Trace-basierte Simulation von Mobilfunkcharakteristiken für die Fahrzeug-zu-Fahrzeug Kommunikation

Masterarbeit

von

Norbert Goebel

aus

Neuss

vorgelegt am

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

August 2010

Betreuer:
Markus Koegel M. Sc.

Danksagung

Ich möchte mich bei all denjenigen bedanken, die mich bei der Erstellung meiner Masterarbeit unterstützt haben.

Meinem Betreuer Markus Koegel möchte ich für die vielen interessanten Diskussionsrunden, die zahlreichen Anregungen und Vorschläge zum Inhalt und die vielseitige Hilfe bei der typographischen Umsetzung danken.

Ganz besonders bedanken möchte ich mich bei meiner Frau Sabine für die tatkräftige Unterstützung. Danke dass du mir den Rücken freigehalten und als Lektorin zur Verfügung gestanden hast.

Vielen Dank auch an meinen Vater und meine Schwiegereltern. Ohne eure Kinderbetreuung wäre diese Masterarbeit nicht möglich gewesen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kapitel 1

Einleitung

Das weltweite Verkehrsaufkommen wächst seit Jahrzehnten und ein Ende des Wachstums ist nicht abzusehen. Durch die erhöhte Verkehrsdichte steigt auch das Stau- und Unfallrisiko. Daher ist es nicht verwunderlich, dass die Automobilindustrie intensiv an Systemen zur Steigerung der Verkehrseffizienz und Sicherheit forscht. So hat zum Beispiel das *Antiblockiersystem (ABS)* in vielen Situationen Unfälle verhindert oder deren Folgen zumindest gelindert. Auch das *Elektronische Stabilitätsprogramm (ESP)* wird in einigen Jahren wahrscheinlich genauso in jedem PKW eingebaut sein wie heute schon der Anschnallgurt [ADA]. All diese Systeme erhöhen die Verkehrssicherheit allerdings ausschließlich durch Informationen, die das jeweilige Fahrzeug selbst gemessen hat.

Die Entwicklung in der Automobilindustrie zeigt, dass viele Hersteller großes Potential für zukünftige Verbesserungen der Verkehrssicherheit und der Verkehrseffizienz darin sehen, Informationen direkt zwischen den Fahrzeugen mittels Fahrzeug-zu-Fahrzeug- (*C2C*) oder Fahrzeug-zu-Infrastruktur- (*C2I*) Kommunikation auszutauschen. Durch diesen Informationsaustausch, der z.B. die aktuelle Position, Geschwindigkeit und Bewegungsrichtung der Fahrzeuge beinhaltet, erhält der Boardcomputer eines mit *C2C*-Technik ausgestatteten Fahrzeugs einen wesentlich besseren Einblick in seine Umgebung. Folglich kann der Boardcomputer den Fahrer frühzeitig über kritische Verkehrssituationen informieren und somit die Situation entschärfen. Er kann den Fahrer zum Beispiel frühzeitig vor einem Stauende hinter einer Kurve warnen und ihm damit ein kontrolliertes Bremsmanöver ermöglichen.

Damit sich die Fahrzeug-zu-Fahrzeug Kommunikation jedoch durchsetzen kann, wird eine Mindestpenetrationsrate – ein genügend großer Prozentsatz an Fahrzeugen, die mit *C2C*-Technik ausgestattet sind – benötigt. Wie die Autoren von [WS05] zeigen, ist diese Penetrationsrate nur zu erreichen, wenn sich alle Automobilhersteller auf einen Standard für die Fahrzeug-zu-Fahrzeug Kommunikation einigen und diesen auch einsetzen. Diese Mindestpenetrationsrate ist zum einen anwendungsabhängig und wird zum anderen auch von der verwendeten Kommunikationstechnik beeinflusst. Die in der Markteinführungsphase niedrige Penetrationsrate kann bei infrastrukturlosem WLAN zur Netz-

werkpartitionierung und damit zu großen Funktionseinschränkungen führen. Dieses Problem hat die flächendeckende, infrastrukturbasierte Mobilkommunikation (GMS, UMTS, LTE) nicht.

Die entscheidende zu beantwortende Frage in der Fahrzeug-zu-Fahrzeug Kommunikation ist die nach der verkehrlichen Wirkung von Anwendungen wie etwa Kollisionswarnsystemen, Stauendewarnern oder Verkehrsinformationssystemen. Oftmals kann eine Anwendung mit unterschiedlichen zugrundeliegenden Fahrzeug-zu-Fahrzeug Kommunikationstechnologien realisiert werden. Dabei konkurriert der klassische Mobilfunk (*GSM/UMTS/LTE* mit Unterstützung zentraler Server) mit infrastrukturlosem WLAN nach dem Standard *IEEEE802.11p*, kombiniert mit dezentraler Datenverarbeitung. Folglich ist die Lösung der Frage, welche Auswirkungem die unterschiedlichen Ansätze auf die verkehrliche Wirkung haben, von wesentlicher Bedeutung.

Wir wollen daher in dieser Arbeit eine Möglichkeit aufzeigen, die Kommunikation in heutigen Mobilfunksystemen zu simulieren, um bestimmen zu können, welche Informationen zu jedem Zeitpunkt in den Fahrzeugen vorliegen.

1.1 Motivation

Das zu lösende Problem ist inhärent schwer, da die betrachteten Mobilfunksysteme (*GSM, UMTS, LTE*) höchst komplex sind und nur mit sehr vielen Einschränkungen modelliert werden können.

Die Lösungsidee besteht darin, dass anstelle der reinen Simulation der Netzwerke und Radiokommunikation trace-basierte Simulationen eingesetzt werden. Hierfür werden zunächst repräsentative Strecken in Messfahrten abgefahren und die charakteristischen Eigenschaften (wie Latenz, Verlustrate, verfügbare Bandbreite, Signalqualität, vertikaler Handover zu anderen Netztypen) des Mobilfunknetzes aufgezeichnet. Diese Aufzeichnungen (*Traces*) werden im folgenden analysiert, um ein Simulationsmodell zu erstellen, das die Beantwortung der Frage, welche Informationen an jedem Zeitpunkt in jedem Fahrzeug vorliegen, ermöglicht.

Gegenüber der reinen Simulation aller Netzwerkparameter und der Radiokommunikation hat dieses trace-basierte Verfahren den Vorteil, dass die Messwerte real sind und nicht erheblich von der Realität abweichen. Außerdem erwarten wir, dass das trace-basierte Verfahren sehr gut skaliert, da keine komplexen Simulationsschritte notwendig sind. Gerade die Skalierbarkeit ist bei Simulationen mit mehreren zehntausend Fahrzeugen immanent wichtig.

1.2 Aufbau der Arbeit

Kapitel 2 fasst die verwandten Arbeiten in den Bereichen *trace-basierte Simulation, Bandbreitenmessung und Zeitsynchronisation* zusammen. In Kapitel 3 erläutern wir, welche Messwerte wir erfassen wollen und welche Messverfahren wir dafür entwickelt und benutzt haben. Das daraus entstandene Loggingframework und die zur Erstellung der Traces verwendete Hardware stellen wir in Kapitel 4 vor. Um diese Traces zu analysieren haben wir ein Analysewerkzeug entwickelt, welches wir in Kapitel 5 vorstellen. Daran anschließend stellen wir in Kapitel 6 unseren Entwurf für ein Simulationsmodell vor, das auf den gemessenen Daten aufbaut und die Netzwerkkommunikation für die gesamte Kommunikationsstrecke zwischen Client und Server auf den OSI-Schichten 1 bis 3 abdeckt. Abschließend fassen wir die Ergebnisse dieser Masterarbeit in Kapitel 7 zusammen und geben einige Anregungen für künftige Forschungen auf dem Gebiet der trace-basierten Simulation von Mobilfunkcharakteristiken für die Fahrzeug-zu-Fahrzeug Kommunikation.

Kapitel 2

Verwandte Arbeiten

In diesem Kapitel geben wir zunächst einen Überblick über verwandte Arbeiten, die sich mit *trace-basierter* Simulation drahtloser Netzwerke beschäftigen (2.1). Anschließend stellen wir zwei Themengebiete vor, die zur Erstellung von Netzwerk-*Traces* von besonderem Interesse sind: Messverfahren zur Ermittlung der verfügbaren Bandbreite (2.2) und Methoden der Zeitsynchronisation (2.3).

2.1 Trace-basierte Simulation drahtloser Netzwerke

Die erste uns bekannte Arbeit [NSNK97], die sich mit *trace-basierter* Simulation mobiler Netzwerke beschäftigt, wurde 1997 veröffentlicht. Ziel der Autoren war es eine synthetische Netzwerkumgebung zu schaffen, die es ihnen ermöglicht Bedingungen für drahtlose Kommunikation zu reproduzieren. *Trace replay*-Verfahren, die als Simulationsschritt einmal erstellte Traces erneut in der aufgezeichneten zeitlichen Abfolge wiederholen, simulieren jedoch nur die Netzwerkauslastung, ermöglichen aber nicht die Erstellung einer synthetischen Netzwerkumgebung. Deshalb haben die Autoren eine von ihnen als *trace modulation* bezeichnete Technik entwickelt, die sich in drei Phasen unterteilt, *collection*, *distillation* und *modulation*. In der ersten Phase haben die Autoren *Traces* ihres *WaveLAN*, einem Vorfahren des heutigen *IEEE802.11* WLAN, erstellt. Aus den gesammelten Daten wurden in der zweiten Phase zeitbezogene Datensätze, die jeweils Werte für die Latenz, Bandbreite und Paketverlustrate für diesen Zeitabschnitt enthielten, generiert. Diese wurden dann im letzten Schritt dazu verwendet, ein einfaches Netzwerkmodell zu erstellen. Ihrer Modellbildung legten die Autoren die Annahme zu Grunde, dass die Latenzen symmetrisch wären, stellten jedoch später fest, dass diese Annahme nicht zutrifft und schlugen als Lösung hochpräzise, synchrone Uhren in den Messstationen vor um die asymmetrischen Latenzen messen zu können.

Einen ganz anderen Weg schlugen die Autoren in [CKLL01] ein. Sie generierten *Traces* des IP-Netzwerkverkehrs von Modem und ISDN Dial-Up Links der Universität Dortmund. Unter Verwen-

dung des *batch Markovian arrival Process (BMAP)* erstellten sie aus diesen Traces ein synthetisches Modell des Netzwerkverkehrs, mit dem sie Rückschlüsse auf den Netzwerkverkehr in UMTS Netzwerken zogen.

Die Arbeit [YRZ+08] beschäftigt sich zwar auch mit trace- basierter Simulation, allerdings simulieren die Autoren nicht das drahtlose Netzwerk, sondern den Upload-Verkehr in 3G Netzwerken. Hierfür haben sie *Traces* des Netzwerkverkehrs eines koreanischen 3G Netzwerkes erstellt und in Bezug auf den immer weiter steigenden Uploadverkehr mit Hilfe ihres Simulationsmodells analysiert. Dabei haben sie als erste festgestellt, dass der Upload-Traffic sowohl in den *Base Stations (BS)*, als auch in den *Subscriber Stations (SS)* selbstähnlich ist. Die durch die Selbstähnlichkeit entstehenden Bursts sind nach ihren Beobachtungen bei allgemein relativ niedriger Last in den SS unproblematisch. Sie werden dann zu einem Problem, wenn eine große Menge zeitkritischer Pakete in vielen SS eintreffen, auch wenn die Netzwerklast unterhalb der Netzwerkkapazität liegt.

Innovativ ist das Simulationsmodell, das die Autoren in dem wissenschaftlichen Artikel [KH09] vorstellen. Die Autoren haben erkannt, dass sich die klassische Simulationstechnik, also die Simulation aller physikalischen Parameter der drahtlosen Kommunikation, nicht zur Simulation von einer Vielzahl von Kommunikationspartnern in einem *Vehicular Ad Hoc Network (VANET)* eignet, da die klassischen Verfahren zu schlecht skalieren. Deshalb entwickelten die Autoren ein analytisches Modell, das die Wahrscheinlichkeit des erfolgreichen Empfangs eines 1-*Hop Broadcasts* in einem *VANET* bestimmt. Unter Verwendung von Netzwerktraces und *curve fitting* Techniken wurde ein Modell entwickelt, das für die Berechnung der 1-*Hop* Empfangswahrscheinlichkeit lediglich die folgenden vier Parameter benötigt:

1. Abstand zwischen Sender und Empfänger
2. Sendeleistung
3. Übertragungsrate
4. Fahrzeugdichte

Abschließend haben die Autoren ihr Modell mit statistischen Methoden validiert.

Eine Arbeit, die unter Verwendung von Traces eine Simulation von Mobilfunknetzwerken durchführt, konnten wir nicht finden. Diese Lücke möchten wir mit dieser Arbeit füllen.

2.2 Bandbreitenmessung

Über Bandbreitenmessung in Netzwerken haben sich schon viele Wissenschaftler Gedanken gemacht. Da wir für unsere Messungen keinen Einblick in das Innere der Mobilfunknetze haben, sind für uns nur Ende-zu-Ende Messverfahren von Interesse. Bei diesen Verfahren helfen beide Enden der Messstrecke aktiv mit. Diese Messverfahren nutzen entweder *Packet Pairs*, bei denen jeweils zwei Pakete back-to-back – also ohne Lücke zwischen den Paketen – verschickt werden, oder *Packet Trains*, bei denen auf variierende Weise eine ganze Reihe von Messpaketen verschickt werden. Die Verfahren unterscheiden sich in der Regel lediglich darin, in welcher Weise der Paketeversand durchgeführt wird und wie die Auswertung erfolgt.

Da in der Literatur eine Reihe von unterschiedlichen Notationen benutzt werden, führen wir hier zunächst die in dieser Arbeit benutzten Definitionen ein.

Die *Kapazität (C)* eines Netzwerklinks ist der maximale *IP-Layer* Durchsatz, der auf diesem Link ohne konkurrierende Datenströme (*cross traffic*) erreicht werden kann. Die *verfügbare Bandbreite (A)* dagegen ist der maximale *IP-Layer* Durchsatz, der auf einem schon durch cross traffic belegten Link noch erreichbar ist. Die *Auslastung (u)* eines Links ist der Anteil der *Kapazität*, der durch cross traffic verbraucht wird, daher gilt:

$$A = C(1-u)$$

Ein Netzwerkpfad ist eine Reihe von Netzwerklinks vom Sender zum Empfänger. Die Kapazität eines Netzwerkpfades wird durch den Link mit der minimalen Kapazität im Pfad bestimmt. Analoges gilt für die verfügbare Bandbreite. Für einen Netzwerkpfad mit H Hops und den Auslastungen u_i auf den Links gilt also:

$$\begin{aligned} C &= \min_{i=0..H} C_i \\ A &= \min_{i=0..H} [C_i(1-u_i)] \end{aligned} \tag{2.1}$$

Der Link auf einem Pfad mit der geringsten *Kapazität* wird *narrow link* genannt, der mit der geringsten *verfügbaren Bandbreite* wird als *tight link* bezeichnet.

Eine der ersten Arbeiten [Kes91] zu dem Thema Bandbreitenmessung beschäftigt sich mit einem Verfahren, das der Autor *Packet-Pair* nennt. Das Verfahren setzt allerdings voraus, dass die Router *fair queueing* benutzen und ist daher für das heutige Internet unpassend. Das später eingeführte Cprobe [CC96] nutzt einen *Packettrain* um die verfügbare Bandbreite zu bestimmen. Die Autoren von [DRM01] haben jedoch gezeigt, dass dieses Verfahren weder die verfügbare Bandbreite, noch die Kapazität, sondern eine von Ihnen als *Asymptotic Dispersion Rate (ADR)* definierte Metrik misst. Die *ADR* ist meist deutlich kleiner als die Kapazität und liegt in der Regel auch unterhalb der ver-

fügbaren Bandbreite und ist für genügend viele Pakete in der Messreihe unabhängig von der Anzahl der Messpakete. Die Autoren entwickeln in Ihrem Paper jedoch ein Verfahren zur Bestimmung der Netzwerkkapazität, und kein Verfahren zur Bestimmung der verfügbaren Bandbreite. Dafür analysieren sie, welche Einflüsse die Messungen, die man mit *Packet Dispersion* Messverfahren macht, in welcher Weise verändern.

Neuere Ende-zu-Ende Bandbreitenmessverfahren verwenden in der Regel entweder eine Variation des *Probe Gap Model (PGM)* oder des *Probe Rate Model (PRM))*.

2.2.1 Das *Probe Gap Model (PGM)*

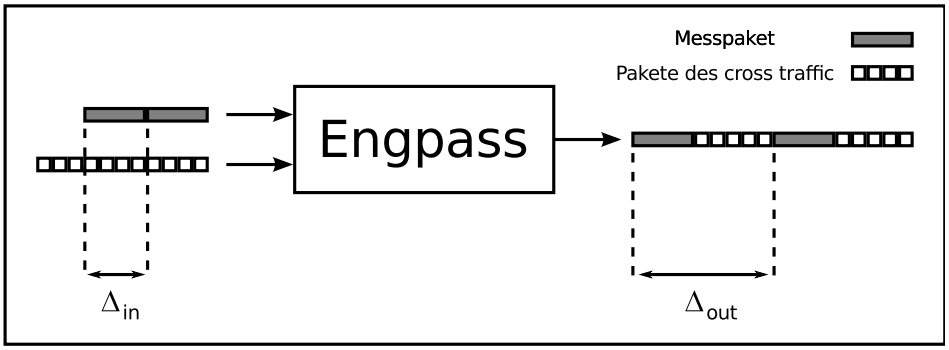

Abbildung 2.1: Funktionsweise des *Probe Gap Model (PGM)*.

Das *Probe Gap Model (PGM)* nutzt die *Packet Dispersion* – die zeitliche Lücke zwischen zwei Messpaketen am Empfänger, die am Sender *back-to-back* verschickt wurden – und errechnet daraus die verfügbare Bandbreite (vergleiche Abbildung 2.1). Eingesetzt wird dieses Verfahren unter anderem in den Tools *Delphi* [RCR+00], *IGI* [HMSM03] und *Spruce* [SKK03]. Ein großer Vorteil des Verfahrens ist, dass die Messungen nur sehr wenig Bandbreite benötigen. Die Nachteile sind, dass es von *FIFO* Queues in den Routern ausgeht und annimmt, dass nur ein Bandbreitenengpass auf dem Weg vom Sender zum Empfänger existiert, der gleichzeitig der *tight link* und der *narrow link* ist. Außerdem muss man die Kapazität des Netzwerkpfades kennen um die verfügbare Bandbreite mit der Formel

$$A = C \cdot \left(1 - \frac{\Delta_{out} - \Delta_{in}}{\Delta_{in}}\right) \tag{2.2}$$

bestimmen zu können. Dabei ist Δ_{in} der Zeitabstand (*packet dispersion*) zwischen dem letzten Byte des ersten Messpakets und dem letzten Byte des zweiten Messpakets am Sender. Analog ist Δ_{out} der Zeitabstand der gleichen Bytes am Empfänger. Da die Daten am Sender *back-to-back* versendet werden gilt für Pakete der Größe L:

$$\Delta_{in} = \frac{L}{C}$$

Die Autoren von [LDS06] stellen außerdem fest, dass das Probe Gap Model die verfügbare Bandbreite unterschätzt, wenn der cross traffic nicht den gleichen Pfad wie die Messpakete benutzt.

2.2.2 Das *Probe Rate Model (PRM)*

Das *Probe Rate Model (PRM)* versucht sich an die Sendedatenrate heranzutasten, die der verfügbaren Bandbreite des Netzwerkpfades entspricht. Dazu nutzt es das Prinzip der *self-induced congestion*: Sendet man Daten mit einer Datenrate, die geringer ist als die verfügbare Bandbreite über einen Netzwerkpfad, so ist die Datenrate am Empfänger gleich der Datenrate am Sender. Ist die Sendedatenrate jedoch größer als die verfügbare Bandbreite, so bauen sich im Netzwerk Warteschlangen auf, die Latenz der Pakete steigt und die Empfangsdatenrate ist geringer als die Senderate. Durch wiederholte Versuche lässt sich die verfügbare Bandbreite ermitteln. Dies geschieht, indem die Schwelle ermittelt wird, ab der die Senderate nicht mehr gleich, sondern größer als die Empfangsrate ist. Vertreter des *Probe Rate Model* sind unter anderem TOPP [MBG00], Pathload [JD02], Pathchirp [RRB+03] und PTR [HMSM03].

Für diese Messverfahren spricht, dass sie die verfügbare Bandbreite bei gleichbleibendem cross traffic nahezu beliebig genau bestimmen können. Solange der cross traffic gleich bleibt kann man die Messreihe beliebig oft mit veränderten Sendedatenraten wiederholen um die gewünschte Genauigkeit der Messung zu erreichen. Die gravierenden Nachteile der *PRM*-Messverfahren für unsere mobilen Messungen sind aber, dass pro Messung mehrere *RTTs* benötigt werden um den Übergangspunkt zur self-induced congestion zu bestimmen und die pro Messung benötigte Datenmenge. So haben die Autoren von [SKK03] festgestellt, dass *Pathload* zwischen 2,5 MB und 10 MB pro Messung benötigt, wogegen die *PGM* Verfahren *IGI* und *Spruce* mit 130 KB beziehungsweise 300 KB Daten pro Messung auskommen.

Würden wir ein *PRM*-Modell zur Messung benutzen und 2,5 MB Daten pro Messung über den UMTS Uplink unseres Modems schicken, der maximal 48 kByte/s zur Verfügung stellt, so würden wir für eine solche Messung alleine ca. 53 Sekunden benötigen. Wir brauchen jedoch ortsbezogene Messungen, ein sich bewegendes Fahrzeug, das unser Messgerät enthält, legt aber bei 30 km/h in dieser Zeit schon 450 m zurück. *PRM* basierte Verfahren sind daher für unsere Messungen nicht geeignet.

2.3 Zeitsynchronisation

Ein Problem, das alle digitalen Uhren, und damit auch jene in Computern, mit sich bringen, ist die Ungenauigkeit der Quarze, die die Frequenz f erzeugen, die für das Voranschreiten der Zeitrechnung

mit Hilfe der Formel $Zeit = Zeit + \frac{1}{f}$ benötigt wird. Die verwendeten Quarze arbeiten meistens nicht exakt mit der Frequenz, die sie nominell haben sollten. Zusätzlich haben Umwelteinflüsse wie Temperatur, Luftdruck, Magnetfelder und die Versorgungsspannung Auswirkungen auf die Frequenz. So steigt in aller Regel die Frequenz eines Quarzes mit steigender Temperatur. Eine Frequenzabweichung von lediglich 0,001% (bezogen auf die gängigen Quarze in PC Uhren) führt jedoch schon zu einer Ungenauigkeit von ca. 1 Sekunde pro Tag. Daher wird bei Zeitmessproblemen auch die Messgröße *PPM (Part Per Million* mit $1PPM = 0,0001\%$) verwendet um geringe Schwankungen besser beschreiben zu können.

Es ist schon kompliziert, den Fehler einer einzigen Uhr zu korrigieren, daher verwundert es nicht, dass es ungleich komplizierter ist, die Uhren zweier beliebiger Computer zu synchronisieren.

Für die Synchronisation in drahtgebundenen Netzwerken wurde für dieses Problem das *Network Time Protocol (NTP)* [Mil85, Mil88, Mil89, Mil92] erfunden. NTP benötigt eine Referenz Uhr, die an einen NTP-Server angeschlossen ist. Ein NTP-Client kann nun seine Uhr mit dem NTP-Server abgleichen, indem er periodisch *Request* und *Reply* Paketpaare mit dem Server austauscht. Der Client fügt dabei einen Sendezeitstempel, den *originate timestamp*, in das *Request*-Paket an den Server ein. Dieser wiederum fügt einen Empfangszeitstempel, den *receive timestamp*, hinzu und sendet das Ganze in einem *Reply* Paket, in das er auch noch einen Sendezeitstempel (*transmit timestamp*) einfügt, an den Client zurück. Beim Empfang der Antwort merkt sich der Client zusätzlich den Empfangszeitstempel des Antwortpakets (*reply receive timestamp*). Nun kann der Client die Übertragungszeit (*delay*) mit der Formel

$$\begin{aligned} delay &= \frac{\text{total delay} - \text{remote processing time}}{2} \\ &= \frac{(\text{reply receive} - \text{originate}) - (\text{transmit} - \text{receive})}{2} \end{aligned} \tag{2.3}$$

näherungsweise berechnen. *NTP* schätzt den Delay also als auf die Hälfte einer *Roundtriptime (RTT)* abzüglich der Bearbeitungszeit im Server. Mit Hilfe des so bestimmten *delay* kann der Client nun mittels des *transmit timestamp* und dem eigenen *reply receive timestamp* den Zeitunterschied zum Server schätzen.

Da die Zeit kontinuierlich verlaufen soll, ändert NTP die Systemzeit nur in kleinen Schritten, dazu nutzt NTP auf Unix/Linux zwei Methoden:

settimeofday setzt die Systemzeit neu wenn der Offset mehr als 128ms beträgt – die Uhr springt.

adjtime beschleunigt bzw. verlangsamt die Uhr indem es die virtuelle Frequenz der Softwareuhr verändert. Diese Veränderung wird in sehr kleinen Schritten vollzogen, in Linux z.B. mit einer Rate von 0,5 ms pro Sekunde.

Da jede Uhr fehlerbehaftet ist, muss NTP diese Korrekturen häufig wiederholen. Damit NTP diese Anpassungen auch fortführen kann, schätzt NTP den systematischen Fehler der Uhr laufend neu und speichert ihn als *Drift* in der Einheit PPM. Sollten nun alle Synchronisationsquellen nicht mehr erreichbar sein, führt NTP die Korrekturen weiterhin anhand des vorher ermittelten Drift weiter aus. Da der Fehler der Uhr jedoch variiert, ist diese Notfallkorrektur natürlich ebenfalls fehlerbehaftet – aber immer noch wesentlich besser als keine Korrektur.

Zeitserver werden bei *NTP* anhand ihres *Stratums* in Klassen eingeteilt. *Stratum 1* Server sind dabei Server, an die eine hochgenaue Zeitquelle angeschlossen ist. Ein *Stratum x* Server dagegen synchronisiert sich nur mit einem oder mehreren *Stratum x-1* Servern und ist daher ungenauer. Als genaue Zeitgeber für *Stratum 1* werden in der Regel GPS-Empfänger mit *Pulse Per Second (PPS)*-Signal, DCF-Empfänger (empfängt das DCF77 Zeitsignal aus Frankfurt) oder MSF-Empfänger (DCF Äquivalent in Großbritannien) verwendet. Nur wenige *Stratum 1* Serverbetreiber können sich noch präzisere Zeitgeber wie zum Beispiel Atomuhren leisten. Mittlerweile hat *NTP* in der Revision 3 weite Verbreitung gefunden und ein RFC-Draft für Revision 4 ist in Arbeit.

Wie das von NTP während der Synchronisation mit einem NTP Server generierte *driftfile*, welches den Gangunterschied in PPM zwischen der Systemuhr und einer hochpräzisen Referenzuhr beschreibt, genutzt werden kann, um die Präzision einer Quarzuhr zu verbessern, zeigen die Autoren in [KZM07]. Dabei beschreiben sie zwar die Korrektur der Uhren von PDAs, die sie für *MANET (Mobile Ad-hoc Network)* Experimente einsetzen, da aber die Uhren von PDAs nach dem gleichen Prinzip funktionieren wie die Uhren in aktuellen PCs sind die gezogenen Schlüsse und das Korrekturprinzip auf unser Szenario übertragbar.

In [KM07] haben die Autoren festgestellt, dass die Zeitsynchronisation über Mobilfunk mittels GPRS oder EDGE keine guten Resultate liefert. Sie entwickelten ein Verfahren auf Applikationsebene, das die Mobilfunkverbindung (GRPS oder EDGE) ausschließlich zur Zeitsynchronisation nutzt. Zur einmaligen Zeitsynchronisation benötigen sie 100 Messungen mit jeweils 2000 ms Abstand. Damit erreichen sie pro Synchronisationsvorgang eine Genauigkeit im Bereich von einigen Zehntel Millisekunden. Diese Genauigkeit ist zwar erstaunlich, da die Synchronisation aber laufend wiederholt werden müsste, wäre an Mobilfunkmessungen für unsere Traces nicht mehr zu denken. Daher ist dieses Verfahren zur Zeitsynchronisation für unsere Zwecke leider nicht brauchbar.

Kapitel 3

Logging-Methodik

Die Grundlage für eine aussagekräftige trace-basierte Simulation sind solide Traces. In diesem Kapitel werden wir daher erläutern, welche Verfahren wir einsetzen, um die Messdaten für unsere Traces zu erzeugen. Dabei wird in jedem Abschnitt ein neues Messproblem und unsere Lösung für selbiges vorgestellt.

Messungen in den von uns getesteten Mobilfunknetzen zeigen, dass zur Zeit alle deutschen Mobilfunkprovider IPv4 mit Adressübersetzung (sehr wahrscheinlich *NAT*) einsetzen. Bei O_2 z.B. erhält man als Mobilfunkkunde nach der Einwahl eine IP-Adresse aus dem Bereich 10.0.0.0/8 und ist vom Internet aus nicht direkt erreichbar. Hat man selbst aber über einen UDP-Port eine Verbindung ins Internet geöffnet, so ist man für diesen Kommunikationspartner auch über den gleichen UDP-Port erreichbar – das spricht eindeutig für eine NAT-Lösung seitens des Providers. Daher ist momentan eine direkte Kommunikation zwischen zwei Mobilfunknutzern nicht möglich. Vermutlich könnte man als Workaround *Holepunching* benutzen, ein Verfahren bei dem jeder Client dafür sorgt, dass ein Adressübersetzungseintrag in dem *NAT*-Router seines Providers zum Kommunikationspartner vorliegt damit jener ihn erreichen kann. Hierfür würde man aber sehr wahrscheinlich einen Rendezvous Server und je nach verwendetem Protokoll und NAT unterschiedliche Vorgehensweisen benötigen.

Um die Komplexität des *Holepunchings* zu vermeiden und um die Messungen nur von einem Mobilfunkgerät abhängig zu machen, haben wir uns dazu entschieden, die Messungen zwischen einem mobilen Endgerät und einem im Universitätsnetz betriebenen Server durchzuführen.

Trotzdem lassen sich mit unseren Messwerten auch gute Abschätzungen für die direkte Kommunikation von Mobilfunkgeräten – falls die Mobilfunkprovider sie irgendwann zulassen – liefern. So zeigen die Autoren in [FJJ+01], dass sich die Latenz $l(A,C)$ zwischen den Mobilgeräten A und C durch bekannte Latenzen von A zu Server B und Server B zu C wie folgt abschätzen lässt:

$$|l(A,B) - l(B,C)| \leq l(A,C) \leq l(A,B) + l(B,C) \tag{3.1}$$

Analog dazu lässt sich die Gleichung natürlich auch für die Latenz von C nach A aufstellen.

Für die Bandbreite können wir unter der Annahme, dass die limitierenden Faktoren die Luftschnittstellen und nicht die Backbones der Provider sind, folgende Abschätzung für die Bandbreite für Übertragungen von A zu C ($bw(A,C)$) mit bekannten Bandbreiten von A nach B und von B nach C machen:

$$bw(A,C) = \min\{bw(A,B), bw(B,C)\} \tag{3.2}$$

3.1 Latenzmessung

Da bei Netzwerkverbindungen über Mobilfunk davon ausgegangen werden muss, dass die Latenzen für den Upstream und Downstream nicht symmetrisch sind, reicht eine Analyse der Round-Trip Zeiten (*RTTs*) zur Bestimmung der Netzwerklatenz nicht aus. Stattdessen müssen wir die Latenzen für den Upstream und den Downstream getrennt messen. Das gängige Verfahren zur richtungsgebundenen Latenzmessung sieht wie folgt aus:

1. Der Sender erstellt ein Messpaket, in dem er die Absendezeit (Sendezeitstempel) möglichst exakt speichert.
2. Der Empfänger merkt sich zu jedem Messpaket, das ihn erreicht, die Sende- und Empfangszeitstempel.
3. Der Empfänger kann nun die Latenz für die Senderichtung Sender nach Empfänger bestimmen:

$$Latenz = Empfangszeitstempel - Sendezeitstempel$$

Um die Latenzen in beiden Richtungen zu messen, müssen die Messungen in beide Richtungen durchgeführt werden.

Der Aufbau unseres Latenzmesspaketes ist denkbar einfach. Wir verwenden UDP Pakete, die als Nutzlast eine Sequenznummer als Integer (4 Byte) und den Sendezeitstempel als Linux Epoche in Nanosekunden (8 Byte) enthalten. Insgesamt ist ein solches Latenzmesspaket 40 Byte lang, da zu den 12 Byte Nutzdaten noch 28 Byte für den IP- und UDP-Header addiert werden müssen.

Zusätzlich zu diesen Latenzmesspaketen minimaler Länge bestimmen wir auch noch die Latenz des jeweils ersten Bandbreitenmesspakets, das in der Regel wesentlich größer als 40 Byte ist (vergleiche Abschnitt 3.3).

Dieses Messverfahren benötigt synchrone Uhren im Client und Server, da sich jeder Zeitfehler zwischen den beiden Uhren direkt in den Messergebnissen niederschlägt. Während sich der Latenzmess-

wert in der einen Richtung durch den Uhrzeitsynchronisationsfehler verringert, wird er in der anderen Richtung in gleicher Weise erhöht. Daher widmet sich der nächste Abschnitt ausführlich dem Thema Zeitsynchronisation und erklärt wie wir dieses Problem gelöst haben.

3.2 Zeitsynchronisation

Wie in 2.3 beschrieben wurde NTP zur Zeitsynchronisation in Netzwerken entwickelt. Dieses Verfahren liefert aber nur dann zufriedenstellende Ergebnisse, wenn die Latenzen im Upstream und Downstream symmetrisch sind, und genau das ist in Mobilfunknetzen selten der Fall. Da NTP die Formel (2.3) zur Schätzung des Delay verwendet, welche den Delay auf die Hälfte einer RTT schätzt, macht NTP bei asymmetrischer Up- und Downloadlatenz folgenden Fehler:

$$\begin{aligned} error &= \text{delay} - \text{download latency} \\ &= \frac{RTT}{2} - \text{download latency} \\ &= \frac{\text{download latency} + \text{upload latency} - 2 \cdot \text{download latency}}{2} \\ &= \frac{\text{upload latency} - \text{download latency}}{2} \end{aligned} \tag{3.3}$$

Der Fehler ist also die Hälfte der Asymmetrie der Latenzen. Daher ist die direkte Nutzung von NTP zur Synchronisation von Client und Server für unsere Messungen nicht möglich, da die Synchronisation über Mobilfunk erfolgen müsste. Die zusätzlichen Request/Reply Pakete könnten außerdem unsere Messwerte verfälschen.

3.2.1 NTP mit *Offline Skew Correction*

Unsere erste Idee zur Lösung des Zeitsynchronisationsproblems unserer Messsysteme bestand darin, den in [KZM07] vorgestellten Ansatz zu verfolgen. Dabei werden die Uhren der Versuchsrechner über einen längeren Zeitraum (mehrere Stunden) mit Hilfe von NTP synchronisiert. Danach verwendet man NTP offline, das heißt ohne die Verwendung von Zeitservern, um mit Hilfe des erstellten *driftfile* den Gangunterschied der Uhren weitestgehend zu kompensieren. Mit diesem Verfahren haben die Autoren Experimente durchgeführt, bei denen nach 5 Stunden der Offset der Uhren im Bereich von 35 ms lag. Während der Versuchsdauer, also nach der initialen Synchronisationsphase von mehreren Stunden, haben dabei alle PDAs ausschließlich die Offline Skew Correction benutzt.

Es war also zu erwarten, dass wir mit unserem Messaufbau, der, wie in Kapitel 4 ausführlich beschrieben wird, aus einem an das Universitätsnetz angeschlossenen Server und einem Notebook mit UMTS

Modem als mobilem Client besteht, noch bessere Synchronisationswerte erhalten würden. Schließlich verfügt der Server über ausreichend Bandbreitenreserve und ist mit niedriger Latenz an das Internet angeschlossen. Daher kann er sich während der gesamten Versuchsdurchführung mit NTP Servern synchronisieren, ohne unsere Messungen signifikant zu beeinflussen. Lediglich der Client muss die Offline Skew Correction benutzen.

Es zeigte sich aber, dass aktuelle PC Uhren – auch sogenannte *Real Time Clocks* – zu ungenau sind. So haben wir bei Testmessungen über jeweils 4 Stunden Zeitunterschiede von mehreren hundert Millisekunden zwischen dem Offline Client und dem ständig mit NTP-Zeitservern verbundenen Client (unserem Server) festgestellt. Dies dürfte vor allem an den relativ großen Temperaturschwankungen des Laptops während der Messfahrten liegen. Vielleicht spielen aber auch noch andere Faktoren wie zum Beispiel die Stromsparfunktionen aktueller CPUs (in diesem Fall ein Core2Duo) eine Rolle, die zu sehr unterschiedlichen Lastzuständen führen. Diese Schwankungen in der Stromversorgung können auch die Versorgungsspannung des Quarzes beeinflussen, was wiederum eine Änderung der Frequenz bewirkt. Bei Latenzen von teilweise unter 100 ms in UMTS Netzwerken ist der oben beschriebene Zeitunterschied zwischen den Messsystemen jedoch unzureichend.

3.2.2 GPS mit PPS

Als weitere Möglichkeit der Zeitsynchronisation der beiden Rechner empfahl es sich nun sowohl den Client als auch den Server jeweils mit einer hochgenauen Referenzuhr zu verbinden und sie für das gesamte Experiment jeweils gegen diese Uhren zu synchronisieren. Gewöhnliche GPS Empfänger können aus den Satellitendaten zwar einen auf wenige Microsekunden genauen Zeitstempel errechnen, sind aber nicht dazu in der Lage diesen auch mit hoher Präzision an einen PC weiterzureichen. Hauptgrund hierfür ist die geringe Bandbreite mit der die Geräte Daten, in der Regel in Form von standartisierten NMEA Datensätzen, an den PC liefern. Normalerweise stehen hier nur 4800 Baud zur Verfügung, was zu erheblichen Übertragungsverzögerungen führt. Zusätzlich kann auch die Schnittstelle, die zur Verbindung mit dem PC genutzt wird, die Latenz (von der Errechnung des Zeitstempels bis zur Ankunft im PC) erheblich steigern und zusätzlich eine hohe Varianz für die Latenz bedeuten. Eine USB-Verbindung z.B. sorgt unter anderem durch die Komplexität des Protokolls für Latenzen von ca. 600ms mit mehreren 100 ms Varianz. Ähnlich verhält es sich mit einer Bluetooth GPS Maus – die meisten Bluetooth PC-Empfänger werden per USB angeschlossen. Auch seriell angeschlossene gewöhnliche GPS-Empfänger leiden unter ähnlichen Problemen. Die Latenzen liegen hier meist auch zwischen 200 ms und 600 ms (je nach Gerät unterschiedlich), allerdings ist hier die Varianz durch den eigenen Hardware Interrupt der seriellen Ports meist deutlich geringer. Aber auch diese Anschlussvariante alleine würde nur für eine Synchronisationsgenauigkeit von ca. 200 ms genügen.

Erheblich verbessern lässt sich die Genauigkeit mit GPS Empfängern mit *Pulse Per Second (PPS)* Signal, die exakt am Anfang jeder Sekunde einen Impuls an den PC senden. Erfolgt der Anschluss nun derart, dass das PPS Signal am PC einen Hardware Interrupt auslöst (z.B. an einem seriellen Port), so kann der PC sehr genau ermitteln wann eine Sekunde anfängt. Als Standard hat sich durchgesetzt, dass das PPS Signal den Anfang der Sekunde beschreibt, deren Zeitstempel im nächsten NMEA Datensatz verwendet wird. Durch diese Art der Synchronisation sind laut NTP Handbuch eine Genauigkeit von 50 μs und eine Stabilität von weniger als 0.1 PPM auf einem Linux Pentium PC erreichbar.

3.2.3 Garmin OEM GPS

In den NTP Mailinglisten wird zum Betrieb von preiswerten Stratum 1 Servern sehr häufig ein Garmin OEM GPS-Empfänger mit PPS Signal, der *Garmin OEM 18 LVC* empfohlen. Dieses Gerät wird dort allgemein als preiswert, empfangsstark und zuverlässig beschrieben. Leider hat Garmin dieses Gerät durch den Nachfolger, den *Garmin OEM 18x LVC* abgelöst. Das Studium der Datenblätter und eine Nachfrage bei Garmin zeigen jedoch, dass der *Garmin OEM 18x LVC* mit der gleichen Anschlussschaltung wie der *Garmin OEM 18 LVC* funktioniert und man lediglich die Empfangsleistung verbessert hat.

In der *Garmin OEM 18x* Serie gibt es mehrere Geräte, aber nur die *LVC* Version bietet ein PPS Signal. Die eigentlich viel praktischere Variante *Garmin OEM 18x PC* bietet zwar einen seriellen Anschluss, aber dort liegt kein PPS Signal an. Die *18x LVC* Variante wird dagegen mit einem 5m Anschlusskabel mit offenem Kabelende geliefert. Die passende Beschaltung für den Anschluß an einen PC liegt in der Hand des Käufers. Aus den zahlreichen im Internet verteilten Schaltplänen (meist noch für den *18 LVC*) und den Schaltplänen (ohne PPS Signalbeschaltung) des technischen Handbuchs von Garmin zum *Garmin OEM 18x LVC* [GI08] haben wir den im Schaltplan (Abbildung 3.1) dargestellten Anschlussplan mit Beschaltung für das PPS Signal abgeleitet. Dieser wurde auf einer Streifenrasterplatine nach dem Boardplan (siehe Abbildung 3.2) aufgebaut.

Das Anschlussboard bezieht die Versorgungsspannung aus einem USB-Anschluss, der jedoch nicht zur Datenübertragung genutzt wird, und speist damit sowohl die LEDs, als auch den Garmin GPS Empfänger. Dabei benötigen die beiden low current LEDs (vergleiche Teileliste in Tabelle 3.1) jeweils 2 mA und der GPS-Empfänger laut Datenblatt (siehe Tabelle 3.2) 90 mA. Der gesamte GPS-Empfänger-Aufbau benötigt also weniger als 100 mA und liegt damit deutlich unter den 500 mA, die ein USB-Port nach Spezifikation zur Verfügung stellen können muss. Die *NMEA* Daten und das *PPS* Signal werden an einem seriellen D-Sub 9 Anschluss zur Verfügung gestellt, der sich direkt an einen seriellen Port am PC anschließen lässt. Abbildung 3.3 zeigt den fertig aufgebauten *Garmin OEM 18x LVC* GPS Empfänger.

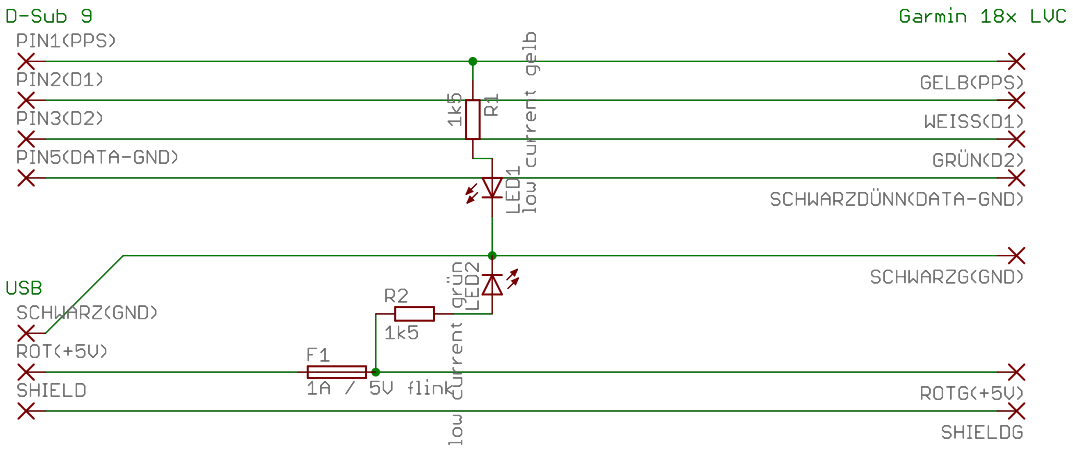

Abbildung 3.1: Schaltplan des Garmin GPS 18x LVC Anschlussboards.

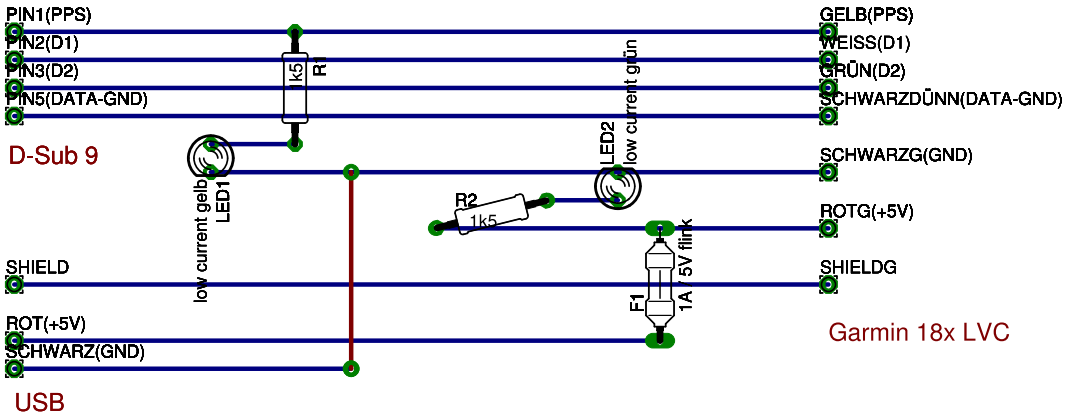

Abbildung 3.2: Boardplan des Garmin GPS 18x LVC Anschlussboards zum Aufbau auf einer Streifenrasterplatine.

Als Software zur Ansteuerung des *18x LVC* haben wir uns für *gpsd*, einen Interface Dämon für GPS Receiver, entschieden. Dieser Daemon wertet zum einen das PPS Signal aus und leitet die Informationen über mehrere *Shared Memory Devices* an den NTP Daemon *ntpd* weiter. Zum anderen bietet er zusätzlich die Möglichkeit, über eine TCP Verbindung sowohl die NMEA Rohdaten, als auch schon aufbereitete Positionsdaten abzufragen.

3.2.3.1 Konfiguration des Garmin OEM GPS

Die Übertragungsrate lässt sich beim *Garmin 18x LVC* von dem Defaultwert von 4800 Baud auf 9600 oder 19200 Baud umkonfigurieren. Dabei entsprechen 19200 Baud 1920 übertragbaren Zeichen pro Sekunde. Zusätzlich kann man frei konfigurieren, welche NMEA Datensätze aus Tabelle 3.3 man empfangen möchte.

Abbildung 3.3: *Garmin OEM 18x LVC* GPS Empfänger mit selbstgefertigten Anschlüssen.

Tabelle 3.1: Teileliste zum Anschluss des Garmin 18x LVC Empfängers.

ANZAHL	BESCHREIBUNG
1	D-Sub Buchse
$\frac{1}{2}$	USB-Kabel mit 2 x USB-Stecker Typ A
2	Widerstand 1,5 kΩ - abgestimmt auf die zwei Low Current LEDs
1	Sicherung 1A 5V flink
1	Low Current LED 3mm gelb $I_F = 2mA$ $U_F = 2.4V$
1	Low Current LED 3mm grün $I_F = 2mA$ $U_F = 2.4V$
1	Streifenrasterplatine ca. 8,0 x 4,5 cm
1	Gehäuse ca. 8,5 x 5,0 cm
70cm	4-adriges Kabel

Gpsd nutzt den *GPRMC* Datensatz zur Synchronisation des PPS Signals mit dem GPS Zeitstempel, daher müssen wir auf jeden Fall den GPRMC Datensatz in der Konfiguration des *Garmin 18x LVC* zur Ausgabe auswählen.

Ferner interessieren uns die aktuellen Positions- und Bewegungsdaten (*GPGGA*) und deren Fehlerabschätzungen (*GPGSA*). Außerdem würden wir gerne wissen, welche und wieviele Satelliten der Empfänger gerade empfängt. Diese Daten liefert uns der *GPGSV* Datensatz.

Nicht abstellbar ist die Übertragung des *PGRMT* Datensatzes, der einmal pro Minute Informationen über den Status des GPS Sensors liefert.

Durch unsere Datensatzauswahl von GPRMC, GPGGA, GPGSA, GPGSV (letzterer wird maximal 3 mal übertragen, weil der GPS-Sensor 12 Satelliten gleichzeitig beobachten kann und ein Datensatz

Tabelle 3.2: Technische Daten (Auszug) des *Garmin OEM 18x LVC*.

Größe	61mm Durchmesser, 19,5mm Höhe
Gewicht	160g
Gehäuse	Polycarbonate, wasserdicht nach IEC 60529 IPX7 level (30 Minuten bei 1m Tiefe)
Kabellänge	5m
Eingangsspannung	4.0-5.5 Vdc
Eingangsstrom	90mA @ 5.0 Vdc
CMOS Serial Output Level	0 Vdc to Vin, between 4 and 5.5 Vdc (Asynchronous Serial, TIA-232-F (RS-232) Compatible Polarity)
GPS Empfindlichkeit	-185 dBw minimum
Betriebstemperatur	$-30\,^{\circ}\mathrm{C}$ bis $+80\,^{\circ}\mathrm{C}$
Acquisition Times	Recquisition <2s, Hot ca. 1s, Warm ca. 38s, Cold ca. 45s
Updaterate	1 pro Sekunde
Genauigkeit	**GPS Standard Positioning Service (SPS)** Position: < 15 meters, 95% typical Velocity: 0.1 knot RMS steady state **WAAS** Position: < 3 meters, 95% typical Velocity: 0.1 knot RMS steady state **Measurement Pulse Output Time** ± 1 microsecond at rising edge of the pulse
NMEA Sätze	GPRMC, GPGGA, GPGSA, GPGSV, PGRME, GPGLL, GPVTG, PGRMV, PGRMF, PGRMB, PGRMM, PGRMT

jeweils nur Daten für 4 Satelliten beinhalten kann) und PGRMT ergeben sich entsprechend Tabelle 3.3 maximal $74 + 82 + 66 + 3 \cdot 70 + 50 = 482$ zu übertragende Zeichen pro Sekunde, also deutlich unter den maximal möglichen 1920 Zeichen pro Sekunde.

Die gesamte Konfiguration, die auch die Einstellung der Länge des PPS Pulses beinhaltet, lässt sich komfortabel über das Garmin Konfigurationsprogramm `SNSRXCFG_240.exe` durchführen. Mit Hilfe von *wine* [Win], das es ermöglicht Windows Programme unter Linux auszuführen, können wir das Garmin Konfigurationsprogramm auch auf unseren Linux basierten Messsystemen einsetzen. Abbildung 3.4 zeigt die Konfiguration, die für beide GPS Empfänger verwendet wurde.

Da die meisten Nutzer der *Garmin 18 LVC* und *Garmin 18x LVC* GPS-Empfänger diese nur zur Zeitsynchronisation nutzen, wird in den ntp-Mailinglisten dazu geraten, nur den GPRMC Datensatz abzufragen und alle anderen abzustellen. Wir wollen aber den hochempfindlichen Empfänger auch zur Positionsbestimmung nutzen und brauchen daher die weiteren Datensätze ebenfalls.

Tabelle 3.3: Verfügbare *NMEA* Datensätze und deren Größe.

DATENSATZ	MAXIMALE ZEICHENANZAHL
GPRMC	74
GPGGA	82
GPGSA	66
GPGSV (nur PC und LVC Version)	70
PGRME	35
GPGLL	44
GPVTG (18x-5Hz Version)	42
PGRMV	32
PGRMF	82
PGRMB	40
PGRMT (Einmal pro Minute)	50

Theoretisch dürften die zusätzlichen NMEA Datensätze keinen Einfluss auf die Zeitsynchronisation haben, da per Konvention das PPS Signal den Anfang der Sekunde definiert, die im nächsten NMEA Datensatz, der einen Zeitstempel enthält, gesendet wird. In der Praxis zeigt sich jedoch, dass gpsd (bis einschließlich Vesion 2.39) hiermit Probleme haben kann. Je nach Anzahl der gerade genutzten Satelliten braucht der GPS-Empfänger mehr oder weniger Zeit um Berechnungen (z.B. der aktuellen Position) durchzuführen. Da aber die NMEA Datensätze mit Zeitstempel alle auch weitere berechnete Daten enthalten, können diese nur verzögert an den PC übertragen werden.

Debug-Logs von gpsd 2.39 zeigen, dass diese Verzögerung bei ca. 5-6 empfangenen und zur Positionsberechnung benutzten Satelliten circa 450 ms beträgt. Steigt die Anzahl dieser zur Positionsberechnung benutzten Satelliten durch gute Empfangsbedingungen auf 8 oder mehr an, so liegt sie im Schnitt bei ca. 500 ms (bei einer geschätzten Varianz von 30ms). Leider hat gpsd in der weit verbreiteten Version 2.39 einen Bug. Die Konstante `PUT_MAX_OFFSET` in `ntpshm.c` sorgt dafür, dass gpsd alle PPS Signale verwirft, auf die nicht spätestens nach 500 ms der GPRMC Datensatz gefolgt ist. Ist wie bei den meisten Nutzern nur der GPRMC Datensatz ausgewählt, gibt es durch die Konstante kein Problem, da der GPRMC Datensatz dann vor den 500 ms den PC erreicht. Da wir jedoch mehr Datensätze abfragen, erreicht der GPRMC Datensatz häufig erst nach mehr als 500 ms den PC und das PPS Signal wird verworfen – dadurch ist eine Synchronisation trotz gutem Satellitenempfang nicht möglich.

In den aktuellen gpsd Versionen (ab Version 2.90), die nach Erstellung der gpsd Loggingkomponente des Loggingframeworks erschienen, wurde diese Konstante auf 1000 ms erhöht. Da sich in der neuen Version aber auch das Abfrageprotokoll grundlegend geändert hat und damit ein komplettes Redesign der gpsd Loggingkomponente erforderlich gewesen wäre, entschieden wir uns, vorerst weiterhin die seit mehr als einem Jahr erprobte *gpsd* Version 2.39 mit manuell auf 800 ms geänderter Konstante zu verwenden.

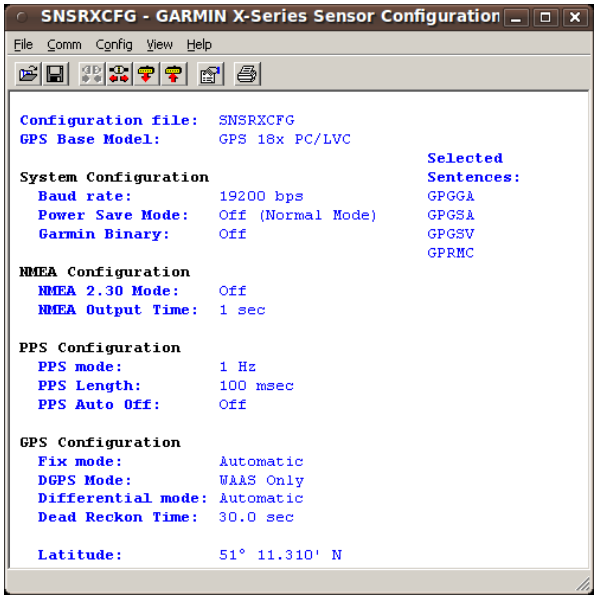

Abbildung 3.4: Konfigurationseinstellungen der verwendeten *Garmin OEM 18x LVC* GPS Empfänger.

Im weiteren Betrieb zeigten sich aber weiterhin sporadische Synchronisationsprobleme, deren Hauptursache nicht diese Konstante, sondern die Auswahl der falschen Signalflanke des PPS Signals ist. Dadurch ist es möglich, dass sich die Uhren der beiden Messsysteme um 100 ms, die Länge des PPS Pulses, unterscheiden. Mehrfache Neustarts des gpsd Dämons schafften hier zwar kurzfristig Abhilfe, allerdings dauerte es nach jedem Neustart mehrere Minuten, bis der Erfolg oder Misserfolg der Synchronisation erkennbar war. Zusätzlich konnte es sogar vorkommen, dass sich gpsd zwischenzeitlich umentschied und von der Synchronisation auf die steigende Flanke zur Synchronisation auf die fallende Flanke wechselte oder umgekehrt. Da ein Backport der Bugfixes aus der aktuellsten Version 2.95 auf die von uns verwendete Version 2.39 nach Rücksprache mit den Autoren extrem kompliziert wäre, wir die Synchronisationsprobleme von gpsd 2.39 jedoch vermeiden wollten, setzten wir in den Messsystemen schließlich doch gpsd in der Version 2.95 ein. Die GPS-Loggingkomponenten wurden daher umgeschrieben und sind jetzt in der Lage, Daten von gpsd Dämonen der Version 2.39 und 2.95 abzufragen und zu verarbeiten.

3.3 Bandbreitenmessung

Die Messung der verfügbaren Bandbreite war die wohl schwierigste Aufgabe bei der Erstellung des Logging Frameworks. Denn Bandbreitenmessungen benötigen zum einen eine je nach Verfahren nicht zu vernachlässigende Bandbreite und zum anderen pro Messung mehr oder weniger viel Zeit. Da wir die Messungen aber unter ständiger Bewegung durchführen, darf eine Messreihe nur relativ kurze

Zeit dauern. Zum einen kann sich der cross traffic schnell ändern, wenn man z.B. eine stark belebte Nahverkehrshaltestelle passiert erwarten wir eher mehr cross traffic als auf einer kaum benutzten Straße. Zum anderen können sich die zugrundeliegenden Mobilfunkcharakteristiken durch den Positionswechsel zum Beispiel durch Abschattungen, Wechsel der Basisstation oder Handover in der Netzwerktechnik dramatisch ändern. Schlussendlich soll die Zeit pro Messung minimiert werden, um sie immer noch einem möglichst kleinen Gebiet zuordnen zu können.

Wir wünschen uns also ein Messverfahren für die verfügbare Bandbreite, das:

wenig Bandbreite benötigt, da wir zum einen zeitweise wenig Bandbreite zur Verfügung haben (z.B. dort wo nur GPRS zur Verfügung steht), zum anderen weil bei allen verfügbaren Privatkundenflatrates nach spätestens 5 GB Datenvolumen eine Drosselung auf GPRS Geschwindigkeit erfolgt.

wenig Zeit für die Messungen benötigt, da sich das Mobilgerät in ständiger Bewegung befindet.

genaue Messergebnisse liefert, die die verfügbare Bandbreite möglichst exakt wiedergeben.

das Übertragungsmedium möglichst wenig belastet um den *cross traffic*, und damit die Messung der verfügbaren Bandbreite, nicht zu beeinflussen.

Nach der Lektüre der verwandten Arbeiten (vergleiche Abschnitt 2.2) schien das Probe Gap Model den besten Kompromiss bezüglich der gerade genannten Kriterien zu bieten, daher haben wir zunächst dieses Verfahren implementiert und im folgenden Abschnitt untersucht.

3.3.1 Probe Gap Model für Mobilfunkmessungen

Das erste Problem, das das PGM Verfahren aus Abschnitt 2.2.1 mit sich bringt ist, dass wir die Kapazitäten der Netzwerkpfade (Hin- und Rückrichtung) kennen müssen, um mit Formel (2.2) die verfügbare Bandbreite zu errechnen. Wir gehen davon aus, dass die Kapazität durch die Luftschnittstelle zwischen Basisstation und UMTS-Modem begrenzt wird. Diese Annahme müssen wir treffen, weil wir keinen Einblick in das Innere der Netzwerke haben und Datenübertragungen über Kabel- oder Lichtwellenleiter weit mehr Kapazitäten bieten als Funkübertragungen. Leider stellt uns das UMTS-Modem nur eine sehr grobe Klassifizierung der Kapazität bereit, da wir lediglich abfragen können welcher Datenmodus (GSM, GPRS, EDGE, UMTS, HSDPA) gerade verwendet wird (vergleiche Tabelle 4.1). Da aber zum Beispiel HSDPA sowohl 1,8 MBit, als auch 3,6 oder 7,2 MBit/s bedeuten kann, können wir die Kapazität nur schätzen.

Wir haben uns dazu entschieden, die verfügbare Bandbreite lieber zu unterschätzen als zu überschätzen und wählten bei HSDPA als Kapazität daher 1,8 MBit.

Ferner können wir, da unser Loggingframework auf der Applikationsebene des OSI-Modells arbeitet, keinen zeitlichen Einfluss auf den Versand der Pakete nehmen. Wir können sie erstellen und an das Betriebssystem weiterleiten, das sie dann dem Treiber zum Versenden übergibt. Da der Aufruf der Sendefunktion nicht blockiert bis das Paket wirklich die Netzwerkhardware passiert hat, können wir Δ_{in} nicht aus den Sendezeitstempeln der back-to-back verschickten Pakete ermitteln. Der Zeitstempel für das erste Paket ist zwar noch ziemlich exakt (vorausgesetzt wir haben nicht selbst den Sendepuffer für das UMTS-Modem gefüllt). Aber der Zeitstempel, den wir auf Applikationsebene in das zweite Paket schreiben, hat in der Regel nichts mit dem Sendezeitpunkt des Paketes zu tun, da er gegenüber dem Zeitstempel des ersten Paketes nur um die Erstellungszeit, nicht aber die Versendezeit eines solchen Paketes abweicht. Wir können Δ_{in} also nicht über die Formel

$$\Delta_{in} = \text{Sendezeitstempel 2. Paket} - \text{Sendezeitstempel 1. Paket}$$

bestimmen. Stattdessen müssen wir Δ_{in} mit Hilfe der Netzwerkkapazität C in Byte/s und der Paketlänge L wie folgt berechnen:

$$\Delta_{in} = \frac{L}{C} \tag{3.4}$$

Dabei müssen wir davon ausgehen, dass die Pakete vom Betriebssystem tatsächlich back-to-back verschickt werden. Deshalb bemühen wir uns die Testsysteme möglichst geringer Last auszusetzen.

Implementiert haben wir eine einfache Form des *PGM*, bei dem wir pro Sekunde und Senderichtung jeweils ein *Packet pair* zur Messung verwendet haben. Zur Überprüfung des Verfahrens haben wir stationäre Testmessungen durchgeführt, um sicherzustellen, dass Änderungen in den Messungen einzig auf cross traffic zurückzuführen sind.

Leider zeigten erste Testmessungen, bei denen mehr als 50% der Messungen eine negative verfügbare Bandbreite vorhersagten, dass mindestens eine der Annahmen für das PGM nicht erfüllt wurden. Eine Analyse der PGM zu Grunde liegenden Formel (2.2)

$$A = C \cdot \left(1 - \frac{\Delta_{out} - \Delta_{in}}{\Delta_{in}}\right)$$

zeigt, dass für $C > 0$ A genau dann negativ ist, wenn

$$\begin{aligned} & 0 > 1 - \tfrac{\Delta_{out} - \Delta_{in}}{\Delta_{in}} \\ \Leftrightarrow \quad & 0 > \Delta_{in} - (\Delta_{out} - \Delta_{in}) \\ \Leftrightarrow \quad & \Delta_{out} > 2 \cdot \Delta_{in} \end{aligned} \tag{3.5}$$

Das Verfahren liefert also negative Schätzungen, wenn die *packet dispersion* am Empfänger mehr als doppelt so groß ist als beim Sender.

3.4 Packettrain

Da das PGM Verfahren in Mobilfunknetzen nicht funktioniert und die PRM Verfahren (vergleiche Abschnitt 2.2.2) zu viel Zeit pro erhaltenem Messwert benötigen, haben wir ein *Packettrain* Verfahren entwickelt, das zum einen dafür sorgt, dass unsere Messungen durch kurze Messzeiten stark ortsbezogen sind, und zum anderen möglichst gute Bandbreitenschätzungen ermöglicht. Die Idee des Packettrains ist dabei nicht neu, wohl aber das Abstimmen des Packettrains auf möglichst kurze Messzeiten damit wir ortsbezogene Messwerte erhalten.

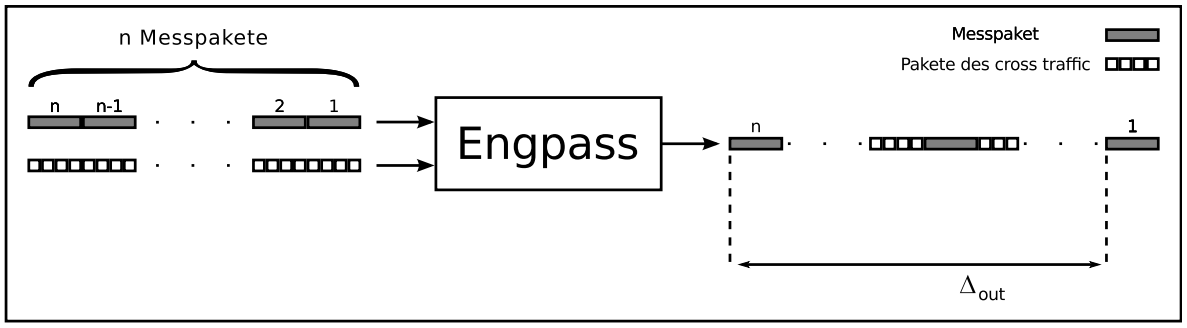

Abbildung 3.5: Funktionsweise unseres Packettrain Verfahrens.

Für jede Messung senden wir eine vorher festgelegte Anzahl n (mit $n \geq 2$) an Messpaketen gleicher Länge L back-to-back als UDP Pakete vom Sender zum Empfänger (vergleiche Abbildung 3.5).

Die verfügbare Bandbreite A für eine Packettrainmessung, bei der mindestens zwei der Messpakete ihr Ziel erreichten, berechnen wir mit der Formel:

$$A = \frac{(\text{Anzahl der erhaltenen Pakete} - 1) \cdot L}{\Delta_{out}} \tag{3.6}$$

Hierbei ist Δ_{out} der Zeitunterschied zwischen dem Eintreffen des letzten Bytes des ersten uns erreichenden Messpaketes eines Packettrains und des letzten Bytes des letzten uns erreichenden Messpaketes des gleichen Packettrains.

L ist die Paketlänge auf Netzwerkebene, sie enthält also auch die 28 Byte für die IP und UDP Header.

Formel (3.6) berücksichtigt insbesondere alle erdenklichen Paketverlustfälle, denn die für die Zeitspanne Δ_{out} verfügbare Bandbreite für Datentransfers in Packettrainrichtung setzt sich einzig aus der Datenmenge zusammen, die auch wirklich in dieser Zeit beim Empfänger eintrifft.

Auch bei diesem Verfahren können wir nur bedingt Einfluss darauf nehmen, dass die Pakete vom Betriebssystem tatsächlich back-to-back verschickt werden, daher haben wir bei der Erstellung des Messframeworks darauf geachtet, die Messsysteme möglichst geringer Last auszusetzen.

Da wir für unsere Traces ortsbezogene Messwerte benötigen, haben wir uns für eine Messung pro Messrichtung und Sekunde entschieden. Damit unsere UDP Messpakete anderen Netzwerkverkehr, vor allem TCP Ströme, nicht unnötig stören und dadurch die Messwerte signifikant verändern, versuchen wir außerdem maximal 50% der verfügbaren Bandbreite für unsere Messpakete zu nutzen.

3.4.1 Statische Packettraingröße

Da sich der Client bewegt und daher Änderungen bei der Übertragungstechnik auftreten können, müssen wir die Größe des Packettrains der Übertragungstechnik anpassen. Daher haben wir Messreihen mit konstanter Clientposition aber sich ändernden Parametern für den Packettrain durchgeführt, um die Parameter 'Anzahl der Pakete im Train (*#Pakete*)' und 'die Länge eines jeden Paketes (*Paketgröße*)' möglichst optimal zu der gerade benutzten Übertragungstechnik zu wählen.

Im folgenden sei Packettraingröße $=$ Anzahl der Pakete $\cdot$ Paketgröße.

Da das O_2 Netzwerk im Großraum Düsseldorf im Freien fast vollständig HSDPA beziehungsweise UMTS bietet und ansonsten bei jeglichen Messungen als Fallback immer GPRS und niemals EDGE (der EDGE Ausbau bei O_2 ist kaum vorhanden) zum Einsatz kam, haben wir für diese beiden Übertragungstechniken Messreihen mit variierenden *Packettraingrößen* gemacht.

Dabei wurden für HSDPA die Parameter wie folgt variiert:

- #Packets: 2,3,4,5,6,8,10,15,20,30
- Packetsize: 200,300,400,500,700,900,1100,1300,1464 Bytes

Maximal wurden also $30 \cdot 1464\text{Bytes} = 43920\text{Bytes}$ pro Packettrain pro Sekunde und Richtung verschickt. Viel mehr Kapazität steht im Upload auch nicht zur Verfügung, da UMTS maximal 48 kByte/s bietet.

In Tabelle 3.4 finden sich die maximal möglichen Datenraten, die das verwendete UMTS-Modem bei der jeweiligen Verbindungstechnik übertragen kann. Dabei sind die Einträge für GPRS und EDGE erklärungsbedürftig. GPRS und EDGE Geräte werden in Klassen eingeteilt. Class 10 heißt hierbei, dass das Gerät maximal 4 Zeitslots pro Sekunde zum Datenempfang und maximal 2 Zeitslots zum

Versenden nutzen kann. Insgesamt kann es aber maximal 5 Zeitslots pro Sekunde für das Senden und Empfangen nutzen.

Tabelle 3.4: Maximal nutzbare Übertragungsraten des Sierra Wireless MC8775 UMTS Modems.

ÜBERTRAGUNGSTECHNIK	DOWNLOAD		UPLOAD	
	kBit/s	kByte/s	kBit/s	kByte/s
GSM	14,4	1,8	9,6	1,2
GPRS Class 10 (4+2=5)	80,0	10,0	40,0	5,0
EDGE Class 10 (4+2=5)	216,0	127,0	108,0	13,5
UMTS bzw. WCDMA	384,0	48,0	384,0	48,0
HSDPA	7372,8	921,6	384,0	48,0

3.4.1.1 Parameterwahl für den UMTS Upload

Die erste statische Messreihe zur Bestimmung der Packettraingrößen für den UMTS Upload und HSDPA Download dauerte ca. 4 Stunden und wurde absichtlich mitten in der Nacht durchgeführt, um Störeinflüsse durch andere Nutzer zu minimieren. Zwischen den einzelnen Messungen wurde jeweils 60 Sekunden pausiert, damit der nächste Loggingrun auf jeden Fall mit leeren Netzwerkpuffern beginnen konnte.

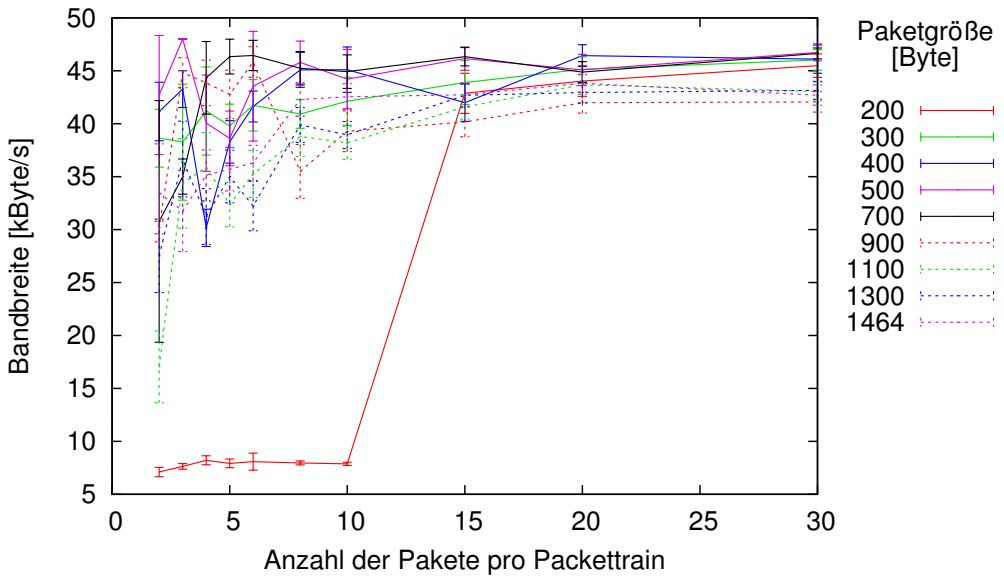

Abbildung 3.6: Statische Messungen verschiedener Packettraingrößen für den UMTS Upload mit Konfidenzintervallen nach Paketgröße.

In Abbildung 3.6 haben wir für verschiedene Paketgrößen die jeweils untersuchte Paketanzahl pro Packettrain auf der X-Achse über die geschätzte verfügbare Bandbreite inklusive 95% Konfidenzin-

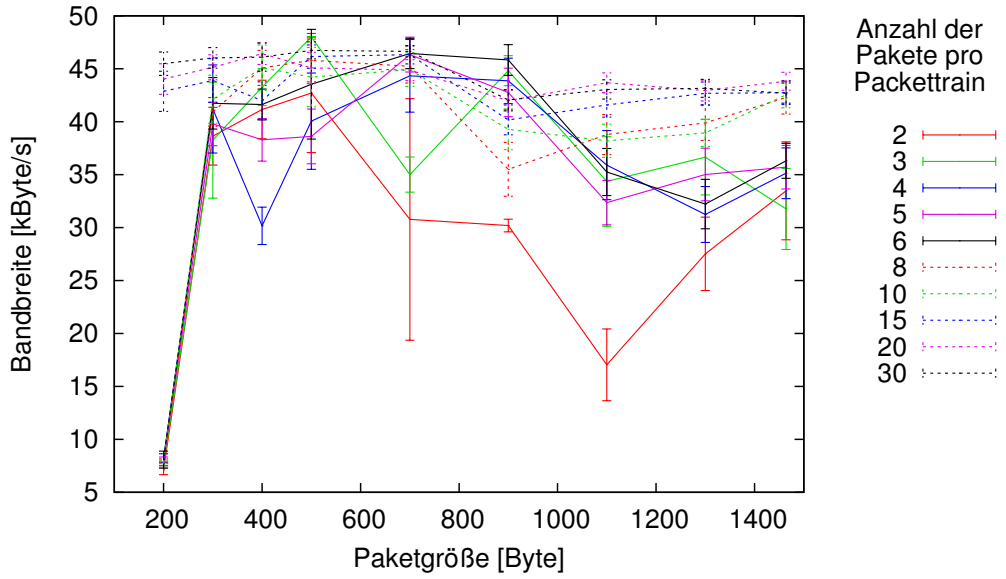

Abbildung 3.7: Statische Messungen verschiedener Packettraingrößen für den UMTS Upload mit Konfidenzintervallen nach Anzahl der Pakete.

tervall auf der Y-Achse für den UMTS Upload aufgetragen. Abbildung 3.7 zeigt die gleiche Datenbasis, jedoch wurde hier die verfügbare UMTS Upload Bandbreite verschiedener Paketgrößen über variierende Paketanzahl pro Packettrain abgebildet. Diese unterschiedlichen Darstellungsformen helfen bei der Analyse der Daten, da Häufungspunkte in der einen Abbildung in der anderen Abbildung gestreut abgebildet werden. So sind zum Beispiel die später noch genauer untersuchten Messausreißer der Packettrains mit den Parametern 2x200 bis 10x200 Byte in Abbildung 3.6 fein differenziert in der Nähe der X-Achse aufgetragen, wohingegen sie in Abbildung 3.7 einen dichten Häufungspunkt in der Nähe des Ursprungs ergeben.

Die Abbildungenen 3.6 und 3.7 zeigen zunächst, dass die geschätzte Uploadbandbreite bei ca. 40-45 kByte/s lag, was sehr nah an den theoretisch möglichen 48 kByte/s liegt. Wie nicht anders zu erwarten, wird die Bandbreitenschätzung genauer je mehr Daten übertragen werden, schließlich haben dann minimale Schwankungen in den Zeitstempeln geringere Auswirkungen auf die Schätzung. Besonders schlecht schneiden Packettrains mit sehr wenigen Paketen oder sehr kleinen Paketen ab. Da wir nicht unnötig Bandbreite verschwenden wollen, haben wir uns entschlossen nach Möglichkeit nicht mehr als 30% bis 50% der Bandbreite zu verwenden. Zum einen bieten aktuelle Flatrates nur ein bestimmtes bandbreitenunbeschränktes Freivolumen, zum anderen ist die Beschränkung verhältnismäßig unkritisch, da wir, wie man den Grafiken entnehmen kann, bei zu geringer Packettraingröße die verfügbare Bandbreite leicht unterschätzen. Eine Überschätzung wäre für eine spätere Simulation wesentlich kritischer.

Da bei unseren Messungen alle Packettrains mit Längen von 15 Paketen relativ gut abschneiden,

haben wir uns für die UMTS-Messungen (auch für reinen UMTS Download) für die Packettraingröße von 15 x 700 Bytes entschieden. Zum einen schneidet sie relativ weit am oberen Spektrum ab, zum anderen ist hier das Konfidenzintervall sehr klein.

Die einleitend schon erwähnten Messausreißer der Parameterkombinationen 2x200 bis 10x200 Byte pro Packettrain lassen sich durch eine starke Auslastung des UMTS-Netzwerkes zu Beginn unserer Messreihen erklären. Als erstes wurden von uns Packettrains mit 2x200 Byte vermessen. Anschließend wurde die Anzahl der Pakete pro Train im Rahmen der oben genannten Liste immer weiter erhöht, bis wir die Messreihe mit 30x200 Byte abgeschlossen hatten. Erst danach haben wir auf gleiche Weise Messungen für die weiteren Paketgrößen durchgeführt.

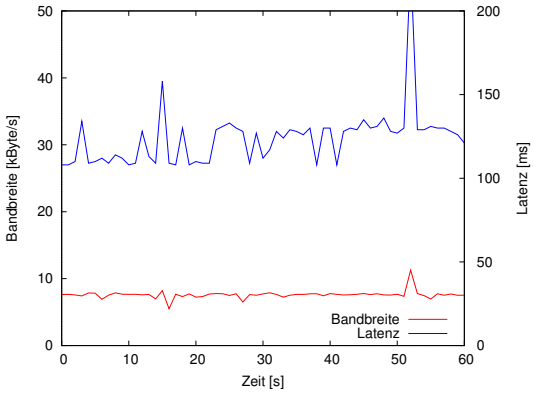

Abbildung 3.8: Messungen für den UMTS Upload mit dem 10x200 Byte Packettrain.

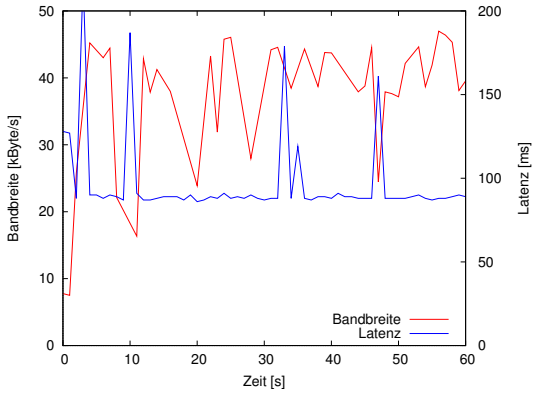

Abbildung 3.9: Messungen für den UMTS Upload mit dem 15x200 Byte Packettrain.

Betrachtet man die Messdaten für den 10x200 Byte Packettrain (Abbildung 3.8), so haben die Pakete dort im Durchschnitt eine Latenz von 123 ms und es wurde eine durchschnittliche verfügbare Bandbreite von 7,54 kByte/s gemessen. Die ersten beiden Messwerte für den anschließend vermessenen 15x200 Byte Packettrain decken sich sehr exakt mit diesen Werten (vergleiche Abbildung 3.9). Sie lieferten Messwerte von 7,8 kByte/s und 128ms beziehungsweise 7,5 kByte/s und 127ms. Der dritte Wert in der Messreihe zeigt jedoch schon eine Latenz von nur noch 88ms und eine Bandbreite von 26 kByte/s. Während die Latenz bei den darauf folgenden Messungen ziemlich stabil bei ca. 90 ms liegt, schwankt die gemessene Bandbreite die nächsten 10 Sekunden noch mehrfach zwischen 16 und 40 kByte/s um danach bis zum Ende der 15x200 Messreihe nur noch zwischen 37 und 44 kByte/s zu variieren. Dies stützt unsere Vermutung, dass der Upload in der für unsere Messreihe verwendeten UMTS-Zelle zu Beginn unserer Messungen stark ausgelastet war.

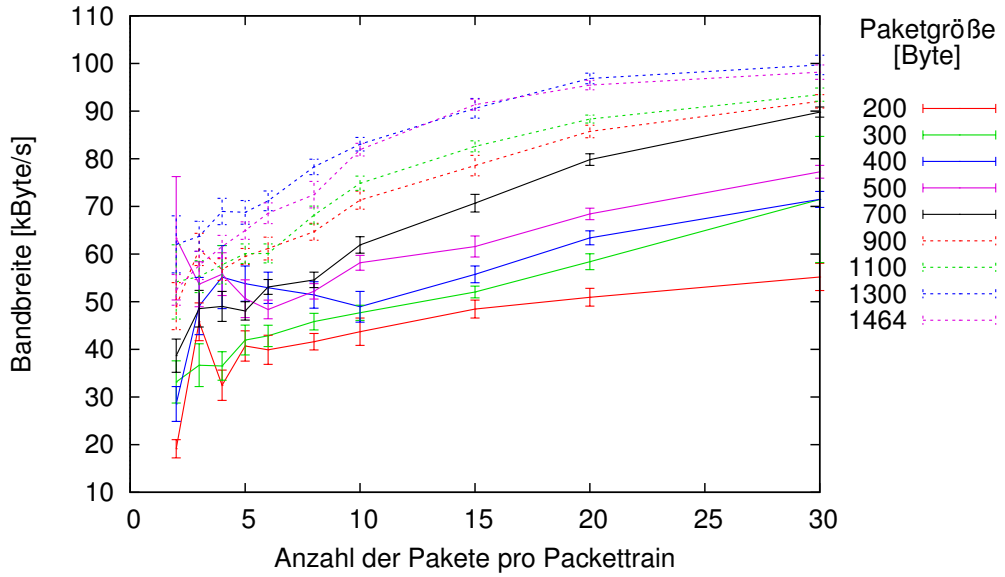

Abbildung 3.10: Statische Messungen verschiedener Packettraingrößen für den HSDPA Download mit Konfidenzintervallen nach Paketgröße.

3.4.1.2 Parameterwahl für den HSDPA Download

Die Abbildungen 3.10 und 3.11, analog zu den Abbildungen 3.6 und 3.7 erstellt, zeigen die Messungen für die Downloadrichtung, die zur gleichen Zeit durchgeführt wurden. Leider stellt uns das UMTS-Modem keine Möglichkeit zur Verfügung um herauszufinden, welche Maximalbandbreite HSDPA gerade anbietet. Das Modem unterstützt zwar den 7,2 MBit Modus, es werden aber auch teilweise nur 3,6 MBit oder 1,8 MBit vom Mobilfunknetz zur Verfügung gestellt. Die Messwerte zeigen jedoch, dass wir um den Faktor 2 unter der niedrigsten HSDPA-Datenrate liegen. Es zeigt sich aber auch, dass wir nicht zu sehr mit der verwendeten Bandbreite für die Messungen sparen dürfen, da die Bandbreitensschätzungen über alle Messreihen mit steigender Packettrainlänge steigen.

Einzelne Ausreißer in den Messdaten sind besonders bei relativ geringen Paketgrößen zu erkennen. Beim HSDPA Download haben wir uns auf eine Packettraingröße von 15x1300 Bytes als Kompromiss zwischen gutem Schätzwert und verwendeter Bandbreite geeinigt.

Bei reiner Bandbreitenmessung und einer Messung pro Sekunde würden wir bei reinem HSDPA Download und UMTS Upload also 15x700 + 15x1300 Bytes/s = 30000 Byte/s über das Mobilfunknetz schicken. Pro Stunde würden wir also ca. 103 MB Datenvolumen verbrauchen.

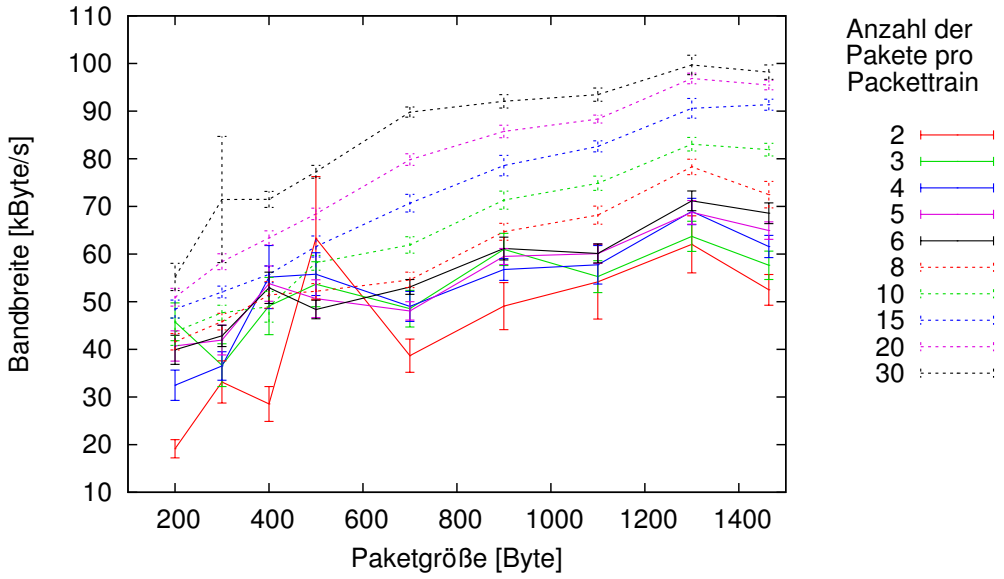

Abbildung 3.11: Statische Messungen verschiedener Packettraingrößen für den HSDPA Download mit Konfidenzintervallen nach Anzahl der Pakete.

3.4.1.3 Parameterwahl für die GPRS Packettrains

Die GPRS Testmessungen zur Packettraingröße zeigen, wie viel schwieriger die Bandbreitenschätzung bei den bei GPRS nur verhältnismäßig geringen verfügbaren Datenraten ist. Negativ wirkt sich hier vor allem die viel größere Konkurrenz der Datenübertragung mit Telefongesprächen und anderen GSM-basierten Datenübertragungen aus. UMTS stellt potentiell wesentlich mehr Bandbreite zur Verfügung und kann diese in viel feineren Abstufungen an die Teilnehmer verteilen. Außerdem kann die UMTS-Basisstation alle 10 ms Einfluss auf die Verteilung der Bandbreite nehmen, indem sie den Mobilgeräten andere *Channelizationcodes*, die *WCDMA* zur Bandspreizung einsetzt, zuteilt. So kann UMTS zum einen sehr schnell auf Bandbreitenanforderungen reagieren, und damit Latenzen gering halten, zum anderen steht bei gleicher Benutzeranzahl mehr Bandbreite pro Nutzer zur Verfügung. Folglich haben andere GPRS Nutzer oder GSM Telefongespräche wesentlich mehr negative Einflüsse auf die verfügbare Bandbreite in GPRS als dies bei UMTS der Fall ist. Schon alleine dadurch ergeben sich signifikante Schwankungen in den GPRS Testmessungen – obwohl auch diese zwischen 1:00 und 5:00 Uhr nachts durchgeführt wurden.

Erste Messungen zeigten, dass im Download die verfügbare Bandbreite bei ca 4 kByte/s und im Upload bei 1 bis 2 kByte/s lag. Da wir maximal die Hälfte der verfügbaren Bandbreite für die Messungen nutzen wollen (um *self-induced congestion* vorzubeugen), haben wir eine lange Messreihe (ca. 4 Stunden) mit weniger Parametervariationen für die Packettrains gemacht. Damit Schwankungen über die Zeit diesmal einen geringeren Einfluss auf die Messergebnisse ausüben, wurde jeweils nach

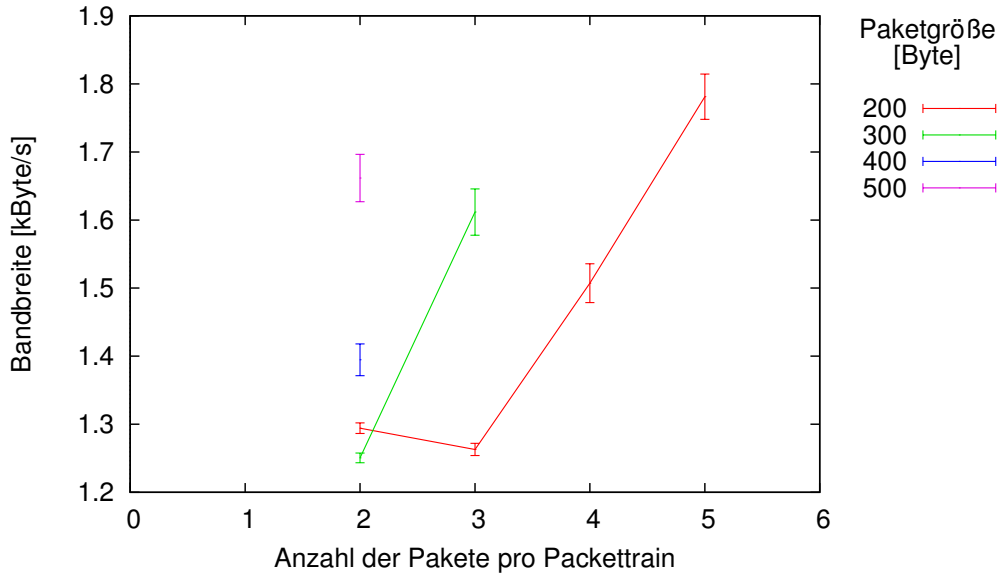

Abbildung 3.12: Statische Messungen verschiedener Packettraingrößen für den GPRS Upload mit Konfidenzintervallen nach Paketgröße.

60 Sekunden eine 30 sekündige Messpause gemacht um dann mit den nächsten Messparametern fortzufahren. So wurden die Messungen für die Packettraingrößen mehrfach und ineinander verschachtelt durchgeführt.

Im Upload haben wir maximal 1 kByte pro Messung verbrauchen wollen und haben daher die Packettrains mit 2x200, 3x200, 4x200, 5x200, 2x300, 3x300, 2x400 und 2x500 Byte untersucht, während gleichzeitig im Download alle Packettraingrößen bis 2000 Byte getestet wurden.

3.4.1.4 Parameterwahl für den GPRS Upload

Die Abbildungen 3.12 und 3.13 zeigen die Messergebnisse für die Uploadmessungen. Erstaunlich ist, dass die Konfidenzintervalle für geringe Packettraingrößen sehr klein ausfallen, allerdings wird mit ihnen die verfügbare Bandbreite massiv unterschätzt. Daher haben wir uns für den GPRS Upload für den Packettrain mit 5x200 Byte entschlossen.

3.4.1.5 Parameterwahl für den GPRS Download

Die GPRS Downloadmessungen (vgl. Abbildungen 3.14 und 3.15) zeigen, dass wir Aufgrund der höheren verfügbaren Bandbreite mehr Auswahl in der möglichen Packettraingröße haben. Letztend-

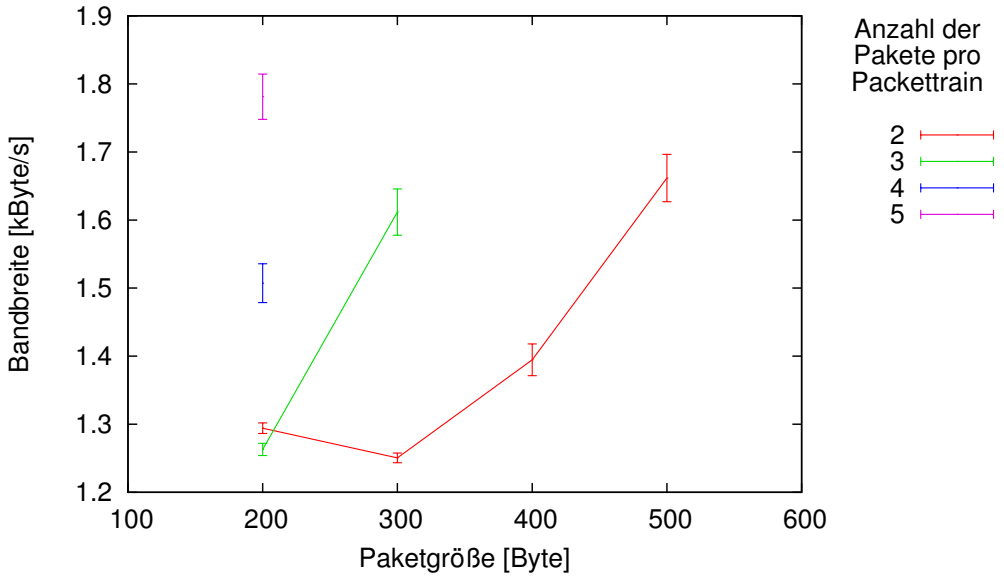

Abbildung 3.13: Statische Messungen verschiedener Packettraingrößen für den GPRS Upload mit Konfidenzintervallen nach Anzahl der Pakete.

lich haben wir uns wegen der relativ guten Bandbreitenschätzung (sowohl die Höhe als auch die Größe 95% Konfidenzintervall sind vielversprechend) im GPRS Download für einen Packettrain mit 6x300 Byte enschieden.

Da zum einen der Ausbau von EDGE im O_2 Netz wenig fortgeschritten ist, und wir zum anderen bei unseren Messfahrten überall entweder GPRS oder UMTS/HSDPA vorfanden, haben wir für GSM und EDGE kein Parametertuning bezüglich der Packettraingröße vorgenommen bzw. vornehmen können.

Aufgrund der sehr geringen Bandbreite die GSM bietet, haben wir hier den minimalen Packettrain mit 2x100 Byte gewählt.

Für EDGE haben wir im Up- und Download die Packettrains von GPRS übernommen.

Letztlich haben wir für unsere Bandbreitenmessungen also die Packettrainparameter aus Tabelle 3.5 per Parametertuning ausgewählt.

Da wir auch (zumindest bei den GPRS Messungen) Packettraingrößen betrachtet haben, die weit über der verfügbaren Bandbreite liegen [erkennbar an den stark steigenden Latenzen (self-induced congestion)], konnten wir auch zeigen, dass unsere Bandbreitenschätzung funktioniert. Denn dabei haben wir wirklich pro Sekunde so viele Daten übertragen wie möglich und haben nicht nur anhand von einigen 100ms Datenübertragung auf die verfügbare Bandbreite des Mediums pro Sekunde geschlossen.

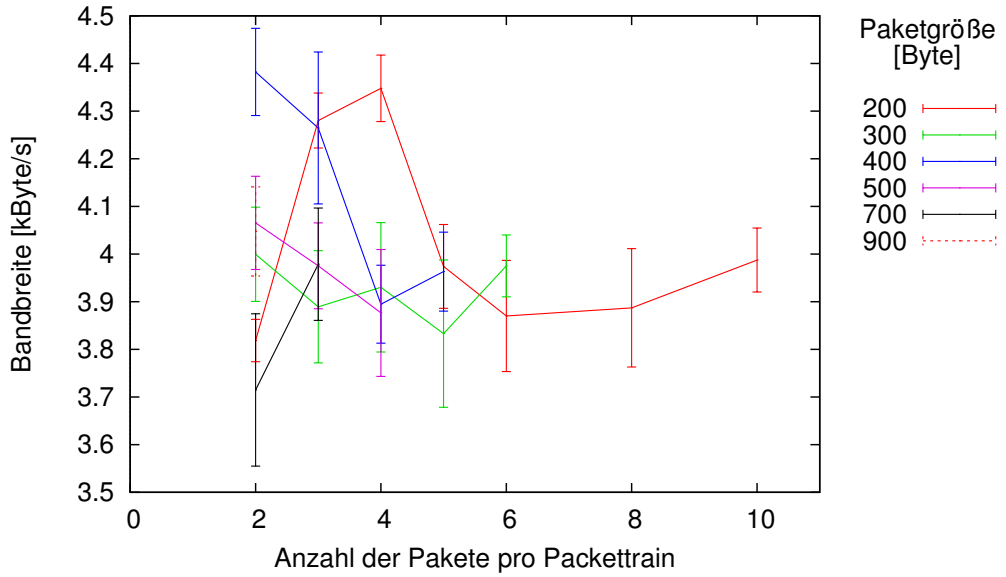

Abbildung 3.14: Statische Messungen verschiedener Packettraingrößen für den GPRS Download mit Konfidenzintervallen nach Paketgröße.

Tabelle 3.5: Packettraingrößen (#Pakete x Paketgröße [Byte]) für die Bandbreitenmessungen.

ÜBERTRAGUNGSTECHNIK	DOWNLOAD	UPLOAD
GSM	2x100	2x100
GPRS	6x300	5x200
EDGE	6x300	5x200
UMTS bzw. WCDMA	15x700	15x700
HSDPA	15x1300	15x700

Da diese Messwerte sehr nahe bei unseren Schätzungen mit Bandbreite > Packettraingröße/s liegen, funktioniert diese Art der Bandbreitenmessung in Mobilfunknetzen.

3.4.2 Filtern eindeutiger Messfehler

Abbildung 3.16 zeigt die kummulativen Verteilungsfunktionen [*cummulative distribution function (CDF)*] für den UMTS Upload und den HSDPA-Download der Traces einer zweistündigen Messfahrt, die in einem komplett mit HSDPA versorgtem Gebiet in Neuss stattfand. Dabei wurde der Download über die linke Y-Achse und der Upload über die rechte Y-Achse aufgetragen. Zusätzlich enthält die Abbildung noch zwei Geraden, die jeweils das Technologielimit für den UMTS Upload (48 kByte/s) und den HSDPA Download (921,6 kByte/s) repräsentieren. Es ist deutlich zu erkennen, dass im Download nur sehr wenige Messwerte für die Bandbreite oberhalb des Technologielimits liegen. Betrachtet man die Rohdaten selbst, so liegen lediglich 12 von 6831 Messwerten im Download über dem Tech-

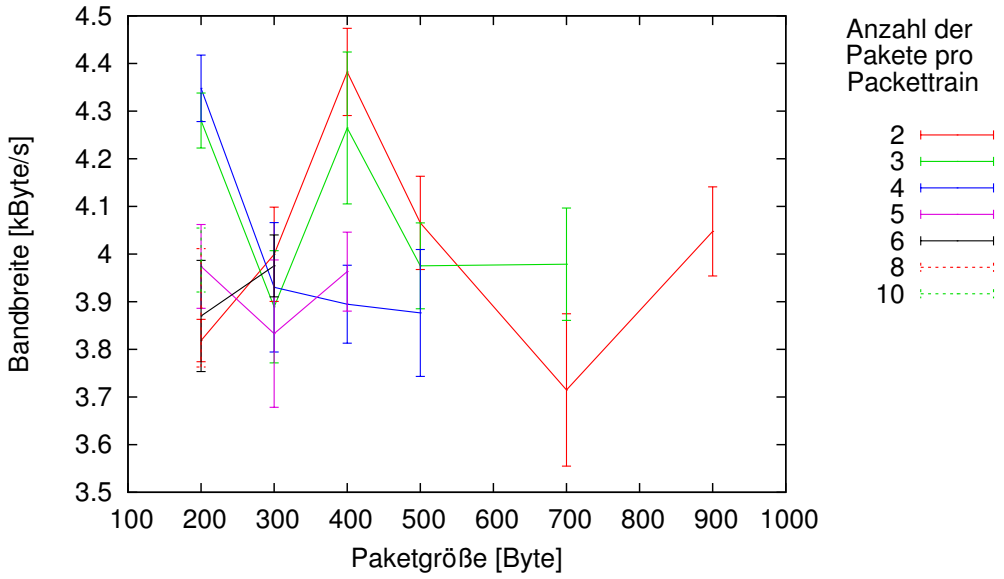

Abbildung 3.15: Statische Messungen verschiedener Packettraingrößen für den GPRS Download mit Konfidenzintervallen nach Anzahl der Pakete.

nologielimit – und das sehr deutlich mit Werten im Bereich von 80 bis 215 MB/s. Im Upload liegen 84,7% der Messwerte unterhalb des Technologielimits von 48 kByte/s, 92% der Messwerte liegen mit unter 49,0 kByte/s nur knapp über dem Limit. Viele der extrem falschen Messwerte für den Upload zeigen sich in Phasen von self-induced congestion mit mehr als 1,5 Sekunden Latenz.

Um offensichtlich falsche Bandbreitenmesswerte auszufiltern, haben wir uns dazu entschieden alle Messwerte zu verwerfen, die oberhalb des Technologielimits liegen.

3.4.3 Variable Packettraingröße

Bei der Auswertung der ersten Messfahrt stellten wir fest, dass die statische Wahl der Packettraingrößen zwar in den meisten Fällen hervorragend funktionierte, es aber kurze Zeitintervalle gab, in denen die verfügbare Bandbreite kleiner als die Packettraingröße war. Daraus ergab sich schnell eine self-induced congestion, die zu Latenzen jenseits von 20 Sekunden führen konnte. Diese hohen Latenzen bauten sich zwar, sobald wieder eine entsprechende Bandbreite verfügbar war, binnen weniger Sekunden wieder ab, zeigten uns aber Verbesserungspotential am Packettrainmodell auf.

Die nun folgende Verbesserung des Packettrainverfahrens ist durch die hohen Latenzen in Mobilfunknetzen in ihrer Reaktionszeit eingeschränkt. Wir wollen nach Möglichkeit verhindern, dass unser Bandbreitenmessverfahren unsere Latenzmessungen signifikant beeinflusst. Wie jedoch im vorigen Abschnitt erläutert, ist dies mit unserem durch Messwerte gestützten Auswahlverfahren für die

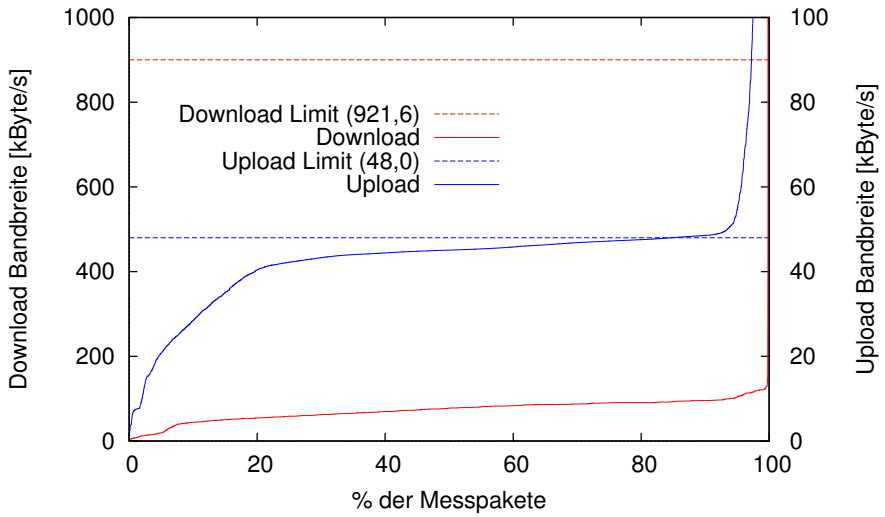

Abbildung 3.16: CDF der Bandbreite für den UMTS Upload und HSDPA-Download einer Messfahrt.

Standard-Packettraingrößen nicht möglich – zumindest solange wir die Packettraingröße nicht auf ein Minimum reduzieren (2x100 Byte), was aber zu extrem schlechten Bandbreitenschätzungen führen würde.

Abbildung 3.17 zeigt einen 100-sekündigen Ausschnitt der Messdaten für den Upload der ersten Loggingfahrt. Wie man an der grünen Geraden erkennen kann, wurde im ganzen Messabschnitt eine Packettraingröße von 10500 Byte benutzt, der eingangs des Kapitels bestimmte Packettrain mit 15x700 Byte für den UMTS Upload.

Es ist deutlich zu erkennen, dass bei ungefähr 10 Sekunden die verfügbare Bandbreite merklich auf ca. 7,5 kByte/s einbricht und dann unterhalb der Bandbreite liegt, die unsere Messpakete pro Sekunde verbrauchen. In der Folge steigt die Latenz durch die self-induced congestion von ehemals ca. 100 ms auf fast 8000 ms an. Ein weiteres Ansteigen der Latenz wird hier scheinbar dadurch verhindert, dass die Droprate stellenweise auf über 50% ansteigt.

Bei ca. 70 Sekunden steigt die verfügbare Bandbreite wieder auf die früheren Werte in der Nähe des maximal verfügbaren Uploads im UMTS Netz an. Die Latenz erreicht allerdings erst ca. 10 Sekunden nach dem Bandbreitenanstieg ein für UMTS normales Maß.

Eine Verkleinerung unserer Standard-Packettraingröße für UMTS auf eine Größe <7,5KByte/s würde die self-induced congestion in diesem speziellen Fall verhindern und dort gute Bandbreitenschätzungen ermöglichen. Für die restlichen Zeitintervalle mit höherer Bandbreite würden wir jedoch große Fehler in der Bandbreitenmessung erhalten. Daher haben wir uns für eine variable Packettrain Größe mit fester Obergrenze entschieden.

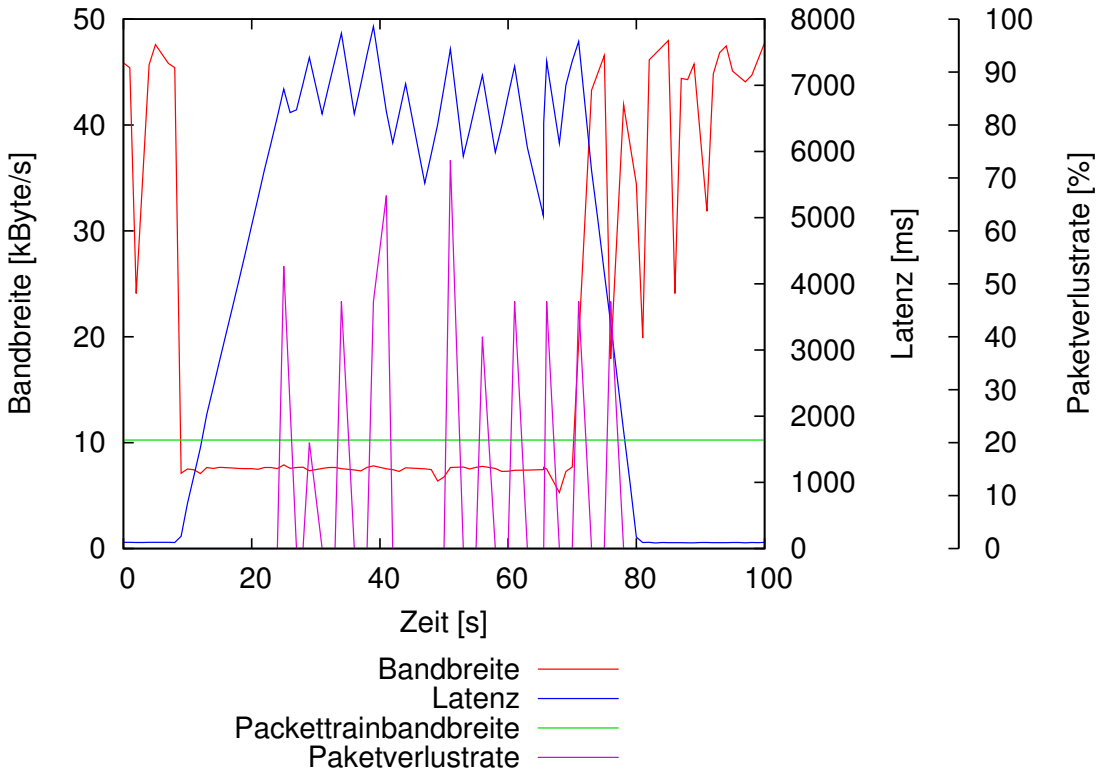

Abbildung 3.17: Beispiel für einen Bandbreiteneinbruch im UMTS Upload auf einer Testfahrt mit der Buslinie 841 in Neuss unter Verwendung statischer Packettraingrößen.

Wir nutzen nun also unsere in Abschnitt 3.4.1 bestimmten Packettraingrößen für jede Verbindungsart als maximale Packettraingröße für die jeweilige Verbindungsart, passen den Packettrain aber wie folgt dynamisch an:

1. Wenn die letzte Latenzmessung eine Latenz > 2 Sekunden liefert, halbieren wir (falls möglich) die Größe des momentanen Packettrains.

2. Andernfalls, wenn der aktuelle Packettrain mehr als $\frac{2}{3}$ des Durchschnitts der letzten x Bandbreitenmessungen belegt, setzen wir den Packettrain auf maximal

$$\frac{\text{Durchschnittsbandbreite der letzten x Messungen}}{2}$$

3. Andernfalls, wenn die aktuelle Latenz < 1s ist, die aktuelle Packettraingröße kleiner als die maximale Packettraingröße für die Übertragungstechnik und die

$$\text{aktuelle Packettraingröße} < \frac{\text{Durchschnittsbandbreite der letzten x Messungen}}{2}$$

ist, setzen wir den Packettrain auf maximal

$$\frac{\text{Durchschnittsbandbreite der letzten x Messungen}}{2}.$$

Der Parameter x ist dabei konfigurierbar und momentan auf 5 eingestellt.

Schritt 1 soll dafür sorgen, dass wir schnell auf große Latenzen reagieren und die Packettraingröße verringern können. In der Regel sind die Latenzen im Mobilfunk ohne self-induced congestion kleiner als 2 Sekunden. Werte zwischen 1 und 2 Sekunden treten jedoch speziell bei GPRS Verbindungen häufiger auf.

Schritt 2 sorgt ebenfalls für eine Reduzierung der Packettraingröße, diesmal auf die Hälfte des Durchschnitts der Bandbreitenschätzung sollten wir mehr als $\frac{2}{3}$ der geschätzten verfügbaren Bandbreite für den Packettrain verbrauchen.

Schritt 3 sorgt letztendlich dafür, dass wir bei wieder genügend verfügbarer Bandbreite nach einer Drosselung der Packettraingröße erneut zu unserer in Abschnitt 3.4.1 gewählten Standard-Packettraingröße zurückkehren können (gegebenenfalls in mehreren Abstufungen).

Die $\frac{2}{3}$ bzw. $\frac{1}{2}$ in den Schritten 2 und 3 wurden unterschiedlich gewählt, um ein permanentes Oszillieren der Packettraingröße weitestgehend zu unterbinden.

Aber auch dieses Vorgehen löst das Problem der self-induced congestion noch nicht vollständig. Zwar reagiert der Algorithmus darauf, braucht aber noch relativ lange bis die erste Reaktion eintritt. Das liegt daran, dass sich erst einmal die self-induced congestion aufbauen muss, was wiederum dazu führt, dass die Latenz schnell auf mehrere Sekunden ansteigt. Wir reagieren aber erst dann, wenn das erste Latenzmesspaket mit großer Latenz bei uns eintrifft.

Daher haben wir zur Unterstützung von Punkt 1 je einen Timer für die Upload- und Download-Latenz eingeführt, die beide immer dann auf 2 Sekunden gesetzt werden, wenn ein zu ihnen gehörendes Latenzmesspaket eingetroffen ist. Trifft vor dem Auslaufen eines Timers kein neues Latenzmesspaket ein – der Timer wird also nicht erneut auf 2 Sekunden gesetzt – so führt der Timer automatisch Punkt 1 aus. Die Länge des Packettrains wird somit halbiert, was zu einer deutlich schnelleren Reaktion auf self-induced-congestion führt.

Abbildung 3.18 zeigt einen Ausschnitt aus einer Messfahrt mit der Straßenbahnlinie 707 in Düsseldorf. Bei dieser Messung sieht man sehr gut, dass die eben eingeführten Änderungen dafür sorgen, dass die Bandbreite, die unser Packettrain verbraucht, fast immer unterhalb der verfügbaren Bandbreite liegt. Ohne diese Anpassung hätte sich im Bereich von 23 bis 30 Sekunden, bei der die verfügbare Bandbreite bei ca. 8 kByte/s und damit unterhalb unserer Standard UMTS Messdatenrate von

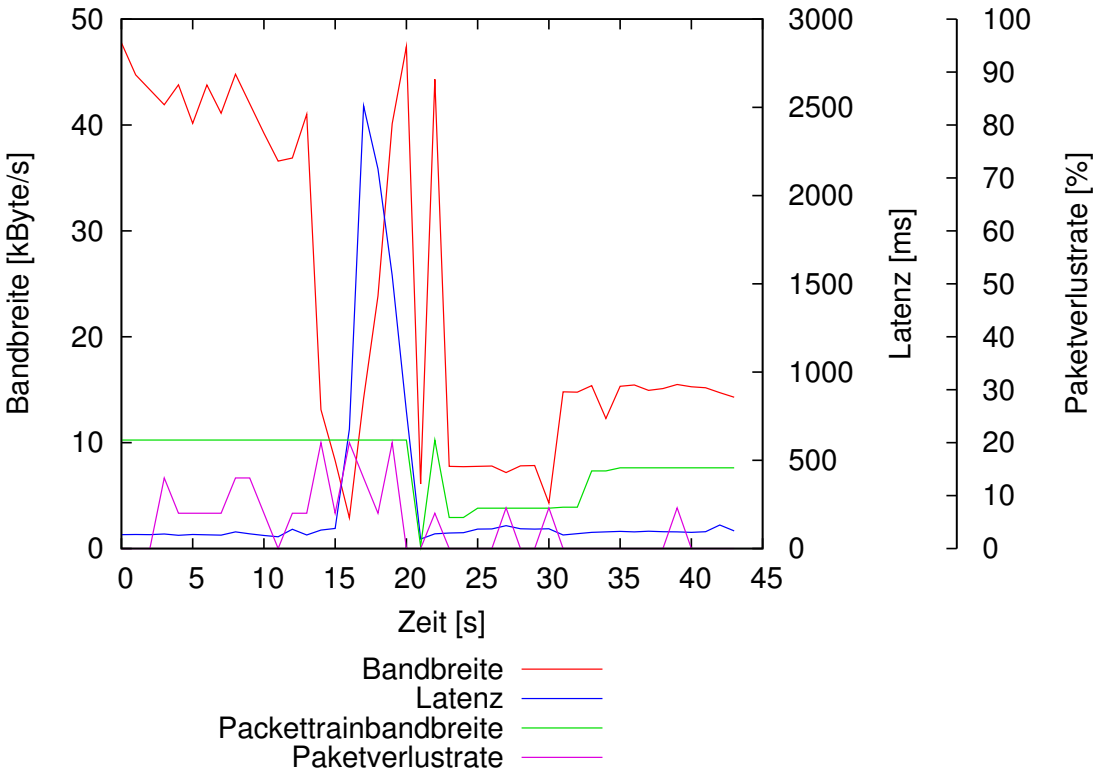

Abbildung 3.18: Beispiel für einen Bandbreiteneinbruch im UMTS Upload auf einer Testfahrt mit der Straßenbahnlinie 707 in Düsseldorf unter Verwendung variabler Packettraingrößen.

10500 Byte/s, ein Datenstau gebildet und die Latenzen wären deutlich angestiegen. Hier bleiben die Latenzen in diesem Abschnitt jedoch auf normalem UMTS Niveau.

Die Abbildung zeigt jedoch auch, dass es noch weiteres Verbesserungspotential gibt. So führt der Bandbreiteneinbruch bei Sekunde 16 erst zu einer, dann allerdings extremen, Reaktion in Sekunde 21. Hier greifen dann gleich alle Reduktionspunkte auf einmal und reduzieren die Packettraingröße auf das Minimum von 2x100 Byte.

Uns ist bewusst, dass dieser Ansatz zudem einen Schönheitsfehler besitzt. Mit unserem Ansatz führen spät eintreffende Latenzmesspakete zu einer größeren Strafe (bei einer Latenz von 2,5 s z.B. zur Viertelung der Packettraingröße) als Latenzpakete, die bei der Übertragung verloren gehen (bei einer eigentlichen Latenz von 2,5 s würde hier die Packettraingröße nur halbiert). Dieses Problem ließe sich mit mehr Zeit, welche im Rahmen dieser Arbeit aber leider nicht mehr zur Verfügung stand, programmiertechnisch lösen. Auch wäre eine Verbesserung denkbar, die mit einem Erwartungswert für die Latenz arbeitet. Würde man diesen Erwartungswert auf Basis der letzten x (wobei x noch zu bestimmen wäre) Latenzmessungen bilden und die nächste Latenzmessung würde eine signifikant höhere

Latenz zeigen, so könnte man schon in diesem Moment gegensteuern anstatt erst auf das Überschreiten eines festen Schwellenwerts von momentan 2 Sekunden zu warten.

Kapitel 4

Das Logging Framework

Zunächst mussten wir entscheiden, welches Betriebssystem wir für das Logging benutzen möchten. Viele UMTS-Modems lassen sich zwar einfacher unter Windows nutzen und abfragen, aber in puncto NTP-Service und GPS mit PPS Synchronisation ist Windows aufgrund der wesentlich ungenaueren Timingroutinen im Windows Kernel Linux eindeutig unterlegen.

Daher wurde das Loggingframework auf Linux abgestimmt, wenngleich der Hauptteil des Frameworks plattformübergreifend funktioniert, da es in Java implementiert wurde.

Als Server setzen wir einen dedizierten Server mit Pentium4 3Ghz CPU, 2GB RAM und einem Raid1 Festplattenverbund ein. Als Client findet ein Thinkpad T61 mit Core2Duo T7300 2GHz CPU, 4GB Ram, nativer serieller Schnittstelle und einer Sierra Wireless MC8775 Mini-PCIe UMTS Karte (maximale Datenraten siehe Tabelle 3.4) Verwendung. An beide Rechner ist seriell jeweils ein *Garmin OEM 18x LVC* GPS Empfänger angeschlossen (vgl. Abschnitt 3.2.3).

Abbildung 4.1 zeigt den Messaufbau des Logging Clients mit angeschlossenem *Garmin OEM 18x LVC* GPS Empfänger.

Als Software läuft auf beiden Systemen:

- Ubuntu 9.10 amd64
- Sun JDK 1.6.0_20 32 Bit
- gpsd 2.95
- ntpd 4.2.4p6
- mysql Ver 14.14 Distrib 5.1.37

Abbildung 4.1: Aufbau des *TBUSLogger*-Client Messsystems.

Als zusätzliche Libraries für Java finden folgende Bibliotheken Verwendung:

- JFreeChart 1.0.13 + JCommon 1.0.16 (zur Darstellung des Netzwerkmonitors)
- MySQL JDBC Driver (mysql-connector-java-5.1.6-bin) und TopLink Essentials zur Anbindung der Mysql Datenbank
- Nanotime (um Zeitstempel mit Nanosekundengenauigkeit vom Linux Kernel zu erhalten)
- batik 1.7 zur Erstellung von SVGs
- RXTX-2.2pre2, eine Bibliothek zur Ansteuerung serieller Ports unter Java
- GSON 1.4, eine Bibliothek zur Verarbeitung von JSON strings
- diverse weitere Netbeans Gui Libraries

4.1 Zeitsynchronisation für das Logging

Die Zeitsynchronisation der beiden Messstationen erfolgt unabhängig vom eigentlichen Loggingframework. Beide PCs synchronisieren ihre Uhren mittels PPS Signal gegen die Garmin 18x LVC GPS Uhren. Zu diesem Zweck läuft auf beiden Geräten *gpsd*, ein Dämon, der die Daten einer oder mehrerer angeschlossener GPS Geräte empfängt, auswertet und zur weiteren Verarbeitung in unterschiedlichen Formen zur Verfügung stellt. Unter anderem wertet gpsd das PPS Signal eines GPS Empfängers aus und stellt es *ntpd* über ein Shared Memory Device als Zeitgeber zur Verfügung. Durch dieses Zusammenspiel können sich die beiden PCs mit dem GPS Zeitsignal synchronisieren. Als Fallback sind im Server zusätzlich auch noch einige Stratum-1 Zeitserver zur Synchronisation eingetragen (am Server spielen die wenigen Byte großen NTP Pakete keine signifikante Rolle, da er über das Universitätsrechenzentrum großzügig ans Internet angebunden ist), bisher wurde das Fallback jedoch nicht benötigt. Der Server erreicht in der Regel eine Synchronisation zum GPS Zeitsignal von unter 0,1 Millisekunden.

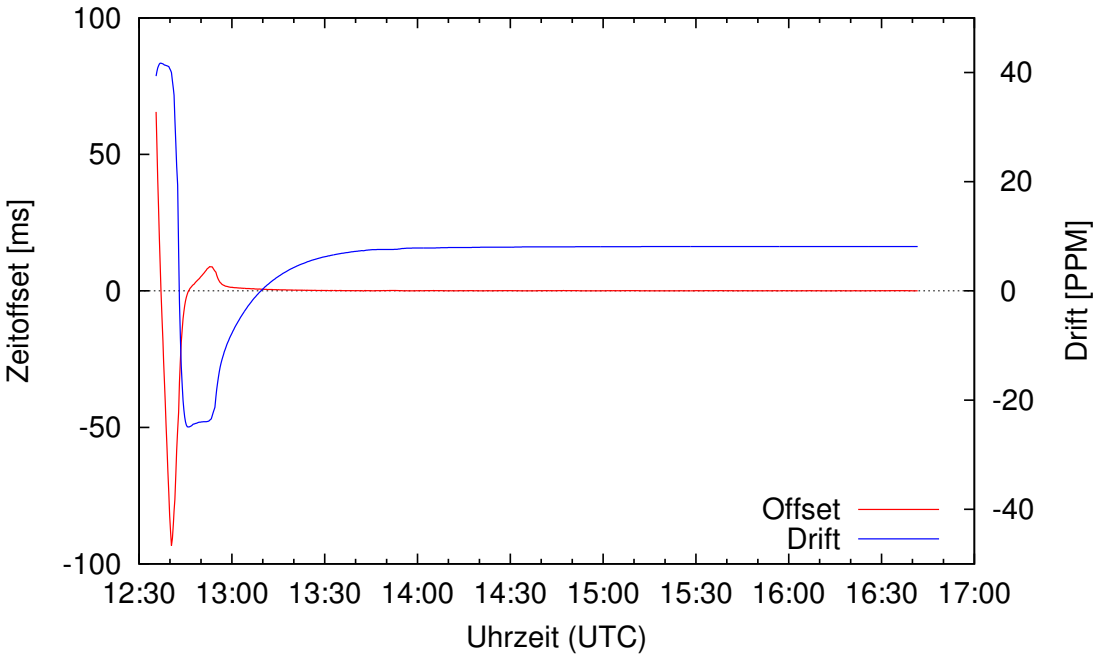

Abbildung 4.2: Einschwingphase der Zeitsynchronisation von *NTP* gegen das *GPS* Zeitsignal mit Hilfe von *PPS*.

Wichtig anzumerken ist, dass der Client vor einer Loggingfahrt ca. 30 Minuten Zeit erhält, um sich gegen das GPS Zeitsignal zu synchronisieren. NTP benötigt diese Zeit um sich einzupendeln. Würde man erst kurz vor – oder gar während – der Loggingfahrt mit der Zeitsynchronisation beginnen, wären die Latenzmessungen zum Anfang der Loggingfahrt sehr ungenau bis unbrauchbar. Beispielhaft zeigt Abbildung 4.2 die Einschwingphase der Zeitsynchronisation von NTP gegen das GPS Zeitsi-

gnal mit Hilfe von PPS. Dabei ist auf der X-Achse die Uhrzeit in UTC, auf der linken Y-Achse der Offset in Millisekunden der Systemuhr zum PPS Signal und auf der rechten Y-Achse die Korrektur in PPM (vergleiche Abschnitt 2.3) des Fehlers der Systemuhr aufgetragen. Deutlich sind die großen Schwankungen zu Beginn der Zeitsynchronisation in der Offset-Kurve zu erkennen, bei der der Offset erst nach circa 30 Minuten konstant deutlich unter 10 ms beträgt. Schaubild 4.3 ist ein Plot der gleichen Daten, allerdings wurde der Wertebereich für den Offset auf den Bereich von -0,1 ms bis 0,3 ms beschränkt um zu zeigen, wie gut die Zeitsynchronisation zur GPS-Referenzzeit mit Hilfe des PPS Signals funktioniert.

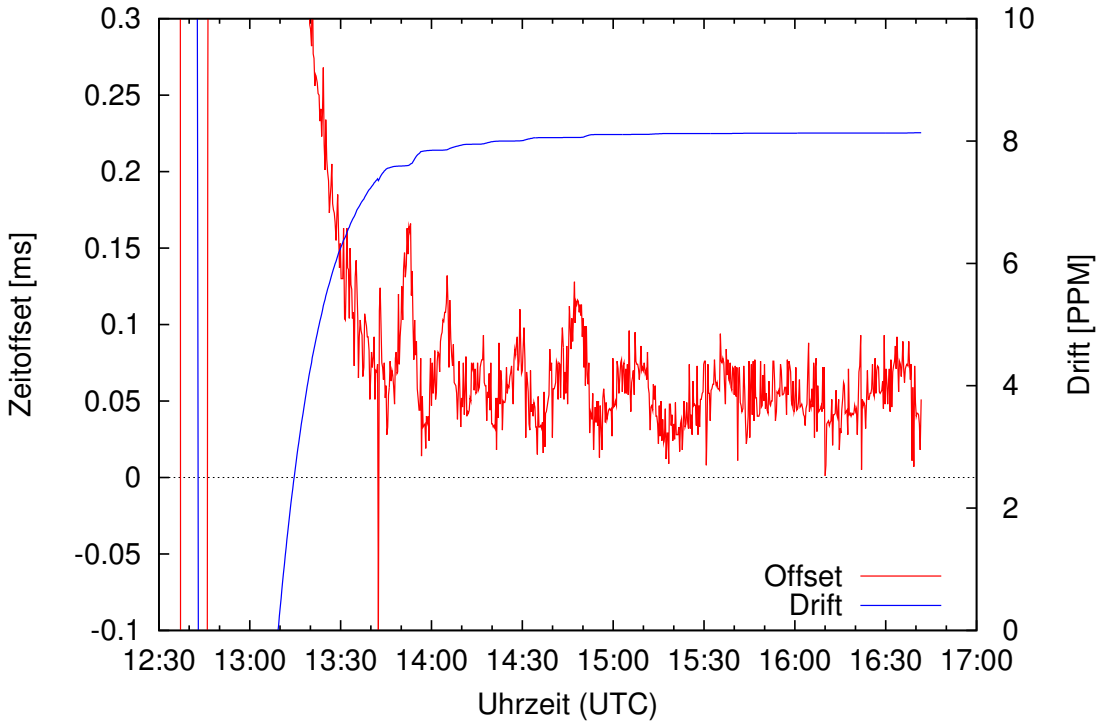

Abbildung 4.3: Einschwingphase der Zeitsynchronisation von *NTP* gegen das *GPS* Zeitsignal mit Hilfe von *PPS* mit beschränktem Wertebereich für den Zeitoffset.

Vor einer Loggingfahrt muss man also immer prüfen, ob der Client und der Server gut gegen die GPS Zeit synchronisiert sind. Erst dann (ohne zwischenzeitliches Abschalten der Rechner oder längere Unterbrechung des GPS Signals) kann man mit der Loggingfahrt beginnen. Als praktische Möglichkeit der Überprüfung einer guten Synchronisation hat sich die Latenzanzeige des Netzwerkmonitors, den wir in Abschnitt 4.2.3 beschreiben, erwiesen. Schließt man den Client und den Server an das gleiche LAN an, so ergeben sich Latenzen für den Up- und Download, die deutlich unter einer Millisekunde liegen. Ist die Uhrzeit der Rechner nicht gut synchronisiert, so erkennt man dies deutlich daran, dass die Anzeige für eine der Latenzen im negativen Bereich liegt, während die andere mit fast dem gleichen betraglichen Wert im positiven Bereich liegt.

4.2 Aufbau des Logging Frameworks (*TBUSLogger*)

Das Loggingframework besteht aus zwei Teilen. Einem Java Gui Client und einem konsolen-basierten Java Server. Das Framework besteht im wesentlichen aus einzelnen Funktions-Paketen (Packages), die möglichst unabhängig voneinander gehalten wurden. Sämtliche Interaktion zwischen den Paketen (mit Ausnahme der Steuerung der einzelnen Pakete durch die GUI) findet über Eventlistener statt. Hierdurch ist eine relativ einfache Erweiterung und Veränderbarkeit der einzelnen Pakete erreicht worden.

Wir haben uns dazu entschlossen, alle Daten in einer MySQL Datenbank [1] zu speichern da diese zum einen von vielen Programmiersprachen nutzbar ist und zum anderen auf Dauer wesentlich handlicher als Textfiles oder eigene Binärfiles ist. Dabei ging es nicht um ein möglichst elegantes Datenbank Design, sondern einzig um die effiziente Sammlung großer Datenmengen.

Das Framework, das wir *TBUSLogger (Trace-Based UMTS Simulation Logger)* genannt haben, besteht aus folgenden Paketen:

1. Tbuslogger – in dem Hauptpaket befinden sich sowohl die Client Gui Klasse als auch der Konsolen Server
2. Entities – enthält alle Entityklassen (Abbildungen von MySQL Tabelleneinträgen auf Java Objekte) und Controller für das gesamte Logging
3. GpsdLogger – dient sowohl dem Logging der NMEA Rohdaten als auch der durch gpsd aufbereiteten GPS Daten
4. ModemLogger – initialisiert das UMTSModem (für das Logging) und fragt periodisch die verbindungsspezifischen Daten des Modems ab und loggt sie
5. Network – enthält jeweils einen Client und einen Serverpart (die Unterschiede sind marginal aber wichtig), die sowohl die Latenz- als auch die Bandbreitenmessungen steuern und loggen.

Sämtliche Daten werden mit mindestens einem Zeitstempel, dem Empfangszeitstempel, in Nanosekunden gespeichert. Bei den Netzwerkdaten gibt es in der Regel natürlich auch noch einen Sendezeitstempel.

Den Aufbau der GUI des TBUSLogger Client möchten wir kurz am Beispiel des Screenshots 4.4 des ModemLoggers, der im folgenden Abschnitt genauer beschrieben wird, erläutern. Die GUI ist in drei Teile gegliedert. Direkt unter der Menüleiste befindet sich eine Tab-Liste (1), mit der man zwischen den unterschiedlichen Ansichten des Loggingclients wechseln kann. Die Statusbar (2) zeigt zu jeder Zeit die wichtigsten Einstellungen des TBUSLogger-Client an. Man kann jederzeit erkennen,

[1] Die für den Betrieb des TBUSLogger nötige mysql Konfiguration haben wir in Anhang A.3 beschrieben.

ob das Logging gerade aktiviert ist, ob gerade Testpakete über das Netzwerk geschickt werden, oder ob Packettrains fixer Größe verschickt werden oder ob wir dynamische Packettrains benutzen. Diese Umschaltfunktionen kann man auch aus jedem Tab heraus durch einen einfachen Mausklick ändern. Zusätzlich gibt die Statusleiste über den aktuell verwendeten Clienttrain (CT) und Servertrain (ST), sowie über die verwendete Netzwerktechnologie, hier HSDPA, Auskunft. Der Hauptteil des Fensters (3) wird jeweils mit den zum ausgewählten Tab gehörenden Informationen gefüllt.

4.2.1 ModemLogger

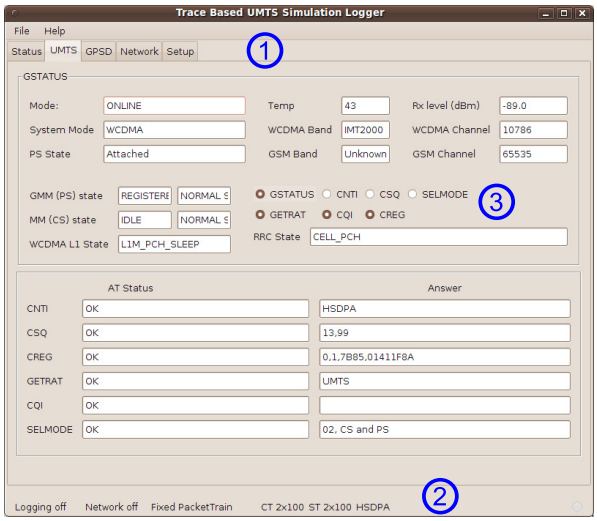

Abbildung 4.4: Übersicht über die aktuellen Mobilfunkdaten im *TBUSLogger*-Client.

UMTS Modems unterscheiden sich von ihrer Ansteuerung wenig von herkömmlichen Modems für analoge Telefonleitungen. So stellen sie in der Regel mindestens zwei serielle Ports (meist über USB Devices) im System zur Verfügung, ein Command-Interface zur Konfiguration und Statusabfrage des Modems über *AT-Kommandos* und ein Data-Interface zur Übertragung der Nutzdaten. Leider ändern die Hersteller die Zuordnung dieser Interfaces auf die /dev/ttyUSBx Geräte ohne erkennbares Muster. Teilweise existieren Geräte mit exakt der gleichen Gerätebezeichnung und auch gleicher USB-Device-ID mit unterschiedlicher Zuordnung und ohne Dokumentation. Dies ist eine wesentliche Fehlerquelle für freie Treiber. Bis heute lässt sich das hier verwendete Modem trotz Bugreports nicht mit dem in Linux verbreiteten Networkmanager für das Logging nutzen, da er für die Datenübertragung das Command-Interface benutzt. Das Command-Interface unterstützt zwar auch die Datenübertragung über UMTS, aber das eigentlich dafür vorgesehene Data-Interface unterstützt nur sehr wenige AT-Commands und eignet sich daher nicht zur Abfrage der Mobilfunkverbindungseigenschaften. Daher muss das verwendete Sierra Wireless MC8775 UMTS Modem unter Linux mit *wvdial* [2] auf der Kom-

[2] Eine detaillierte Konfigurationsbeschreibung inklusive Konfigurationsdatei befindet sich in Anhang A.4

mandozeile konfiguriert und mit dem Mobilfunkprovider manuell eine Verbindung aufgebaut werden.

Erst danach kann der ModemLogger das Command-Interface in Beschlag nehmen und zunächst das UMTS Modem so initialisieren, dass wir all die Daten, die uns interessieren, auch sammeln können.

Nach der Initialisierungsphase sendet der ModemLogger einen AT-Befehl über den CommandPort an das Modem, wartet auf die Antwort und sendet daraufhin den nächsten AT-Befehl usw.

Der ModemLogger fragt also nonstop das Modem nach Informationen. Da das Modem je nach AT-Befehl bzw. gerade herrschender Verbindung unterschiedlich lange für die Antworten benötigt, können wir nicht genau beeinflussen, wann welcher AT-Befehl gesendet und beantwortet wird. So benötigt die Antwort auf *AT+USET?0*, welches bei UMTS-Verbindungen das aktuelle Active-Set, also die sichtbaren UMTS-Basisstationen und die zu den Stationen gehörenden Messwerte, zurückliefert natürlich unterschiedlich lang, je nachdem wieviele Basisstationen gerade gemeldet werden.

Im Durchschnitt wird jedoch jeder der in Tabelle 4.1 aufgeführten AT-Befehle alle 600ms an das Modem gesendet und dementsprechend auch eine Antwort empfangen. In der Tabelle sind jeweils nur die wichtigsten Antwortparameter aufgeführt. Eine komplette Liste findet sich im Quelltext (tbuslogger.modemlogger.ModemLoggerAllInOne.Java), eine ausführliche Beschreibung der einzelnen Parameter zu den unterstützten AT-Befehlen im Handbuch [Sie06] von Sierra Wireless.

In seltenen Fällen kann es bei dem Empfang einer Antwort jedoch zu Fehlern kommen, da das Modem – trotz Initialisierung, die dies eigentlich laut Handbuch verhindern müsste – selbstständig eine nicht angeforderte AT-Statusmitteilung übermittelt. Zudem wird nicht erst die aktuell laufende AT-Antwort zu Ende übertragen, sondern die neue AT-Antwort wird direkt gesendet. Das wäre sogar AT-konform, wenn denn die vorige AT-Antwort mit einer Antwortzeile mit dem String `ERROR` beendet würde. Eine solche Antwort fehlt jedoch.

Durch diesen Firmwarebug kommt es ab und zu (im Schnitt ca. alle 2 Minuten einmal) dazu, dass eine AT-Antwort 'zerstört' wird. In Anbetracht der Häufigkeit der Abfragen ergibt sich aber trotzdem eine sehr dichte Datenbasis der Mobilfunkdaten.

Während des Loggings kann man jederzeit den aktuellen Mobilfunkstatus im *TBUSLogger*-Client im UMTS-Tab (siehe Screenshot 4.4) einsehen.

Tabelle 4.1: Benutzte AT-Befehle und die wichtigsten Antwortdaten

AT-BEFEHL	BESCHREIBUNG & ERHALTENE DATEN
AT*CNTI=0	Fragt ab welcher Datenmodus gerade genutzt wird. Die Antwort enthält einen der Werte: GSM, GPRS, EDGE, UMTS, HSDPA
AT!GSTATUS?	Fragt allgemeine Statusinformationen ab. Man erhält unter anderem: Modemtemperatur, Zeit seit Aktivierung des Modems, die aktuell benutzten WCDMA und GSM Kanäle, die aktuelle Empfangsstärke
AT!GETRAT?	Fragt ab ob gerade GSM oder UMTS benutzt wird. Kann zusätzlich zu UMTS und GSM auch mit `No Service` antworten.
AT!SELMODE?	Fragt die aktuellen Service Domains ab. CS oder PS oder CS und PS.
AT+CSQ	Fragt die Signalqualität ab
AT+CREG?	Fragt die aktuelle CellID und den aktuellen Location Area Code ab
AT+USET?0	Fragt das aktuelle Active Set ab. Die Antwort ist `ERROR` wenn gerade kein UMTS bzw. HSDPA benutzt wird. Ansonsten enthält sie eine Liste der gerade sichtbaren Basisstationen mit jeweils den Informationen zu Scrambling Codes, received signal power code und einigen weiteren Daten.
AT+CQI?	Fragt den Channel Quality Indicator ab (funktioniert nur bei UMTS Betrieb) – laut Datenblatt und Sierra Wireless sollte dieses Kommando funktionieren, wir haben aber im gesamten Logging nie einen Wert erhalten.
AT+CREG=0	Aktiviert die manuelle Abfragemöglichkeit für die aktuelle CellID und den aktuellen LAC (Location Area Code).
AT+CQI=1	Schaltet die Möglichkeit ein, CQI Informationen abzufragen.
AT!GVER?	Fragt Revisionsinformationen des UMTS Modems ab

4.2.2 GPSDLogger

Das Logging der GPSD Daten erfolgt auf zwei unterschiedliche Weisen. Zum einen werden mit dem *GpsdRawReader* die Roh-*NMEA*-Datensätze vom GPSD Dämon ausgelesen und in der Datenbank gespeichert, zum anderen lässt sich der *GpsdWatcher* von *gpsd* aufbereitete Daten bezüglich Position/Geschwindigkeit (o) und Satfix (y) (vergleiche Tabellen 4.2 und 4.3) schicken.

Der *GpsdRawReader* speichert den kompletten unbehandelten *NMEA*-Trace und enthält die *NMEA*-Sätze *GPRMC*, *GPGGA*, *GPGSA* und *GPGSV*.

Auch der Status des *GpsdWatcher* lässt sich am Client während des Loggings jederzeit über den GPSD-Tab (Abbildung 4.5) einsehen.

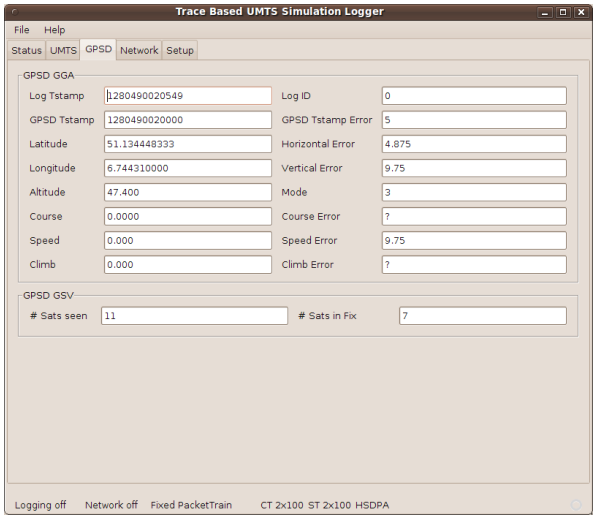

Abbildung 4.5: Übersicht über die aktuellen GPSD-Daten im *TBUSLogger*-Client.

Wie schon in Kapitel 3.2.3.1 erwähnt, sind die GPSD-Loggingkomponenten in der Lage, sowohl mit gpsd in Version 2.39, als auch mit gpsd in der Version 2.95 zu kommunizieren und die Daten in einer einheitlichen Datenbank abzuspeichern. Zusätzlich zu den im Screenshot 4.5 ersichtlichen Daten werden auch die Empfangsdaten der Satelliten (siehe Tabelle 4.3) gespeichert.

4.2.3 Netzwerkmonitor

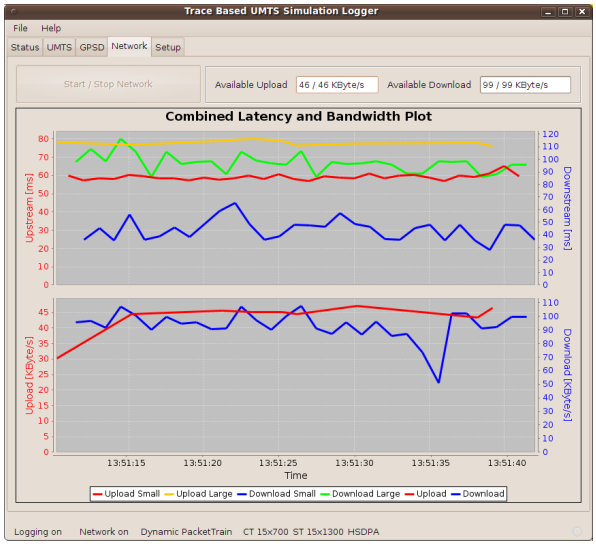

Abbildung 4.6: Netzwerkmonitor des *TBUSLogger*-Client.

Um während der Messfahrten Einblick in die aktuellen Bandbreiten und Latenzmessungen nehmen zu können, haben wir im Netzwerk-Tab (siehe Abbildung 4.6) einen Netzwerkmonitor implemen-

Tabelle 4.2: GPSD O Daten (gpsd 2.39) beziehungsweise TPV Daten (gpsd 2.95)

NAME	BESCHREIBUNG
timestamp	Sekunden seit der Unix Epoche in UTC.
time error	Von *gpsd* geschätzter Fehler des Zeitstempels und das 95% Konfidenzintervall
latitude	Breitengrad
longitude	Längengrad
altitude	Höhe in Metern, wenn mode != 3 handelt es sich bei dem Wert um eine Schätzung die als unsicher betrachtet werden sollte.
horizontal error estimate	Fehlerschätzung für den horizontalen Fehler in Metern
vertical error estimate	Fehlerschätzung für den vertikalen Fehler in Metern
course over ground	Bewegungsrichtung in Grad
speed over ground	Bewegungsgeschwindigkeit in m/s
climb/sink	Vertikale Geschwindigkeit in m/s
estimated error in course over ground	Fehlerabschätzung für die Bewegungsrichtung (Grad und 95% Konfidenzintervall)
estimated error in speed over ground	Fehlerabschätzung für die Bewegungsgeschwindigkeit (m/s und 95% Konfidenzintervall)
estimated error in climb/-sink	selbige Fehlerabschätzung für die vertikale Geschwindigkeit
mode	Der Satfix Modus von NMEA (?=noch kein mode gesehen, 1=no fix, 2=2D, 3=3D).

tiert. Dieser gibt einen Überblick über die letzten 30 Sekunden (dieser Wert ist konfigurierbar) der Netzwerkmessungen.

In der Grafik werden die Messungen für den Upload immer in Bezug zur linken Y-Achse aufgetragen, die des Downloads gegen die rechte Y-Achse. Dies ist sinnvoll, da sich Messbereiche gerade für die Bandbreite zwischen Up- und Download in großem Maße unterscheiden können.

In dem oberen Teil der Grafik sind die Latenzen aufgetragen. Dabei werden jeweils zwei Messreihen für jede Übertragungsrichtung dargestellt. Diese zwei Latenzmesswerte pro Übertragungsrichtung ergeben sich, weil wir zum einen eine separate Latenzmessung mit Paketen minimaler Größe durchführen (40 Byte, die sowohl den UDP Header als auch den Zeitstempel enthalten), diese haben wir `Upload Small` beziehungsweise `Download Small` genannt. Zum anderen gewinnen wir eine Latenzmessung aus dem jeweils ersten Paket einer Packettrain Bandbreitenmessung (`Upload/Download Large`. Die Messwerte der Uploadrichtung sind in rot und orange, die der Downloadrichtung in blau und grün abgetragen (vergleiche die Legende unter der Grafik).

Der untere Teil der Grafik gibt einen Einblick in die Bandbreitenmessungen der letzten 30 Sekunden.

Tabelle 4.3: GPSD Y Daten (gpsd v 2.39) beziehungsweise SKY Daten (gpsd v 2.95)

NAME	BESCHREIBUNG
timestamp	Sekunden seit der Unix epoche in UTC.
count	Anzahl der Satelliten für die Daten folgen und dann jeweils pro Satellit
PRN	Die Satellitennummer
elevation	Integer im Wertebereich 0-90
azimuth	Integer von 0-359
signal strength	Signalstärke in dB
in fix	Wurde der Satellit im letzten Fix benutzt (1) oder nicht (0)

Der kritische Leser wird erkennen, dass die einzelnen Kurven zwar jeweils ungefähr die gleiche Länge haben, aber an unterschiedlichen Zeitpunkten der Grafik starten beziehungsweise enden. Hierfür gibt es drei Gründe:

1. Die Messwerte für die Uploadmessungen werden vom Server erhoben, denn dieser erhält ja die Messpakete. Diese Messwerte muss er als Nutzlast huckepack in den Bandbreitenmesspaketen für den Download an den Client zurücksenden, damit dieser sie anzeigen kann. Damit treffen die Messwerte frühestens nach einer weiteren Downloadlatenz nach ihrer Erhebung beim Client ein.

2. Die Messreihen werden in Datenstrukturen des JFreeChart Grafikpaketes gespeichert, das für die Anzeige des Netzwerkmonitors verwendet wird. Diese sorgen automatisch dafür, dass zu jeder Messreihe nur die vorher konfigurierte Zeitspanne an Daten vorgehalten wird, daher erstrecken sich die Kurven nie von einer zur anderen Seite, solange auch nur eine von ihnen Versatz zu den anderen hat.

3. Da wir davon ausgehen, dass der Engpass, und damit die Hauptursache für die Latenzen und die zur Verfügung stehende Bandbreite, die Luftschnittstelle ist, haben wir uns dazu entschieden, für die Messungen im Upload den Zeitstempel des ersten gesendeten Paketes als Messzeitpunkt festzusetzen. Für Downloadmessungen haben wir dagegen den Zeitpunkt des Eintreffens des ersten Messpaketes beim Client als Messzeitpunkt festgesetzt. Dadurch ergibt sich für die Download Bandbreitenmessung ein deutlicher Zeitversatz, obwohl die Messpakete für die Download- und Uploadmessungen eigentlich zeitgleich vom Client beziehungsweise Server abgeschickt werden. Die Downloadlatenzmesswerte aus den ersten Bandbreitenmesspaketen werden natürlich den Eintreffzeipunkten eben dieser Pakete beim Client zugeordnet.

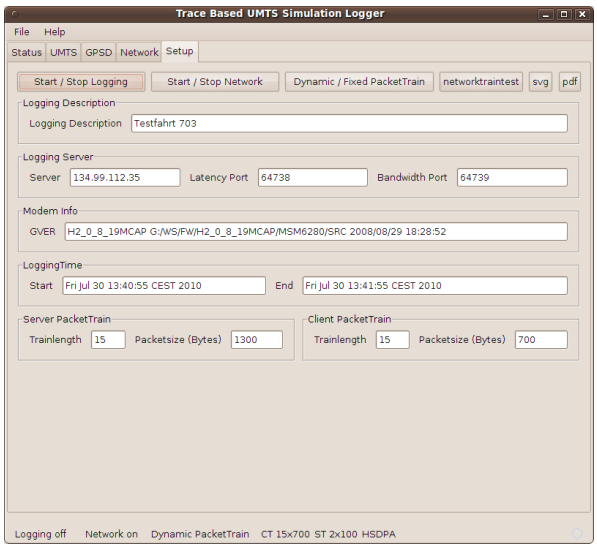

Abbildung 4.7: Setup-Tab im *TBUSLogger*-Client.

4.2.4 TBUSLogger Setup

Das Setup-Tab (Screenshot 4.7) bietet Möglichkeiten das Logging für die nächste Messreihe zu konfigurieren. So kann man jedem Loggingrun eine Beschreibung mitgeben, die in der Datenbank gespeichert wird und es erleichtert, ein bestimmtes Ereignis wiederzufinden. Man kann IP-Adresse und Ports des Loggingservers konfigurieren, die Netzwerkmessungen starten und stoppen und zwischen dynamischen (vgl. Abschnitt 3.4.3) und statischen Packettrains (siehe Abschnitt 3.4.1) für die Bandbreitenmessungen wählen. Ferner kann man auch selbst fixe Packettrainparameter für den *Server Packettrain* – also den Packettrain, der vom Server zum Client geschickt wird – und den *Client Packettrain* vorgeben.

Das Starten beziehungsweise Stoppen des Loggingvorgangs setzt die Start- und Endzeit der aktuellen Messreihe.

Kapitel 5

Das Analyse Framework

5.1 TBUS Analyzer

Zur Analyse der Messdaten haben wir den *TBUS Analyzer*, eine ebenfalls unter Java arbeitende Applikation, erstellt, der den Umgang mit den Messdaten erleichtert und es ermöglicht, Messdaten unterschiedlichen Umfangs genauer zu betrachten und für das Plotten mit gnuplot aufzuarbeiten.

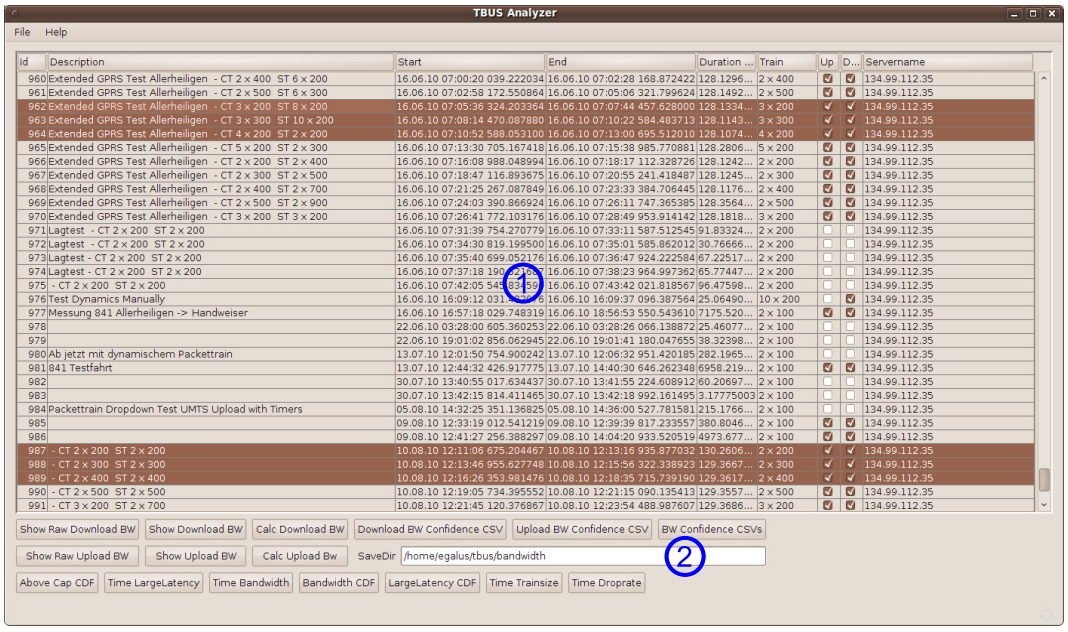

Abbildung 5.1: Hauptfenster des *TBUS Analyzer*.

Das Hauptfenster (Screenshot 5.1) ist in zwei Teile unterteilt. Teil 1 enthält eine Übersicht über alle durchgeführten Messreihen. Zu jeder Messreihe kann man direkt die Datenbank ID für den Loggingvorgang, den selbst vergebenen Loggingnamen, den Start- und Endzeitpunkt der Messung, die Messdauer, die bei dem Logging benutzte initiale Trainsize und die IP-Adresse des Servers, gegen den

gemessen wurde, einsehen. Außerdem gibt es zwei Spalten, die Auskunft darüber geben, ob für diese Messreihe schon aufbereitete Up- und Downloadbandbreitenschätzungen vorliegen. Die Tabelle kann man durch Anklicken der Überschriften in vielfacher Weise sortieren. Ferner kann man eine beliebige Anzahl von Zeilen beliebig kombiniert auswählen, die man dann weiter untersuchen kann.

Abschnitt 2 des Fensters enthält eine Reihe von Buttons, die es ermöglichen Auswertungen durchzuführen. Das Betätigen von `Calc Upload/Download BW` löst die Berechnung der Upload- und Downloadbandbreiten für die ausgewählte(n) Messreihe(n) aus. Diese Berechnung kann je nach Messdauer einige Zeit in Anspruch nehmen, daher werden einmal berechnete Werte in einer eigenen Datenbanktabelle vorgehalten. Ein Druck auf `Show Raw Upload/Download BW` öffnet weitere Fenster, die die unbehandelten Bandbreitenmessdaten zeigen. Dies ermöglicht eine gezielte Analyse der Bandbreitenmessdaten. `Show Download/Upload BW` schließlich öffnet ein weiteres Fenster, das einem die aufbereiteten Bandbreitenmessdaten für die ausgewählten Messreihen anzeigt (siehe Abschnitt 5.1.1).

Die mit `*BW Confidence CSV` bezeichneten Buttons generieren *.csv Dateien im Verzeichnis `SaveDir`, die mit Hilfe eines Ruby Scripts die *.csv Dateien mit den 95% Konfidenzintervallen der Messdaten generieren.

`Bandwidth CDF` erstellt *.csv Dateien, die die CDF der Up- und Downloadbandbreite enthalten. Analog dazu erstellt `LargeLatency CDF` *.csv Dateien der CDF für die Up- und Downloadlatenzen des ersten Pakets eines jeden zur Bandbreitenmessung benutzten Packettrains der ausgewählten Messreihen.

`Time Bandwidth` erstellt *.csv Dateien, die die verfügbare Bandbreite gegen die Zeit abbilden. Selbiges macht `Time LargeLatency` für die Latenzen großer Pakete, also für die Latenz des ersten Pakets eines Packettrains zur Bandbreitenmessung. `Time Trainsize` und `Time Droprate` erstellen ebenfalls *.csv Dateien, die jedoch die Packettraingröße beziehungsweise die Paketverlustrate gegen die Zeit abbilden.

5.1.1 TBUS Analyzer - Bandbreitendetailansicht

Der TBUS Analyzer stellt vier Detailansichtsfenster für die Bandbreitenmessdaten zur Verfügung: jeweils eine Ansicht für die Rohdaten der Messpakete und eine Ansicht für die ausgewerteten Bandbreitenmessdaten für jede Übertragungsrichtung.

Stellvertretend für alle Bandbreitendetailansichten stellen wir hier die Detailansicht (siehe Abbildung 5.2) für die ausgewerteten Download-Bandbreitenmessungen vor. In diesem Fenster werden

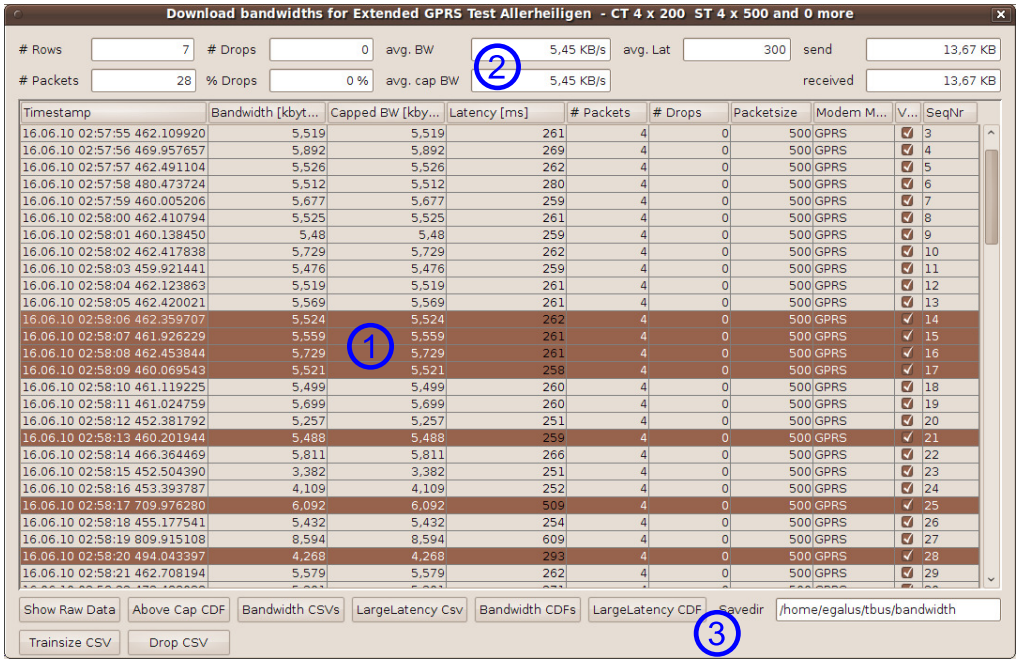

Abbildung 5.2: Bandbreitendetailansicht (in diesem Fall für den Download) des *TBUS Analyzer*.

alle verfügbaren Download Bandbreitenmessdaten zu den im Hauptfenster des TBUS Analyzer ausgewählten Messfahrten angezeigt. Auch in der Tabelle (1) in diesem Fenster kann nach jeder Spalte sortiert werden, und es können beliebige Zeilen der Tabelle ausgewählt werden. Die Anzeigen im Bereich (2) und die Buttons im Bereich (3) des Fensters beziehen sich immer nur auf die gerade ausgewählten Datensätze der Tabelle.

Jeder Tabelleneintrag enthält:

1. Den Zeitstempel, dem dieser Bandbreitenmesswert zugeordnet wird.
2. Die geschätzte Bandbreite.
3. Die durch das Technologielimit bearbeitete Bandbreite (zu hohe Werte werden als nicht gültig markiert).
4. Die Latenz des ersten Messpaketes.
5. Die Anzahl der Pakete in diesem Packettrain.
6. Die Anzahl der gedroppten Pakete dieses Packettrains .
7. Die Paketgröße der Messpakete.
8. Die eingesetzte Übertragungstechnologie.
9. Die Sequenznummer des Packettrains.
10. Ein Flag das angibt, ob diese Messung ein gültiges Ergebnis ($\leq$ Technologielimit) enthält.

Die Wahl des Zeitstempels, dem die Messung zugeordnet ist, haben wir für den Upload und den Download unterschiedlich gewählt. Für den Upload haben wir den Zeitpunkt des Versandes des ersten Messpaketes gewählt, da dieses daraufhin die Luftschnittstelle, unseren designierten Flaschenhals, benutzt. Für die Downloadpackettrains haben wir dagegen den Zeitpunkt des Eintreffens des ersten Messpaketes beim Mobilgerät als Messzeitpunkt festgesetzt. Die als für diese Messreihe gültig ausgewählte Übertragungstechnik haben wir aus der CNTI-Tabelle (vergleiche Kapitel 4.2.1) der Datenbank entnommen. Dabei haben wir den letzten gültigen Eintrag in der Datenbank ausgewählt, der an dem Zeitstempel des Packettrains oder möglichst dicht davor, existiert.

Teil (2) des Fensters enthält eine Übersicht der in der Tabelle ausgewählten Datensätze:

#Rows gibt an, wieviele Zeilen ausgewählt wurden.

#Packets gibt an, wieviele Pakete zu den ausgewählten Messreihen gehören.

#Drops gibt an, wieviele Pakete den Messpartner nicht erreicht haben.

%Drops gibt an, wieviel Prozent der Pakete ihr Ziel nicht erreicht haben.

avg. BW gibt die durchschnittliche Bandbreite der nicht bereinigten Bandbreitenmesswerte an.

avg. cap BW gibt die durchschnittliche Bandbreite der gültigen Bandbreitenmessungen an.

avg. Lat gibt die durchschnittliche Latenz der ersten Bandbreitenmesspakete an.

send gibt an, wieviele Daten versendet wurden.

received gibt an, welche Datenmenge den Empfänger erreicht hat.

In Teil (3) des Fensters können analog zu dem Hauptfenster wieder eine Reihe von *.csv Dateien im unter `SaveDir` angegebenen Verzeichnis zur weiteren Analyse erstellt werden, diesmal allerdings beschränkt auf die in diesem Fenster ausgewählten Tabellenzeilen der Downloadbandbreitendaten.

Kapitel 6

Simulationsmodell

Nachdem wir in den vorigen Kapiteln ausführlich auf die Erstellung der Traces eingegangen sind, stellen wir in diesem Kapitel mögliche Simulationsmodelle vor. In Abschnitt 6.1 geben wir zunächst einen Überblick, wie wir aus den erstellten Traces eine Datenbasis für unsere Simulation erstellen können. Anschließend beschreiben wir in 6.2 ein einfaches Simulationsmodell, das auf dieser Datenbasis aufsetzt. Abschließend diskutieren wir am Ende dieses Kapitels, in Abschnitt 6.4, Erweiterungen und Verbesserungsmöglichkeiten unseres Simulationsmodells.

6.1 Datenaufbereitung

Zunächst müssen wir aus unseren gesammelten Traces eine Datenbasis für die Simulation generieren. Hierzu werden die Messdaten zunächst auf Kartenmaterial abgebildet, das danach in Messsegmente eingeteilt wird, in denen die Daten mehrerer Messpunkte aus einem oder mehreren Messfahrten zu einem repräsentativen Wert zusammengefasst werden.

6.1.1 Map-Matching

Die Ungenauigkeit der Positionsbestimmung mittels GPS führt dazu, dass viele der gespeicherten Längen- und Breitengrad Paare nicht auf, sondern neben der von uns während der Messfahrt befahrenen Straße liegen. Wir müssen also einen Weg finden, um diese Positionierungsfehler weitestgehend zu eliminieren (vergleiche Abbildung 6.1). Das gleiche Problem müssen auch alle GPS-gestützten Navigationssysteme lösen. Daher existieren bereits eine Reihe von *Map-Matching* Verfahren, die mit unterschiedlichen Mitteln versuchen, GPS-Messwerte für Längen- und Breitengradpaare auf vorhandene (Straßen-)Karten abzubilden.

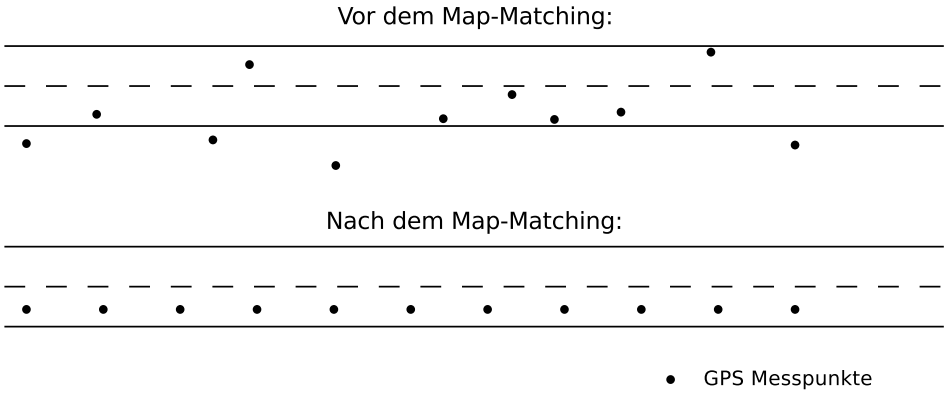

Abbildung 6.1: Vorher-Nachher-Beispiel eines Map-Matching Verfahrens unter der Annahme gleichbleibender Bewegungsgeschwindigkeit.

In [QON07] geben die Autoren einen Überblick über aktuelle Map-Matching Algorithmen. Da das Map-Matching für unsere Daten im Nachhinein erfolgt, umgehen wir viele Zuordnungsprobleme, die real-time Verfahren aufgrund von nicht vorhandenem Zukunftswissen haben. So lässt sich im Nachhinein an einer Kreuzung viel einfacher der Straßenabschnitt finden, auf dem der Weg fortgesetzt wird als in Echtzeit, in der eine Prognose getroffen werden muss und sich die GPS-Ungenauigkeit negativ auf die Entscheidung auswirken kann. Einen vielversprechenden offline Ansatz stellen die Autoren in [MHA04] vor.

6.1.2 Zuordnung der Messdaten

Nachdem die Positionsdaten unserer Traces mit Hilfe eines Map-Matching Verfahrens korrigiert wurden, gilt es nun verschiedene Traces der gleichen Strecke zu einer Datenbasis zusammenzufassen. Dabei müssen folgende Probleme beachtet und gelöst werden:

1. Nur durch Zufall werden Messpunkte von zwei Traces, die die gleiche Messstrecke beschreiben, exakt die gleichen Koordinaten haben (vergleiche das Beispiel in Abbildung 6.2).

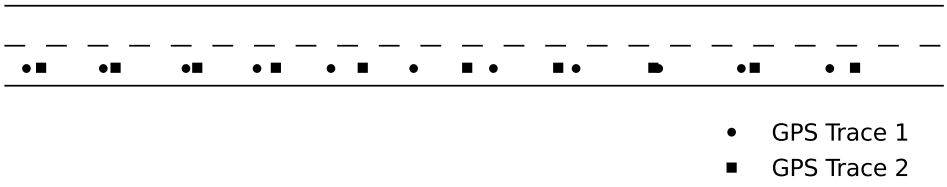

Abbildung 6.2: Versatz verschiedener Traces auf dem selben Straßenabschnitt.

2. Wie Abbildung 6.3 veranschaulicht, können wir nicht alle Messungen zeitgleich durchführen. So haben wir zum Beispiel keinen Einfluss darauf, wann die NMEA Datensätze und die Band-

breitenmesspakete unser Messsystem erreichen. Daher existiert beispielsweise weder zu jedem Bandbreitenmesspaket eine zeitgleiche Positionsangabe, noch sind alle Daten des UMTS-Modems zu diesem Zeitpunkt verfügbar.

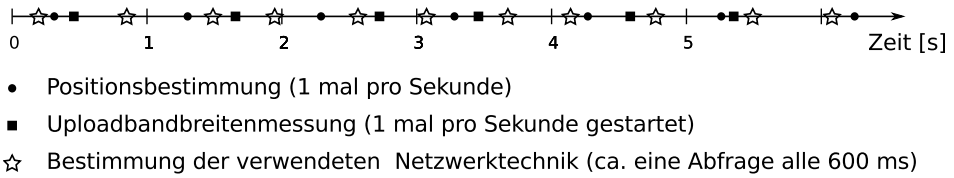

Abbildung 6.3: Veranschaulichung der Zeitunterschiede der Messdatenerfassung anhand einiger Beispieldaten.

3. Es ist nicht zu erwarten, dass zu unterschiedlichen Messzeiten an den selben Orten der Messstrecke jeweils exakt die gleichen Netzwerkbedingungen angetroffen werden. Schließlich hängt die Auslastung des Netzwerkes zum Beispiel entscheidend davon ab, wieviele Personen das Mobilfunknetzwerk gerade benutzen. Und das dürften zum Beispiel um 4:00 Uhr nachts erheblich weniger sein als um 16:00 Uhr.

4. In [YRZ+08] haben die Autoren nachgewiesen, dass der Upload-Traffic in Mobilfunknetzwerken selbstähnlich ist – der Download Traffic wurde von ihnen nicht untersucht. Daher müssen wir davon ausgehen, dass wir auch kurzfristige Einbrüche in der verfügbaren Bandbreite auf Grund von Traffic Bursts anderer Benutzer sehen werden.

6.1.2.1 Unterteilung des Straßennetzes in Segmente

Unser Vorschlag zur Lösung der Punkte 1 und 2 ist es, eine geschickte Unterteilung des Straßennetzes in Straßenabschnitte vorzunehmen und die Messwerte nicht nach absoluten Koordinaten zu gruppieren, sondern nach diesen Straßenabschnitten (vergleiche Abbildung 6.4). Ziel ist es, später für jeden Straßenabschnitt eine Aussage darüber treffen zu können, welche charakteristischen Mobilfunkeigenschaften auf ihm gelten.

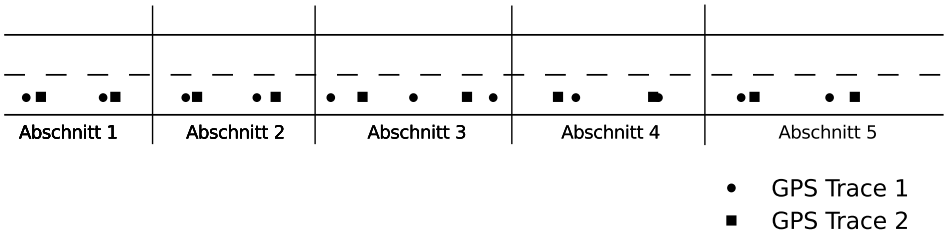

Abbildung 6.4: Einteilung einer Straße in Teilstücke zur Messdatenzusammenfassung.

Dabei spielt die Art der Unterteilung eine entscheidende Rolle. So ist es zum Beispiel wenig sinnvoll eine Straße, auf der im Schnitt 30 km/h gefahren werden, in 5-Meter Segmente zu unterteilen, da ein Fahrzeug, das 30 km/h fährt in einer Sekunde mehr als 8 Meter zurücklegt. Dadurch würden entweder Straßenabschnitte ohne Messdaten entstehen, oder es könnte die Situation eintreten, dass wir im einen Abschnitt Messdaten von Trace 1 verwenden, im nächsten von Trace 2, dann wieder Trace 1 und so weiter. Dies gilt es auf Grund von Punkt 3 unbedingt zu vermeiden, denn sonst könnten wir unserer Simulation ungewollt oszillierende Netzwerkzustände für eine Straße zuweisen, nur weil der eine Trace während starker Mobilfunknutzung und der andere bei wenig belastetem Mobilfunk erstellt wurde.

Ebensowenig ist es wünschenswert zu große Straßensegmente zu erzeugen, denn dann steigt die Gefahr, dass wir Orte mit normalerweise geringer Netzwerklast mit Orten hoher Netzwerklast in einem Segment zusammenfassen, was das Ergebnis der Simulation in ungewollter Weise beeinflussen könnte.

Welches Verfahren für die Unterteilung in Straßensegmente am besten geeignet ist, muss noch untersucht werden. Mögliche – teilweise miteinander kombinierbare – Kriterien für eine Unterteilung sind zum Beispiel:

- Grenzen der Mobilfunkzellen auch als Teilungspunkte nutzen.
- Regionen bestimmen, an denen Zellwechsel oder gar Technologiewechsel stattfinden und diese als eigene Straßensegmente betrachten.
- Einteilungen anhand der verfügbaren Bandbreiten oder Latenzen. So gibt es zum Beispiel Straßensegmente entlang von großen Haltestellen des öffentlichen Nahverkehrs, die (nicht unerwartet) bei den Messungen durch geringe verfügbare Bandbreiten aufgefallen sind.
- Straßenkreuzungen könnten als weitere Teilungspunkte dienen. Es kann sich aber auch herausstellen, dass es sinnvoller ist, ganze Kreuzungsbereiche als eigenes Straßensegment zu betrachten.

Schon an der Zahl der genannten möglichen Parameter ist ersichtlich, dass hier einiger Forschungsbedarf besteht.

6.1.2.2 Bildung der Simulationsdatenbasis

Sind die gesammelten Traces wie in 6.1 mit einem Map-Matching Verfahren behandelt und anschließend wie in 6.1.2.1 erläutert in Straßensegmente eingeteilt worden, so müssen sie nun noch segmentweise aggregiert werden. Das heißt, wir müssen insbesondere die Punkte 3 und 4 aus 6.1.2 beachten.

Ziel dieses Abschnittes ist es also, alle Messdaten, die innerhalb eines Straßensegmentes liegen, zu einem Datensatz zusammenzufassen.

Auch diese Aufgabe ist alles andere als trivial und im Rahmen dieser Arbeit nicht abschließend zu klären.

Als naive Anfangslösung wäre eine einfache Mittelwertsbildung mit 95% Konfidenzintervallangaben für die Netzwerkparameter in den Segmenten denkbar, in denen nach den Messdaten kein Technologiewechsel stattfindet. Aussagen über Segmente mit Technologiewechsel – diese kamen bei unseren stark begrenzten Messfahrten im Freien im Großraum Neuss/Düsseldorf nie vor – mit Hilfe einfacher Mittelwertbildung über alle Messwerte des Segmentes dürften dagegen eher wenig Sinn machen. Hier wäre wahrscheinlich eine Mittelwertbildung ausschließlich über Messwerte der 'schlechteren' Technologie, quasi als untere Grenze für die Netzwerkcharakteristiken, die bessere Wahl.

Bei genügend Messwerten pro Segment könnte man auch versuchen, 'offensichtliche' Ausreißer in den Messwerten, die zum Beispiel durch die Bursts, die durch die Selbstähnlichkeit des Netzwerkverkehrs verursacht werden, auszufiltern

Alternativ ist auch denkbar, bei jeder Anfrage für ein Segment zufällig ein vorhandenes Messwertset (Bandbreite, Latenz, Paketverlustrate usw.) dieses Segmentes auszuwählen und für die Simulation zu nutzen.

6.2 Entwurf eines einfachen Simulationsmodells

Eine erste Fassung des Simulationsmodells unter Verwendung der in Abschnitt 6.1 erstellten Simulationsdatenbank stellen wir uns als Replay-Simulation vor, die reine Client-Server Kommunikation simuliert.

Für das Simulationsmodell haben wir folgende Annahmen getroffen:

1. Der Flaschenhals bei den Übertragungen ist stets die Luftschnittstelle.
2. Die Netzwerkcharakteristiken werden durch unsere zu testende Anwendung im Testgebiet nicht signifikant beeinflusst.

Sind diese Bedingungen erfüllt, so können wir mit Hilfe unserer Datenbank und dem folgenden runden-basierten Simulationsverfahren Aussagen darüber treffen, welche Daten wie lange im Netzwerk unterwegs sind und welche Datenmengen anderweitig simulierte Versuchsfahrzeuge (zum Beispiel durch den Verkehrssimulator *sumo*) zu welcher Zeit senden und empfangen können.

In einem runden-basierten Simulationsverfahren wird die Zeit in Runden eingeteilt. Das heißt, dass die Zeit in einer solchen Simulation schrittweise vergeht. In jedem Simulationsschritt werden dabei Funktionen ausgeführt, die alle zur gleichen Simulationszeit stattfinden und die Zeitspanne – die Rundenzeit – simulieren, die eine Runde dauert. Eine solche Funktion simuliert also alles, was während einer Rundenzeit passiert. Hierbei spielt die Frequenz, das heißt die Anzahl der simulierten Zeitabschnitte pro Sekunde, eine große Rolle. Wählt man die Frequenz zu klein, dann wird die Simulation ungenau. Wählt man die Frequenz zu hoch, dann werden die Rundenfunktionen zu oft ausgeführt und die Simulation skaliert sehr schlecht.

Obwohl uns bewusst ist, dass in den meisten Fällen eine runden-basierte Simulation einer timer- oder ereignis-basierten Simulation in Genauigkeit und Skalierbarkeit unterlegen ist, haben wir aus Gründen der Veranschaulichung des Funktionsprinzips unseres Simulationsmodells ein runden-basiertes Modell gewählt. In Abschnitt 6.4.1 erklären wir, was geändert werden muss um die gleiche Funktionsweise bei höherer Genauigkeit der Simulationsergebnisse mit einem ereignis-basierten Verfahren zu erreichen.

6.2.1 Simulation des Uploads

Für die Simulation der Bandbreitenbeschränkung und der Paketverluste im Upload, also der Daten, die vom Client zum Server übertragen werden, verfügt jeder Client über eine eigene *Client Bandbreiten Sendewarteschlange (CBSQ)*. Für die Simulation der Latenz (abgesehen von congestion in der Bandbreitenwarteschlange) erhält jeder Client zusätzlich eine *Client Latenz Sendewarteschlange (CLSQ)* (vergleiche Abbildung 6.5).

Für jede CBSQ wird einmal pro Simulationsrunde mit Hilfe des in Listing 6.1 dargestellten Pseudocodes ermittelt, welche Pakete in dieser Runde versendet werden konnten oder verloren gegangen sind.

`CBSQ.head` zeigt hierbei immer auf den Kopf der Warteschlange oder ist NULL, wenn die Warteschlange gerade leer ist. Die Anweisungen in den Zeilen 2 bis 6 werden genau dann ausgeführt, wenn die Warteschlange leer ist. Zum einen wird in diesem Fall die `Gesamtkapazitaet` auf 0 gesetzt, da nun keine angesammelte Kapazität mehr für nachfolgende Pakete genutzt werden kann. Zum anderen wird die `LossSumme`, aus der sich zusammen mit den `WarteRunden` die Paketverlustrate berechnen lässt, zurückgesetzt. Außerdem wird der Wert für die `WarteRunden`, die angeben wie lange das erste Paket schon in der Warteschlange verweilen muss weil die angesparte `Gesamtkapazitaet` noch nicht für den Versand des Paketes reicht, ebenfalls auf Null gesetzt.

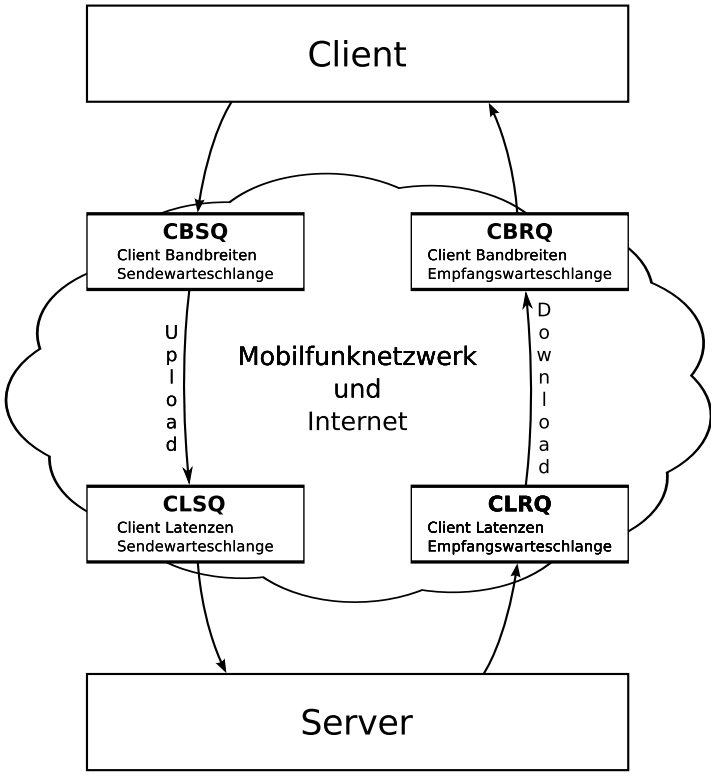

Abbildung 6.5: Schematische Darstellung des Simulationsmodells.

Die Zeilen 8-39 werden dagegen genau dann ausgeführt, wenn mindestens ein Paket in der Warteschlange vorhanden ist. Zunächst wird dann einmalig die `Gesamtkapazitaet` um die für die aktuelle Clientposition und die llänge der `Rundenzeit` anteilmäßig von der Datenbank vorgegebene Uploadbandbreite erhöht (`DatenbankUploadBandbreite * Rundenzeit`), die Anzahl der Warterunden um eins inkrementiert und die `Losssumme` um die Paketverlustrate an der aktuellen Clientposition (`DatenbankUploadLoss`) erhöht. Anschließend wird der Inhalt der `while`-Schleife (Zeilen 15 bis 32) so lange abgearbeitet, bis entweder kein weiteres Paket mehr in der Warteschlange ist, oder das erste Paket in der Warteschleife größer ist als die aktuelle `Gesamtkapazitaet`.

Die Zeilen 17 und 18 im Inneren der `while`-Schleife behandeln zunächst den Fall des Paketverlusts. Dieser tritt in unserer Simulation immer genau dann auf, wenn ein gleichverteilter Zufallswert zwischen 0 und 1 kleiner oder gleich dem Mittelwert der Paketverlustrate (ebenfalls ein Wert zwischen 0 und 1) über die gesamte Wartezeit des ersten Paketes in der Warteschlange ist (Zeile 16). Tritt ein Paketverlust ein, so wird das Paket aus der Warteschlange entfernt. Die Paketgröße wird aber nicht von der `Gesamtkapazitaet` abgezogen. Dieses Vorgehen erscheint zunächst falsch, jedoch haben wir bei den Messungen der verfügbaren Bandbreite die zur Messzeit aufgetretenen Paketverluste schon berücksichtigt (vergleiche Abschnitt 3.4). Eine Verkleinerung des Wertes der Variablen `Gesamtkapazitaet` bei aufgetretenem Paketverlust würde dazu führen, dass wir nicht mehr die

gemessene verfügbare Bandbreite simulieren, sondern eine geringere Bandbreite.

Listing 6.1: Pseudocode der Routine der Client Bandbreiten Sendewarteschlange (CBSQ), die in jeder Simulationsrunde einmal abgearbeitet wird

```
1  if (CBSQ.head == null) {
2      //wenn die Warteschlange leer ist, setzen wir alle
3      //Werte zurueck
4      Gesamtkapazitaet = 0;
5      LossSumme = 0;
6      WarteRunden = 0;
7  } else {
8      //es befindet sich mindestens ein Paket in der Warteschlange
9      Kapazitaet = DatenbankUploadBandbreite * Rundenzeit;
10     Gesamtkapazitaet = Gesamtkapazitaet + Kapazitaet;
11     WarteRunden++;
12     LossSumme = LossSumme + DatenbankUploadLoss;
13     while (CBSQ.head != null
14             && CBSQ.head.packetsize() <= Gesamtkapazitaet) {
15         //pruefen ob das Paket verloren gehen soll
16         if (Math.random() <= LossSumme / WarteRunden) {
17             //Paketverlust
18             CBSQ.removeHead();
19         } else {
20             //speichere den fruehesten Zustellzeitpunkt fuer das headpaket
21             CBSQ.head.fruehesterZustellzeitpunkt = AktuelleSimulationsZeit +
                   DatenbankUploadLatenz - (WarteRunden * Rundenzeit);
22             //fuege das Paket in die Latenz Sendewarteschlange ein
23             CLSQ.add(CBSQ.head);
24             //ziehe die verbrauchte Bandbreite ab
25             Gesamtkapazitaet = Gesamtkapazitaet - CBSQ.head.packetsize();
26             //entferne das Paket aus dieser Warteschlange
27             CBSQ.removeHead();
28             //setze den Summenrundenloss auf den an der aktuellen Position
29             //gueltigen Loss
30             LossSumme = DatenbankUploadLoss;
31             WarteRunden = 1;
32         }
33     }
34     if (CBSQ.head == null) {
35         //alle Werte zuruecksetzen
36         Gesamtkapazitaet = 0;
37         LossSumme = 0;
38         WarteRunden = 0;
39     }
40 }
```

Ergibt die Auswertung der Zufallszahl, dass kein Paketverlust stattfindet, so werden die Zeilen 20 bis 30 abgearbeitet. Hier wird zunächst der frühest mögliche Zustellzeitpunkt für das aktuell erste Paket der Warteschlange als Zeitstempel (`fruehesterZustellzeitpunkt`) bestimmt, der sich aus der aktuellen simulierten Zeit (`AktuelleSimulationsZeit`) zuzüglich der für die aktuelle Clientposition gültigen Uploadlatenz (`DatenbankUploadLatenz`) abzüglich der Wartezeit, die das Paket als Kopf der Warteschlange warten musste (`WarteRunden` $\cdot$ `Rundenzeit`), zusammensetzt. Anschließend (Zeile 23) wird das Paket mit diesem Zeitstempel in die Client Latenz Sendewarteschlange (CLSQ) eingefügt und die `Gesamtkapazitaet` um die Paketgröße verringert (Zeile 25) und es gilt $0 \leq$ Gesamtkapazitaet $<$ DatenbankUploadBandbreite. Anschließend werden die `LossSumme` auf die an der aktuellen Clientposition gültige Paketverlustrate und die Warterunden auf 1 gesetzt (Zeilen 30 und 31), denn das jetzt (falls vorhanden) erste Paket der Warteschlange hat bisher noch nicht als erstes Paket in der Warteschlange gewartet.

Die Zeile 34 des Quelltextes wird erst dann erreicht, wenn entweder kein Paket mehr in der Warteschlange ist, oder die aktuelle `Gesamtkapazitaet` nicht mehr ausreicht um das erste Paket der Warteschlange zu verschicken. Sollte sich nun kein Paket mehr in der Warteschlange befinden, so werden analog zu den Zeilen 3 bis 6 einige Werte zurückgesetzt.

Auch für die Client Latenz Sendewarteschlange (CLSQ) gibt es eine Routine, die einmal pro Runde abgearbeitet wird (siehe Listing 6.2). Sie sorgt dafür, dass immer nur der Kopf der Warteschlange entfernt werden kann. Allerdings muss dafür zusätzlich die Simulationszeit (`AktuelleSimulationsZeit`) die früheste Zustellzeit dieses Paketes erreicht oder überschritten haben (Zeilen 1 und 2). Erst dann gilt das Paket als zugestellt (Zeile 5) und wird der nächst höheren OSI-Schicht, der Transportschicht (vergleiche Abbildung 6.6 in Abschnitt 6.3), übergeben. Anschließend wird das Paket aus der CLSQ gelöscht (Zeile 6). Da immer nur der Kopf der Latenzwarteschlange entfernt, und damit zugestellt, werden kann, ist in dieser Simulation keine out-of-order Paketzustellung simulierbar, das heißt Pakete können sich im Netzwerk nicht überholen. Dies kann allerdings kaum als Einschränkung gesehen werden, da wir bei keiner der durchgeführten Testfahrten auch nur ein Paket out-of-order empfangen haben. Unsere Messpakete kamen entweder in-order beim Empfänger an oder gingen verloren.

Listing 6.2: Pseudocode der Routine der Client Latenz Sendewarteschlange (CLSQ), die in jeder Simulationsrunde einmal abgearbeitet wird

```
1 while (CLSQ.head != null &&
2         CLSQ.head.fruehesteZustellzeit <= AktuelleSimulationsZeit)
3 {
4     //der Server empfaengt jetzt das Paket
5     Server.empfaengt(CLSQ.head);
6     CLSQ.removeHead();
7 }
```

Natürlich hat die CBSQ auch Einfluss auf die Latenzen der gesendeten Pakete. Zum einen wird der Versand von Paketen nur alle $\frac{1}{\text{Anzahl der Runden pro Sekunde}}$ s durchgeführt, was zu einer durchschnittlichen Latenzerhöhung von $\frac{1}{2 \cdot \text{Anzahl der Runden pro Sekunde}}$ s für jedes Paket führt. Hier zeigt sich, dass die Rundenfrequenz einen bedeutenden Einfluss auf die Simulation hat. Zum anderen beschränkt die verfügbare Bandbreite die Anzahl der Pakete, die die CBSQ verlassen. Damit kann eine gefüllte CBSQ, wie im Realfall auch, die Latenz der Pakete durch congestion erhöhen.

Der Versand eines Paketes vom Client zum Server wird also wie folgt simuliert:

1. Das Paket wird an die CBSQ hinten angefügt.
2. Sobald das Paket zum Kopf der Bandbreiten Sendewarteschlange geworden ist und sobald genügend Bytes übertragen werden können (Gesamtkapazitaet), wird es an die Latenz Sendewarteschlange (CLSQ) mit dem Zeitstempel AktuelleSimulationsZeit + DatenbankUploadlatenz − WarteRunden ∗ Rundenzeit als frühesten Zustellzeitpunkt angehängt.
3. Ist das Paket zum Kopf der CLSQ geworden und die simulierte Zeit hat den frühesten Zustellzeitpunkt erreicht oder überschritten, so gilt das Paket als zugestellt und wird der nächst höheren OSI-Schicht des Servers übergeben.

6.2.2 Simulation des Downloads

Die Simulation der Downloadrichtung, also der Übertragung vom Server zu einem Client, erfolgt nahezu analog zu der Simulation des Uploads (vergleiche Abbildung 6.5). Diesmal heißen die Warteschlangen, die jeder simulierte Client besitzt, jedoch *Client Latenz Empfangswarteschlange (CLRQ)* und *Client Bandbreiten Empfangswarteschlange (CBRQ)*. Allerdings wird bei der Simulation des Downloads die Reihenfolge der Warteschlangen umgedreht. Auf ausführliche Listings verzichten wir hier. Stattdessen beschreiben wir kurz die Änderungen.

Der Pseudocode der CLRQ ist bis auf Zeile 5 (und den Austausch jedweden Vorkommens von CLSQ durch CLRQ) identisch zum Pseudocode der CLSQ (vergleiche Listing 6.1). Zeile 5 wird ersetzt durch `CBRQ.add(CLRQ.head);`, da bei der Simulation des Uploads zuerst die Latenzwarteschlange abgearbeitet wird und die Pakete erst danach in die Bandbreitenwarteschlange übertragen werden.

Auch die Rundenroutine der Bandbreiten-Empfangswarteschlange (CBRQ) ist nahezu identisch mit der der CBSQ. Es wird jedoch jedes Vorkommen von CBSQ durch CBRQ ersetzt und alle Datenbankabfragen beziehen sich nun auf den Download und nicht auf den Upload. Zeile 21 wird gestrichen und Zeile 23 wird durch `Client.empfaengt(CBRQ.head);` ersetzt.

Auch in der Downloadrichtung gibt es keine Simulationsmöglichkeit für out-of-order Paketzustellungen.

Der Versand eines Paketes vom Server zu einem Client wird wie folgt simuliert:

1. Das zu versendende Paket wird mit dem `fruehestensZustellzeitpunkt` als Zeitstempel an die CLRQ angehängt.
2. Ist das Paket an erster Stelle in der Latenzwarteschlange und die aktuelle Simulationszeit hat den frühesten Zustellzeitpunkt erreicht oder passiert, so wird das Paket aus der Latenzwarteschlange (CLRQ) entfernt und an die Bandbreitenwarteschlange (CBRQ) angehängt.
3. Sobald das Paket zum Kopf der Bandbreiten Empfangswarteschlange geworden ist, und genügend Bytes übertragen werden können (Gesamtkapazitaet), wird entschieden, ob das Paket zugestellt wird oder verloren geht. Im Fall der Zustellung gilt es unmittelbar als zugestellt und wird der nächst höheren OSI-Schicht des Empfangsclients übergeben.

Die Korrektur der Wartezeit in der Latenzwarteschlange um die Zeit, die das Paket als Kopf der Bandbreitenwarteschlange warten musste, ist im Upload kein Problem, da wir die Latenzwarteschlange hinter die Bandbreitenwarteschlange geschaltet haben. Im Download durchlaufen Pakete allerdings zuerst die Latenzwarteschlange (CLRQ) und erst danach die Bandbreitenwarteschlange (CBRQ). Daher können wir die Latenz nach dem Durchlaufen der Bandbreitenwarteschlange nicht mehr beeinflussen. Wir müssten also beim Einfügen des Paketes in die Latenzwarteschlange (CLRQ) schon wissen, welche Übertragungsverzögerung (*transmission delay*) über die Luftschnittstelle (dies entspricht in unserer Simulation der Wartezeit als Kopf der Bandbreitenwarteschlange) dieses Paket in der Zukunft haben wird. Das ist natürlich nicht möglich. Daher können wir an dieser Stelle einen kleinen Fehler in der Simulation nicht verhindern, da wir die Übertragungsverzögerung im Upload schätzen müssen. Daher schlagen wir vor, den `fruehestenZustellzeitpunkt` für die Downloadrichtung mit der Formel

$$\text{fruehesterZustellzeitpunkt} = \text{AktuelleSimulationsZeit} + \text{DatenbankDownloadlatenz} - \frac{\text{Paketgröße}}{\text{DatenbankDownloadbandbreite}} \tag{6.1}$$

zu schätzen. Hierdurch kann es aber dazu kommen, dass wir uns verschätzen. Gilt für die Downloadbandbreite des Client beim Absenden eines Paketes mit einer Größe von 1000 Byte am Server zum Beispiel eine Bandbreite von 50000 Byte/s, so würden wir die Übertragungsverzögerung auf 20 ms schätzen. Bewegt sich der Client nun aber während das Paket die Latenzwarteschlange passiert und zum Kopf der Bandbreitenwarteschlange wird, dann ergibt sich für die wirkliche Übertragungsverzögerung gegebenenfalls ein anderer Wert. Stehen jetzt zum Beispiel nur noch 25000 Byte/s Downloadbandbreite zur Verfügung, so müsste das Paket 40 ms Übertragungsverzögerung warten und wäre damit 20 ms länger unterwegs als die Datenbank eigentlich vorgibt. Andersherum kann es aber auch

passieren, dass nun 100000 Byte/s Datenrate zur Verfügung stehen und das Paket schon nach 10 ms die Bandbreitenwarteschlange verlässt, dann wäre das Paket 10 ms schneller als die Datenbank es voraussagt.

Da wir davon ausgehen, dass unser Engpass die Luftschnittstelle ist und dieser Engpass vor allem die verfügbare Bandbreite – bei congestion natürlich auch die Latenz – beeinflusst, ist ein Umdrehen der Reihenfolge der Warteschlangen auch keine Lösung. Würden wir im Download zuerst die Bandbreite und dann die Latenz simulieren, so könnte es gut möglich sein, dass ein Client an einer Position, an der bei den Messfahrten immer eine geringe Datenrate vorlag, in der Simulation mehr Daten erhält, als es die Übertragungstechnik überhaupt zulassen würde. Hierfür bräuchte es nur vorher eine hohe Datenrate und eine ständig abnehmende Latenz.

Würde die Übertragungsverzögerung nicht geschätzt, so würden wir im Download alle Pakete $\frac{\text{Paketgröße}}{\text{DatenbankDownloadbandbreite}}$ Sekunden später zustellen als es die Messwerte ausdrücken. Bei durchschnittlich 50kByte/s verfügbarer Bandbreite würden wir also im Mittel die Latenz aller Downloadpakete um ca. 20 ms erhöhen.

6.2.3 Simulationsreihenfolge

Damit die Latenzen und Bandbreiten die die Simulationsdatenbank für ein Straßensegment vorgibt eingehalten werden, spielt die Reihenfolge, in der die Warteschlangenrundenalgorithmen abgearbeitet werden, eine wichtige Rolle. In Uploadrichtung müssen in jedem Client zuerst alle diese Runde an unsere Simulationsschicht übergebenen Pakete in die CBSQ eingefügt werden. Danach muss der CBSQ-Rundenalgorithmus (Listing 6.1) ausgeführt werden. Erst danach sollte der CLSQ-Rundenalgorithmus (Listing 6.2) ausgeführt werden. Hier spielt die Reihenfolge allerdings nur dann eine Rolle, wenn die CLSQ Warteschlange leer ist und an der aktuellen Clientposition eine Latenz von 0 herrscht.

In der Downloadrichtung (vom Server zum Client) spielt die Reihenfolge der Queueabarbeitung aber eine entscheidende Rolle. Eine Vertauschung würde hier bei leerer CBRQ zu einer Erhöhung der Latenzen um eine Rundenzeit führen.

Am Anfang einer Runde müssen alle Pakete, die diese Runde vom Server verschickt werden sollen, in die CLRQ des Zielclients eingetragen werden. Danach muss der CLRQ-Rundenalgorithmus ausgeführt werden und erst danach der CBRQ-Algorithmus. Es spielt allerdings keine Rolle, ob erst alle CLRQ-Algorithmen aller Clients und danach alle CBRQ-Algorithmen abgearbeitet werden oder ob die Abarbeitung clientweise erfolgt.

6.3 Vollständige Netzwerksimulation

Unser in Abschnitt 6.2 vorgestelltes Simulationsmodell behandelt lediglich die Simulation der Datenübertragung über die Luftschnittstelle, das Core-Netzwerk und eine kurze Strecke des Internets zwischen Client und Server. Es simuliert also nur die OSI-Schichten 3 bis 1: Vermittlungsschicht, Sicherungsschicht und Bitübertragungsschicht. Wir simulieren mit unserem Modell also lediglich die in Abbildung 6.6 rot markierten Schichten.

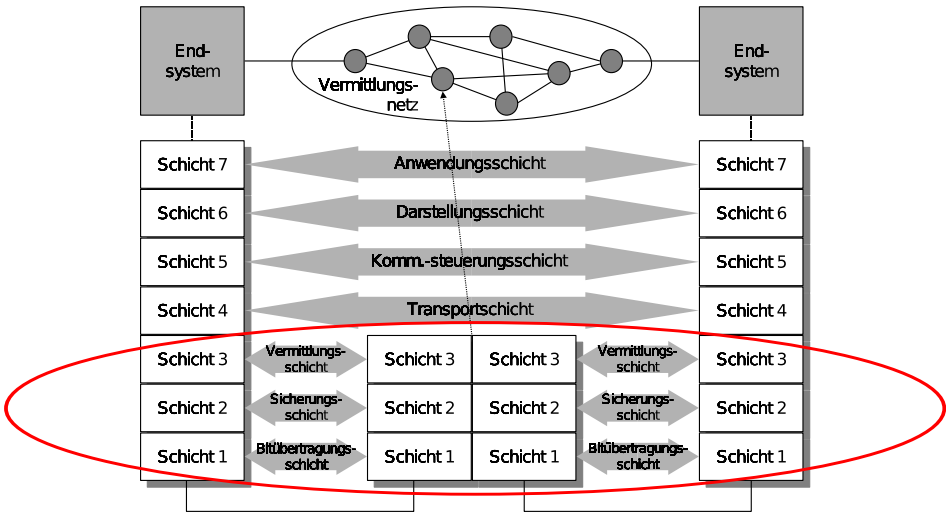

Abbildung 6.6: OSI-Schichtenmodell [Mau06], in dem die durch unser Simulationsmodell simulierten Netzwerkschichten rot umkreist sind.

Für eine Simulation von reinem UDP Datenverkehr würde dieses Vorgehen schon ausreichen. Um jedoch eine vollständige Netzwerksimulation zu erhalten, muss zusätzlich noch die Transportschicht simuliert werden. Da es für diese Aufgabe schon langerprobte Simulatoren gibt, schlagen wir vor, unser Simulationsmodell zur Simulation der Schichten 1 bis 3 in einem Netzwerksimulator, zum Beispiel *ns-2* [NS-], zu implementieren. So kann dieser mit seinen vorhandenen Funktionen die Transportschicht – und damit alle in ihm implementierten Netzwerkprotokolle – simulieren und greift dann zur Datenübertragung auf unser Simulationsmodell zurück.

6.4 Erweiterungs- und Verbesserungsmöglichkeiten für das Simulationsmodell

Unser einfaches Simulationsmodell ist die Grundlage für vielschichtige Verbesserungs- und Erweiterungsmöglichkeiten. Einige davon wollen wir hier als Ausblick vorstellen.

6.4.1 Änderung in ein ereignis-basiertes Simulationsmodell

Wie in Abschnitt 6.2 angekündigt, zeigen wir an dieser Stelle am Beispiel des Uploads welche Änderungen an unserem einfachen Simulationsmodell nötig sind um es in eine ereignis-basierte Simulation zu konvertieren. Dabei wird ersichtlich, dass diese Umwandlung zwar möglich ist, die Erklärung der Vorgehensweise aber unnötig verkompliziert hätte.

In einer ereignis-basierten Simulation gibt es eine Ereignisliste, in die neue Ereignisse eingefügt und zeitlich sortiert werden. Die Zeit springt in einer solchen Simulation immer vom Zeitpunkt des letzten Ereignisses zum Zeitpunkt des nächsten Ereignisses aus der Ereignisliste. Dies kann zu erheblichen Verbesserungen der Skalierbarkeit gegenüber runden-basierter Simulation führen, da nur dann Programmcode ausgeführt wird wenn wirklich ein zu simulierendes Ereignis eingetreten ist.

Im folgenden beschreiben wir den Versand eines Paketes vom Client zum Server inklusive aller nötigen Ereignisse.

Zunächst muss ein Ereignis ausgelöst werden, das uns mitteilt, dass ein Paket verschickt werden soll. Die Ereignisbehandlungsroutine muss dann dafür sorgen, dass die Funktion `CBSQ.add(Paket)` ausgeführt wird. Diese Funktion fügt das Paket in die Bandbreitenwarteschlange (CBSQ) ein und prüft, ob dieses Paket nun Kopf der Warteschlange ist. Ist dies nicht der Fall, so wird es einfach an die Warteschlange angefügt. Ist es jedoch der Kopf der Warteschlange, so gilt es die Übertragungsverzögerung zu simulieren. Dafür muss ein Ereignis für den Zeitpunkt des wahrscheinlichen Endes der Übertragungsverzögerung erzeugt werden. Damit aber auch Änderungen durch die Bewegung des Clients berücksichtigt werden (Bandbreiten- und Paketverluständerungen), muss außerdem ein Ereignis erzeugt werden, das ausgelöst wird wenn der Client in ein Gebiet mit anderen Netzwerkeigenschaften wechselt. Wenn dieser Wechsel zuerst eintritt, dann muss zunächst das Ereignis für den Paketversand gelöscht werden, da sich die Übertragungsverzögerung geändert hat. Danach muss ein neues Ereignis mit der aktualisierten Berechnung der Übertragungsverzögerung erstellt werden. Wird ein Paket versendet oder geht es verloren, so muss geprüft werden, ob es ein weiteres Paket in der Warteschlange gibt, für das dann ein neues Ereignis für die Simulation der Übertragungsverzögerung erstellt werden muss.

Auch in der Latenz Sendewarteschlange muss jeweils für das erste Paket der Warteschlange ein Ereignis existieren, das nach Verstreichen der Latenzwartezeit ausgelöst wird. Dieses Ereignis muss den Versand simulieren (also wieder ein Event in der nächst höheren Netzwerkschicht auslösen) und prüfen, ob ein weiteres Paket in der Warteschlange existiert, das nun zum Kopf der Warteschlange geworden ist und für dessen Latenzwartezeit wieder ein Ereignis generiert werden muss.

Die Änderungen für die Downloadrichtung müssen analog durchgeführt werden.

6.4.2 Verbesserung der Simulation von Paketverlusten

Bei der Erstellung unserer Traces haben wir die Paketverlustrate in Prozent für jeden Packettrain durch die einfache Gleichung

$$\text{Paketverlustrate} = \frac{\text{Anzahl der empfangenen Pakete des Packettrains}}{\text{nominelle Anzahl der Pakete des Packettrains}}$$

berechnet. Wie aber Abbildung 3.17 gut veranschaulicht, haben wir dabei in Bereichen mit self-induced congestion sehr wahrscheinlich nicht die Paketverlustraten, die durch Übertragungsfehler auf der Luftschnittstelle verursacht werden, gemessen. Stattdessen haben Paketverluste auf Grund von vollen Netzwerkpuffern einen deutlichen Anteil an der Paketverlustrate während der self-induced congestion.

Wären wir in der Lage, bei den Messungen in Phasen von self-induced congestion, die wir mit unserem Messverfahren nicht verhindern können, Paketverluste, die nicht durch Übertragungsverluste auf der Luftschnittstelle herrühren, auszufiltern, so würde unsere Simulation in solchen Messgebieten stark verbessert. Eine Analyse der Verteilung der Paketverluste muss aber aus Zeitmangel auf später verschoben werden.

Des Weiteren haben wir in unserem Simulationsmodell – abgesehen von Stellen in denen wir durch unsere Messdaten selbst bei den Messungen für Datenstau gesorgt haben – Paketverlust durch verworfene Pakete in Warteschlangen nicht simuliert, in diesem Bereich lässt sich die Simulation also verbessern, indem das im untersuchten Mobilfunknetzwerk verwendete Warteschlangenmodell besser simuliert und die entsprechenden Pakete bei congestion verworfen werden.

6.4.3 Simulation von Client-zu-Client Kommunikation

Wie wir schon einleitend in Kapitel 3 erklärt haben, können wir mit unseren rein auf Client-Server-Kommunikation basierenden Messdaten auch eine worst case Einschätzung für die Bandbreite und die Latenz bei direkter Client-zu-Client Kommunikation geben. Dies allerdings nur, wenn wir annehmen, dass die Mobilfunkprovider Peerings ohne Penalty bezüglich Latenz oder Bandbreite untereinander betreiben. Das heißt die beiden an der Kommunikation beteiligten Mobilfunkprovider müssen die Paketdaten untereinander ohne künstliches Bandbreitenmanagement oder künstliche Latenzerhöhung austauschen.

Wir können unser Simulationsmodell auf einfache Weise so erweitern, dass für die minimale Latenz zwischen zwei Clients, C_i und C_j und für die maximal verfügbare Bandbreite zwischen ihnen die

folgenden Gleichungen gelten:

$$\begin{aligned} l_{min}(C_i, C_j) &= l(C_i, \text{Server}) + l(\text{Server}, C_j) \\ bw_{max}(C_i, C_j) &= \min\left\{bw(C_i, \text{Server}), bw(\text{Server}, C_j)\right\} \end{aligned} \tag{6.2}$$

Hierzu müssen wir für ein Paket, das von C_i an C_j geschickt wird, den Verlauf durch die bereits eingeführten Warteschlangen nur minimal abändern.

1. Das Paket wird an die $CBSQ_i$ hinten angefügt.
2. Sobald das Paket zum Kopf der Bandbreiten Sendewarteschlange geworden ist und genügend Bytes übertragen werden können (Gesamtkapazitaet), wird es an die Latenz Sendewarteschlange ($CLSQ_i$) mit dem Zeitstempel AktuelleSimulationsZeit + DatenbankUploadlatenz$_i$ als frühesten Zustellzeitpunkt angehängt.
3. Ist das Paket zum Kopf der $CLSQ_i$ geworden und die simulierte Zeit hat den frühesten Zustellzeitpunkt erreicht oder überschritten, wird das Paket nun mit dem frühesten Zustellzeitpunkt (AktuelleSimulationsZeit + DatenbankDownloadlatenz$_j$) an die $CLRQ_j$ angehängt.
4. Ist das Paket an erster Stelle in der Latenzwarteschlange und die aktuelle Simulationszeit hat den Zustellzeitpunkt erreicht oder passiert, so wird das Paket aus der Latenzwarteschlange ($CLRQ_j$) entfernt und an die Bandbreitenwarteschlange ($CBRQ_j$) angehängt.
5. Sobald das Paket zum Kopf der Bandbreiten Empfangswarteschlange geworden ist und genügend Bytes übertragen werden können (Gesamtkapazitaet), wird entschieden, ob das Paket zugestellt wird oder verloren geht. Im Fall der Zustellung gilt es unmittelbar als zugestellt und wird der nächst höheren OSI-Schicht des Empfangsclients übergeben.

Die Simulation wird also dahingehend geändert, dass der Server diesmal quasi nur ein Router ist und die Pakete direkt, ohne weitere Verzögerung, an den richtigen Empfänger weiterleitet. Die Reihenfolge der Abarbeitung der Rundenalgorithmen muss nicht verändert werden. Im Pseudoquellcode ändert sich nur die Zeile 5 in Listing 6.2.

Leider hat diese Simulationserweiterung einen gravierenden Schwachpunkt. Solange die Bestimmung der Paketverlustraten (vergleiche Abschnitt 6.4.2) bei der Datenaggregation der Traces für die Messdatenbank nicht verbessert wird, liefert die hier beschriebene Client-zu-Client Simulation in bestimmten Situationen falsche Ergebnisse. Dies ist immer dann der Fall, wenn Kommunikation zwischen zwei Clients simuliert wird und sich mindestens einer der beiden Clients auf einem Straßensegment befindet, auf dem während der Messungen self-induced congestion aufgetreten ist. Denn dann wird in der Simulation ohne die Anpassungen aus Abschnitt 6.4.2 eine zu hohe Paketverlustrate simuliert.

6.4.4 Simulation auf Zellebene

Bei dem vorgestellten Simulationsmodell haben wir die Annahme gemacht, dass sich die Netzwerkcharakteristiken durch unsere (zu testende) Anwendung im Testgebiet nicht signifikant verändern. Durch diese Annahme haben wir ausgeschlossen, dass sich mehrere in einer Zelle befindliche simulierte Clients in irgendeiner Weise beeinflussen. Für Szenarien mit geringer Clientdichte hat diese Einschränkung nur geringen Einfluss auf die Nähe des Simulationsergebnisses zur Realität. Steigt jedoch die Clientdichte, so wird das Simulationsergebnis durch das Ignorieren der gegenseitigen Beeinflussungen innerhalb einer Mobilfunkzelle weiter von der Realität abweichen.

Daher gehen wir davon aus, dass das Simulationsmodell für hohe Clientdichten in Richtung einer zellbasierten Simulation geändert werden muss. Wie genau diese Änderung aussehen könnte, können wir hier noch nicht sagen, da dies entscheidend davon abhängt, wie sehr sich z.B. ein zweiter oder dritter Messclient in der gleichen Zelle auf die Latenz beziehungsweise Bandbreite auswirkt. Wenn unser Messclient in den Messungen die volle Kapazität als verfügbare Bandbreite ermittelt hat, so können wir relativ einfach Rückschlüsse auf die verfügbare Bandbreite bei 2 Clients in der Zelle schließen, dann dürften beide die Hälfte der Kapazität erhalten. Was aber passiert, wenn wir nur einen Bruchteil der maximal möglichen Bandbreite gemessen haben? Sehr wahrscheinlich können wir dann nicht einfach sagen, dass ein 2. Client in dieser Zelle für eine Halbierung der gemessenen Bandbreite pro Client sorgen würde. Denn damit hätten wir die pro Client verfügbare Bandbreite sehr wahrscheinlich deutlich unterschätzt, da zu dem Messzeitpunkt mehrere andere Mobilfunkteilnehmer Bandbreite verbraucht haben, die ihre Bandbreite gegebenenfalls auch noch weiter drosseln müssten, wenn unser 2. Client in der gleichen Zelle Datenverkehr erzeugt.

Ziel muss es also sein, mit Hilfe unserer Messwerte für die Bandbreite möglichst gute Rückschlüsse zu ziehen, wieviel Bandbreite für x simulierte Clients vom RNC bereitgestellt würde.

Kapitel 7

Zusammenfassung

Im Rahmen dieser Masterarbeit wurde ein Logging- und Analyseframework (TBUSLogger und TBUS Analyzer) zur Erfassung von Mobilfunkcharakteristiken erstellt. Es wurden Messfahrten im Großraum Düsseldorf durchgeführt und die hierbei erfassten Messdaten untersucht. Abschließend wurde der Entwurf eines einfachen Simulationsmodells auf Grundlage der gesammelten Daten vorgestellt.

Hierzu haben wir zunächst eine Einführung in das Thema gegeben und zugrunde liegende Arbeiten in den Bereichen (trace-basierte) Simulation, Bandbreitenmessung und Zeitsynchronisation zusammengefasst.

Anschließend haben wir die Methodik unseres Loggingverfahrens dargelegt. Dabei wurden folgende Themen intensiv behandelt:

- Mit Hilfe eines Garmin OEM 18x LVC GPS-Empfängers und eines selbstgebauten Interfaces wurde eine hochgenaue Referenzuhr zur Zeitsynchronisation unserer Messsysteme mittels NTP hergestellt.
- Das Probe Gap Model wude als Bandbreitenmessmethode für Mobilfunkmessungen durch Versuche ausgeschlossen.
- Wir haben ein eigenes packettrain-basiertes Bandbreitenmessverfahren für Mobilfunkmessungen entwickelt, das der Ortsbezogenheit unserer Messwerte Rechnung trägt und im Durchschnitt für einen Messwert pro Sekunde sowohl in Upload- als auch in Downloadrichtung sorgt.

Als nächstes haben wir das Loggingframework TBUSLogger vorgestellt, das Traces mit Latenzmessungen, Bandbreitenmessungen, Daten des GPS Sensors und Daten des UMTS Modems sammelt.

Im daran anschließenden Kapitel haben wir unser Analysewerkzeug, den TBUSAnalyzer, vorgestellt, der Einblicke in die gewonnenen Messdaten gibt.

Abschließend erklärten wir, wie unsere Traces zu einer Simulationsdatenbank aggregiert werden können und wir entwarfen ein Simulationsmodell für Client-Server Kommunikation über Mobilfunknetze, das diese Simulationsdatenbank nutzt. Ferner diskutierten wir Verbesserungsmöglichkeiten des Simulationsmodells und erläuterten, wie unser Simulationsmodell erweitert werden kann, um Prognosen für eine Client-Client Kommunikation über Mobilfunk zu erzeugen.

7.1 Ausblicke

Während der Erstellung dieser Masterarbeit zeigten sich an vielen Stellen weitere Möglichkeiten für Verbesserungen oder weiteren Forschungsbedarf, für die im Rahmen dieser Arbeit jedoch die Zeit fehlte.

Von zentraler Bedeutung ist natürlich zunächst die Implementierung und Evaluation des vorgestellten Simulationsmodells. Ein wichtiger Aspekt ist dabei die Wahl eines guten offline Map-Matching Verfahrens und die Suche nach optimalen Unterteilungsstrategien der Messdaten für die Aggregation in der Simulationsdatenbank.

Ferner muss untersucht werden, welches Aggregationsverfahren in den einzelnen Unterteilungsgebieten angewendet werden sollte.

Darüber hinaus bietet auch das von uns entwickelte Packettrain Verfahren zur schnellen, adaptiven Bandbreitenmessung weiteres Verbesserungspotential. Die Timer, die für eine Bandbreitendrosselung bei ausbleibendem Netzwerkverkehr sorgen, können noch wesentlich besser an die aktuellen Gegebenheiten angepasst werden. So könnte der Timeout zum Beispiel erwartungswert-gesteuert sein oder sich mehr an der gerade verwendeten Netzwerktechnik orientieren. Auch die optimale Größe für den back-off sollte weiter untersucht werden. Und schließlich sollte untersucht werden, ob die Droprate nicht auch direkt zu einem back-off führen sollte.

Auch die Auswirkungen vieler Teilnehmer einer Zelle auf die Übertragungsraten gilt es zu untersuchen, um das Simulationsmodell weiter zu verbessern.

In Abschnitt 6.4.2 haben wir schon darauf hingewiesen, dass eine Simulation von Paketverlusten in stark gefüllten Warteschlangen bisher nicht durchgeführt wird, und dass auch die Erhebung der Paketverlustraten im Falle von self-induced congestion während der Messfahrten weiter untersucht und geändert werden sollte. Es wäre auch vorstellbar, bei wiederholten Messfahrten auf der gleichen Messstrecke aus den vergangenen Messfahrten zu lernen und vor Gebieten, an denen bisher Bandbreiteneinbrüche aufgetreten sind, vorsorglich die Packettraingröße zu verringern.

Anhang A

Anhang

A.1 Konfiguration von NTP

Anbei finden sich `/etc/ntp.conf` Dateien, die wir auf dem Client (Listing A.1) und dem Server (Listing A.2) während des Loggings eingesetzt haben. Damit *cron* die Logfiles von NTP nicht nach ein paar Tagen einfach entsorgt, muss auf beiden Rechnern noch die Datei `/etc/cron.daily/ntp` geändert werden (siehe Listing A.3).

Sollen die Logfiles nicht am Standardspeicherort `/var/log/ntpstats` gespeichert werden, so blockiert bei Ubuntu 9.10 `apparmor` den Zugriff, wenn man ntpd nicht die nötigen Zugriffsrechte in der Datei `/etc/apparmor.d/usr.sbin/ntpd` einräumt.

Listing A.1: Client /etc/ntp.conf

```
# /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/
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

# You do need to talk to an NTP server or two (or three).
server 127.127.28.0 minpoll 4 noselect
fudge 127.127.28.0 time1 0.510 refid GPS
server 127.127.28.1 minpoll 4 prefer true
fudge 127.127.28.1 refid PPS

# server rohrpostix.cs.uni-duesseldorf.de iburst minpoll 4 maxpoll 5
#server 0.de.pool.ntp.org minpoll 4 maxpoll 6
```

```
#server ptbtime1.ptb.de minpoll 9 maxpoll 9
#server hora.cs.tu-berlin.de minpoll 9 maxpoll 9
#server uhr.uni-trier.de minpoll 9 maxpoll 9
#server ntp1.sda.t-online.de minpoll 9 maxpoll 9

# 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

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

# Clients from this (example!) subnet have unlimited access, but only if
# cryptographically authenticated.
#restrict 192.168.123.0 mask 255.255.255.0 notrust

# If you want to provide time to your local subnet, change the next line.
# (Again, the address is an example only.)
#broadcast 192.168.123.255

# If you want to listen to time broadcasts on your local subnet, de-comment the
# next lines.  Please do this only if you trust everybody on the network!
#disable auth
#broadcastclient
```

Listing A.2: Server /etc/ntp.conf

```
# /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/
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

# You do need to talk to an NTP server or two (or three).
```

```
server 127.127.28.0 minpoll 4 noselect
fudge 127.127.28.0 time1 0.510 refid GPS
server 127.127.28.1 minpoll 4 prefer true
fudge 127.127.28.1 refid PPS

# server rohrpostix.cs.uni-duesseldorf.de iburst minpoll 4 maxpoll 5
#server 0.de.pool.ntp.org minpoll 4 maxpoll 6
server ptbtime1.ptb.de minpoll 9 maxpoll 9
server hora.cs.tu-berlin.de minpoll 9 maxpoll 9
server uhr.uni-trier.de minpoll 9 maxpoll 9
server ntp1.sda.t-online.de minpoll 9 maxpoll 9

# 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

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

# Clients from this (example!) subnet have unlimited access, but only if
# cryptographically authenticated.
#restrict 192.168.123.0 mask 255.255.255.0 notrust

# If you want to provide time to your local subnet, change the next line.
# (Again, the address is an example only.)
#broadcast 192.168.123.255

# If you want to listen to time broadcasts on your local subnet, de-comment the
# next lines.  Please do this only if you trust everybody on the network!
#disable auth
#broadcastclient
```

Listing A.3: Server und Client /etc/cron.daily/ntp

```
#!/bin/sh

# The default Debian ntp.conf enables logging of various statistics to
# the /var/log/ntpstats directory.  The daemon automatically changes
```

```
# to a new datestamped set of files at midnight, so all we need to do
# is delete old ones, and compress the ones we're keeping so disk
# usage is controlled.

statsdir=$(cat /etc/ntp.conf | grep -v '^#' | sed -n 's/statsdir \([^ ][^ ]*\)
    /\1/p')

if [ -n "$statsdir" ] && [ -d "$statsdir" ]; then
        # only keep a week's depth of these

        #no, we do not want to delete them!!!
        #find "$statsdir" -type f -mtime +7 -exec rm {} \;

        # compress whatever is left to save space
        cd "$statsdir"
        ls loopstats.???????? peerstats.???????? > /dev/null 2>&1
        if [ $? -eq 0 ]; then
                # Note that gzip won't compress the file names that
                # are hard links to the live/current files, so this
                # compresses yesterday and previous, leaving the live
                # log alone.  We supress the warnings gzip issues
                # about not compressing the linked file.
                gzip --best --quiet loopstats.???????? peerstats.????????
                return=$?
                case $return in
                    2)
                        exit 0                  # squash all warnings
                        ;;
                    *)
                        exit $return            # but let real errors through
                        ;;
                esac
        fi
fi
```

A.2 Konfiguration von GPSD

Zur Konfiguration von gpsd (sowohl für Version 2.39, als auch für Version 2.95) nutzen wir auf beiden Rechnern das in Listing A.4 abgedruckte Konfigurationsfile `/etc/defaults/gpsd`. Die Option `-D 2` setzt das Debuglevel auf 2 und sorgt damit für mehrere Logeinträge pro Sekunde in `/var/log/messages`. Bei dauerhaftem Einsatz von gpsd sollte diese Option auf `-D 0` geändert werden.

Listing A.4: Server und Client /etc/defaults/gpsd

```
# Default settings for gpsd.
# Please do not edit this file directly - use 'dpkg-reconfigure gpsd' to
# change the options.
START_DAEMON="true"
DAEMON_OPTS="-n -F /var/run/gpsd.sock -D 2"
DEVICES="/dev/ttyS0"
USBAUTO="false"
```

A.3 Konfiguration der MySQL Datenbank

Damit TBUSLogger und TBUSAnalyzer funktionieren, muss sowohl auf dem Client als auch auf dem Server ein mysql Server laufen. Wir haben die Version `14.14 Distrib 5.1.37, for debian-linux-gnu (x86_64)` benutzt. Außerdem muss die Datenbank `tbus` existieren, auf die der User `tbuslogger` mit dem Passwort `tbuslogger` Zugriff hat. Es reicht allerdings lokaler Zugriff. Sämtliche benötigten Datenbanktabellen werden automatisch beim ersten Logging vom TBUSLogger erstellt.

Soll die mysql Datenbank nicht wie gewöhnlich unter /var/lib/mysql liegen, so müssen auch hier die Zugriffsrechte in der apparmor Konfigurationsdatei (`/etc/apparmor.d/usr/sbin/mysqld`) entsprechend gesetzt werden.

A.4 Konfiguration des Modems

Da das von uns verwendete UMTS Modem von Network-Manager nicht richtig unterstützt wird, haben wir `wvdial` zur manuellen Einwahl in das Mobilfunknetz von O_2 benutzt.

Damit der Network-Manager während einer Messfahrt weder die Netzwerkrouten, noch die Einträge in der Datei `/etc/resolv.conf` verändern kann, haben wir ihn vor einem Loggingrun mit dem Befehl `stop network-manager` abgestellt.

Danach haben wir das UMTS Modem durch das Kommando `wvdial on` eingeschaltet, einige Sekunden gewartet, bis das UMTS Modem das O_2 Netz gefunden hat, und danach mit dem Kommando `wvdial o2-pin` die SIM-Karte für diese Session entsperrt. Danach haben wir mit `wvdial surfo2` die Verbindung zu dem für unseren Vertrag vorgesehenen APN aufgebaut.

Bei der Prozedur gilt es zu beachten, dass die wvdial Befehle zur Aktivierung des Modems und zur Aktivierung der SIM-Karte über das Command-Interface abgewickelt werden, die Datenübertragung, die der dritte Befehl startet, aber das Data-Interface benutzt. Daher darf zum Absetzen der ersten beiden wvdial Befehle TBUSLogger noch nicht gestartet sein, da dieser sonst das Device, das das Command-Interface zur Verfügung stellt, blockiert.

Das verwendete Konfigurationsfile für wvdial ist in Listing A.5 abgedruckt. Allerdings wurden die Einträge für die PINs der SIM-Karten auf 0815 geändert.

Listing A.5: Client /etc/wvdial.conf

```
[Dialer klarmobil]
Carrier Check = no
Init3 = AT+CFUN=1
Init4 = AT+CGDCONT=1,"IP","internet.mobilcom"
Stupid Mode = 1
Phone = *99***1#
Password = egal
Username = egal
Dial Command = ATD
Dial Attempts = 1
Modem = /dev/ttyUSB2

[Dialer surfo2]
Carrier Check = no
Init3 = AT+CFUN=1
Init4 = AT+CGDCONT=1,"IP","surfo2"
Stupid Mode = 1
Phone = *99***1#
Password = egal
Username = egal
Dial Command = ATD
Dial Attempts = 1
Modem = /dev/ttyUSB2

[Dialer off]
Modem = /dev/ttyUSB0
Init3 = AT+CFUN=0

[Dialer on]
Modem = /dev/ttyUSB0
Init3 = AT+CFUN=1

[Dialer klarmobil-pin]
Modem = /dev/ttyUSB0
Init4 = AT+CPIN=0815
```

```
[Dialer o2-pin]
Modem = /dev/ttyUSB0
Init4 = AT+CPIN=0815

[Dialer Defaults]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
Modem Type = Analog Modem
; Phone = <Target Phone Number>
ISDN = 0
; Password = <Your Password>
New PPPD = yes
; Username = <Your Login Name>
#Modem = /dev/ttyUSB0
Baud = 9600
```

Literaturverzeichnis

[ADA] ADAC: *Elektronisches Stabilitäts-Programm*. http://www1.adac.de/Tests/fahrerassistenzsysteme/esp/.

[CC96] CARTER, Robert; CROVELLA, Mark: Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks. Boston, MA, USA : Boston University, 1996. Forschungsbericht.

[CKLL01] CHRISTOPH, Alexander K.; KLEMM, Er; LINDEMANN, Christoph; LOHMANN, Marco: Traffic Modeling and Characterization for UMTS Networks. In: *Proc. IEEE Globecom 2001*, 2001, S. 1741–1746.

[DRM01] DOVROLIS, Constantinos; RAMANATHAN, Parameswaran; MOORE, David: What Do Packet Dispersion Techniques Measure? In: *IN PROCEEDINGS OF IEEE INFOCOM*, 2001, S. 905–914.

[FJJ+01] FRANCIS, Paul; JAMIN, Sugih; JIN, Cheng; JIN, Yixin; RAZ, Danny; SHAVITT, Yuval; ZHANG, Lixia: IDMaps: a global internet host distance estimation service. In: *IEEE/ACM Trans. Netw.* 9 (2001), Nr. 5, S. 525–540. http://dx.doi.org/http://dx.doi.org/10.1109/90.958323. DOI http://dx.doi.org/10.1109/90.958323. ISSN 1063–6692.

[GI08] GARMIN INTERNATIONAL, Inc.: *GPS 18x Technical Specifications*. Olathe, KS 66062 USA, 2008. http://www8.garmin.com/manuals/GPS18x_TechnicalSpecifications.pdf

[HMSM03] HU, Ningning; MEMBER, Student; STEENKISTE, Peter; MEMBER, Senior: Evaluation and Characterization of Available Bandwidth Probing Techniques. In: *IEEE Journal on Selected Areas in Communications* 21 (2003), S. 879–894.

[JD02] JAIN, Manish; DOVROLIS, Constantinos: Pathload: A Measurement Tool for End-to-End Available Bandwidth. In: *In Proceedings of Passive and Active Measurements (PAM) Workshop*, 2002, S. 14–25.

[Kes91] KESHAV, Srinivasan: A control-theoretic approach to flow control. In: *SIGCOMM Comput. Commun. Rev.* 21 (1991), Nr. 4, S. 3–15. http:

//dx.doi.org/http://doi.acm.org/10.1145/115994.115995. DOI http://doi.acm.org/10.1145/115994.115995. ISSN 0146–4833.

[KH09] KILLAT, Moritz; HARTENSTEIN, Hannes: An empirical model for probability of packet reception in vehicular ad hoc networks. In: *EURASIP J. Wirel. Commun. Netw.* 2009 (2009), S. 1–12. http://dx.doi.org/http://dx.doi.org/10.1155/2009/721301. DOI http://dx.doi.org/10.1155/2009/721301. ISSN 1687–1472.

[KM07] KOVAR, Petr; MOLNAR, Karol: Precise Time Synchronization over GPRS and EDGE. In: *PWC*, 2007, S. 277–284.

[KZM07] KIESS, Wolfgang; ZALEWSKI, Stephan; MAUVE, Martin: Improving System Clock Precision With NTP Offline Skew Correction. In: *MedHocNet 2007: Proceedings of the The Sixth Annual Mediterranean Ad Hoc Networking Workshop*, Corfu, Greece, 2007, S. 159–164.

[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), Nr. 5, S. 29–34. http://dx.doi.org/http://doi.acm.org/10.1145/1163593.1163599. DOI http://doi.acm.org/10.1145/1163593.1163599. ISSN 0146–4833.

[Mau06] MAUVE, Martin: *Vorlesungsunterlagen Rechnernetze, Kapitel 1*. Heinrich Heine Unversität Düsseldorf, Deutschland, 2006.

[MBG00] MELANDER, Bob; BJORKMAN, Mats; GUNNINGBERG, Per: A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks. In: *GLOBECOM*, 2000.

[MHA04] MARCHAL, F.; HACKNEY, J.; AXHAUSEN, K. W.: Efficient Map-Matching of Large GPS Data Sets - Tests on a Speed Monitoring Experiment in Zurich, volume 244 of Arbeitsbericht Verkehrs und Raumplanung. 2004. Forschungsbericht.

[Mil85] MILLS, D.L.: *Network Time Protocol (NTP)*. RFC 958. http://www.ietf.org/rfc/rfc958.txt. Version: September 1985 (Request for Comments). Obsoleted by RFCs 1059, 1119, 1305

[Mil88] MILLS, D.L.: *Network Time Protocol (version 1) specification and implementation*. RFC 1059. http://www.ietf.org/rfc/rfc1059.txt. Version: Juli 1988 (Request for Comments). Obsoleted by RFCs 1119, 1305

[Mil89] MILLS, D.L.: *Network Time Protocol (version 2) specification and implementation*. RFC 1119 (Standard). http://www.ietf.org/rfc/rfc1119.txt. Version: September 1989 (Request for Comments). Obsoleted by RFC 1305

[Mil92] MILLS, D.: *Network Time Protocol (Version 3) Specification, Implementation and Analysis*. RFC 1305 (Draft Standard). http://www.ietf.org/rfc/rfc1305.txt. Version: März 1992 (Request for Comments).

[NS-] *The Network Simulator - ns-2*. http://www.isi.edu/nsnam/ns/.

[NSNK97] NOBLE, Brian D.; SATYANARAYANAN, M.; NGUYEN, Giao T.; KATZ, Randy H.: Trace-based mobile network emulation. In: *SIGCOMM Comput. Commun. Rev.* 27 (1997), Nr. 4, S. 51–61. http://dx.doi.org/http://doi.acm.org/10.1145/263109.263140. DOI http://doi.acm.org/10.1145/263109.263140. ISSN 0146–4833.

[QON07] QUDDUS, Mohammed A.; OCHIENG, Washington Y.; NOLAND, Robert B.: Current map-matching algorithms for transport applications: State-of-the art and future research directions. In: *Transportation Research Part C: Emerging Technologies* 15 (2007), Nr. 5, 312 - 328. http://dx.doi.org/DOI:10.1016/j.trc.2007.05.002. DOI DOI: 10.1016/j.trc.2007.05.002. ISSN 0968–090X.

[RCR+00] RIBEIRO, Vinay; COATES, Mark; RIEDI, Rudolf; SARVOTHAM, Shriram; HENDRICKS, Brent; BARANIUK, Richard: Multifractal cross-traffic estimation. In: *Proc. ITC Specialist Seminar on IP Trac Measurement, Modeling, and Management*, 2000, S. 15–1.

[RRB+03] RIBEIRO, Vinay J.; RIEDI, Rudolf H.; BARANIUK, Richard G.; NAVRATIL, Jiri; COTTRELL, Les: pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In: *In Passive and Active Measurement Workshop*, 2003.

[Sie06] SIERRA WIRELESS: *UMTS Modems - Supported AT Command Reference Rev. 2.1*. Richmond, BC, Canada, Oct 2006. http://www.sierrawireless.com/en/sitecore/content/Sierra%20Wireless/Su%pport/Downloads/AirCard/Legacy_Products/~/media/Support_Downloads/AirCard/docs%/2130617_Supported_AT_Command_Reference.ashx

[SKK03] STRAUSS, Jacob; KATABI, Dina; KAASHOEK, Frans: A measurement study of available bandwidth estimation tools. In: *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. New York, NY, USA : ACM, 2003. ISBN 1–58113–773–7, S. 39–44.

[Win] WINE: *Wine*. http://www1.winehq.org.

[WS05] WILL SPECKS, Kirsten M. u.: Car-to-Car Communication - Market Introduction and Success Factors. In: *ITS 2005: 5th European Congress and Exhibition on Intelligent Transport Systems and Services*, 2005.

[YRZ+08] YUN, Mira; RONG, Yanxia; ZHOU, Yu; CHOI, Hyeong-Ah; 0004, Jae-Hoon K.; SOHN, JungKyo; CHOI, Hyeong I.: Analysis of Uplink Traffic Characteristics and Impact on Performance in Mobile Data Networks. In: *ICC*, IEEE, 2008, S. 4564–4568.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbständig 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.

Neuss, 23. August 2010 Norbert Goebel