilfe eines Empfängers und dem Programm gpsd an einen Computer übertragen werden.

2.5.1. Einführung in NTP

Das *Network Time Protocol (NTP)* ist ein Protokoll zur Synchronisation der Uhr eines Computers gegen lokale oder entfernte Zeitquellen. Bei entfernten Quellen wird die Entfernung zu einer primären Zeitquelle durch das sogenannte Stratum beschrieben. Ein Stratum 1 Server hat einen direkten Kontakt zu einer Referenzquelle, wie zum Beispiel eine Atom- oder Funkuhr. Nutzt ein NTP Client einen Stratum 1 Server als Quelle, so hat der Client selbst ein Stratum von 2. Somit ist das Stratum ein Indiz für die Entfernung zur Primärquelle und Güte des Zeitsignals.

2.5.1.1. Bestimmen des Offsets

Um den Offset der eigenen Uhr zu einem entfernten Zeitserver zu bestimmen, sendet der Client periodisch eine Anfrage an den Server als UDP Paket. Diese Anfrage enthält den Sendezeitpunkt des Pakets aus Sicht des Clients (T_1). Der Zeitserver antwortet mit einem weiteren UDP Paket, welches den Zeitpunkt des Empfangs aus Sicht des Servers (T_2), sowie den Sendezeitpunkt aus sicht des Servers (T_3) enthält. Empfängt der Client dieses Paket, merkt er sich den Empfangszeitpunkt (T_4). Mit

Hilfe dieser Zeitstempel kann nach Formel 2.1 der Offset (Θ) bestimmt werden. Die Rundlaufzeit (δ) wird nach Formel 2.2 berechnet.

$$\Theta = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)] \tag{2.1}$$

$$\delta = (T_4 - T_1) - (T_3 - T_2) \tag{2.2}$$

Diese Berechnungen gehen von synchronen Latenzen zwischen dem Server und dem Client aus. Sind die Latenzen lat_{up} und lat_{down} nicht synchron, kann es maximal zu dem in Formel 2.3 beschriebenen Fehler kommen. Die Herleitung findet sich in [Goe10].

$$|E| = \frac{|lat_{up} - lat_{down}|}{2} \tag{2.3}$$

2.5.1.2. Auswahl korrekter Quellen

Es werden mehrere Verfahren angewendet, um aus den erhaltenen Zeitstempeln den besten Wert für die Synchronisation zu bestimmen. Zunächst werden für jeden Zeitserver die letzten acht erhaltenen Werte gespeichert. Unter diesen acht Werten wird nur derjenige weitergegeben, welcher die kleinste Rundlaufzeit aufweist, da diese Werte die kleinste Varianz im Offset aufweisen.

Ein Auswahlalgorithmus nutzt anschließend die Offsets, um zu bestimmen welche Zeitquellen korrekt (Truechimmers) und welche falsch (Falsetickers) sind. Dazu wird für jeden Zeitserver das Intevall gebildet, in dem sich der korrekte Zeitwert befinden muss. Das Intervall bestimmt sich aus dem berechneten Offset (Θ) zum Zeitserver und der sogenannten Root Distanz (λ), welche sich aus der halben Rundlaufzeit zuzüglich weiterer Faktoren berechnet.

$$[\Theta - \lambda, \Theta + \lambda] \tag{2.4}$$

Aus diesen Intervallen für die einzelnen Zeitserver wird dann das sogenannte Intersection Intervall bestimmt. Dazu wird eine minimale Schnittmenge zwischen den Intervallen der Zeitserver bestimmt, unter der Bedingung dass die Anzahl der an der Schnittmenge beteiligten Intervalle maximiert wird Die Abbildung 2.3, welche sich an der Abbildung aus [clo] orientiert, verdeutlicht diesen Vorgang.

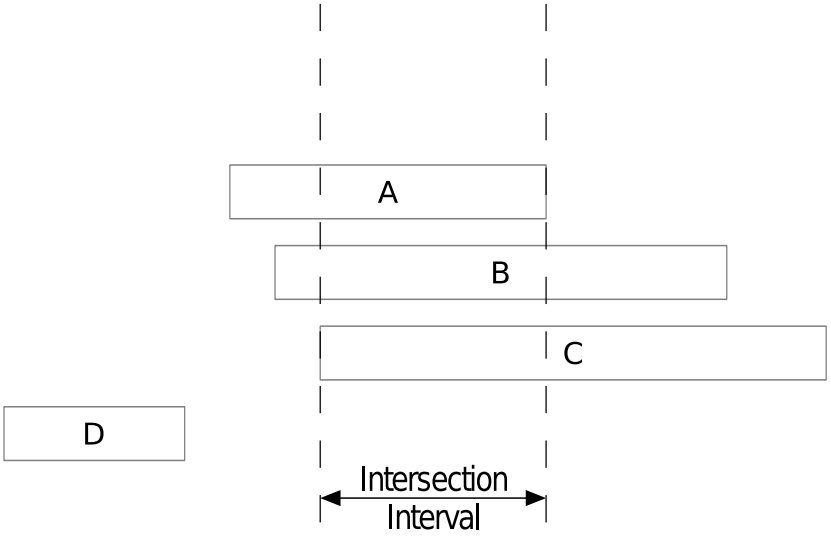

Abbildung 2.3.: Intersection Intervall

Mit Hilfe des Intersection Intervalls können anschließend die korrekten Zeitserver bestimmt werden. Alle Zeitserver, deren Intervall mindestens einen gemeinsamen Punkt mit dem Intersection Intervall haben, sind Truechimmers. Alle anderen sind Falsetickers und werden verworfen.

Anschließend wird aus den Truechimmers in mehreren Runden immer derjenige entfernt, dessen Wert am weitesten vom Mittelwert aller Truechimmers entfernt ist. So wird am Ende ein Wert bestimmt, welcher zur Synchronisation der lokalen Uhr genutzt wird.

2.5.1.3. Korrektur der lokalen Zeit

Wie in Abschnitt 2.5 beschrieben, nutzen Betriebssystem meist eine Kombination von Hardware- und Softwareuhr. Durch die Softwarekomponente, gibt es zwei Möglichkeiten die Zeit eines Systems zu korrigieren.

Die erste Möglichkeit ist die direkte Anpassung der Zeit durch Manipulation der aktuellen Softwarezeit. Dies ist insbesondere erwünscht, wenn ein großer Offset zur als gültig betrachteten Zeitquelle besteht. Da diese Methode jedoch zu Problemen, zum Beispiel bei schlafenden Threads führen kann, wird diese Methode nur beim Start der NTP Referenzimplementierung genutzt. Des Weiteren wird durch das NTP Protokoll keinesfalls die Zeit zurückgestellt, da dies zu Problemen zum Beispiel mit Logdaten führen kann.

Die zweite Möglichkeit ist die Anpassung des Verhältnisses von Takten der Hardwareuhr zu einer Sekunde der Softwareuhr. Durch die Änderung dieses Verhältnisses kann die Dauer einer Sekunde

der Softwareuhr gekürzt oder verlängert werden, so dass sich über die Zeit der Offset korrigieren kann.

Sollten alle externen Signalgeber, zum Beispiel durch eine Netzwerkpartitionierung ausfallen, erhält die Referenzimplementierung keine gültigen Zeitstempel. In diesem Fall versucht das Programm über die bisher bekannte Abweichung des lokalen Oszillators die absolute Abweichung bis zum nächsten Empfang eines gültigen Signals zu minimieren. Dazu dient das sogenannte Driftfile.

2.5.2. GPS als Zeitquelle

Das *Global Positioning System (GPS)* ist ein Satellitengestütztes System zur Positionsbestimmung. Dazu sendet jeder GPS Satellit kontinuierlich Nachrichten, welche neben der Position des Satelliten im Orbit auch den Sendezeitpunkt der Nachricht enthalten. Bei Empfang von mehr als vier Satelliten kann ein GPS Empfänger aus diesen Daten seine eigene Position bestimmen. Da die Genauigkeit der Positionsbestimmung von der des Zeitsignals abhängt ist dieses sehr genau. Somit bietet es sich an, dass GPS Signal zur Synchronisation von Uhren zu nutzen.

2.5.2.1. Pulse-Per-Second Signal

Die Genauigkeit des Zeitsignals eines GPS Empfängers hängt vom Empfänger selbst und der Anzahl der sichtbaren Satelliten ab. Je mehr Satelliten sichtbar sind, um so mehr Signale muss ein Empfänger verarbeiten. Da die Geschwindigkeit, mit der Signale verarbeitet werden können konstant ist, variiert die Zeitdifferenz zwischen dem Empfang der Signale und dem übermitteln der Daten an einen angeschlossenen Computer mit der Anzahl der sichtbaren Satelliten. Dies beeinflusst die Genauigkeit mit der sich der Computer auf das ursprüngliche GPS Signal synchronisieren kann.

Die besten Resultate erreicht man mit Empfängern, welche ein *Pulse-Per-Second (PPS)* Signal emittieren. Dieses Signal löst beim Computer einen Interrupt aus, damit dieser den Beginn der Sekunde markieren kann. Erhält der Computer den nächsten Zeitstempel, so kann er diesen dem PPS Signal zuordnen und kann den durch die Verarbeitung entstandenen variablen Offset korrigieren.

Damit kann der Offset bei der Synchronisation gegen das GPS Signal auf wenige Mikrosekunden begrenzt werden [PPS].

2.5.2.2. gpsd

Um die Daten des GPS Empfängers nutzen zu können, müssen dessen Signale zunächst interpretiert werden. Dies ist nötig, da die einzelnen Hersteller unterschiedliche, teils propriätere Codierungen nutzen. Teilweise können auch verschiedene Codierungen mit einem Gerät genutzt werden. Das Programm *gpsd* interpretiert die Signale verschiedener GPS Empfänger und stellt diese in einem einheitlichen Format über einen TCP Port zur Verfügung [gpsb]. Um die Nutzung des Programms zu vereinfachen verwendet gpsd einen Autokonfigurationsmechanismus, so dass nur die Schnittstelle zum GPS Empfänger definiert werden muss.

Neben der Möglichkeit die Daten über eine TCP Verbindung abzufragen, besteht des Weiteren die Möglichkeit, die empfangenen Zeitstempel über einen gemeinsamen Speicherbereich an die NTP Referenzimplementierung weiterzuleiten. Dies gilt auch für das PPS Signal.

Somit kann mit Hilfe eines GPS Empfängers, sowie des Programms gpsd und der NTP Referenzimplementierung ein eigener Stratum 1 Zeitserver betrieben werden, welcher seine lokale Uhr unabhängig von einer Netzwerkverbindung synchronisieren kann.

2.6. Datenratenmessverfahren

Datenratenmessverfahren lassen sich, abhängig davon welche Art von Datenrate sie messen (z.B. Layer2 oder Layer3) und welche Methode sie zum Messen verwenden in verschiedene Kategorien einteilen. Im Folgenden werden, neben einer kurzen Einführung zu den verwendeten Begriffen, zwei allgemeine Ansätze für Datenratenmessverfahren, sowie die Idee des in [Goe10] verwendeten Verfahrens beschrieben.

2.6.1. Relevante Eigenschaften von Netzwerkverbindungen

Jede Verbindung im Netzwerk besitzt eine *Kapazität C*. Diese wird bei Segmenten der Sicherungsschicht durch die verwendete Hardware und Übertragungsmethode bestimmt und vom Hersteller angegeben.

Die Kapazität eines einzelnen Hops auf der Vermittlungsschicht wird definiert als die maximal mögliche Transferrate von IP Schicht Daten über diesen Hop (Formel 2.5). Betrachtet man eine Ende-zu-Ende Verbindung in der Vermittlungsschicht, so ist die maximale Kapazität definiert als die maximal

mögliche Transferrate von der Quelle zum Empfänger (Formel 2.6). Der Hop mit der minimalen Kapazität wird als *narrow link* bezeichnet [PDMC03].

$$C_i = \frac{Header_{IP} + Data}{C_{link}} \tag{2.5}$$

$$C = \min_{i=1,\ldots} C_i \tag{2.6}$$

Eine Weitere definierte Eigenschaft von Netzwerkverbindungen ist die *verfügbare Datenrate A*. Die verfügbare Datenrate ist der zu einem Zeitpunkt ungenutzte Teil der Kapazität einer Verbindung. Dieser Wert ist somit abhängig von der momentanen Auslastung des Hops u_i und variiert über die Zeit. Formel 2.7 zeigt die Berechnung der verfügbaren Datenrate für einen Hop. Die verfügbare Datenrate für eine Ende-zu-Ende Verbindung ist das Minimum über alle Hops des Ende-zu-Ende Pfads (Formel 2.8). Der Hop mit der minimalen verfügbaren Datenrate wird als *tight link* bezeichnet [PDMC03].

$$A_i = (1 - u_i)C_i \tag{2.7}$$

$$A = \min_{i=1,\ldots} A_i \tag{2.8}$$

2.6.2. Bekannte Messverfahren

In dieser Arbeit werden nur Verfahren besprochen, welche eine Ende-zu-Ende Messung erlauben. Es werden für die Messmethoden keine Informationen über das interne Netzwerk der Mobilfunkprovider genutzt. Das interne Netzwerk wird in den Mobilfunkstandards beschrieben, die konkrete Implementierung ist jedoch den Providern vorbehalten und voraussichtlich jeweils unterschiedlich. Somit sind Provider unabhängige Ende-zu-Ende Messungen gegenüber der Analyse des internen Netzes eines einzelnen Providers zu bevorzugen.

Die bekannten Messverfahren zur Bestimmung der verfügbaren Datenrate einer Ende-zu-Ende Verbindung lassen sich in zwei Gruppen einteilen. Eine Gruppe nutzt das *Probe Gap Modell (PGM)*, die andere das *Probe Rate Modell (PRM)*.

2.6.2.1. Probe Rate Modell

Verfahren, welche nach dem Probe Rate Modell arbeiten, versuchen durch das Senden von Messpaketen mit einer bestimmten Senderate und einer Analyse der Latenzen beim Empfänger die aktuell verfügbare Datenrate zu bestimmen.

Indikator ist die sogenannte Self-Induced Congestion. Wird mit einer Datenrate gesendet, welche höher ist als die verfügbare Datenrate, so werden die Pakete auf dem Weg zum Empfänger gepuffert. Dieser Zustand kann durch steigende Latenzen erkannt werden. Ändert sich die Latenz nicht, so kann davon ausgegangen werden, dass die Senderate kleiner als die verfügbare Datenrate war. Die Senderate wird, je nach Implementierung entweder bereits innerhalb einer Messrunde oder nach jeder Messrunde angepasst. Diese Anpassungen erfolgen nach Regeln, welche integraler Bestandteil der Messmethode sind.

Mit Hilfe dieser Art von Messmethoden kann die verfügbare Datenrate, unter der Voraussetzung das die Datenrate der Verbindung relativ stabil ist, sehr genau bestimmt werden.

Diese Voraussetzung ist jedoch im Mobilfunk, insbesondere wenn sich der Nutzer bewegt, nicht gegeben. Durch einen Handover in eine andere Funkzelle oder ein Fallback auf eine andere Übertragungsmethode kann sich die verfügbare Datenrate schnell ändern. Somit sind Messverfahren nach dem PRM nur schlecht für die zukünftig geplanten Messfahrten geeignet.

2.6.2.2. Probe Gap Modell

Bei Verfahren welche nach dem Probe Gap Modell arbeiten, werden die Messpakete in Form von *Packet Pairs* bzw. *Packet Trains* verschickt. Im Falle eines Packet Pairs werden zwei Messpakete *back-to-back*, also direkt aufeinander folgend verschickt. Dies impliziert, dass die maximale Datenrate zum Senden genutzt wird.

Existiert auf dem Weg zum Empfänger ein tight link, so werden die Pakete auf der Leitung wie in Abbildung 2.4 gestreckt. Dies erhöht jedoch die *Packet Dispersion*, welche als der zeitliche Abstand zwischen dem letzten Bit des ersten Pakets und dem letzten Bit des zweiten Pakets definiert ist. Im Falle eines unbelasteten Pfads kann der Empfänger aus der Packet Dispersion Δ_R zusammen mit der Größe eines Messpakets L die Kapazität des Pfads nach Formel 2.9 bestimmen.

$$C = L/\Delta_R \tag{2.9}$$

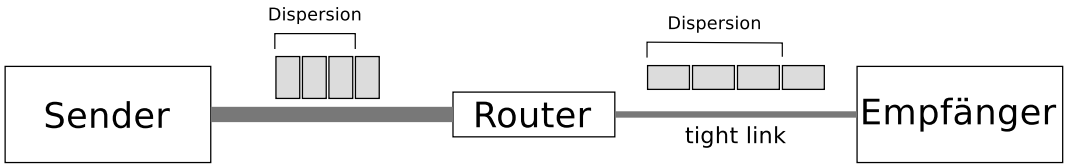

Abbildung 2.4.: Streckung der Messpakete am Tight Link

Dies gilt jedoch nur für eine unbelastete Ende-zu-Ende Verbindung. Im Falle von Cross Traffic, welcher nicht den gleichen Pfad wie die Messpakete nutzt oder wenn tight und narrow link nicht gleich sind, kann es zum Unterschätzen der verfügbaren Datenrate kommen [LDS06].

Da bei diesem Verfahren mit einem Packet Pair die verfügbare Datenrate bestimmt werden kann und die Messung somit sehr schnell ist, wurde vermutet das PGM mit Packet Pairs ein guter Kandidat für Datenratenmessungen im Mobilfunknetz im Rahmen von Messfahrten wäre.

In der Arbeit [Goe10] wurde jedoch festgestellt, dass bei Messungen im Mobilfunk mit dem Packet Pair Verfahren negative Werte gemessen werden können. Dies ist genau dann der Fall, wenn die Packet Dispersion beim Empfänger mehr als doppelt so groß ist wie beim Sender. Somit wurde PGM mit Packet Pairs als potentielle Messmethode verworfen.

2.6.2.3. Bulk Traffic

Da sowohl das Probe Rate Modell, als auch das Probe Gap Modell mit Packet Pairs nur unzuverlässig über Mobilfunk arbeiten, wurde in der Arbeit [Goe10] eine Methode unter Verwendung von Packet Trains entwickelt. Ein Packet Train besteht aus $N > 2$ Paketen. Die Datenrate beim Empfänger berechnet sich dann nach der Formel 2.10, wobei die Train Dispersion Δ_R definiert ist als der zeitliche Abstand zwischen dem Empfang des letzten Bits des ersten Pakets und dem letzten Bit des letzten Pakets des Trains.

$$A = \frac{(N-1) \cdot L}{\Delta_R} \tag{2.10}$$

Die Senderate sollte sich dabei für jeden Packet Train dynamisch an die durch die Übertragungstechnik potentiell zur Verfügung stehende Senderate anpassen. Dabei wurde eine Auslastung von $30-50\%$ angedacht um durch den Mobilfunkbetreiber im Falle von Übertragungstechnik der dritten Generation (UMTS) einen möglichst guten Spreizcode zu erhalten und damit auch eine hohe Senderate. Sollte es zu einer Self Induced Congestion kommen, könnte diese in der Zeit abgebaut werden, in der nicht gesendet wird. Die Latenzmessung, für welche das erste Paket eines Trains genutzt wird,

sollte somit durch die Datenratenmessung nicht gestört werden. Kommt es während des Sendens eines Trains zu einem Technologiewechsel oder zu hohem Cross Traffic, kann es dennoch passieren, dass die Zeit bis zum nächsten Train nicht zum Abbau reicht. Dann wird die Latenzmessung durch die Datenratenmessung beeinflusst und ist somit nicht mehr gültig.

2.7. Mobilfunk

Im folgenden wird eine kurze Zusammenfassung der Funktionsweise der verschiedenen Mobilfunktechnologien gegeben. Die in den Mobilfunkstandards des *Third Generation Partnership Project (3GPP)* [3gp] definierten Grenzwerte für die Datenraten der einzelnen Technologien werden in Tabelle 2.4 dargestellt.

2.7.1. GSM

Das Global System for Mobile Communication (GSM) ist eine Technologie der zweiten Mobilfunk Generation. Während die erste Mobilfunkgeneration noch auf analoge Sprachübertragung setzte, werden bei GSM die Sprachinformationen digitalisiert.

Die Datenübertragung erfolgt bei GSM durch eine Mischung von Frequenz- und Zeitmultiplexing. Mit Hilfe des Frequenzmultiplexing wird das zur Verfügung stehende Frequenzband in 248 Kanäle aufgeteilt. Jeweils zwei Kanäle werden wiederrum gepaart, um getrennte Kanäle zum Senden und Empfangen anzubieten. Auf diesen Frequenzbändern werden GSM Rahmen aufgeprägt, welche jeweils 8 Zeitschlitze haben. Ist ein Mobilfunkgerät aktiv, so belegt es jeweils einen Zeitschlitz in Sende- und Empfangsrichtung, wobei diese zusätzlich zeitlich getrennt sind. Für die Datenkommunikation bietet GSM Datenraten von bis zu 14,4 kbps an (TCH/F 14,4) [3GP03].

2.7.2. HSCSD

Um die Nutzung höherer Datenraten im GSM System zu ermöglichen, wurde *High Speed Circuit Switched Data (HSCSD)* standardisiert [3GP99]. Dabei wird es dem mobilen Teilnehmer erlaubt, mehrere Zeitschlitze in einem GSM Rahmen zu belegen. HSCSD erreicht mit Endgeräten der Multislot Klasse 10 Datenraten von bis zu 57,6 kbps im Downlink [3GP03].

2.7.3. GPRS

GSM und HSCSD bieten jeweils einen leitungsvermittelnden Datendienst. Das bedeutet, dass ein aktives Mobilgerät einen belegten Zeitschlitz komplett nutzt, ohne dass es in diesem Zeitraum Daten senden muss. Des Weiteren werden zu den Downlink Zeitschlitzen jeweils auch die zugehörigen Uplink Zeitschlitze belegt.

Um eine effizienter Datenübertragung zu ermöglichen und parallel das interne Netz der Provider auf die kommende Mobilfunkgeneration vorzubereiten, wurde der *General Packet Radio Service (GPRS)* standardisiert [3GP12]. Bei GPRS wird der leitungsvermittelnde Datendienst durch einen paketvermittelnden Datendienst ersetzt und das interne Netz des Providers entsprechend angepasst. Eine weitere mit GPRS eingeführte Änderung betrifft die Fehlerkorrekturcodes. Die neuen Codes CS-1 bis CS-4 können abhängig von der Signalqualität genutzt werden und bieten gegebenenfalls ein höhere Datenrate pro Zeitschlitz. Theoretisch beträgt die maximale Datenrate bei GPRS 171,2kbit pro Sekunde, wenn alle acht Zeitschlitze genutzt werden. Die tatsächlich verfügbare Datenrate hängt jedoch von der Geräteklasse, welche definiert wie viele Zeitschlitze im Up- und Downlink genutzt werden können. Üblicherweise können maximal vier Zeitschlitze im Downlink belegt werden, was beim Kodierungschema CS-2 53,6 kbps entspricht.

2.7.4. EDGE

Enhanced Data Rates for GSM Evolution (EDGE) ist eine standardisierte Erweiterung von GSM. Durch die Verwendung eines zusätzlichen Modulationsverfahrens (8-PSK) kann die Datenrate eines Zeitslots auf bis zu 59,2kbit pro Sekunde gesteigert werden. Somit sind mit EDGE theoretische Datenraten von bis zu 473kbit pro Sekunde möglich. Die erreichbare Datenrate wird durch die Geräteklasse bestimmt, wobei zum Beispiel mit Geräten der Klasse 10 maximal vier Zeitslots im Downlink belegt werden können, was bedeutet, dass eine Downlink Datenrate von bis zu 220 kbps möglich ist [3GP12].

2.7.5. UMTS

Das *Universal Mobile Telecommunication System (UMTS)* ist eine Mobilfunktechnologie der dritten Generation. Für das UMTS wurde die Luftschnittstelle überarbeitet, um höhere Datenraten zwischen den mobilen Endgeräten und dem Kernnetz zu ermöglichen.

UMTS verwendet auf der Luftschnittstelle zur Trennung der einzelnen Datenströme *Wideband Code*

Technologie	Datenrate (uplink)	Datenrate (downlink)	Anmerkungen
GSM	14,4kbps	14,4kbps	
HSCSD	28,8kbps	57,6kbps	MS-Class 10
GPRS	26,8	53,6kbps	MS-Class 10, CS-4
EDGE	118,4kbps	236,8kbps	MS-Class 10, MCS-9
UMTS	64kbps	384kbps	
HSPA	2Mbps	7,2Mbps	HSUPA Cat.5, HSDPA Cat.8

Tabelle 2.4.: Maximale Datenraten nach Übertragungstechnologie

Division Multiple Access (WCDMA). Dabei wird zunächst das zu sendende Signal mit Hilfe eines Spreizcodes vergrößert, wobei die verfügbare bei größeren Spreizcodes sinkt. Anschließend wird das so entstandene Signal mit einem quasiothogonalen Code codiert. Durch die Verwendung mehrerer, zueinander quasiorthognaler Codes ist es möglich, dass mehrere Endgeräte ein gemeinsames Frequenzband parallel nutzen. UMTS bietet eine maximale Datenrate von 384 kbps im Download.

2.7.6. HSPA

High Speed Packet Access (HSPA) ist ein Sammelbegriff für Erweiterungen zum UMTS Standard, welche eine höhere Datenrate im Up- bzw. Downlink ermöglichen. Verbesserungen werden dabei durch die Verwendung zusätzlicher Modulationsverfahren, das MIMO oder Dual Cell Verfahren erreicht. Die mögliche Datenrate hängt bei den Verfahren von der Gerätekategorie ab. Das in dieser Arbeit genutzte Modem unterstüzt HSDPA nach Kategorie 8 und HSUPA nach Kategorie 5, was eine Uplink Datenrate von 2Mbit pro Sekunde sowie einen Downlink Datenrate von 7,2Mbit pro Sekunde ermöglicht.

Kapitel 3.

Anpassungen am Messgerät

Im Rahmen der Masterarbeit wurde das Messgerät weiterentwickelt. Dabei wurden insbesondere zwei Themenbereiche, welche sich beim mobilen Einsatz des Messgeräts negativ auswirken könnten betrachtet. Um Datenverluste zu vermeiden, sowie die Schreibgeschwindigkeit zu optimieren wurden mehrere Dateisysteme bezüglich ihrer Eignung für Flash Speicher und Toleranz gegenüber Stromverlusten untersucht (Abschnitt 3.1). Des Weiteren wurde die Synchronisation des Messgeräts mit Hilfe eines GPS Empfängers verbessert, womit sich die Genauigkeit von Latenzmessungen steigern lässt (Abschnitt 3.2). Zusätzlich wurde die Installation gegenüber dem Stand der Projektarbeit vereinfacht. Die Anpassungen am Voyage Linux Abbild wurden automatisiert und können jetzt leichter nachvollzogen sowie geändert werden (Abschnitt 3.3).

3.1. Dateisystem

Bei der Auswahl eines passenden Dateisystems müssen die Eigenschaften des nichtflüchtigen Speichermediums sowie des verwendeten *Single Board Computers (SBC)* berücksichtigt werden. Die folgenden beiden Abschnitte beschreiben die kritischen Eigenschaften des Speichermediums und des verwendeten SBC. In Abschnitt 3.1.3 wird dargestellt, wie Voyage Linux diese Herausforderungen löst und warum diese Lösung für das Messgerät nicht verwendet werden kann. Anschließend werden verschiedene Dateisysteme vorgestellt und bezüglich ihrer Eignung diskutiert, bevor die Auswahl für das Messgeräts begründet wird.

3.1.1. Eigenschaften von Flash Speicher

Als nichtflüchtiger Speicher ist bei den SBC vom Typ Alix6F2 eine CompactFlash Karte vorgesehen. Diese ist für den mobilen Einsatz des Messgeräts von Vorteil, da CF Karten intern Flash Speicher

benutzen und somit keine beweglichen Teile aufweisen, welche durch stoßartige Bewegungen im Rahmen von Messfahrten beschädigt werden können.

Ein Nachteil bei der Verwendung von Flash Speicher ist die Limitierung der Schreibzyklen für jede Speicherzelle. Ist die Anzahl der möglichen Schreibzyklen überschritten, kann die Speicherzelle nicht mehr genutzt werden und wird vom Controller als Defekt markiert. Somit sinkt der zur Verfügung stehende Speicher mit der Zeit, abhängig vom Nutzerverhalten. Um die Lebensdauer der einzelnen Speicherzellen und damit auch des gesamten Speichers zu erhöhen nutzen Hersteller Wear-Leveling Algorithmen, welche die Schreibzugriffe auf den gesamten Speicher verteilten. Ob ein solcher Algorithmus in der von uns verwendeten Karte genutzt wird, ist jedoch nicht ohne weiteres festzustellen.

Eine weitere Eigenschaft von Flash Speicher ist, dass sequentielles Schreiben schneller möglich ist, als randomisiertes Schreiben. Wenn eine Zelle des Flash Speichers geschrieben werden soll, muss diese zunächst gelöscht werden. Das Löschen von Speicherzellen ist jedoch nur Blockweise möglich. Beim randomisierten Schreiben bedeutet dies, dass zum Schreiben einer Zelle zunächst der Inhalt der umliegenden Zellen innerhalb des betreffenden *Erase Blocks* zwischengespeichert werden muss. Nach dem Löschen des Blocks müssen anschließend, neben der Zelle die geschrieben werden sollte, alle anderen Zellen des Erase Blocks geschrieben werden um deren Zustand wiederherzustellen. Beim sequentiellen Schreiben über mehrere Zellen kann das Zwischenspeichern und Wiederherstellen für die betreffenden Zellen entfallen, so dass weniger Zeit zum Wiederherstellen benötigt wird.

Ein für die Verwendung von Flash Speicher ausgelegtes Dateisystem sollte also den gesamten Speicherbereich gleichmäßig nutzen und möglichst wenige randomisierte Schreibzugriffe verursachen.

3.1.2. Spannungsverluste

Der verwendete SBC ist für einen 24 Stunden Betrieb ausgelegt. Das geordnete Herunterfahren des Systems ist nur mit Hilfe einer Terminalverbindung über die serielle Schnittstelle oder eine Netzwerkverbindung möglich. Da diese Voraussetzung insbesondere im mobilen Einsatz nicht immer erfüllt ist, muss davon ausgegangen werden, dass das System durch einen plötzlichen Verlust der Stromzufuhr gestoppt wird. Dies kann dazu führen, dass Schreibvorgänge in den nichtflüchtigen Speicher nicht abgeschlossen werden können, was die betreffenden Daten und eventuell auch das Dateisystem korrumpieren kann.

Das verwendete Dateisystem sollte den potentiellen Datenverlust minimieren und unempfindlich gegenüber unvollständigen Schreibzyklen sein.

3.1.3. Voyage Linux Lösung

Um eine Beschädigung des Dateisystems zu vermeiden und die Lebensdauer des Flash Speichers zu erhöhen, nutzt Voyage Linux das in Abschnitt 3.1.4.1 beschriebene *ext2* Dateisystem im Read-Only Modus. Damit trotzdem Logdateien geschrieben werden können, werden einige Verzeichnisse mit Hilfe des *tmpfs* Dateisystems eingebunden. Das tmpfs Dateisystem stellt einen Teil des Arbeitsspeichers als reguläres Verzeichnis zur Verfügung. Da der Inhalt des Arbeitsspeichers bei einem Stromverlust verloren geht, ist ein persistentes Speichern in einem tmpfs Verzeichnis nicht möglich.

Bei einer Nutzung als Messgerät muss jedoch potentiell eine große Anzahl an Daten erfasst und sicher gespeichert werden. Dies steht in Konflikt mit der beschränkten Größe des Arbeitsspeichers des SBC und der nicht-persistenten Speichermöglichkeit. Somit ist diese Lösung für das Messgerät nicht sinnvoll.

Eine Lösung für das Messgerät ist das Einrichten einer zusätzlichen Datenpartition mit einem Dateisystem, welches die in den Abschnitten 3.1.1 und 3.1.2 beschriebenen Eigenschaften berücksichtigt.

3.1.4. Diskussion verschiedener Dateisysteme

Um das persistente Speichern von Messdaten unter den in den Abschnitten 3.1.1 und 3.1.2 beschriebenen Voraussetzungen zu ermöglichen, wird ein passendes Dateisystem benötigt. In den folgenden Abschnitten werden verschiedene Dateisysteme bezüglich ihrer Eignung diskutiert.

3.1.4.1. ext2

ext2 (second extended filesystem) ist das ehemalige Standard-Dateisystem des Linux Kernels. Im folgenden werden die für das Messgerät relevanten Eigenschaften diskutiert. Eine Beschreibung des Dateisystems kann dem Paper [CTT10] entnommen werden.

Änderungen an Dateien finden beim ext2 Dateisystem *in-place* statt. Das bedeutet, dass der Schreibzugriff direkt in den Blöcken der zu ändernden Datei stattfindet. Kommt es während des Schreibzugriffs zu einem Stromverlust und damit zu einem unvollständigen Schreiben, kann die Datei korrumpiert werden.

Zusätzlich zu den eigentlichen Dateien werden auch Metadaten vorgehalten, wie zum Beispiel die Organisation der Dateien in Verzeichnissen. Auch diese Daten können durch einen unvollständigen Schreibzugriff in Folge eines Stromverlusts korrumpiert werden. Defekte Metadaten können jedoch

in den meisten Fällen durch die Verwendung eines Programms zur Überprüfung der Metadaten korrigiert werden. Dieses Programm kann jedoch den Start des Systems abhängig von der Größe des Dateisystems verzögern.

Für Flash Speicher ist die Änderung von Dateien in-place zusätzlich problematisch, da dies ohne entsprechende Wear-Leveling Algorithmen einen randomisierten Schreibzugriff bedeutet. Dieser ist zum einen langsam und kann auf Dauer, wenn immer wieder die gleiche Datei geschrieben wird, die betreffenden Speicherzellen schnell zerstören.

Somit erfüllt das ext2 Dateisystem keine der dargestellten Anforderungen und ist kein guter Kandidat für die Datenpartition.

3.1.4.2. Dateisysteme mit Journaling Funktion

Um es dem Dateisystem zu ermöglichen durch unvollständiges Schreiben beschädigte Dateien zu reparieren, wurden Journaling Dateisysteme eingeführt. Beispiele für Journaling Dateisysteme sind ext3 [Twe98] und ext4 [MCB+07].

Ein Journaling Dateisystem schreibt die anstehenden Dateioperationen in einen speziellen Bereich des Dateisystems, das Journal. Sollte es während des Schreibens einer Datei zu einem Stromverlust kommen, kann das Journal genutzt werden, um die Wiederherstellung der Metadaten zu beschleunigen. Zusätzlich kann versucht werden, eventuell beschädigte Dateien zu reparieren, indem die für diese Datei nicht abgeschlossenen Operationen beendet werden.

Die Verwendung von Journaling Dateisystemen zusammen mit Flash Speicher ist jedoch kritisch. Da das Journal in einem bestimmten Speicherbereich liegt, ist dieser einer deutlich erhöhten Frequenz von randomisierten Schreibzugriffen ausgeliefert. Dies kann wie in 3.1.1 beschrieben dazu führen, dass die betreffenden Speicherzellen schneller zerstört werden. Zusätzlich kann trotz des Journals ein Datenverlust nicht komplett ausgeschlossen werden.

Aus diesen Gründen sind Journaling Dateisysteme nicht für die Datenpartition des Messgeräts geeignet.

3.1.4.3. NILFS2

Bei *NILFS2 (New Implementation of a Log-Structured File System)* ([NIL]) handelt es sich um die Implementierung eines Dateisystem Prinzips, welches 1988 in [RO91] vorgeschlagen wurde. Die Idee

der Autoren war, den nichtflüchtigen Speicher als zirkuläres Medium zu betrachten. Anstatt Änderungen an Dateien in-place durchzuführen, sollten die Änderungen sequentiell in einen Datenstrom geschrieben werden. Dies sollte sowohl für normale Dateien, als auch für Metadaten gelten. Das Ziel dieses Ansatzes war ein möglichst hoher Schreibdurchsatz.

NILFS2 erweitert diesen Ansatz um ein kontinuierliches Checkpointing. Jede Sekunde oder nach einem synchronen Schreiben wird ein solcher Checkpoint, der einen konsistenten Punkt markiert, angelegt. Der Nutzer kann beliebige Checkpoints zu Snapshots erklären, welche auch separat eingebunden werden können. Somit ist es zum Beispiel möglich, versehentlich gelöschte Dateien wiederherzustellen, indem man einen alten Snapshot des Dateisystems einbindet und die Datei in das aktuelle System kopiert.

Beim sequentiellen Schreiben in den Speicher füllt sich das Medium schneller als bei in-place Dateisystemen. Um nicht mehr benötigte Speicherblöcke zu identifizieren und für eine weitere Verwendung freizugeben, wird daher ein Garbage Collector benötigt, welcher den freien Speicherplatz verwaltet. Im Falle von NILFS2 berücksichtigt dieser auch, ob ein Speicherblock durch einen Snapshot benötigt wird oder nicht.

Auch wenn NILFS2 nicht für Flash Speicher entwicklet wurde, so ist das Design des Dateisystems sehr gut dafür geeignet. Das sequentielle Schreiben sorgt für ein automatisches Wear-Leveling und nutzt den Geschwindigkeitsvorteil gegenüber randomisiertem Schreiben aus.

Der Nachteil eines LogFS Dateisystems ist für Flash Speicher nicht relevant. Bei einem LogFS können die Blöcke einer Datei über das gesamte Medium verteilt sein. Bei einem rotierenden Speichermedium mit vergleichsweise langen Zugriffszeiten wird dadurch das Lesen der Datei verlangsamt.. Da bei Flash Speicher das Lesen eines Blocks immer die gleiche Zeit in Anspruch nimmt, egal an welcher Stelle er sich befindet, ist dies kein Problem.

NILFS2 ist daher ein potentieller Kandidat als Dateisystem für das Messgerät. Das kontinuierliche Checkpointing beschränkt den Datenverlust im Falle eines Spannungsverlusts auf eine Sekunde und durch das sequentielle Schreiben findet automatisch ein Wear-Leveling statt.

3.1.4.4. Btrfs

Das *Btrfs (b-tree file system)* [RBM12] ist ein Dateisystem, welches die Idee eines copy-on-write B-tree für die Verwaltung eines Dateisystems nutzt. Dabei werden Änderungen nicht in-place ausgeführt, sondern jeweils eine Kopie der betreffenden Speicherstruktur angelegt, welche dann die aktuelle Sicht darstellt. Die alte Sicht auf den Speicherbereich bleibt dabei erhalten, so dass diese verfügbar ist,

bis sie durch eine Garbage Collection gelöscht wird. Somit bietet Btrfs ein Checkpointing bei jeder Änderung am Dateisystem.

Zusätzlich können manuell Snapshots angelegt und separat eingebunden werden. Diese werden dann bei einer Garbage Collection nicht berücksichtigt. Des Weiteren besteht die Möglichkeit, durch Hardware Fehler korrumpierte Dateien an einer Checksumme zu erkennen. Diese Checksummen werden sowohl für normale, als auch für Metadaten erstellt.

Die Copy-on-Write Eigenschaften von Btrfs sorgen für ein weitgehend sequentielles Schreiben auf dem Medium, was widerrum für Flash Speicher vorteilhaft ist. Des Weiteren sorgt es dafür, dass Dateien durch einen abgebrochenen Schreibvorgang nicht beschädigt werden. Zusätzlich verhindert das laufende Checkpointing, sowie die Checksummen über alle Dateien einen Datenverlust.

Somit ist Btrfs ein potentieller Kandidat als Dateisystem für das Messgerät. Es muss aber beachtet werden, dass Btrfs momentan noch in der Entwicklung ist und viele Änderungen erfährt, weshalb ein aktueller Kernel von Vorteil ist.

3.1.5. Auswahl des Dateisystems

NILFS2 und Btrfs kommen beide mit den Herausforderungen bezüglich Spannungsverlusten und Flash Speicher zurecht. NILFS2 hat zusätzlich den Vorteil, dass die Entwicklung schon weiter fortgeschritten ist, was die Stablilität des Dateisystems erhöht. Btrfs ist allerdings aus Sicht der Kernel Entwickler das kommende Standarddateisystem des Linux Kernels und wird im größeren Rahmen durch die Kernel Entwickler und Unternehmen unterstützt, was ein schnelleres Feedback bei Problemen ermöglicht.

Um eine Entscheidung für ein Dateisystem zu erleichtern, wurden Performance Tests durchgeführt. Dazu wurde der Benchmark bonnie++ [bon] verwendet. Der Benchmark erstellt Dateien, fragt ihren Status ab und löscht sie anschließend wieder. Die Operationen werden sowohl sequentiell als auch randomisiert ausgeführt.

Die Tests lieferten für ext2 und Btrfs ähnliche Werte, was dafür spricht, dass nicht das Dateisystem den Durchschsatz beschränkt, sondern die Schnittstelle zur CF Karte. Dies ist dadurch zu erklären, dass CompactFlash Karten intern einen IDE Controller nutzen.

NILFS2 wurde in mehreren Versuchen mit verschiedenen Konfigurationen getestet. Keiner der Versuche konnte jedoch beendet werden, da der Speicher immer komplett gefüllt wurde. Darauf hin beendete sich der Benchmark. Das spricht dafür, dass der Garbage Collector zumindest in unserer

Konfiguration nicht ordentlich funktioniert hat.

Somit wurde Btrfs als Dateisystem für die Datenpartition des Messgeräts ausgewählt. Die Systempartition sollte weiterhin als Read-Only ext2 Dateisystem betrieben werden.

3.2. Zeitsynchronisation

Neben der Messung von Datenraten ermöglicht das *Rate Measurement Framework (RMF)* (siehe Abschnitt 2.2) auch die Messung der Latenzen zwischen den beteiligten Systemen. Um eine möglichst genaue Messung zu ermöglichen, werden genaue Zeitstempel für das Senden und Empfangen von Paketen benötigt. Außerdem müssen die Uhren der beteiligten Systeme synchonisiert sein.

Für die Synchronisation der Uhren von Computern, welche mit dem Internet verbunden sind, kann das in Abschnitt 2.5 beschriebene Network Time Protocol in Kombination mit mehreren im Internet erreichbaren Zeitservern genutzt werden. Im Fall des Messgeräts ist dies jedoch nicht sinnvoll, da die einzige Verbindung zum Internet über den Mobilfunkkanal besteht. Dieser hat für gewöhnlich die Eigenschaft, dass die Latenzen in Up- und Downlink asynchron sind. Das hat zur Folge, dass ein Fehler nach Formel 2.3 entsteht, da NTP von synchronen Latenzen zum Server ausgeht, worauf Herr Goebel in seiner Arbeit hingewiesen hat [Goe10].

Somit wurde eine andere Zeitquelle benötigt, welche sowohl dem Messclient, als auch dem Messserver gemeinsam zur Verfügung stehen musste. Wie in Abschnitt 2.5.2 erwähnt, kann das GPS als gemeinsame Zeitquelle genutzt werden. Im Rahmen der Projektarbeit wurde eine entsprechende Konfiguration zusammengestellt und getestet. Der maximal erreichbare Offset zum PPS Signal lag dabei bei 50μs, was den in [Goe10] beschriebenen Werten entsprach. Dieser Wert liegt jedoch über dem in [PPS] beschriebenen Wert. Zusätzlich wurde festgestellt, dass die Signale des GPS Empfängers teilweise nicht durch das NTP Referenzprogramm berücksichtigt wurden.

3.2.1. Clock Select Algorithmus und PPS

Eine Analyse der Logdateien des NTP Daemon zeigte, dass das vom GPS Empfänger emittierte PPS Signal teilweise verworfen wurde. Im normalen Betrieb wurde das PPS Signal als primäre Quelle genutzt, während das Standardsignal des GPS Empfängers als zusätzliche Quelle vorgehalten wurde. Zeitweise wurden jedoch beide Signale durch den in Abschnitt 2.5.1.2 beschriebenen Clock Select Algorithmus verworfen. Eine Recherche im Online Handbuch ([ntp]), sowie im Buch [Mil10] lieferte die Erklärung. Beide Signale haben einen sehr geringen Offset, da sie zu einer lokalen Uhr, dem GPS

Empfänger gehören. Liegen die übermittelten Zeitstempel so weit auseinander, dass die mit Hilfe der Offsets gebildeten Intervalle sich nicht mehr überschneiden, so kann kein Intersection Intervall gebildet werden. Da der Algorithmus nun nicht bestimmen kann, welcher der beiden Zeitstempel korrekt ist, werden beide verworfen.

Um dieses Problem zu lösen, muss der Intersection Algorithmus für eine der beiden Uhren deaktiviert werden. Das hat jedoch zur Folge, dass diese Quelle immer zur Bestimmung der aktuellen Zeit genutzt wird. Da das PPS Signal, immer genutzt werden soll wenn es empfangen wird, wurde der entsprechende Eintrag in der NTP Konfiguration um den Parameter *true* ergänzt.

```
server 127.127.28.1 true minpoll 4 maxpoll 4
fudge 127.127.28.1 refid PPS
server 127.127.28.0
fudge 127.127.28.0 time1 0.600 refid GPS
```

Listing 3.1: Auszug aus ntp.conf für GPS Empfang

3.2.2. Geringe Genauigkeit trotz PPS

Sowohl das Kapitel im NTP Handbuch bezüglich PPS [PPS], sowie die technischen Spezifikationen des verwendeten Garmin GPS 18x LVC beschreiben eine erreichbare Genauigkeit im einstelligen Mikrosekundenbereich bei Verwendung des PPS Signals. Dieser Wert konnte jedoch auch nach mehreren Tagen nicht erreicht werden. Wie oben beschrieben lag der maximal erreichbare Offset bei 50μs. Die Ursache für den erhöhten Offset war in diesem Fall die Konfiguration der seriellen Schnittstelle.

Standardmäßig puffern serielle Schnittstellen, sofern sie die Möglichkeit besitzen, eingehende Signale über einen kurzen Zeitraum um die CPU zu entlasten. Dies sorgt im Fall des PPS Signals für eine zusätzliche Latenz. Der Puffer kann mit Hilfe der Option *low_latency* deaktiviert werden. Diese kann mit Hilfe des Programms *setserial* eingestellt werden. Um diese Änderung persistent zu machen, muss die Konfigurationsdatei */etc/serial.conf* angepasst werden (siehe Listing A.7). Nach der Deaktivierung des Puffers konnte der Offset zum PPS Signal in den einstelligen Mikrosekundenbereich gesenkt werden.

3.3. Installation

Um Einflüsse durch unterschiedliche Softwareversionen oder Konfigurationen auf die späteren Messungen zu vermeiden, sollte ein einheitliches Installationsabbild für die an den Messungen beteiligten

Systeme genutzt werden. Dies ist insbesondere für die Alix Systeme sinnvoll, da diese weitgehend identisch bezüglich der Hardwareausstattung sind. Ausnahmen sind eine CF Karte und ein System, welches durch eine Commell MP-954 Erweiterungskarte zusätzliche serielle Schnittstellen anbietet.

Im folgenden wird beschrieben, wie ein modifiziertes Abbild von Voyage Linux [Voya] erstellt wurde und wie es mit Hilfe der Methoden des Debian Live Projekts [Debb] weiter modifiziert werden kann.

3.3.1. Virtuelle Maschine als Basis

Als Basis für die Modifikation der Abbilder, sowie die Kompilierung neuer Softwareversionen und Kernel dient eine virtuelle Maschine. Für die Virtualisierung wird die Software VirtualBox [Vir] der Firma Oracle genutzt, welche für mehrere Plattformen verfügbar ist. Zusätzlich ist VirtualBox teilweise als Open Source Software verfügbar. Somit kann die virtuelle Maschine mit hoher Wahrscheinlichkeit auch zukünftig genutzt werden.

Als Betriebssystem für die virtuelle Maschine wurde Debian 6.0 (Squeeze), in der Variante i386 ausgewählt. Das für die Messgeräte genutzte Betriebssystem Voyage Linux 0.8.5 (siehe Abschnitt 3.3.2) basiert auf dieser Debian Version. Die Verwendung der gleichen Basis sollte Komplikationen durch unterschiedliche Softwareversionen vermeiden.

3.3.2. Modifikationen am Voyage Linux Abbild

Um die Installation mehrerer Messgeräte zu vereinfachen und eine einheitliche Softwareauswahl auch bezüglich der Versionen zu gewährleisten, wurde ein modifiziertes Installationsabbild benötigt. Im Folgenden wird beschrieben, welche Werkzeuge durch das Voyage Linux Projekt genutzt werden um die Distribution zu erstellen und wie diese modifiziert werden können, um eine an die Bedürfnisse des Messgeräts angepasste Version zu generieren.

3.3.2.1. Bezug und Bau eines Voyage Linux Abbilds

Voyage Linux nutzt zur Modifikation des Debian Basissystems und zum Erstellen des Abbilds die Werkzeuge des Debian Live Projekts in Kombination mit einigen Shell Skripten.

Mit Hilfe des Programms *lb_config* wird ein Konfigurationsverzeichnis mit Standardwerten für die Erstellung eines Debian Live Systems generiert. Änderungen an der Konfiguration können zum einen

durch die Parameter von lb_config vorgegeben, oder später manuell im Konfigurationsverzeichnis durchgeführt werden. Neben der Paketauswahl können mit Hilfe der Konfigurationsverzeichnisse auch der verwendete Kernel, sowie die Konfiguration des Grundsystems nach der Installation beeinflusst werden. Zusätzlich können zu verschiedenen Zeitpunkten während der Erstellung des Abbildes eigene Skripte zur weiteren Modifikation genutzt werden.

Wurde eine gültige Konfiguration erstellt, können mit Hilfe des Programms *lb_build* Abbilder in verschiedenen Formaten wie ISO-Images oder tar-Archiven erstellt werden. Für die Messgeräte wird ein tar-Archiv genutzt, da die Voyage Installationsskripte nach dem Entpacken verfügbar sind und die Installation erleichtern.

Die von Voyage Linux genutzte Konfiguration kann von [Voyb] zusammen mit dem Skript *build.sh* bezogen werden. build.sh wird dazu genutzt, um den Bau eines Voyage Linux Abbildes zu vereinfachen und eine zusätzliche Modifikation im Grundsystem vorzunehmen. Während Änderungen an der Paketauswahl in build.sh nachvollzogen werden können, müssen zusätzliche Modifikationen im Verzeichnis *config/chroot_local_hooks* betrachtet werden.

3.3.2.2. Modifkation des Voyage Linux Abbilds

Da für das Messgerät diverse Anpassungen bezüglich der Software und den Konfigurationsdateien, sowie am Kernel notwendig sind, wurde ein modifiziertes Image von Voyage Linux benötigt. Um die Änderungen zu Dokumentieren und damit die zukünftige Verwendung zu erleichtern, wurden die Skripte *alixBuild* und *alixConfig* entwickelt. alixBuild wird genutzt um das modifizierte Abbild zu erstellen, während alixConfig genutzt werden kann, um ein laufendes System zu konfigurieren.

Bei der Nutzung des Skripts alixBuild.sh wird das Konfigurationsverzeichnis von Voyage Linux aus dem Internet heruntergeladen. Anschließend werden die benötigten Änderungen vorgenommen und das neue Abbild gebaut. Das Skript alixConfig.sh wird dabei in das Abbild integriert, so dass es auf einem neu installierten System direkt zu Verfügung steht.

Der Quellcode der Skripte ist auf der beiliegenden CD zu finden.

3.3.3. Installation auf der CF Karte

Die Installation des Voyage Linux Abbilds auf einer CF Karte wird durch Skripte unterstützt. Diese richten jedoch nur eine Partition ein. Da für das Messgerät eine zusätzliche Datenpartition benötigt wird, muss die Partitionierung vor der eigentlichen Installation manuell durchgeführt werden.

Partition	Sektoren (Sektorgröße 512 Byte)		Größe (MB)	Dateisystem
	Beginn	Ende		
System	8192	2097151	996,1	ext2(Read-Only)
Daten	2097152	15662303	6623,6	Btrfs

Tabelle 3.1.: Alix Partitionen

3.3.3.1. Einrichten der CF Karte

Werden bei Flash Speicher Daten gelöscht, geschieht dies in Blöcken fester Größe, den sogenannten *Erase Blocks*. Um die Performance des Flash Speichers zu maximieren, sollte man die Partitionen an der Größe der Erase Blocks ausrichten. Durch die Ausrichtung wird vermieden, dass sich ein Block des Dateisystems über zwei Erase Blöcke verteilt, sofern die Blockgröße des Dateisystems nicht größer als als die Erase Block Größe ist. Somit werden unnötige Löschoperationen vermieden und die Lebensdauer des Flash Mediums erhöht. Da nicht jeder Hersteller die Größe der Erase Blocks veröffentlicht, wurde das Programm *flashbench* [fla] entwickelt. Dieses misst die Schreibgeschwindigkeit für verschiedene Blockgrößen. Mit Hilfe dieser Daten kann man die Erase Blockgröße bestimmen.

Die verwendeten CF Karten haben eine Erase Blockgröße von 4MB. Eine Übersicht über die eingerichteten Partitionen liefert Tabelle 3.1.

Die Formatierung der Systempartition sollte durch das im folgenden vorgestellte Installationsskript erfolgen. Es aktiviert zusätzliche Tuning Optionen des ext2 Dateisystems. Die Datenpartition wird mit Hilfe der *btrfs-tools* formatiert. Ist das Messgerät gestartet, genügt dazu der Aufruf von *mkfs.btrfs /dev/hda2*.

3.3.3.2. Installation des Abbilds

Die Installation des modifizierten Voyage Linux Abbilds erfolgt nach der gleichen Methode wie die des regulären Abbilds. Eine Beschreibung der Installation liefert die README Datei des entpackten Abbilds oder die *Getting Started* Webseite von Voyage Linux [Voya]. Als *Target Profile* sollte *Alix* gewählt werden.

Kapitel 4.

Anpassungen der Messsoftware

Im Laufe der Masterarbeit wurden mehrere Anpassungen an den verwendeten Programmen nötig. Die Anpassungen am *Rate Measurement Framework (RMF)* werden in Abschnitt 4.1 beschrieben und waren meist durch kritische Fehler bedingt, welche korrekte Messungen verhinderten. Abschnitt 4.2 beschreibt die Implementierung einer eigenen Messmethode, welche den Backchannel der RMF nutzt um die Senderate an die letzte bekannte Empfangsrate der Gegenstelle anzupassen. Abschließend werden in Abschnitt 4.3 die Anpassungen am SEC beschrieben.

4.1. Rate Measurement Framework

Die folgenden Abschnitte beschreiben die Probleme welche bei der Verwendung des RMF aufgetreten sind und deren Lösung. Der Abschnitt 4.1.1 werden die Schwierigkeiten im Zusammenhang mit Multithreading erklärt, welche ein korrektes Beenden des RMF verhindert haben. Im folgenden Abschnitt 4.1.2 werden Anpassungen am Netzwerkquellcode dargestellt, welche einen Verbindungsaufbau über ein *Network Address Translation (NAT)* System ermöglichen. Anschließend beschreibt Abschnitt 4.1.3 Probleme bei der Übergabe von Paketdaten zwischen den Threads.

4.1.1. Thread Problematik

In [Wil12] wird erklärt, dass es bei den Tests des RMF vereinzelt zu Problemen beim Beenden der Threads und somit auch beim Beenden des Programms gekommen ist. Da spätere Experimente automatisiert ablaufen sollen, sind solche Probleme für unsere Arbeit kritisch und wurden deshalb zu Beginn der Arbeit behandelt.

4.1.1.1. Sleep Error

Um den Fehler, welcher sporadisch das korrekte Beenden des RMF verhinderte zu lokalisieren, wurden mehrere Testreihen ausgeführt bis der Fehler auftrat. Eine Analyse der Logdateien lieferte keinen Hinweis auf die Ursache. Bei genauerer Betrachtung des Programmablaufs auf der Standardausgabe wurde jedoch eine mehrmals aufblitzende Fehlermeldung bemerkt, welche nicht im Logfile zu finden war. Eine Suche im Quellcode nach der Fehlermeldung lieferte den Codeausschnitt aus Listing 4.1.

```
while( clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &sleeping, NULL)
    != 0 )
 perror("timeHandler.sleepUntil() => nanosleep_error");
```

Listing 4.1: timeHandler.cpp (Rev. 52560); Zeilen 185-186

Im Falle eines Fehlers der Funktion clock_nanosleep() wird in diesem Codeausschnitt eine Endlosschleife geschaffen. Das führte dazu, dass sich das Hauptprogramm nicht beenden konnte, da dieses darauf wartet, dass sich alle Threads beenden. Die Fehlermeldung wurde durch den Aufruf von perror() nicht in das Logfile geschrieben.

Somit war bekannt, warum sich das Programm nicht beenden konnte, aber nicht der Auslöser. Ein Blick in die Dokumentation zu clock_nanosleep() zeigt, dass die Funktion einen Wert ungleich Null zurückgibt, wenn die Auführung durch einen Signal Handler unterbrochen, oder ein ungültiger Eingabeparameter verwendet wurde.

Die Zeit, die clock_nanosleep() schlafen soll, wird der Funktion durch eine timespec Struktur übergeben.

```
struct timespec {
  time_t tv_sec;   /* seconds */
  long   tv_nsec;  /* nanoseconds [0 .. 999999999] */
};
```

Listing 4.2: Auszug aus [man12a]

Der Wertebereich von tv_nsec ist größer, als der von der Funktion clock_nanosleep() als gültig definierte Bereich. Somit muss ein Überlauf durch den Nutzer selbst abgefangen werden. Listing 4.3 zeigt, dass dies auch versucht wurde, allerdings nicht korrekt.

```
timing.getTime();
if( (timing.lastProbedTime.nanoseconds + 10000) > 1000000000 ) {
  timing.lastProbedTime.seconds += 1;
  timing.lastProbedTime.nanoseconds -= 10000;
}
else
  timing.lastProbedTime.nanoseconds += 10000;
```

Listing 4.3: networkHandler.cpp (Rev. 52560); Zeilen 4269-4276

Ziel dieses Auschnitts war es, den Zeitpunkt zum nächsten Aufwachen zu bestimmen. Die Einträge des Objekts timing wurden anschließend in ein timespec Objekt übertragen, welches dann als Parameter für ein clock_nanosleep() diente. Die if-Abfrage lässt jedoch einen Wert von 1000000000 für das spätere tv_nsec zu. Damit wird die Endlosschleife aus Listing 4.1 ausgelöst. Der Fehler konnte leicht durch eine minimale Änderung der Bedingung korrigiert werden.

Durch die Änderung der Bedingung konnte zwar die Endlosschleife verhindert werden, aber der Quellcode war dennoch nicht korrekt. Wird die Bedingung erfüllt, so wird 999990000ns statt der geplanten 10000ns geschlafen. Eine korrigierte Implementierung kann Listing 4.4 entnommen werden.

```
timing.getTime();
if( (timing.lastProbedTime.nanoseconds + 10000) >= 1000000000 ) {
  timing.lastProbedTime.seconds += 1;
  timing.lastProbedTime.nanoseconds -= 999990000;
}
else
  timing.lastProbedTime.nanoseconds += 10000;
```

Listing 4.4: networkHandler.cpp (Rev. 53288); Zeilen 4290-4297

Dieser Fehler war an mehreren Stellen in verschiedenen Klassen im Quellcode präsent und wurde entsprechend geändert. Des Weiteren prüft die Funktion sleepUntil() inzwischen selber die Eingabeparameter und gibt eine Medlung im Fehlerfall aus, welche in das Logfile geschrieben wird.

4.1.1.2. Mutex Error

Nachdem das clock_nanosleep() Problem gelöst war, zeigten weitere Testreihen, dass es immer noch Probleme mit dem Beenden des Framworks gab. Mit Hilfe von Logdateien, welche unter Verwendung

des Debug Modus des RMF auch die Belegung der Mutexe speichern, wurde der Fehler analysiert. Dabei zeigte sich, dass die Threads ThreadedSenderDataDistributor, sowie ThreadedReceiverDataDistributor manchmal bei dem Versuch Mutexe zu belegen stoppten. Eine zeilenweise Analyse der Funktionen zeigte schliesslich, dass die Mutexe für die Sent- bzw. Receivequeue beim Beenden der Threads nur dann freigegeben wurden, wenn die Queues leer waren (siehe Listing 4.5). Da jedoch im weiteren Ablauf der Threads die selben Mutexe nochmal reserviert werden mussten, um die Queues zu leeren, blockierten die Threads. Zur Korrektur dieses Fehlers wurde der Quellcode so geändert, dass die Mutexe auch unabhängig vom Zustand der Queues wieder freigegeben werden.

```
if(flag_sendingDone) {
// If both queues are empty exit otherwise empty queues
pthread_mutex_lock( &networkHandler_mSentQueue_1 );
pthread_mutex_lock( &networkHandler_mSentQueue_2 );
if(mSentQueue_1.empty() && mSentQueue_2.empty()) {
  pthread_mutex_unlock( &networkHandler_mSentQueue_2 );
  pthread_mutex_unlock( &networkHandler_mSentQueue_1 );
  break;
}
```

Listing 4.5: networkHandler.cpp (Rev. 53288); Zeilen 4290-4297

4.1.1.3. Test der Echtzeitfähigkeit

Wie in der Einführung zum RMF erwähnt, unterstützt das Programm die Verwendung von Echtzeit Scheduling um die Performance zu erhöhen. Sowohl in [GRT09], als auch in [NSGR05] wird betont, dass auch eine höhere Geschwindigkeit beim Senden und Empfangen von Netzwerkpaketen zu erwarten ist.

Während der Testläufe wurde festgestellt, dass keine Pakete gesendet wurden, wenn als Scheduler SCHED_FIFO ausgewählt wurde, während SCHED_RR keine Probleme offenbarte. Auf Grund mangelnder Erfahrung mit Echtzeit Programmierung unter Linux, wurde zunächst [Lov10] konsultiert. Insbesondere folgendes Zitat erklärte das Problem:

> Two or more SCHED_FIFO tasks at the same priority run round robin, but again only yielding the processor when they explicitly choose to to so.

Es war also naheliegend, dass ein Thread mit hoher Priorität den Prozessor nicht mehr freiwillig zurückgibt. Der fehlerhafte Thread war in dem Fall der ThreadedMeasurementSender. Dieser nutzt eine busy-wait Schleife, um schnellstmöglich auf Pakete in der SendingQueue zu reagieren. Wenn

die Priorität dieses Threads jedoch gleich oder höher ist als die Priorität des Hauptthreads, blockiert er dadurch alle anderen Threads. Die Lösung dieses Problems bestand in dem Aufruf der Funktion sched_yield() durch den Thread. Diese Methode gibt den Prozessor freiwillig frei und erlaubt es so, das andere Threads einen Teil der Prozessorzeit zugeteilt bekommen.

Eine anschließende Analyse des Quellcodes der Klasse systemOptimization zeigte zusätzlich folgende fehlerhafte Verwendung auf. In der Methode activateRealtimeThreadPriority() wird versucht, den Threads eine Priorität zuzuweisen, auch wenn der Standardscheduler genutzt wird. Dieser untertützt jedoch laut [Lov10] keine Prioritäten und insbesondere keine Verdrängung von Prozessen niedriger Priorität durch Prozesse hoher Priorität. Statt dessen wird versucht den Threads jeweils einen *fairen* Anteil an der zur Verfügung stehenden CPU Zeit zuzuteilen. Dieser Anteil kann jedoch nicht durch die Priorität, sondern nur durch den Nice-Level eines Prozesses beeinflussen.

4.1.2. Überarbeitung des Netzwerkcodes

Im Rahmen der Tests bezüglich der Echtzeitfähigkeit des RMF zeigte sich, dass der Verbindungsaufbau des Measurement Channels reproduzierbar scheiterte. Somit konnten keine Messungen gestartet werden. Abschnitt 4.1.2.1 beschreibt den Fehler und die passende Lösung. Da bei der Implementierung der Lösung größere Änderungen am Netzwerkcode vorgenommen werden mussten, wurde der Quellcode der Netzwerkklasse parallel neu organisiert um diesen zu vereinfachen. Eine Beschreibung dieser Änderungen liefert Abschnitt 4.1.2.2.

4.1.2.1. Reimplementierung der NAT Überbrückung

Der Verbindungsaufbau für den Measurement Channel sieht vor, dass der Client zunächst UDP Pakete an den Server schickt, um ein eventuell auf Clientseite vorhandenes NAT System zu überbrücken. Ist ein Client über ein NAT System mit dem Internet verbunden, kann dieser nicht direkt aus dem Internet heraus kontaktiert werden. Zunächst muss der Client eine Nachricht an eine aus dem Internet erreichbare Serverinstanz senden, damit das NAT eine Zuordnung zwischen dem Client und dem Server vornehmen kann. Zusätzlich werden durch das NAT die Sender IP Adresse und die Portnummer geändert, damit Antwortpakete durch das NAT angenommen werden können. Hat der Server ein Paket erhalten, kann er Antwortpakete an das NAT schicken, welche durch dieses weitergeleitet werden.

Da UDP Pakete auf dem Weg zum Empfänger verloren gehen können, müssen durch den Client gegebenenfalls mehrere Pakete an den Server geschickt werden. In der Implementierung von Herrn Wilken wurden dazu von Clientseite fünf UDP Pakete direkt nacheinander an den Server gesendet und anschließend auf eine Bestätigung durch den Server gewartet. Bei Verwendung des Echtzeit Schedulers

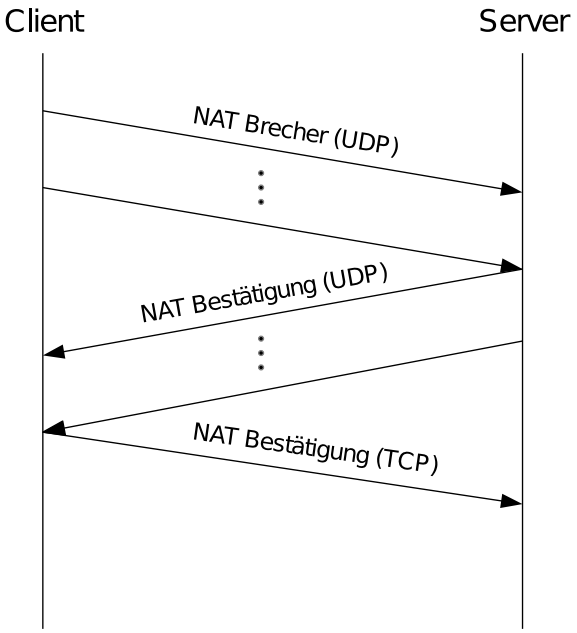

Abbildung 4.1.: Verbindungsaufbau UDP

wurden die Pakete jedoch so schnell abgeschickt, dass die Server Instanz noch gar nicht zum Empfang bereit war und die Pakete verworfen wurden. Da die Serverinstanz kein eingehendes Paket registrieren konnte, wurde nach einer Wartezeit der Verbindungsaufbau für gescheitert erklärt und somit die Messung nicht durchgeführt.

Der Fehler konnte dadurch behoben werden, dass die Pakete nicht direkt nacheinander gesendet wurden, sondern über einen längeren Zeitraum. So hatte der Server mehr Zeit die UDP Verbindung auf seiner Seite einzurichten. Der Empfang eines UDP Pakets vom Client durch den Server erlaubt jedoch keine Aussage darüber, ob der Server UDP Pakete an den Client schicken kann. Daher wurde der Verbindungsaufbau durch das Senden von UDP Bestätigungen erweitert, wobei wieder mehrere Nachrichten über einen längeren Zeitraum gesendet wurden. Empfängt der Client eine solche Bestätigung, kann er verifizieren, dass in beide Richtungen gesendet werden kann (Abbildung 4.1).

Um den Server über den erfolgreichen Verbindungsaufbau zu informieren, wird anschließend eine TCP Nachricht über den Control Channel gesendet. Sollte diese Nachricht nicht erfolgreich übertragen werden können, ist die TCP Verbindung und damit der Control Channel unterbrochen, so dass eine Messung nicht mehr stattfinden kann.

Die Pakete zum Überbrücken und Bestätigen sind durch eine 32bit Typnummer kenntlich gemacht. Zusätzlich wird eine zufallsgenerierte 32bit Zahl zur Identifizierung einzelner Clients genutzt. Diese wird im Vorfeld bei der Synchronisation der allgemeinen Parameter über den Control Channel ausgetauscht.

Funktion	Beschreibung
createSocket()	Fordert abhängig von der gewünschten IP Version einen Socket an und bindet ihn an einen Port.
createTcpChannel()	Wartet auf eine eingehende TCP Verbindung und stellt diese her.
connectToTcpChannel()	Verbindet sich zu entferntem TCP Socket.
createUdpChannel()	Stellt auf Anfrage UDP Verbindung inklusive NAT Überbrückung her.
connectToUdpChannel()	Initiiert UDP Verbindungsaufbau inklusive NAT Überbrückung.

Tabelle 4.1.: Neue Netzwerkfunktionen

4.1.2.2. Reimplementierung des Netwerkcodes

Während der Begutachtung des Netzwerkquellcodes bezüglich der NAT Reimplementierung, ist aufgefallen dass im Quellcode der Netzwerkklasse die Funktionen für den Verbindungsaufbau für jeden Kanal neu implementiert wurden. Zusätzlich wurde für IPv4 und IPv6 Verbindungen jeweils eigener Quellcode geschrieben, so dass eine große Menge duplizierten Quellcodes vorhanden war. Um die spätere Pflege des Quellcodes zu erleichtern, wurden darauf hin die Funktionen zum Verbindungsaufbau neu geschrieben. Dabei wurde darauf geachtet, dass der gleiche Quellcode IPv4 und IPv6 Verbindungen annehmen kann. Um nicht die Schnittstelle zu anderen Klassen zu brechen, wurden die alten Schnittstellenfunktionen genutzt um die neuen Funktionen aufzurufen. Eine Auflistung der neuen Netzwerkfunktionen liefert Tabelle 4.1.

4.1.3. Probleme des Queue Designs

Um eine möglichst hohe Übertragungsrate zu erreichen, wurden bei der Implementierung des RMF jeweils eigene Threads zum Senden und Empfangen von Paketen entwickelt. Die Aufgabe des TMSE und TMRE beschränkt sich auf das Entnehmen/Eintragen von Paketen aus/in FIFO Queues und das Senden/Empfangen. Eine möglichst hohe Performance sollte dadurch erreicht werden, dass jeweils zwei Queues abwechselnd genutzt werden. Das Ziel war dabei, die Wartezeit für den exklusiven Zugriff auf die durch Mutexe geschützten Queues zu minimieren. Bei der Implementierung wurden jedoch Fehler gemacht, die teilweise erst bei der Nutzung von dynamischen Messverfahren auffallen konnten. Die folgenden Abschnitte beschreiben die Fehler und wie sie behoben wurden.

```
[...]
while( ! mSentQueue_1.empty() ) {
  toCopy = mSentQueue_1.front();
  toProcess.id = toCopy.id;
  toProcess.sendtime_seconds = toCopy.sendtime_seconds;
  toProcess.sendtime_nanoseconds = toCopy.sendtime_nanoseconds;
  toProcess.fixed_payload_length = toCopy.fixed_payload_length;
  memcpy(&toProcess.fixed_payload, toCopy.fixed_payload, toCopy.
      fixed_payload_length);
  toProcess.dynamic_payload_length = toCopy.fixed_payload_length;
  memcpy(&toProcess.dynamic_payload, toCopy.fixed_payload, toCopy.
      fixed_payload_length);
  mSentQueue_1.pop();
[...]
while( ! mSentQueue_2.empty() ) {
  toCopy = mSentQueue_2.front();
  toProcess.id = toCopy.id;
  toProcess.sendtime_seconds = toCopy.sendtime_seconds;
  toProcess.sendtime_nanoseconds = toCopy.sendtime_nanoseconds;
  toProcess.fixed_payload_length = toCopy.fixed_payload_length;
  memcpy(&toProcess.fixed_payload, toCopy.fixed_payload, toCopy.
      fixed_payload_length);
  mSentQueue_2.pop();
[...]
```

Listing 4.6: networkHandler.cpp (Rev. 52560); Zeilen 3614-3625 und 3695-3704

4.1.3.1. Fehlerhafte Übergabe der Paketinhalte

Bei der Entwicklung eines eigenen Messverfahrens zur Datenratenmessung in Mobilfunknetzen wurden bei der Nutzung des dynamischen Payloads (siehe 2.2.2) zunächst nicht nachvollziehbare Werte durch das Framework im Senderlog eingetragen. Anstatt wie erwartet den dynamischen Payload zu erhalten, wurde in der Logdatei immer wieder die gleiche Zahlenkombination eingetragen. Eine Überprüfung der eigenen Messmethoden war hierbei nicht zielführend. Eine Analyse der beteiligten Funktionen der Netzwerkklasse lieferte den in Listing 4.6 gezeigte, fehlerhaften Quellcode.

In diesem Fall konnten direkt zwei Fehler eliminiert und der Quelltext beträchtlich gekürzt werden. In der oberen while-Schleife wird beim Kopieren der Daten der dynamische Payload durch den fixierten Payload ersetzt. Somit geht der Inhalt des dynamischen Payloads verloren. In der unteren Schleife wird der dynamische Payload komplett ignoriert. Die korrigierte Fassung zeigt das Listing 4.7.

```
[...]
  while( ! mSentQueue_1.empty() ) {
    toProcess = mSentQueue_1.front();
    mSentQueue_1.pop();
[...]
while( ! mSentQueue_2.empty() ) {
  toProcess = mSentQueue_2.front();
  mSentQueue_2.pop();
[...]
```

Listing 4.7: networkHandler.cpp (Rev. 54401); Zeilen 2079-2081 und 2142-2144

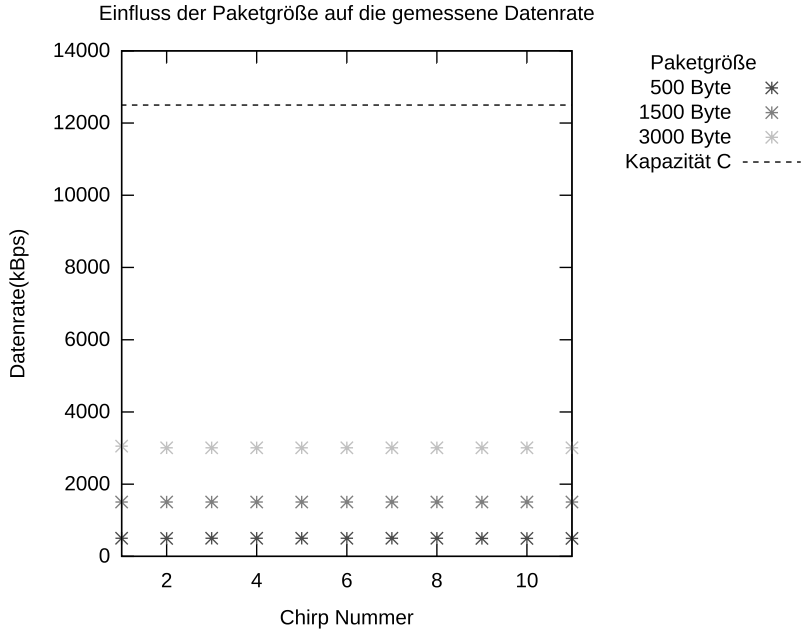

Abbildung 4.2.: Effekt einer Sendegeschwindigkeit unterhalb der Linespeed

4.1.3.2. Eingeschränkte Sendeleistung

Für Probemessungen wurden die Messgeräte an einem 10/100Mbps Switch angeschlossen. Danach wurde die Sende- und Empfangsrate an jedem Gerät auf 10Mbps reduziert und über die Kontrolleuchten am Switch verifiziert. Auch erste Messungen mit dem statischen Basic Verfahren und dem variablen Mobile Verfahren konnten die Rate von 10Mbps bestätigen. Die angezeigte Datenrate lag bei einer Paketgröße von 1500 Byte in der Nähe der erwarteten 1280 kBps.

Wurden jedoch Messungen mit einer Paketgröße kleiner als einem Tausendstel der Kapazität der Messleitung durchgeführt (<1280Byte), fiel die gemessene Datenrate immer ungefähr auf das Tausendfache der Paketgröße. Diese Werte konnten jedoch nicht stimmen, da zum Zeitpunkt der Messung kein Cross-Traffic am Switch anlag. Wiederholte Messungen lieferten das gleiche Ergebnis.

Eine Überprüfung der Logdateien auf Senderseite konnte bestätigen, dass die Zeit zwischen dem Senden einzelner Pakete eines Chirps jeweils ungefähr 1ms betrug. Mehrere Testläufe mit verschiedenen Paketgrößen zeigten, dass diese Zeitspanne unabhängig von der Paketgröße war. Somit wurden die Pakete nicht mit der maximalen Leitungsgeschwindigkeit gesendet, sondern jeweils mit einer Rate die dem Tausendfachen der Paketgröße entsprach. Die maximale Leitungsgeschwindigkeit wurde erst erreicht, wenn die Paketgröße höher als 1280Byte war.

Um die Sendeleistung zu erhöhen, wurden Änderungen am *Threaded Measurement Sender (TMSE)* vorgenommen. Während die ursprüngliche Implementierung vor jedem Senden eines Pakets die Methode sleepUntil() und damit über Umwege die Bibliotheksfunktion clock_nanosleep() aufgerufen hat, wird dies in der neuen Version nur noch gemacht, wenn das aktuell zu sendende Paket eine Sendezeit hat, die größer als die des zuletzt gesendeten Pakets ist. Des Weiteren wurde vor jedem Senden über die Funktion checkIfSocketIsReadyToWrite() die Methode select() aufgerufen, um zu prüfen ob es möglich ist zu Senden, ohne das der Aufruf blockiert. Da der TMSE jedoch als einzige Funktion den Measurement Socket zum Senden nutzt, ist diese Überprüfung unnötig.

Zusätzlich wurde die Übergabe von Daten an die Loggingqueues vereinfacht, um einer Beschränkung der Sendeleistung durch das Loggen der gesendeten Pakete zu entgehen.

Durch diese Maßnahmen und weitere Korrekturen im Quellcode des TMSE konnte das Zeitintervall zwischen dem Senden von zwei Paketen eines Chirps fast um den Faktor 10 gesenkt werden. Abbildung 5.1 zeigt, dass nun auch mit kleinen Paketgrößen gute Ergebnisse erziehlt werden können. Die Sendeleistung reicht jedoch immer noch nur aus, um einen 10Mbps Messkanal zu vermessen. Da das eingebaute Mobilfunkmodem maximal 5,8Mbps im Uplink unterstützt, sollte dies jedoch ausreichen.

4.1.3.3. Queue Größe beeinflusst Dynamik

Bei der Analyse der Implementierung des TMSE im Rahmen der Fehlersuche fiel das Design der Übergabe von Daten an die Loggingqueue auf. Dieses sah vor, dass durch den TMSE zunächst die Queue mindestens bis zur Hälfte gefüllt wird, bevor der *Threaded Sender Data Distributer (TSDD)* die Möglichkeit bekam, die Queue zu leeren. Ähnlich war die Situation beim *Threaded Measurement Receiver (TMRE)*. Um ein schnelles Verarbeiten der empfangenen Pakete durch den TMRE zu ermöglichen, wurden die Queues nur gewechselt, wenn die Queue Größe das vorgegebene Maximum überschritten hatte. Dies hat bei der Verwendung des Backchannels jedoch den Effekt, dass dessen Daten bei Verwendung einer großen Queue erst sehr spät durch den *Threaded Receiver Data Distributer (TRDD)* an die Messmethode weitergeleitet werden könnnen.

Um dennoch mit dynamischen Methoden arbeiten zu können, wurde die maximale Queue Größe an die Größe eines Chirps angepasst. So konnte zumindest nach jedem Chirp ein Backchannel Wert ausgelesen werden. Aus Zeitgründen wurde zunächst auf eine Anpassung der Queues verzichtet. Die in Abschnitt 4.1.3.4 beschriebene Neuimplementierung berücksichtigt jedoch das dynamik Problem.

4.1.3.4. Fehlerhafte Logging Queues

Wie in Abschnitt 4.1.3.3 beschrieben, wurden die folgenden Experimente mit einer kleinen Queue Größe durchgeführt. Dabei fiel auf, dass die gemessenen Datenraten stark schwankten. Eine Analyse der mSenderRaw Logdatei legte nahe, dass die Pakete auf Senderseite in der falschen Reihenfolge verschickt wurden.

Da der Fehler jedoch auch durch das Logging bedingt sein konnte, wurde die Ausgabe des Programms erweitert. Nach dem Senden eines Pakets wurden Chirp- und Paketnummer in das Status Logfile eingetragen und zusätzlich in die Standardausgabe eingefügt. Ein weiterer Testlauf zeigte schließlich, dass die Pakete in der korrekten Reihenfolge gesendet wurden. Die Reihenfolge der Pakete musste also beim Eintragen in das Logfile geändert werden. Der gleiche Effekt konnte auch auf Empfängerseite beobachtet werden.

Während ein Tausch der Pakete innerhalb eines Chirps nur die schnelle Datenratenmessung und die Latenzmessungen störte, sorgte das Vermengen von Chirps dafür, dass die gemessene Datenrate bei der Endauswertung massiv unterschätzt wurde. Das lag daran, dass sich die Sendezeitpunkte von zwei Chrips um die Interchirptime unterscheiden. Wird ein Paket des zweiten Chirps zwischen den Paketen des ersten Chirps gesendet, werden die folgenden Pakete des ersten Chirps mindestens um die Interchirptime verzögert. Dies erhöhte die bei der Gegenstelle gemessene Dauer des Chirps, woraufhin die Datenrate unterschätzt wurde.

Der Grund für das Verwürfeln der Pakete lag im Design der Kommunikation zwischen TMSE und TSDD bzw. TMRE und TRDD begründet. Bei beiden Implementierung wird ein exklusiver Zugriff auf die Queues durch die Verwendung von Mutexen gesichert. In Kombination mit Echtzeit-Scheduling und einer geringen Queue Größe, konnte der TRDD durch den TMSE verdrängt werden, was den folgenden Fehler auslöste.

Anhand von Abbildung 4.3 kann das Problem nachvollzogen werden. In Zustand a hat der TMSE seine Queue *Q2* gefüllt und versucht vor jedem Einfügen den Mutex von *Q2* zu belegen. Da der TRDD jedoch die Queue noch nicht geleert hat und somit den Mutex belegt hält, scheitert der Versuch des TMSE, welcher darauf hin das Paket in Q1 einfügt. Hat der TRDD die Queue geleert, wird der Mutex von *Q1* freigegeben und der TMSE kann beginnen die Queue zu füllen (Zustand b).

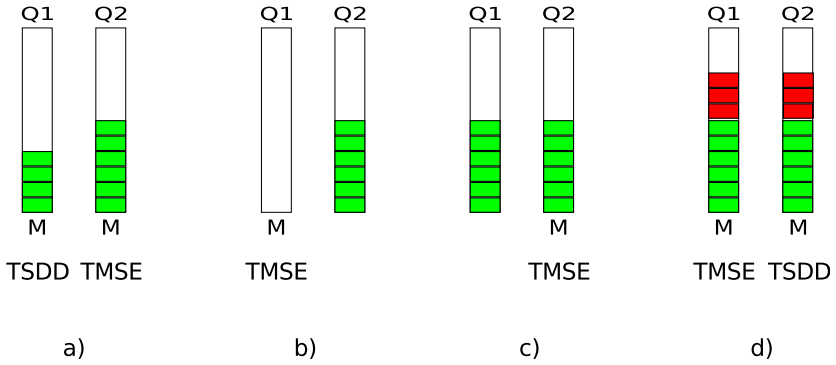

Abbildung 4.3.: Logging Queue Fehler

Wird der Echtzeit Scheduler verwendet und es sind durchgängig Pakete mit dem gleichen Sendezeitpunkt zu senden, wird der TMSE so lange die Queues füllen, bis ein Paket mit einem höheren Sendezeitpunkt vorliegt oder die Sendequeue leer ist. Das Verhindert, dass der TRDD den Mutex von *Q2* belegen kann. Daher wird der TMSE, sobald beide Queues mindestens zur Hälfte gefüllt sind (Zustand c), damit beginnen bei jedem Einfügen zwischen den Queues zu wechseln, so dass die Pakete gleichmäßig auf beide Queues verteilt werden (Zustand d).

Sobald der TSDD wieder Prozessorzeit zugeteilt bekommt, wird er die Queues nacheinander komplett leeren. Damit stimmt die Reihenfolge der Pakete im Sender- bzw. Empfängerlog nicht mit der eigentlichen Sende- bzw. Empfangsreihenfolge überein. Die Korrektur dieses Fehlers beschreibt Abschnitt 4.1.3.5.

4.1.3.5. Neues Design der Logging Queues

Um den in Abschnitt 4.1.3.4 erläuterten Effekt zu korrigieren, wurden die Logging Queues neu entwickelt. Dabei wurde die Kontrolle über die Queues komplett an den TRDD übergeben, was die Implementierung im TMSE einfach und kurz gestaltet. Der TMSE prüft nun eine Klassenvariable um festzustellen, in welche Queue geschrieben werden soll. Ist die entsprechende Queue vor dem Senden leer, wird ein Signal an den TRDD übermittelt, damit dieser die Möglichkeit hat aufzuwachen.

Der TRDD übernimmt wie erwähnt nun die Kontrolle über die Logging Queues. Anhand der Klassenvariable wird bestimmt, welche Queue gerade nicht durch den TMSE genutzt wird und geprüft, ob Daten vorhanden sind. Ist dies nicht der Fall, wird auch die zweite Queue geprüft. Wenn die Mutexe für beide Queues reserviert werden können und keine Daten in den Queues vorhanden sind, legt sich der TSDD schlafen, bis der TMSE ein Signal sendet. Sind in der zweiten Queue Daten vorhanden, so wird ein Wechsel der Queues initiiert. Enthält die durch die Klassenvariable angezeigt Queue bereits Daten, so werden diese direkt entnommen.

4.2. Variable Datenratenmessung

In [Wil12] wurden zwei Kommunikationskanäle eingerichtet, um dem Empfänger von Messpaketen eine Möglichkeit zu geben, dem Sender der Pakete Informationen über die aktuell gemessene Datenrate mitzuteilen. Damit ist es möglich, ein dynamisches Datenratenmessverfahren zu entwickeln, bei dem sich die Senderate der Empfangsrate der Gegenstelle anpasst. Dies ist insbesondere für Messungen im Mobilfunk interessant, da in diesem Fall die Kapazität des Messkanals durch Effekte wie Handover und Fallbacks reduziert bzw. gesteigert werden kann. Im Folgenden wird das für diese Arbeit implementierte Datenratenmessverfahren vorgestellt.

4.2.1. Design des Messverfahrens

Das dynamische Messverfahren basiert auf der Implementierung des Verfahrens Basic aus [Wil12]. Um dieses Verfahren dynamisch zu gestalten, wurde eine Klasse BackChannel implementiert, deren Aufgabe das Einfügen und Entnehmen von Backchannel Informationen aus dem dynamischen Payload der Messpakete ist. Die Klasse SafeguardChannel übernimmt die gleiche Aufgabe für ein- und ausgehende Pakete im Safeguard Channel.

Mit diesen Informationen berechnet die Methode CalculateChirp() die neue Chirpgröße s_{chirp} nach der Formel 4.1 aus der letzten über den Back- oder Safeguard Channel empfangenen Datenratenschätzung a_{fast}, sowie der beim Start der Messung vorgegebenen Paketgröße s_{packet} und der vorgegebenen Last $l_{channel}$. Anschließend werden die Pakete mit Hilfe der Methode EnqueueChirp() an den TMSE übergeben.

$$s_{chirp} = \frac{a_{fast}}{s_{packet}} \cdot \frac{l_{channel}}{100} \tag{4.1}$$

Neben der bereits genannten Parameter Last [%] (-mM_l) und Paketgröße [Byte] (-mM_p) können auch die vom Basic Verfahren bekannten Parameter Interchirptime [s] (-mM_it) und Startrate [kBps] (-mM_s) genutzt werden. Zusätzlich besteht die Möglichkeit über den Parameter (-mM_pt) [s] eine Processtime zu übergeben. Diese definiert, wie viele Sekunden vor dem Sendezeitpunkt eines Chirps mit dem Füllen der Sendequeue begonnen werden soll. Da beim dynamischen Verfahren immer die aktuellsten Informationen genutzt werden sollten, wird empfohlen diesen Wert möglichst klein zu halten.

Die Parameter TimeToBuffer (-mM_t) [s] und PauseOnCongestion [0|1] (-mM_c) steuern den Abbau einer Überlast im Netzwerk. Meldet die Gegenstelle über den Safeguard Channel eine sehr niedrige

Empfangsrate, soll der Sender für eine gewisse Zeitspanne stoppen, falls PauseOnCongestion aktiviert ist, um ein Abbauen der Überlast zu erlauben. Der Parameter TimeToBuffer bestimmt dabei zusammen mit der Interchirptime, die größe eines Puffers, der die Chirpgröße der letzten gesendeten Chirps sichert. Mit Hilfe dieser Daten kann die Wartezeit genau bestimmt werden, so lange der Puffer nicht zu klein dimensioniert ist.

Der Parameter PacketBufferSize (-mM_pb) [#] bestimmt die Größe des Paketpuffers im FastDataCalculator. Wird ein neues Paket empfangen, wird es zunächst in diesem Puffer eingetragen. Die aktuelle Datenrate wird durch den Mittelwert der Empfangsraten aller Pakete eines Chirps im Puffer ermittelt, um den Einfluss von Messausreissern zu verringern.

4.2.2. Gesendete Daten

Das *Mobile* Verfahren speichert die Werte zur Chirp- und Paketgröße im statischen Payload der Messpakete. Dies hat den Hintergrund, dass sich die Chirpgröße ändern kann und ohne die Informationen zur Größe im statischen Payload der Verlust von Paketen nicht erkannt werden könnte, sofern dieser erst am Ende des Chirps auftritt. Die Daten müssen in den statischen Payload geschrieben werden, da sich die Daten des dynamischen Payloads auch während der Sendezeit eines Chirps ändern könnten.

Des Weiteren wird im dynamischen Payload festgehalten, welches Paket als letztes Empfangen wurde und was das Ergebniss der schnellen Datenratenmessung durch dieses Paket ist. Ohne diese Daten könnte ein dynamisches Verfahren nicht realisiert werden. Wird nur eine Senderichtung ausgenmessen, so findet momentan keine dynamische Anpassung statt, da keine Messpakete in Gegenrichtung gesendet werden, welche im Backchannell die benötigten Informationen tragen.

4.3. Stateful Experiment Control

Bei der Verwendung des SEC (siehe Abschnitt 2.3) zur Steuerung von RMF Experimenten traten Fehler auf, welche die Ausführung der Messexperimente zunächst verhinderten. Die folgenden Abschnitte beschreiben die kritischen Fehler und ihre Lösung.

4.3.1. Fehler beim Bestimmen der Anzahl verfügbarer Clients

Bei Tests des SEC mit eigenen Konfigurationsdateien, welche neben der Serverinstanz nur einen Client benötigten, meldete der Server das zu wenig Clients verfügbar wären. Somit wurde das Experi-

ment nicht durchgeführt. Bei der Suche nach dem Fehler wurde in der Datei taskDivider.cpp folgende Quellcodeausschnitt gefunden.

```
m_isCountingClients= true;
// das erste Mal parsen, um Anzahl der Clients zu zaehlen
if (xmlread::parse(path) < 0)
  return -1;
m_nrOfClients+=2;
m_isCountingClients=false;
```

Das setzten der Variable m_isCountingClients und der anschließende Aufruf von xmlread::parse(path) sorgte dafür, dass die Datei einmal eingelesen und zum bestimmen der Clientanzahl die Funktion taskDivider::countClients() genutzt wurde. Die Funktion lieferte jedoch unabhängig von der Anzahl der benötigten Clients immer den Wert Null zurück, da die Implementierung fehlerhaft war.

Die Anweisung *m_nrOfClients+=2* wurde offenbar eingefügt, um andere Funktionen mit Hilfe des Demoprogramms zu testen. Anschließend wurde jedoch vergessen, den eigentlichen Fehler zu korrigieren.

Um nicht zu viel Zeit für die Implementierung einer korrekten Methode zu verwenden, wurde eine Lösung entwickelt, welche unter bestimmten Voraussetzungen die Anzahl der benötigten Clients überschätzen konnte. Da diese Anpassung ausreichte um die für die eigenen Messreihen benötigte Funktionalität herzustellen, wurde dies akzeptiert und der Fehler in einem Bugfile eingetragen.

4.3.2. Unvollständige Übertragung

Bei weiteren Tests mit eigenen Experimentdateien wurde festgestellt, dass die Übertragung der Experimentdateien vom Server zum Client nicht richtig funktionierte. Statt der kompletten Datei wurde immer nur der Anfang übertragen. Durch entsprechende Anpassungen an der Datei NetHost.cpp konnte die vollständige Übertragung eingerichtet werden.

4.3.3. Hohe CPU Last

In der ursprünglichen Implementierung weist das SEC eine sehr hohe CPU Last auf. Diese hohe CPU last wurde durch diverse Busy-Wait Schleifen in der Implementierung der Client- und Serverinstanzen verursacht. Da eine hohe CPU Last die Messungen potentiell beeinflussen kann, wurden diese durch entsprechende sleep() Aufrufe ersetzt.

Kapitel 5.

Experimente

Das Ziel dieser Arbeit war die Untersuchung der Auswirkungen paralleler Datenratenmessungen in einer Mobilfunkzelle. Während der Entwicklung beschränkten sich die Versuche darauf, zu prüfen ob die Version des *Rate Measurement Framework (RMF)* (Abschnitt 2.2) lauffähig und die gelieferten Werte plausibel waren. Die Vielzahl und Komplexität der vorhandenen Programmfehler sorgte dafür, dass eine umfangreiche Untersuchung der Auswirkung paralleler Messungen im Mobilfunknetz aus Zeitgründen nicht mehr möglich war. Die im Folgenden dargestellten Versuchsergebnisse können daher auf Grund mangelnder Quantität nur Hinweise bezüglich eventueller Effekte geben.

Alle folgenden Experimente wurden mit aktiviertem FIFO Echtzeit Scheduler ausgeführt (RMF Option -P 2). Sofern möglich, wurden das Swapping (RMF Option -L 1), sowie die dynamische Anpassung der CPU Frequenz (RMF Option -G 1) deaktiviert.

5.1. Genauigkeit der Messwerte

Um festzustellen, ob die durch das RMF gelieferten Messwerte den Erwartungen entsprechen, wurden Messungen in einer kontrollierten Umgebung mit einem Ethernet Netzwerk durchgeführt. Die folgenden Abschnitte beschreiben den Aufbau, die Durchführung und Ergebnisse der Messungen.

5.1.1. Versuchsaufbau

Für die Messungen standen vier Messgeräte zur Verfügung (Alix1-4), welche über einem 10/100 Mbps Ethernet Switch verbunden waren. Zusätzlich war ein PC (worf) angeschlossen, welcher für die Steuerung der Experimente mit Hilfe des Programms *Stateful Experiment Control (SEC)* (Ab-

Prozessor	Intel Core2Quad Q9400
Arbeitsspeicher	4 GB
Netzwerkinterface	10/100/1000 Mbps
Betriebssystem	Debian 6.0
SEC Version	Revision 54434
RMF Version	Revision 54638

Tabelle 5.1.: Technische Daten von worf

schnitt 2.3), sowie die Generierung von Cross Traffic mit Hilfe des Programms *brute* [NSGR05] genutzt wurde. Eine Auflistung der Eigenschaften des PCs liefert Tabelle 5.1.

5.1.2. Messungen auf unbelasteten Link

Um zu betrachten, wie genau eine unbelastete Verbindung durch das RMF ausgemessen werden kann, wurden mehrere Messungen mit den Messverfahren *Basic* und *Mobile* ausgeführt. Dabei wurden verschiedene Kombinationen aus Paket- und Traingröße genutzt. Die Traingröße bezeichnet hier die Anzahl der Pakete eines Packettrains.

Da das im Messgerät verwendete Mobilfunk Modem maximal Daten mit 7,2 Mbps empfangen kann (Abschnitt 2.4.2.2), wurde die Empfangs- und Sendeleistung der Ethernet Schnittstellen von Alix1-4 mit Hilfe des Programms *mii-tool* [man04] auf 10 Mbps (Duplex) beschränkt. Als Paketgrößen wurden 100-1500 Byte in 100 Byte Abständen gewählt.

Die Größe des Packettrains wurde beim *Mobile* Verfahren mit Hilfe des Last Parameters (-mM_l) beschränkt. Die Last beschreibt, wie hoch der prozentuale Anteil der Messdaten an der zur Verfügung stehenden Datenrate sein soll. Um die Methoden vergleichen zu können, wurde die Packettrain Größe für das Basic Verfahren mit dem entsprechenden Parameter (-mB_c) an den gewünschten Lastwert angepasst.

Für die Steuerung des Experiments wurde das SEC verwendet. Im Anhang ist jeweils ein Ausschnitt der Konfigurationsdatei für eine Messung mit 10% Last abgebildet (Abbildungen B.1 und B.2). Die exemplarische dargestellten Konfigurationsdateien nutzen Alix1 als Server und Alix2 als Client. Für die Testreihe wurden jedoch sämtliche Kombinationen der Messgeräte als Client und Server genutzt. Neben einer Last von 10% wurde auch mit einer Last von 30% gemessen.

Die Last wurde für sehr kleine Paketgrößen um den Faktor 10 verringert, da die Auslastung einer 10 Mbps Verbindung mit sehr kleinen Paketgrößen zu sehr großen Packettrains führt, welche nicht schnell genug von den Messgeräten verarbeitet werden können. Da die Pakete jedoch weiterhin für

eine spätere Verarbeitung im Arbeitsspeicher gesichert werden, läuft der Arbeitsspeicher voll und die Messung wird durch den Kernel beendet.

Während der Testmessungen zeigt Alix4 ein Verhalten, dass auf einen Hardware Defekt schließen ließ. Aus Zeitgründen wurde darauf verzichtet die Ursache zu prüfen und statt dessen Alix1-3 verwendet.

5.1.2.1. Vergleich der Messungen der einzelnen Messgeräte

Für den weiteren Verlauf der Messungen war es interessant, zu wissen ob sich alle Messgeräte gleich verhalten. Daher wurde nach dem Ablauf der oben genannten Messungen jeweils ein CDF für jede Paketgröße, über alle durch ein Messgerät als Client und Server gelieferten Datenratenmesswerte berechnet. Anschließend wurden die CDFs für die einzelnen Paketgrößen bei gleicher Auslastung für Alix1-3 in einem Plot aufgetragen und verglichen.

Wie zu erwarten, war das Verhalten über alle Paketgrößen für die einzelnen Messverfahren gleich. Bei Paketgrößen unter 400 Byte streuten die Ergebnisse. Die Messwerte liegen bei diesen Paketgrößren meist unter 1100 kBps und sind damit relativ weit von den zu erwartenden 1280 kBps entfernt. Für reale Messungen empfiehlt es sich also, Paketgrößen kleiner als 400 Byte für Datenraten von 10 MBps zu vermeiden.

Da sich die einzelnen Messgeräte gleich verhalten, wurden im Folgenden nur die Messergebnisse von Alix3 betrachtet.

Beim Vergleich der CDFs für Auslastungen von 10% und 30% fiel unabhängig vom Verfahren auf, dass die gemessene Datenrate für eine Auslastung von 30% niedriger war, als bei einer Auslastung von 10%. Dies kann damit erklärt werden, dass bei einer höheren Auslastung der Sender und der Empfänger Thread über einen längeren Zeitraum um die CPU konkurieren. Der Empfänger Thread verdrängt dabei immer wieder den Sender, was jedes mal einen Overhead erzeugt. Das sorgt zum einen dafür, dass die Zeitstempel am Empfänger ungenauer werden und zum anderen kann es passieren, dass der Sender die Daten nicht mehr mit Linespeed verschicken kann. Bei welcher Auslastung diese Effekte die Messergebnisse zu stark beeinflussen ist abhängig von der Kapazität des Sendemediums sowie der verwendeten Paketgröße.

Interessant ist, dass das Mobile Verfahren durchgängig Messwerte liefert, welche näher am theoretischen Maximum liegen, als das Basic Verfahren bei gleicher Auslastung. Da beide Verfahren die gleiche Codebasis nutzen und das dynamische Verfahren zusätzlich Zeit zum Auswerten des dynamischen Payloads benötigt, waren schlechtere Messwerte erwartet worden.

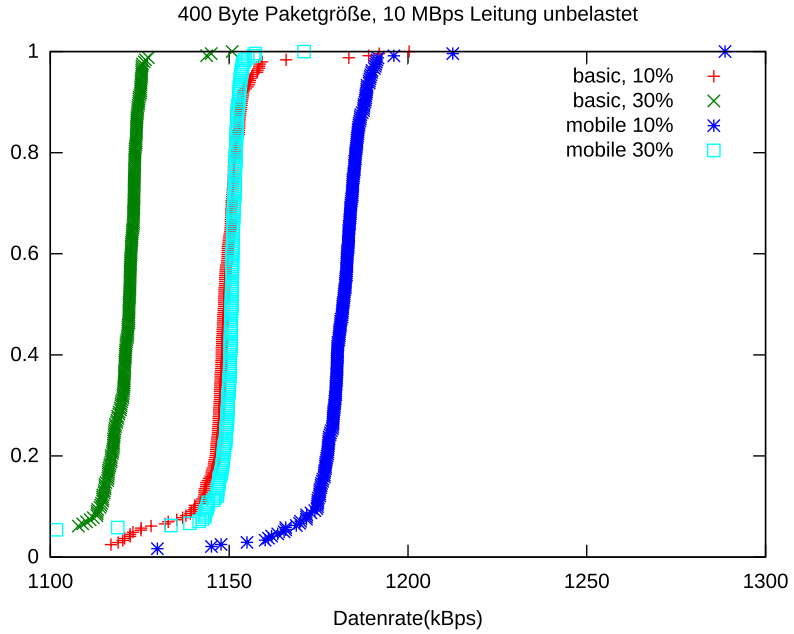

Abbildung 5.1.: Vergleich der Messverfahren mit 400 Byte Paketen (CDF)

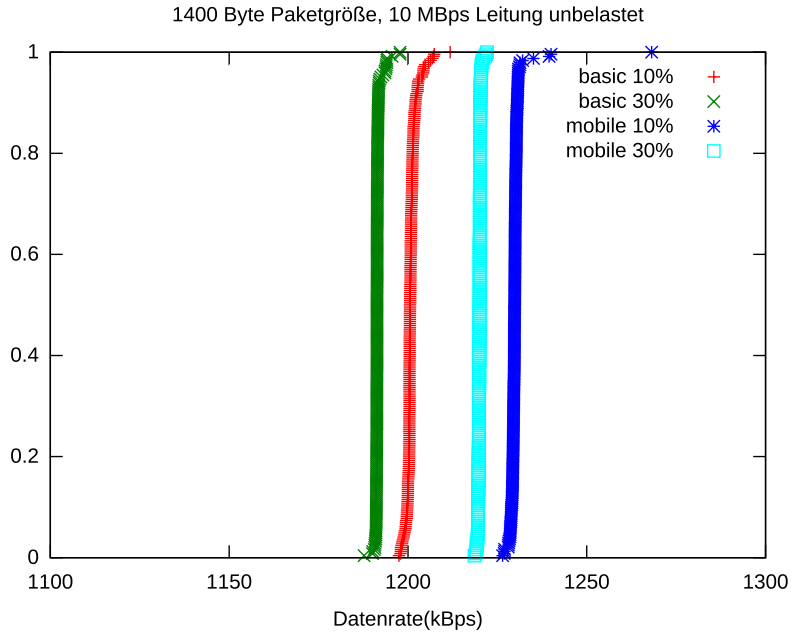

Abbildung 5.2.: Vergleich der Messverfahren mit 1400 Byte Paketen (CDF)

Das die Messwerte für 400 Byte Pakete (Abbildung 5.1) mit gleicher Auslastung schlechter sind als bei 1400 Byte Paketen (Abbildung 5.2) ist logisch. Der Aufwand zum Senden des Packettrains mit 400 Byte Paketen ist deutlich größer, da der Train bei gleicher Auslastung mehr als dreimal so groß ist als bei 1400 Byte Paketen. Damit steigt auch das Risiko, dass die Werte durch den ständigen Wechsel von Sender und Empfänger Thread beeinflusst werden.

5.1.3. Messungen mit einem belasteten Link

Im Gegensatz zum Basic Verfahren, kann das Mobile Verfahren die Größe der Messdaten an die Empfangsrate der Gegenstelle anpassen, sofern über den Backchannel entsprechende Informationen übermittelt werden.

Um das Verhalten des Verfahrens bei Cross Traffic in einer kontrollierten Umgebung beobachten zu können, wurde die verfügbare Datenrate während der Messungen mit Hilfe des Programms *brute* limitiert. brute sendet durch ein Konfigurationsskript gesteuert mit frei wählbaren Datenraten UDP Pakete. Dieser Cross Traffic senkt die verfügbare Datenrate entsprechend.

Die Konfigurationen für das SEC und brute sind im Anhang abgebildet (Listings B.3 und B.4). Aus Zeitgründen wurde darauf verzichtet, mit mehreren Messgeräten zu testen. Es wurde nur mit Alix1 und Alix2 als RMF Server bzw. Client gearbeitet. Der Cross Traffic wurde durch worf erzeugt, welcher auch als SEC Server Instanz diente.

5.1.3.1. Anpassung der Datenrate

Wie in Abschnitt 4.2, kann sich die Senderate beim Mobile Verfahren an die Empfangsrate der Gegenstelle anpassen, sofern Pakete mit Backchannel Informationen eintreffen. Die Anpassung der Senderate von Alix1 an die gemeldete Empfangsdatenrate von Alix2 ist in Abbildung 5.3 deutlich zu erkennen. Neben der Datenrate ist dort auch die Chirpgröße als Produkt von Packettrain Größe und Paketgröße dargestellt. Vergleicht man die die Cross Traffic Werte aus Tabelle 5.2 mit den gemessenen Datenraten, so fällt auf, dass die Messungen sehr genau sind.

Des Weiteren fällt auf, dass die Senderaten nach dem Start zunächst eine Zeit lang zwischen zwei Werten schwanken, obwohl die verfügbare Datenrate durch den Cross Traffic nur beschränkt wird. An dieser Stelle muss auf jeden Fall überprüft werden, ob ein Fehler auf Seiten der Implementierung des *FastDataCalculator* für diesen Effekt verantwortlich sein könnte. Im weiteren Verlauf verhält sich das Verfahren jedoch wie gewünscht. Auch die Messwerte entsprechen im Rahmen der zu erwartenden Fehler bei einer relativ geringen Paketgröße von 700 Byte den Erwartungen.

Zeit [s]	Cross Traffic [kBps]
30	0
30	126
30	250
30	375
30	501
30	0
30	501
30	1000
30	501
30	0
30	1251

Tabelle 5.2.: Cross Traffic Daten

Es ist wichtig zu erwähnen, dass die dargestelle Messreihe unter den durchgeführten eine Ausnahme darstellt. Die meisten Messungen mussten auf Grund von äußeren Einflüssen abgebrochen werden oder waren auf Grund einer fehlerhaften Verwendung von brute nicht vergleichbar.

Bei den ersten Messungen wurde die Möglichkeit der CPU, die Frequenz der Lastsituation anzupassen nicht deaktiviert. Da brute das TSC Register für Zeitmessungen nutzt und dieses abhängig von der Frequenz erhöht wird, wurde der Cross Traffic schneller bzw. langsamer gesendet als in der Konfigurationsdatei angegeben. Für einen korrekten Ablauf von brute auf einem Mehrkernsystem muss das Programm einem festen CPU Kern zugeordnet werden (z.B. mit taskset) und für diesen Kern die dynamische Frequenzanpassung deaktiviert werden (z.B. *echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor*).

5.2. Messungen im Mobilfunk

Nachdem durch die Messungen in einer kontrollierten Umgebung bekannt war, dass das Mobile Verfahren funktioniert, wurden Messungen im Mobilfunk durchgeführt. Die Messungen haben nachts stattgefunden, um eine Beeinflussung der Messergebnisse durch andere Teilnehmer zu minimieren.

Bei den Messungen waren alle vier beteiligten Alix Systeme in einem Büro untergebracht. Die Antennen standen jeweils in einem Abstand von ungefähr 10cm am Fenster. Für die Messungen wurde das Netz des Betreibers *fonic* genutzt.

Als Endpunkt für die Messungen diente ein extra eingerichteter Messerver, welcher über eine öffentliche IP Adresse aus dem Internet erreichbar war. Um mehrere RMF Instanzen nutzen zu können,

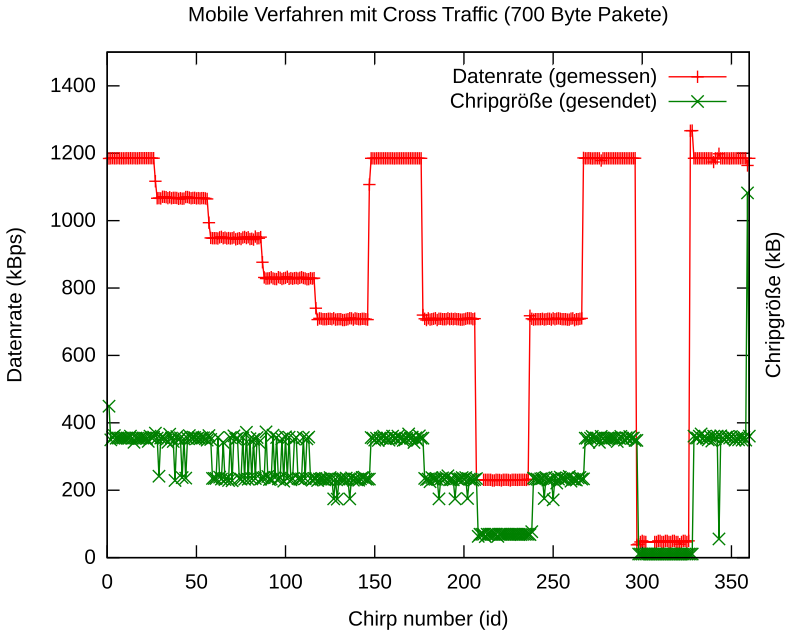

Abbildung 5.3.: Messung mit Mobile Verfahren und Cross Traffic

Prozessor	Intel Pentium D 2,8GHz (Dualcore)
Arbeitsspeicher	3 GB
Netzwerkinterface	10/100/1000 Mbps
Betriebssystem	Debian 6.0
RMF Version	54638

Tabelle 5.3.: Technische Daten von brute

wurden UDP und TCP Portnummern im Bereich 53221-53224 freigeschaltet. Auf diese Weise konnten bis zu vier RMF Instanzen parallel betrieben werden. Während der Messungen wurden die RMF Serverinstanzen bei Bedarf manuell gestartet. Die Ergebnisse der Messungen wurden für jede Instanz in einem separaten Ordner gesichert.

Eine kurze Auflistung der Eigenschaften des Messsystems liefert Tabelle 5.3. Die RMF Instanzen liefen auf diesen Systemen in der Server Rolle und wurden, sofern sie nicht durch Fehler auf Seiten der Clients unterbrochen wurden, durchgängig betrieben.

5.2.1. Einzelmessung gegen den Server

Zunächst wurden für Alix1-3 seperat drei Messungen durchgeführt, um Referenzwerte für einen späteren Vergleich bei parallelen Messungen zu erhalten. Die Messungen konnten jedoch nicht immer

beendet werden. Teilweise waren die Messgeräte nach einem Neustart über einen Zeitraum von zwei Stunden nicht mehr nutzbar, da das Modem keine Einwahl zugelassen hat und allgemein sehr träge reagierte. Eine Ursache ist momentan nicht bekannt. Die Auswertung erfolgt in Abschnitt 5.2.4.

5.2.2. Parallel Messungen gegen den Server

Nachdem die Einzelmessungen abgeschlossen waren, wurden parallele Messungen gegen den Server ausgeführt. Dazu wurden auf dem Server manuell drei RMF Instanzen gestartet, welche jeweils eine eigene Kombination von Portnummer für den Control und Measurement Channel nutzten. Die Client Instanzen wurden durch ein SEC Skript gestartet (Listing B.5). Die Auswertung erfolgt in Abschnitt 5.2.4.

5.2.3. Parallele Messungen mit zeitlich versetzten Packettrains

Neben der rein parallelen Messung im Mobilfunk, wurden auch Messungen durchgeführt, bei denen jede Kombintion aus RMF Server und Client ihre Packettrains mit einem festen Versatz gegenüber den anderen Instanzen gesendet hat. Um dies zu realisieren, wurde auf die Startzeitpunkte in der Funktion *FillSendingQueue* der Klasse *measureMobile* ein Offset von 0, 0,3 oder 0,6 Sekunden auf den Startzeitpunkt addiert und anschließend das RMF jeweils neu kompiliert. Dies bewirkt bei einer Interchirptime von 1s, das die Packettrains versetzt gesendet werden. Die geänderten Softwareversionen wurden anschließend auf die Alix Boards und den Messserver verteilt.

Die Serverinstanzen wurden widerrum manuell gestartet, wobei auf die Portnummern geachtet wurden, um später den Client Instanzen ihre entsprechenden Pendants zuweisen zu können. Die Client Instanzen wurden durch das gleiche Skript wie in Abschnitt 5.2.2 gestartet. Die Auswertung erfolgt in Abschnitt 5.2.4.

5.2.4. Auswertung der Messungen im Mobilfunk

Abbildung 5.4 stellt die CDF über die verschiedenen Testläufe mit Alix1 als Client im Mobilfunknetz dar. Es fällt auf, dass die Messwerte geringer ausfallen, wenn die Startzeitpunkte der Packettrains leicht versetzt sind. Das gleiche Ergebnis konnte auf Serverseite beobachtet werden. Auch Alix2 zeigt das gleiche Verhalten.

Die Datenbasis ist jedoch gering und die Messzeitpunkte zwischen den rein parallelen und leicht versetzten Messungen liegen 3 Stunden auseinander. Um diesen Effekt zu belegen, werden jedoch

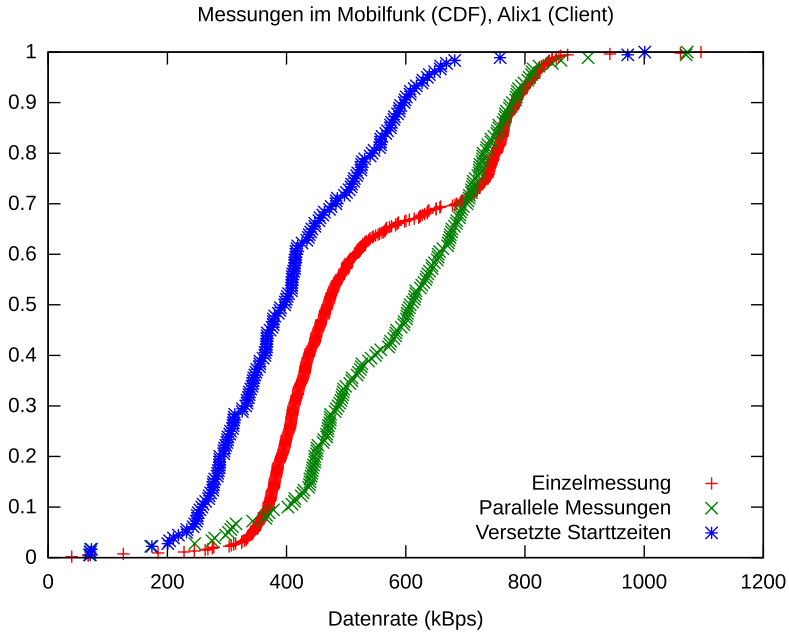

Abbildung 5.4.: Parallele Messungen (CFD)

weitere Messungen benötigt, welche im zeitlichen Rahmen dieser Arbeit nicht mehr möglich waren.

Kapitel 6.

Zusammenfassung und Ausblick

6.1. Zusammenfassung

Im Rahmen dieser Masterarbeit wurde die aus der Projektarbeit bekannte Hardwarebasis mit dem Rate Measurement Framework verbunden, um Datenratenmessungen im Mobilfunk zu ermöglichen. Dabei wurde eine Vielzahl von Optimierungen und Fehlerkorrekturen vorgenommen.

Es wurde ein gegenüber Stromausfällen tolerantes und für Flash Speicher geeignetes Dateisystem gewählt, sowie die Zeitsynchronisation gegen das PPS Signal eines GPS Empfängers verbessert. Die Modifikationen am Standard Image von Voyage Linux können nun anhand von Skripten nachvollzogen werden, so dass eigene Modifikationen erleichtert werden.

Im Rate Measurement Framework wurde mehrfach kritische Fehler korrigiert, welche ordentliche Messungen verhindert haben. Es wurde eine funktionierende Implementierung zum Überwinden eines Network Address Translation Systems entwickelt, ohne die eine Messung im Mobilfunknetz nicht möglich wäre. Die Geschwindigkeit, mit der Pakete gesendet werden können wurde erhöht, damit eine Senden mit Linespeed auf einem 10 Mbps Kanal möglich wurde. Des Weiteren wurde das Design der Loggingqueues korrigiert, welche sonst die ankommenden Pakete verwürfelt und damit die Messungen gestört haben.

Zusätzlich wurden einige Fehler in der Experimentensteuerung Stateful Experiment Control korrigiert, welche einen Betrieb im Netzwerk und mit einer beliebigen Anzahl an Clients ermöglichen.

Um die Möglichkeit der Rückmeldung über den Backchannel des RMF zu testen, wurde das dynamische Datenratenmessverfahren *Mobile* entwickelt. Dieses Verfahren reagiet auf die durch den Backchannel empfangenen Informationen zur gemessenen Datenrate der Gegenstelle und passt die eigene Senderate entsprechend an.

Abschließend wurden Vergleichstest mit dem Basic Verfahren durchgeführt, wobei das Mobiel Verfahren die genaueren Messergebnisse erziehlen konnte. Danach wurden parallele Messungen im Mobilfunk ausgeführt, deren Ergebnisse jedoch weitere Überprüfungen benötigen da die Testabdeckung aus Zeitgründen zu gering ausgefallen ist.

6.2. Ausblick

Auch wenn im Rahmen dieser Arbeit das Verhalten von parallelen Datenratenmessungen im Mobilfunk nicht umfänglich untersucht werden konnte, so liefert diese Arbeit dennoch eine Grundlage für weitergehende Fragestellungen und Entwicklungen.

Konkret konnte dies bereits bei der Bachelorarbeit von Malte Olfen gezeigt werden. Seine Arbeit beschäftigte sich mit der Entwicklung eines zum RMF kompatiblen Frameworks für die Andoid Plattform. Am Ende konnte das RMF auf die Android Plattform portiert werden. Somit kann diese von zukünftigen Entwicklungen des RMF profitieren.

Da im Laufe der Arbeit nur wenig Zeit für die Entwicklung eines an die Eigenschaften des Mobilfunks angepassten Datenratenmessverfahrens verwendet werden konnte, kann an dieser Stelle weiter angesetzt werden. So könnten zum Beispiel die Berechnungen zur schnellen Datenratenschätzung überarbeitet und durch einen Filter ergänzt werden, welcher Abhängig von der momentan genutzten Übertragungstechnologie arbeitet. Da die Ergebnisse der schnellen Datenratenschätzung für die Anpassung der Senderate bei dynamischen Verfahren genutzt werden, könnten damit starke Schwankungen der Senderate durch Messfehler vermieden werden. Des Weiteren wurden während der Arbeit noch nicht die Auswirkung der Safeguard Channel Implementierung betrachtet.

Auch die Implementierung des Frameworks bietet noch viele Möglichkeiten zur Optimierung. So müssen neue Methoden derzeit sehr umständlich an mehreren Stellen im Quellcode integriert werden. Die Tests neuer Methoden und insbesondere der Vergleich von Methoden könnte durch eine Replay Möglichkeit erleichtert werden. Gerade das Verhalten bei Paketverlusten lässt sich ohne eine solche Funktion nur schwer untersuchen.

Zusätzlich deuten die durchgeführten Experimente auf eine Beeinflussung der Messungen durch das parallele, zeitversetzte Senden von Packet Trains hin. An dieser Stelle sollten weitere Messungen durchgeführt werden um dieses Verhalten zu prüfen.

Anhang A.

Alix Konfiguration

Während der vorangegangenen Projektarbeit wurden mehrere Konfigurationsdateien des auf den Messgeräten installierten Voyage Linux Systems geändert. Im Folgenden werden wichtige Änderungen vorgestellt und begründet.

A.1. NTP Konfiguration

Über das DHCP Protokoll können den anfragenden Clients neben der zugewiesenen Adresse auch andere Informationen übermittelt werden. Eine dieser Informationen ist die Konfiguration für einen lokalen NTP Server. Akzeptiert der DHCP Client diese, so wird die eigene Konfiguration verworfen und statt dessen die des DHCP Servers genutzt.

Um dieses Verhalten zu unterbinden, wird die Datei */etc/dhcp/dhclient.conf* angepasst. In der Zeile *request* muss die Option *time-server* entfernt werden. Der modifizierte Teil ist in Listing A.1 dargestellt.

```
request subnet-mask, broadcast-address, time-offset, routers,
        domain-name, domain-name-servers, domain-search, host-name,
        netbios-name-servers, netbios-scope, interface-mtu,
        rfc3442-classless-static-routes;
```

Listing A.1: /etc/dhcp/dhclient.conf, Zeile 20-24

A.1.1. GPSD Client Konfiguration

Um das Alix System mit dem GPS Empfänger zu synchronisieren, müssen mehrere Vorraussetzungen erfüllt sein. Zunächst muss die Konfiguration für die serielle Schnittstelle wie in Abschnitt A.3

angepasst werden. Des Weiteren müssen Konfiguration und Init-Skript von gpsd nach den Angaben aus Abschnitt A.2 modifiziert werden.

Sind diese Vorraussetzungen erfüllt, kann die Konfiguration von NTP modifiziert werden, um die Informationen von gpsd verarbeiten zu können. Dazu müssen die *server* Einträge auf die Werte *127.127.28.0* (GPS) und *127.127.28.1* (PPS) gesetzt werden. Für das PPS Signal werden die Optionen *true*, sowie *minpoll* und *maxpoll* gesetzt.

Die Option true verhindert, dass das PPS Signal durch den Clock Select Algorithmus von NTP (vgl. Abschnitt 3.2.1) verworfen werden kann. Minpoll und Maxpoll sorgen dafür, dass alle 16 Sekunden ein neuer Wert abgefragt wird.

Listing A.2 zeigt die Konfiguration, welche für Alix3 während der Arbeit genutzt wurde.

```
driftfile /var/lib/ntp/ntp.drift

# Enable this if you want statistics to be logged.
statsdir /var/log/ntpstats/
#statsdir /root/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

server 127.127.28.1 true minpoll 4 maxpoll 4
fudge 127.127.28.1 refid PPS
server 127.127.28.0
fudge 127.127.28.0 time1 0.600 refid GPS

# Access control configuration; see /usr/share/doc/ntp-doc/html/accopt.html for
# details.  The web page <http://support.ntp.org/bin/view/Support/
    AccessRestrictions>
# might also be helpful.
#
# Note that "restrict" applies to both servers and clients, so a configuration
# that might be intended to block requests from certain clients could also end
# up blocking replies from your own upstream servers.

# By default, exchange time with everybody, but don't allow configuration.
restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery

# Allows the users of the subnet 192.168.2.0 to sync to our server
restrict 192.168.2.0
```

```
# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1
restrict ::1
```

Listing A.2: /etc/ntp.conf (gpsd Version)

A.1.2. Network Client Konfiguration

Während der Masterarbeit wurde von den Alix Systemen nur Alix3 mit Hilfe des GPS Empfängers synchronisiert. Die anderen Messgeräte waren über einen 10/100 Mbps Switch mit Alix3 verbunden und nutzten diese als einzigen NTP Server. Die Konfiguration für die per Netzwerk angebunden Messgeräte zeigt Listing A.3.

```
# /etc/ntp.conf, configuration for ntpd; see ntp.conf(5) for help
driftfile /var/lib/ntp/ntp.drift

# Enable this if you want statistics to be logged.
statsdir /var/log/ntpstats/
#statsdir /root/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

# Access control configuration; see /usr/share/doc/ntp-doc/html/accopt.html for
# details.  The web page <http://support.ntp.org/bin/view/Support/
    AccessRestrictions>
# might also be helpful.
#
# Note that "restrict" applies to both servers and clients, so a configuration
# that might be intended to block requests from certain clients could also end
# up blocking replies from your own upstream servers.

# By default, exchange time with everybody, but don't allow configuration.
restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery

# Allows the users of the subnet 192.168.2.0 to sync to our server
restrict 192.168.2.0

# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1
restrict ::1
```

```
server 192.168.2.182 minpoll 6 maxpoll 6 iburst
```

Listing A.3: /etc/ntp.conf

A.2. gpsd Konfiguration

Damit gpsd die Möglichkeit hat, die Signale des GPS Empfängers verarbeiten zu können, muss zunächst die Konfiguration für den seriellen Port angepasst werden (Abschnitt A.3). Danach sollte mit Hilfe eines Terminal Programms (z.B. minicom) geprüft werden, welcher serielle Anschluß genutzt wird. Im Folgenden wird davon ausgegangen, das der GPSD Empfänger über den Port */dev/ttyS4* angebunden ist.

Normalerweise wartet gpsd auf Anfragen an TCP/IP Port 2947 bevor ein GPS Empfänger angesprochen wird. Die Verbindung mit NTP erfolgt jedoch über ein geteiltes Speichersegment. Um GPSD mitzuteilen, dass nicht auf Anfragen gewartet werden muss, wird die Option *-n* benötigt.

Die Option *-G* sorgt dafür, dass gpsd Anfragen von allen Adressen annimmt. Dies ist sehr nützlich, um Hilfe von Programmen wie *xgps* den Empfang zu prüfen. Die Konfiguration ist in Listing A.4 abgebildet.

```
START_DAEMON="true"
GPSD_OPTIONS="-n -G"
DEVICES="/dev/ttyS4"
USBAUTO="false"
GPSD_SOCKET="/var/run/gpsd.sock"
```

Listing A.4: /etc/default/gpsd

Für die vorliegende Arbeit wurde die gpsd Version 2.95 genutzt. Der Autokonfigurationsalgorithmus dieser Version kann eine Baudrate von 19200 Bd der serielle Verbindung nicht erkennen. Diese wird aber während dieser Arbeit beim GPS Empfänger genutzt. Um dieses Problem zu umgehen, kann kurz vor dem Starten des gpsd der Befehl *stty speed 19200 </dev/ttyS4* genutzt werden. Damit dies nicht vor jedem Start manuell geschehen muss, wurde das Init-Skript von gpsd entsprechend angepasst. Listing A.5 zeigt die Änderung an der Funktion *do_start*

```
do_start()
{
        stty speed 19200 </dev/ttyS4
        sleep 1
        # Return
        #   0 if daemon has been started
```

```
        #   1 if daemon was already running
        #   2 if daemon could not be started
        start-stop-daemon --start --quiet --pidfile $PIDFILE --exec $DAEMON --
            test > /dev/null \
                || return 1
        start-stop-daemon --start --quiet --pidfile $PIDFILE --exec $DAEMON -- \
                $GPSD_OPTIONS -P $PIDFILE $DEVICES \
                || return 2
}
```

Listing A.5: /etc/init.d/gpsd, Zeile 57-70

A.3. Serial Port Konfiguration

Um die Erweiterungskarte Commell MP-954 nutzen zu können, müssen Anpassungen an der Boot bzw. Kernelkonfiguration gemacht und die Datei /etc/serial.conf erstellt werden.

Standardmäßig sind im Linux Kernel nur vier serielle Schnittstellen vorgesehen. Da das Alix System selber zwei Ports anbietet, muss die Anzahl der unterstüzten Schnittstellen entweder beim Kompilieren eines eigenen Kernels oder durch den Startparameter *8250.nr_uarts=8* erweitert werden, damit alle vier Schnittstellen der Erweiterungskarte genutzt werden können.

Listing A.6 zeigt eine entsprechende Änderung an der Boot Konfiguration.

```
title Voyage Linux 0.9 (Build Date 20130123)
root (hd0,0)
kernel /vmlinuz root=LABEL=VOYAGE_FS  console=ttyS0,38400n8 8250.nr_uarts=8
initrd /initrd.img
```

Listing A.6: /boot/grub/menu.lst

Die Erweiterungskarte arbeitet mit einer viermal höheren Frequenz als die vom Linux Kernel vorgesehene Standard Frequenz für serielle Schnittstellen. Dieser Unterschied muss dem Kernel mitgeteilt werden. Dazu wird der Befehl *setserial /dev/ttySX baud_base 460800* benötigt. Damit dieser nicht bei jedem Start neu eingegeben werden muss und für die in Abschnitt 3.2.2 beschriebene Anpassung an eine geringe Latenz eine weitere Änderung vorgenommen werden muss, wurde die Datei */etc/serial.conf* erstellt (Listing A.7. Diese wird beim Start des Debian Systems ausgelesen und die Konfiguration des Serial Ports entsprechend angepasst.

```
# This is a configuration file for the Commell MP-954 in combination with
# the PCEngines Alix board and the Garmin GPS18 LVC
#
```

```
# ttyS[0|1] are the original ports if the Alix board
# ttyS[2-5] are the ports of the Commell Card
#
# The Commell card uses its own oscillator, which is why baud_base needs to be
    set
# to 460800
# Source: https://azug.minpet.unibas.ch/wikiobsvermes/index.php/Embedded_Control
#
# Because the Alix board ports just support Rx/Tx, but no other signals, the
    Garmin
# GPS receiver needs to use one of the Commell ports to submit a PPS signal
#
# The Commell ports are set to low_latency to support a very low offset of about
# ~1us. Without this setting, we will not get a better result as ~50us.
#
/dev/ttyS0 uart undefined port 0x03f8 irq 4 baud_base 921600 spd_normal skip_test
/dev/ttyS1 uart undefined port 0x02f8 irq 3 baud_base 921600 spd_normal skip_test
/dev/ttyS2 uart 16950/954 port 0x1810 irq 2 baud_base 115200 spd_normal skip_test
/dev/ttyS3 uart 16950/954 port 0x1818 irq 2 baud_base 115200 spd_normal skip_test
/dev/ttyS4 uart 16950/954 port 0x1800 irq 2 baud_base 460800 spd_normal skip_test
    low_latency
/dev/ttyS5 uart 16950/954 port 0x1808 irq 2 baud_base 460800 spd_normal skip_test
    low_latency
```

Listing A.7: /etc/serial.conf

A.4. UMTS Einwahl

Um dem System zu ermöglichen, dass UMTS Modem zu nutzen, wird ein Einwahlprogramm benötigt. Für das Messgerät wird wvdial genutzt. Die Konfiguration für die während der Arbeit genutzten SIM Karten des Anbieters Fonic ist in Listing A.8 dargestellt. Die Pin wurde durch #### ersetzt.

```
[Dialer Defaults]
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
#; Modem Type = USB Modem
#; Phone = <Target Phone Number>
ISDN = 0
#; Username = <Your Login Name>
Init1 = ATZ
#; Password = <Your Password>
Modem = /dev/ttyUSB3
Baud = 38400

[Dialer pin]
```

```
Init3 = AT+CPIN=####

[Dialer fonic]
Init5 = AT+CGDCONT=1, "IP","pinternet.interkomm.de","",0,0
Baud = 38400
Username=fonic
Password=fonic
Dial Command = ATDT
Carrier Check = No
Phone = *99#
Stupid Mode = 1
New PPPD = yes
```

Listing A.8: /etc/wvdial.conf

Anhang B.

SEC Experimentdateien

Für die im Verlauf der Arbeit durchgeführten Experimente wurde die Stateless Experiment Control (SEC) Experimentensteuerung genutzt. In diesem Abschnitt werden die Konfigurationsdateien für die im Hauptteil referenzierten Experimente abgedruckt.

Die in Listing B.1 dargestellte Konfigurationsdateien wurde für den Test der Genauigkeit des Basic Verfahrens über eine 10 Mbps Ethernet Leitung genutzt. Dabei wurde eine Auslastung von 1% bzw. 10% angestrebt.

```
<experiment>
  <run>
    <client id="alix1">
      <system>./rmf -r server -sc eth0 -o BTRFS/Experimente/
          genauigkeit_unbelastet/server/alix2/ -P 2 -a 1 -G 1 -L 1</system>
      <system offset="2710s">./killscript.bash</system>
    </client>
    <client id="alix2">
      <system offset = "10s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS/
          Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
          60 -M basic -mB_c 125 -mB_p 100</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
          /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
          60 -M basic -mB_c 63 -mB_p 200</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
          /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
          60 -M basic -mB_c 42 -mB_p 300</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
          /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
          60 -M basic -mB_c 32 -mB_p 400</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
          /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
          60 -M basic -mB_c 25 -mB_p 500</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
```

```
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 21 -mB_p 600</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 179 -mB_p 700</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 157 -mB_p 800</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 139 -mB_p 900</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 125 -mB_p 1000</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 114 -mB_p 1100</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 105 -mB_p 1200</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 97 -mB_p 1300</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 90 -mB_p 1400</system>
      <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
        /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
        60 -M basic -mB_c 84 -mB_p 1500</system>
      <system offset = "180s">date</system>
    </client>
  </run>
  [...]
</experiment>
```

Listing B.1: Auszug aus gen_unb_basic_10_s1.xml

Listing B.2 beschreibt den gleichen Versuchsablauf wie Listing B.1, allerdings für das Mobile Verfahren.

```
<experiment>
  <run>
    <client id="alix1">
      <system>./rmf -r server -sc eth0 -o BTRFS/Experimente/
        genauigkeit_unbelastet/server/alix2/ -P 2 -a 1 -G 1 -L 1</system>
      <system offset="2710s">./killscript.bash</system>
    </client>
```

```
<client id="alix2">
  <system offset = "10s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS/
      Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 1 -mM_p 100</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 1 -mM_p 200</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 1 -mM_p 300</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 1 -mM_p 400</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 1 -mM_p 500</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 1 -mM_p 600</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 700</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 800</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 900</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 1000</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 1100</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 1200</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 1300</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 1400</system>
  <system offset = "180s">./rmf -r client -cc eth0 -SC 192.168.2.180 -o BTRFS
      /Experimente/genauigkeit_unbelastet/client/alix1 -P 2 -a 1 -G 1 -L 1 -t
      60 -M mobile -mM_l 10 -mM_p 1500</system>
  <system offset = "180s">date</system>
```

```
    </client>
  </run>
  [...]
</experiment>
```

Listing B.2: Auszug aus gen_unb_mobile_10_s1.xml

Das Listing B.3 zeigt die Konfigurationsdatei für die in Kapitel 5.1.3 beschriebene Messung über einen mit Cross Traffic belegten Kanal. Listing B.4 stellt die verwendete Konfiguration für den brute Traffic Generator dar.

```
<experiment>
 <run>
   <client id="worf">
     <system offset = "10s">taskset -c 0 ./brute -d 00:0d:b9:28:86:3c -s 00:27:0
         e:0f:95:92 -p 1 -i eth0 -f traffic.script</system>
   </client>
   <client id="alix1">
     <system>./rmf -r server -SC 192.168.2.180 -o BTRFS/Experimente/
         genauigkeit_belastet/30/ -P 2 -a 1 -G 1 -L 1</system>
     <system offset="720s">./killscript.bash</system>
   </client>
   <client id="alix2">
     <system offset = "10s">./rmf -r client -CC 192.168.2.181 -SC 192.168.2.180
         -o BTRFS/Experimente/genauigkeit_belastet/30/ -P 2 -a 1 -G 1 -L 1 -t 360
          -M mobile -mM_l 30 -mM_p 700</system>
     <system offset = "720s">date</system>
   </client>
 </run>
 [...]
</experiment>
```

Listing B.3: Auszug aus gen_bel_mobile_30_s1.xml

```
off msec=30000;
cbr msec=30000; rate=84; len=1500; saddr=192.168.2.102; sport=9999 daddr
    =192.168.2.181; dport=9999;
cbr msec=30000; rate=167; len=1500; saddr=192.168.2.102; sport=9999 daddr
    =192.168.2.181; dport=9999;
cbr msec=30000; rate=250; len=1500; saddr=192.168.2.102; sport=9999 daddr
    =192.168.2.181; dport=9999;
cbr msec=30000; rate=334; len=1500; saddr=192.168.2.102; sport=9999 daddr
    =192.168.2.181; dport=9999;
off msec=30000;
cbr msec=30000; rate=334; len=1500; saddr=192.168.2.102; sport=9999 daddr
    =192.168.2.181; dport=9999;
cbr msec=30000; rate=667; len=1500; saddr=192.168.2.102; sport=9999 daddr
    =192.168.2.181; dport=9999;
```

```
cbr msec=30000; rate=334; len=1500; saddr=192.168.2.102; sport=9999 daddr
    =192.168.2.181; dport=9999;
off msec=30000;
cbr msec=30000; rate=834; len=1500; saddr=192.168.2.102; sport=9999 daddr
    =192.168.2.181; dport=9999;
off msec=30000;
```

Listing B.4: traffic.skript - Brute Traffic Konfiguration

Das Listing B.5 wurde für die Experimente zur parallelen Messung im Mobilfunknetz verwendet.

```
<experiment>
  <run>
    <client id="alix1">
      <system>./rmf -r client -CC 10.35.253.153 -SC 134.99.112.37 -Sc 53221 -SM
          134.99.112.37 -Sm 53221 -o BTRFS/Experimente/mobilfunk/exp2/ -P 2 -a 1 -
          G 1 -L 1 -t 60 -M mobile -mM_l 30 -mM_p 900 -mM_s 90</system>
      <system offset="61s">date</system>
    </client>
    <client id="alix2">
      <system offset = "20s">./rmf -r client -CC 10.68.214.31 -SC 134.99.112.37 -
          Sc 53222 -SM 134.99.112.37 -Sm 53222 -o BTRFS/Experimente/mobilfunk/exp2
          / -P 2 -a 1 -G 1 -L 1 -t 40 -M mobile -mM_l 30 -mM_p 900 -mM_s 90</
          system>
      <system offset="41s">date</system>
    </client>
    <client id="alix3">
      <system offset = "40s">./rmf -r client -CC 10.33.1.195 -SC 134.99.112.37 -
          Sc 53223 -SM 134.99.112.37 -Sm 53223 -o BTRFS/Experimente/mobilfunk/exp2
          / -P 2 -a 1 -G 1 -L 1 -t 20 -M mobile -mM_l 30 -mM_p 900 -mM_s 90</
          system>
      <system offset="21s">date</system>
    </client>
  </run>
</experiment>
```

Listing B.5: Auszug aus exp2.xml

Literaturverzeichnis

[3gp] *3rd Generation Partnership Project*. Available online at http://www.3gpp.org; Last visited june 23th 2013

[3GP99] 3GPP: Digital cellular telecommunications system (Phase 2+); High Speed Circuit Switched Data (HSCSD) - Stage 2 / 3rd Generation Parntership Project (3GPP). Version: 1999. http://www.3gpp.org/ftp/Specs/html-info/0334.htm. 1999. Forschungsbericht.

[3GP03] 3GPP: Digital cellular telecommunications system (Phase 2+); Multiplexing and Multiple Access on the Radio Path / 3rd Generation Parntership Project (3GPP). Version: 2003. http://www.3gpp.org/ftp/Specs/html-info/0502.htm. 2003. Forschungsbericht.

[3GP12] 3GPP: Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); Overall Description of the GPRS radio interface; Stage 2 / 3rd Generation Parntership Project (3GPP). Version: 2012. http://www.3gpp.org/ftp/Specs/html-info/43064.htm. 2012. Forschungsbericht.

[Ali] *PCEngines Alix6f2 Specification*. Available online at http://www.pcengines.ch/alix6f2.htm; Last visited june 23th 2013

[Amf12] AMFT, Tobias: *Eine zustandsbasierte Experimentsteuerung über einen fehleranfälligen Kommunikationskanal*, Departement of Computer Science, Heinrich Heine University Düsseldorf, bachelor thesis, june 2012

[Ber11] BERGMANN, Arnd: *Optimizing Linux with cheap flash drives*. website, feb 2011. Available online at http://lwn.net/Articles/428584/; Last visited june 23th 2013

[BM99] BRADNER, S.; MCQUAID, J.: *Benchmarking Methodology for Network Interconnect Devices*. RFC 2544 (Informational). http://www.ietf.org/rfc/rfc2544.txt. Version: März 1999 (Request for Comments). Updated by RFC 6201

[bon] *bonnie++ Website*. Online verfügbar unter http://www.coker.com.au/bonnie++/; Letzter Besuch: 23.6.2013

[Bra91] BRADNER, S.: *Benchmarking Terminology for Network Interconnection Devices*. RFC 1242 (Informational). http://www.ietf.org/rfc/rfc1242.txt. Version: Juli 1991 (Request for Comments). Updated by RFC 6201

[Bro12] BROWN, Neil: *A NILFS2 score card*. nov 2012. Available online at http://lwn.net/Articles/522507/; Last visited june 23th 2013

[Btr] *Btrfs Wiki*. Available online at http://btrfs.wiki.kernel.org; Last visited june 23th 2013

[CHPI+00] *Kapitel* 5.3.3. In: COMPAQ; HEWLETT-PACKARD; INTEL; LUCENT; MICROSOFT; NEC; PHILIPS: *Universal Serial Bus Specification Revision 2.0*. 2000

[clo] *NTP Clock Select Algorithm*. Online verfügbar unter http://www.eecis.udel.edu/~mills/ntp/html/select.html; Letzter Besuch: 23.6.2013

[CTT10] CARD, Remy; TS'O, Theodore; TWEEDIE, Stephen: Design and Implementation of the Second Extended Filesystem. In: *Proceedings of the First Dutch International Symposium on Linux* (2010).

[Deba] *Debian Project*. Available online at http://www.debian.org; Last visited june 23th 2013

[Debb] *Debian Live Project*. Available online at http://live.debian.net; Last visited june 23th 2013

[fla] *flashbench - flash media benchmark*. Available online at https://github.com/bradfa/flashbench; Last visited june 23th 2013

[Goe10] GOEBEL, Norbert: *Trace-basierte Simulation von Mobilfunkcharakteristiken für die Fahrzeug-zu-Fahrzeug Kommunikation*, Departement of Computer Science, Heinrich Heine University Düsseldorf, master thesis, august 2010

[GPSa] *Garmin GPS 18x LVC Datasheet*. Available online at http://www.garmin.com/manuals/GPS18x_TechnicalSpecifications.pdf; Last visited june 23th 2013

[gpsb] *gpsd Project*. Available online at `http://catb.org/gpsd/`; Last visited june 23th 2013

[GRT09] GOLDONI, Emanuele; ROSSI, Giuseppe; TORELLI, Alberto: Assolo, a New Method for Available Bandwidth Estimation. In: *Proceedings of the 2009 Fourth International Conference on Internet Monitoring and Protection*. Washington, DC, USA : IEEE Computer Society, 2009 (ICIMP '09). ISBN 978–0–7695–3612–5, 130–136.

[GRT10] GOLDONI, Emanuele; ROSSI, Giuseppe; TORELLI, Alberto: ASSOLO: an Efficient Tool for Active End-to-end Available Bandwidth Estimation. In: *International Journal On Advances in Systems and Measurements* 2 (2010), S. 283–292.

[GS10] GOLDONI, Emanuele; SCHIVI, Marco: End-to-end available bandwidth estimation tools, an experimental comparison. In: *Proceedings of the Second international conference on Traffic Monitoring and Analysis*. Berlin, Heidelberg : Springer-Verlag, 2010 (TMA'10). ISBN 3–642–12364–3, 978–3–642–12364–1, 171–182.

[ipe] *iperf Website*. Available online at `http://code.google.com/p/iperf`; Last visited june 23th 2013

[LDS06] LAO, Li; DOVROLIS, Constantine; SANADIDI, M. Y.: The probe gap model can underestimate the available bandwidth of multihop paths. In: *SIGCOMM Comput. Commun. Rev.* 36 (2006), oct, Nr. 5, 29–34. `http://dx.doi.org/10.1145/1163593.1163599`. DOI 10.1145/1163593.1163599. ISSN 0146–4833.

[Lov10] LOVE, Robert: *Linux Kernel Development*. 3rd. Addison-Wesley Professional, 2010. ISBN 0672329468, 9780672329463

[man04] *mii-tool(8) - Linux man page*. mar 2004.

[man12a] *clock_nanosleep(2) - Linux man page*. nov 2012.

[man12b] *ethtool(8) - Linux man page*. june 2012.

[man12c] *sched_setscheduler(2) - Linux man page*. oct 2012.

[MC8] *Sierra Wireless MC8790 Datasheet*. Available online at `http://www.datasheetarchive.com/MC8790-datasheet.html`; Last visited june 23th 2013

[MCB+07] MATHUR, Avantika; CAO, Mingming; BHATTACHARYA, Suparna; DILGER, Andreas;

TOMAS, Alex; VIVIER, Laurent: The new ext4 filesystem: current status and future plans. In: *Proceedings of the Linux Symposium* Bd. 2, 2007, S. 21–33.

[Mil10] MILLS, David L.: *Computer Network Time Synchronization: The Network Time Protocol on Earth and in Space, Second Edition*. 2nd. Boca Raton, FL, USA : CRC Press, Inc., 2010. ISBN 1439814635, 9781439814635

[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).

[NIL] *NILFS2 Online Manual*. Available online at `http://nilfs.org/en/manual.html`; Last visited june 23th 2013

[NSGR05] N.BONELLI; S.GIORDANO; G.PROCISSI; R.SECCHI: BRUTE: A High Performance and Extensible Traffic Generator. In: *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'05)*. Philadelphia, PA, july 2005.

[ntp] *NTP Online Dokumentation*. Available online at `http://www.ntp.org/documentation.html`; Last visited june 23th 2013

[PDMC03] PRASAD, Ravi; DOVROLIS, Constantinos; MURRAY, Margaret; CLAFFY, KC: Bandwidth estimation: metrics, measurement techniques, and tools. In: *Network, IEEE* 17 (2003), Nr. 6, S. 27–35.

[PPS] *NTP PPS Documentation*. Available online at `http://www.eecis.udel.edu/~mills/ntp/html/pps.html`; Last visited june 23th 2013

[RBM12] RODEH, Ohad; BACIK, Josef; MASON, Chris: BRTFS: The Linux B-tree Filesystem / IBM Research Report RJ10501 (ALM1207-004). 2012. Forschungsbericht.

[RO91] ROSENBLUM, Mendel; OUSTERHOUT, John K.: The design and implementation of a log-structured file system. In: *ACM SIGOPS Operating Systems Review* 25 (1991), Nr. 5, S. 1–15.

[Ser] *Commell MP-954 Specification*. Available online at `http://www.commell.com.tw/Product/Peripheral/MiniPCI/MP-954.HTM`; Last visited june 23th 2013

[Twe98] TWEEDIE, Stephen C.: Journaling the Linux ext2fs filesystem. In: *The Fourth Annual Linux Expo*, 1998.

[Vir] *VirtualBox*. Available online at `https://www.virtualbox.org/`; Last visited june 23th 2013

[Voya] *Voyage Linux*. Available online at `http://linux.voyage.hk/`; Last visited june 23th 2013

[Voyb] *Voyage Linux 0.8.5 Build Konfiguration*. Available online at `http://svn.voyage.hk/repos/voyage/branches/voyage-live/0.8.5/`; Last visited june 23th 2013

[Wil12] WILKEN, Sebastian: *Verfahren zur Datenratenmessung in Mobilfunknetzwerken*, Departement of Computer Science, Heinrich Heine University Düsseldorf, master thesis, june 2012

[Yag08] YAGHMOUR, Karim: *Building embedded Linux systems*. Beijing Sebastopol, CA : O'Reilly, 2008. ISBN 9780596529680

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit 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, 24.Juni 2013 Christian Simon Lange

Diese CD enthält:

- eine *pdf* -Version der vorliegenden Masterarbeit
- die LaTeX- und Grafik-Quelldateien der vorliegenden Masterarbeit samt aller verwendeten Skripte
- Quelldateien des Rate Measurement Framework
- Quelldateien des Stateful Experiment Control
- Virtual Box Image zum Bau eigener Voyage Linux Images
- Messdaten zu den in der Arbeit beschriebenen Experimenten
- die Websites der verwendeten Internetquellen