Live Fernüberwachung von Mobilfunkmessungen

Bachelorarbeit

von

Daniel Glanert

aus

Neuss

vorgelegt am

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

April 2012

Betreuer:
Norbert Goebel M. Sc.

Abstract

Ziel dieser Bachelorarbeit war die Implementierung einer effizienten Netzwerkübertragung von Messwerten und deren Visualisierung mit Hilfe einer grafischen Benutzeroberfläche während der Messung. Da die Messhardware über geringe Rechenleistung und Ressourcen verfügt, war ein besonderes Augenmerk auf eine ressourcensparende und die Messungen möglichst wenig störende Implementierung zu legen.

Zu diesem Zweck wurden zwei Programme erstellt: Ein *Monitor-Server*, der auf dem Computer der bildschirmlosen Messbox läuft und ein *Monitor-Client*, der auf einem externen mit der Messbox per TCP/IP-Protokoll verbundenen Rechner bedient wird. Über die grafische Benutzeroberfläche (GUI) des Client lassen sich Informationen der Messbox abrufen, indem diese vom Server gesammelt und durch ein Public-Subscribe-Verfahren bereitgestellt werden. Weiterhin kann sich der Client direkt mit Serverprogrammen verbinden, die ihre Daten über eine TCP-Verbindung zur Verfügung stellen.

Der Monitor-Client fordert Informationen mit Hilfe einer menschenlesbaren Befehlsliste in folgender Form per TCP-Verbindung an:
`<Befehl>[,<Pause nach einer Befehlsausführung in ms>[,`
`<Anzahl der Befehlsausführungen>]]`
Die Antwort des Servers ist die Ausgabe des Befehls in der entsprechenden Anzahl und mit der gewünschten Pause.

Danksagung

Ich möchte allen Personen, die mich direkt oder indirekt bei der Erstellung meiner Bachelorarbeit unterstützt haben, meinen Dank aussprechen.

Meinem Betreuer Norbert Goebel möchte ich für die ständige Hilfe während der drei Monate und speziell für die Mühe bei der Erstellung und Ausarbeitung des Exposés danken.

Ganz besonders bedanken möchte ich mich bei meiner Frau Ruth, ohne die diese Arbeit nicht möglich gewesen wäre. Sie war es, die mir den Rücken zu jeder Zeit freihielt ohne sich über die entstandene Mehrarbeit zu beschweren.

Vielen Dank auch an meine Eltern, die nie aufgehört haben an mich zu glauben, auch wenn das Erreichen einiger Lebensziele mehr, anderer dafür weniger Zeit als geplant in Anspruch genommen hat.

Inhaltsverzeichnis

Abbildungsverzeichnis **ix**

Tabellenverzeichnis **xi**

1 Einleitung **1**

1.1 Motivation 1

1.2 Aufbau 2

2 Verwandte Arbeiten **3**

2.1 Telnet / SSH 3

2.2 GPSd 4

2.3 NTP 4

3 Design **7**

4 Implementierung **11**

4.1 Monitor-Client 11

4.2 Monitor-Server 17

5 Evaluierung **21**

6 Zusammenfassung und Ausblick **25**

6.1 Zusammenfassung 25

6.2 Ausblick 26

Literaturverzeichnis **27**

Abbildungsverzeichnis

4.1 DockWidgets innerhalb des Hauptfensters 13
4.2 DockWidgets außerhalb des Hauptfensters 15
4.3 Hauptfenster mit GPS- und NTP-Plot 16

Tabellenverzeichnis

5.1 Ausgewählte Befehle und deren Informationsausgabe 21
5.2 Benötigte CPU-Zeit nach 10min bei 1000ms Pause nach jeder Befehlsausführung und verwendeter Arbeitsspeicher 22
5.3 Benötigte CPU-Zeit nach 10min bei 100ms Pause nach jeder Befehlsausführung und verwendeter Arbeitsspeicher 22

Kapitel 1

Einleitung

1.1 Motivation

In der Automobilindustrie ist die sogenannte *Car2Car Communication* (im folgenden *C2C*) Gegenstand der Forschung. Dabei sollen Boardcomputer von Fahrzeugen per Funk Informationen austauschen um Unfälle zu vermeiden und das Stauaufkommen zu verringern. Der im Juli 2010 veröffentlichte IEEE 802.11p Standard [IEE10] wurde deswegen entwickelt. Eine weitere Möglichkeit besteht darin, den bereits vorhandenen Mobilfunk (GSM, UMTS, LTE) dazu zu nutzen. Ob sich dieser für bestimmte C2C-Dienste eignet, wird zur Zeit von Fahrzeugherstellern erforscht. Da es sehr aufwendig ist, Mobilfunknetze und mögliche darauf basierende Anwendungen unter realen Bedingungen im Straßenverkehr zu testen, wird am Lehrstuhl für Rechnernetze der Heinrich-Heine-Universität Düsseldorf ein Verfahren zur Simulation von Mobilfunknetzwerken auf Basis von Messungen entwickelt.

In einer vorangegangenen Arbeit wurde der Prototyp eines Loggingframeworks zur Sammlung von Mobilfunkmessungen [Goe10] mit Hilfe eines Laptops erstellt. In Zukunft sollen die Messungen jedoch mit fahrzeugtauglicher Messhardware durchgeführt werden. Diese Messcomputer besitzen allerdings keinen Bildschirm. Gerade während der Entwicklungsarbeit ist es wichtig live Daten der Messungen zu sehen, um etwaige Probleme und Bugs schneller zu finden. Zwar ist es möglich sich mit einem Laptop über

ein Netzwerkkabel mit der Messhardware zu verbinden und sich per SSH anzumelden, die Anzeige von aktuellen Messwerten ist auf diesem Weg allerdings unkomfortabel.

In dieser Bachelorarbeit wurden zu diesem Zweck zwei Programme entwickelt. Ein *Monitor-Client* mit grafischer Benutzeroberfläche, welcher die Messwerte einfach und übersichtlich anzeigt und ein *Monitor-Server*, der auf dem Computer der Messhardware läuft und die Messwerte sowohl sammelt, als auch dem Client auf Anfrage per TCP/IP zur Verfügung stellt.

Die Messcomputer sind jedoch mit vergleichsweise geringer Rechenleistung und wenig Arbeitsspeicher ausgestattet. Daher soll der Server ressourcensparend programmiert werden, damit die Messungen möglichst wenig beeinflusst werden.

1.2 Aufbau

Im nächsten Kapitel beschäftigen wir uns mit bereits vorhandenen Programmen, die das Anzeigen der Messwerte ermöglichen und gehen auf deren Vor- und Nachteile ein. Das dritte Kapitel wird zum einen das Design der grafischen Oberfläche und zum anderen das zwischen den zwei Programmen verwendete Netzwerkprotokoll behandeln.

Die Implementierung dieser zwei Aspekte ist Thema des vierten Kapitels, welchem die Evaluierung des Ressourcenverbrauchs auf verschiedenen Systemen im fünften folgt. Auf die Zusammenfassung mit Ausblick gehen wir im sechsten Kapitel ein.

Kapitel 2

Verwandte Arbeiten

2.1 Telnet / SSH

Eine Möglichkeit zur Anzeige von Messwerten ist der Zugang zum Messcomputer per *Telnet* oder *SSH*. Es ist mit diesen Programmen möglich, Befehle auf der Box auszuführen und deren Ausgabe zu lesen. Somit können auf diesem Weg auch Messwerte ausgelesen werden.

Vorteile:

- Ein Telnet- oder SSH-Server wird in jedem Fall auf der Box laufen, sodass kein zusätzliches Programm verwendet und somit keine zusätzlichen Ressourcen benutzt werden müssen.

- Kompletter Zugriff auf alle relevanten Programme und Logdateien (entsprechende Rechte vorausgesetzt) -> große Flexibilität

Nachteile:

- Falsche Benutzung (versehentlich oder absichtlich) kann zur Schädigung des Systems oder Verlust von Daten führen.

- Der Benutzer muss mit der Benutzung einer Shell und den zu verwendenden Befehlen vertraut sein und benötigt für den Zugriff einen Benutzernamen und ein Passwort.

- Der Aufwand für das Anzeigen eines Messwertes oder einer Befehlsausgabe ist per SSH/Telnet deutlich höher als bei einer GUI, die vorgefertigte Abfragen per Mausklick ermöglicht. Bei mehreren gleichzeitig anzuzeigenden Daten vervielfacht sich der Aufwand durch Verwendung von entsprechend vielen Instanzen.

2.2 GPSd

Um Daten von verschiedenen GPS-Empfängern einheitlich auswerten zu können, wird das Serverprogramm GPSd (http://catb.org/gpsd) verwendet, welches diese Daten per serieller Schnittstelle von den Geräten abfragt und in *JavaScript Object Notation* (im folgenden *JSON*) aufbereitet über einen TCP-Port per Public-Subscribe-Verfahren anbietet. Es ist somit unnötig den Monitor-Server GPS-Daten verarbeiten zu lassen. Stattdessen verbindet sich der Monitor-Client direkt per Netzwerkverbindung mit dem GPSd (siehe Kapitel 4).

2.3 NTP

Die Synchronisation aller beteiligten Uhren ist für die Genauigkeit der Messungen äußerst wichtig. Bei früheren Messungen zeigte sich, dass eine Zeitsynchronisation nur mit Hilfe des *Network Time Protocol* (*NTP*) über das Mobilfunknetzwerk zu ungenau ist, da NTP die Latenz der Verbindung durch das Halbieren der Round-Trip-Time berechnet. Aber im Gegensatz zu einem klassischen Netzwerk (kabelgebundenes LAN oder WLAN) sind die im Mobilfunk vorhandenen Up- und Downloadlatenzen aufgrund der verwendeten unterschiedlichen Technologien und Protokolle weder annähernd gleich, noch ist die Differenz der Latenzen konstant. Dies führt dazu, dass der eingeflossene Feh-

ler durch den Latenzunterschied nicht nur zweimal in die Berechnung einfließt, sondern sich zusätzlich laufend ändert und damit nicht einfach herausgerechnet werden kann.

Deswegen wird als Zeitgeber für den *NTP daemon* (*NTPd*) das Pulse-per-Second-Signal eines speziellen GPS-Empfängers verwendet. Hierdurch wird der Messcomputer selbst zum Stratum-0-Server (ein NTP-Server, der direkt an einen hochgenauen Zeitgeber angeschlossen ist). Der vom Empfänger zur Verfügung gestellte Zeitstempel alleine wäre als Referenz ebenfalls zu ungenau, da dessen Berechnung und damit auch die Auslieferung nach dem Empfang des GPS-Signals abhängig von der Anzahl der verwendeten Satelliten ist und somit unterschiedlich viel Zeit in Anspruch nimmt. Zusätzliche Verzögerungen enstehen durch die Latenz der Datenübertragung über die serielle Schnittstelle.

Um den Versatz zwischen den Uhrzeiten des GP-Systems und der Messhardware abzufragen wird das *NTP query* Programm (*NTPq*) verwendet. Dieses gibt u.a. eine einfach formatierte Textliste der verbundenen Zeitserver mit den jeweiligen Abweichungen aus. Die Ausgabe wird vom Monitor-Server eingelesen und an den Monitor-Client weitergeleitet.

Kapitel 3

Design

Um die in Kapitel 2 angesprochenen Nachteile einer Telnet- und SSH-basierten Verbindung zu vermeiden, wird in diesem Kapitel die Entwicklung einer geeigneteren Software behandelt.

Für die Überwachung von Messwerten bietet sich eine Client-Server-Lösung an, bei der der Server die Aufgabe übernimmt Daten auf Anfrage des Clients zu sammeln und zur Verfügung zu stellen. Der Monitor-Server muss wegen der vergleichsweise geringen Prozessorleistung der verbauten AMD Geode LX 500MHz CPU und des kleinen Arbeitsspeichers von 256MB ressourcenschonend programmiert werden. Der Monitor-Client, welcher auf handelsüblichen Desktop-PCs oder Notebooks läuft, kann dagegen mit einer komfortablen und intuitiv bedienbaren grafischen Benutzeroberfläche ausgestattet werden.

Um den Server so einfach wie möglich und das System leicht erweiterbar zu gestalten, öffnet der Client für jeden Befehl eine TCP-Verbindung zum Server und schickt diesem ein Datentupel als Bytestream bestehend aus mindestens einem Wert und maximal drei Werten, die durch Kommata getrennt sind. Diese werden vom Server ausgewertet und beantwortet.

Dieses einfache menschenlesbare Publish-Subscribe-Verfahren, hat den Vorteil, dass ein Client und sein Benutzer nur die Struktur des Datentupels kennen müssen, um Befehle des Messsystems auszuführen und somit gewünschte Informationen zu abonnieren.

Gleichzeitig muss der Quelltext des Server nicht verändert werden, um neue Informationen anbieten zu können.

Syntax:
`<Befehl>[,<Pause nach einer Befehlsausführung in ms>[,`
`<Anzahl der Befehlsausführungen>]]`

Der erste Wert, der als einziger unbedingt mitgesendet werden muss, ist ein String. Dieser repräsentiert den gewünschten Befehl, der vom Server auf der Messhardware ausgeführt werden soll.

Der zweite Wert ist optional und ein Integer. Er gibt die Pause in Millisekunden nach jeder (bis auf die letzte) Befehlsausführung an. Fehlt der Wert, wird der Befehl nur einmal ausgeführt; ist er negativ oder keine ganze Zahl wird der Befehl nicht bearbeitet, sondern stattdessen eine Fehlermeldung zurückgesandt und auf STDOUT ausgegeben.

Der dritte Wert ist ebenfalls eine optionale Ganzzahl, die der Anzahl der Befehlsausführungen auf dem Server entspricht. Er wird allerdings nur dann berücksichtigt, wenn ein gültiger zweiter Wert angegeben wurde. Fehlende Werte führen dazu, dass der Befehl bis zur Trennung der zugehörigen TCP-Verbindung wiederholt wird. Auch hier führen negative oder nicht ganzzahlige Werte zu keiner Befehlsausführung sondern zu einer Fehlermeldung.

Wurde die vorgegebene Anzahl der Befehlsausführungen erreicht, trennt der Server die entsprechende TCP-Verbindung zum Client. Auch der Client hat jederzeit die Möglichkeit ein bestehendes Abonnement vorzeitig durch Trennung der dazu genutzten TCP-Verbindung zu beenden.

Beispiele:
`ls /bin`
Listet die Dateien des Verzeichnisses /bin einmal auf und trennt die zugehörige TCP-Verbindung.

`ls /bin,1000`
Listet die Dateien des Verzeichnisses /bin mit einer Pause von jeweils einer Sekunde

solange auf, bis die entsprechende TCP-Verbindung vom Client geschlossen wird oder abbricht.

`ls /bin,1000,3`
Listet die Dateien des Verzeichnisses /bin dreimal mit einer Pause von jeweils einer Sekunde auf. Die dafür verwendete TCP-Verbindung wird anschließend vom Server getrennt.

Der Server ist somit in der Lage jedes Programm auf der Box auszuführen und dessen Ausgabe (STDOUT und STDERR) an den Client zu schicken. Das ermöglicht zum Beispiel das Lesen von Dateien wie Logdateien oder Systeminformationen per *cat* oder *tail*, das Auflisten von Dateinamen durch *ls* oder die Anzeige der Netzwerkeinstellungen mit *ifconfig*.

Es muss in diesem Zusammenhang beachtet werden, dass Befehlsausgaben, die Terminalsteuerzeichen zur Anzeigenformatierung enthalten, nicht wie in einer Shell mit eingebauter Terminalemulation üblich aussehen. Die Steuerzeichen werden als ASCII-Zeichen interpretiert und das dafür in der Anzeigenschriftart vorgesehene Symbol ausgegeben. Dadurch können Befehle wie *wait* oder *top* gar nicht oder nicht in der gewohnten Art und Weise benutzt werden. Top ist im Gegensatz zu *wait* in der Lage durch Angabe des Parameter „-b“ eine Ausgabe ohne Steuerzeichen zu produzieren und kann so verwendet werden. Allerdings beendet sich *top* dann - wie auch ohne zusätzlichen Parameter - nicht von selbst, was bedeutet, dass nur eine kontinuierliche Ausgabe angezeigt werden kann. Abhilfe schafft der Parameter „-n1“ der *top* nach einer Ausgabe beendet.

Um das Programm einfach zu halten, damit es möglichst wenig Ressourcen in Anspruch nimmt, können über eine separate Netzwerkschnittstelle (z.B. vom *GPSd*) bereitgestellte Daten nicht vom Server, sondern nur vom Monitor-Client abgerufen werden. Dazu wird eine TCP-Verbindung direkt mit der entsprechenden Anwendung aufgebaut.

Die erhaltenen Informationen können dann vom Client in Rohform als Log oder tabellarisch aufbereitet angezeigt werden. Der Client speichert eine festgelegte Anzahl von aufbereiteten Daten, sodass diese auch als Plot über die Zeit darstellbar sind. Die Elemente der GUI, die für die Anzeige der Messwerte und Befehlsausgaben zuständig sind,

lassen sich frei im Hauptfenster verschieben und können auch von diesem losgelöst werden, damit der Benutzer die für ihn optimale Ansicht frei wählen kann.

Der Server kann mit zwei optionalen Parametern gestartet werden:

```
monitor [<IP-Adresse> [<Portnummer>]]
```

Es werden dann nur TCP-Verbindungen an die angegebene IP-Adresse und an den gewählten Port akzeptiert. Als Standard-IP-Adresse ist zur Sicherheit 127.0.0.1 eingestellt, damit keine Verbindungen über andere Netzwerkschnittstellen möglich sind. Der Standardport ist auf 54321 festgelegt. Der Monitor-Server sollte nur mit den notwendigen Rechten gestartet werden, damit beispielsweise ein Löschen von System- oder anderen wichtigen Dateien nicht möglich ist. Für die meisten Befehle sind einfache Benutzerrechte ausreichend. Andererseits haben viele Systemlogdateien z.B. im Verzeichnis /var/log keine Leserechte für Benutzer oder sogar Gruppen gesetzt. Diese Rechte müssten bei Bedarf entsprechend angepasst werden.

Kapitel 4

Implementierung

4.1 Monitor-Client

Die Implementierung des Monitor-Clients erfolgte in C++ mit Hilfe des Qt Frameworks in der Version 4.8. Dadurch ist es möglich, das Programm mit grafischer Benutzeroberfläche für Windows, Linux und Mac OS zu erstellen.

Außer den Qt-Standard-Bibliotheken wurden für das Parsen von JSON-Daten die *Json* Klasse [RBK12] in der zu diesem Zeitpunkt aktuellsten stabilen Version vom 05.03.2012 (das Projekt besitzt keine Versionsnummern) und für die Erstellung und Anzeige der Plots die *Qwt*-Bibliothek (*Qt Widgets for Technical Applications*), in der Version 6.0.1 [RW11] benutzt.

Das Programm verwendet neben den üblichen Bestandteilen einer GUI wie z.B. einer Menü- oder Werkzeugleiste, sogenannte *Dock-Widgets*. Diese Elemente, die zur Anzeige aller angeforderten Daten dienen, können frei in, über und neben dem Hauptfenster angeordnet oder auch ausgeblendet werden. Das gewählte Layout wird zusammen mit den anderen Einstellungen beim Beenden der Anwendung in einer Textdatei mit der Endung .ini (Windows) oder .conf (Linux/MacOS) gespeichert und beim Starten wiederhergestellt.

In den Einstellungen werden die IP-Adressen und Ports des Monitor-Servers, des GPSd und anderer Messmodule angegeben. Dort ist auch die Konfiguration der zu übermittelnden Befehle (siehe Kapitel 3) zur Datenanforderung möglich. Diese wird gestartet, indem rechts neben dem jeweiligen Befehl auf den Knopf mit dem „>“-Symbol (Nr. 1 in Abb. 4.1) geklickt wird. Solange die vom Server versendeten Daten vom Client empfangen werden, bleibt der entsprechende Knopf eingedrückt. Die Daten werden währenddessen im verknüpften Textfeld angezeigt. Wird die Verbindung durch den Server oder per erneutem Klick auf „>“ getrennt, kehrt der Knopf in seinen Ausgangszustand zurück.

Werden mehrere Informationen gleichzeitig benötigt, kann mit klick auf den „+“-Knopf (Nr. 4 in Abb. 4.1) ein weiteres Befehlsfeld mit zugehörigem Knopf und Ausgabetextfeld dynamisch erzeugt werden. Das Entfernen durch den „-“-Knopf (Nr. 5 in Abb. 4.1) findet dann in zur Erzeugung umgekehrten Reihenfolge statt. Jedes Textfeld, die Plots und das Einstellungsfenster können entweder wie in Abbildung 4.1 im Hauptfenster angedockt sein oder sich wie in Abbildung 4.2 außerhalb dessen befinden. Das geschieht entweder über den ersten Knopf in der Dock-Widget-Titelleiste oder per Ziehen und Fallen lassen des Fensters. Der zweite Knopf schließt das Fenster ohne seinen Inhalt zu löschen. Zur Öffnung des Fensters, wird mit der rechten Maustaste im Hauptfenster auf die Menü-, Werkzeug-, oder eine Dock-Widget-Titelleiste geklickt. In dem erscheinenden Kontextmenü muss dann der Name des gewünschten Fensters angeklickt werden. Dort kann es auf die gleiche Weise auch wieder geschlossen werden.

Rechts neben dem Knopf zum Starten und Beenden der Befehlsausführung befindet sich ein Knopf mit einem „v“-Symbol (Nr. 2 in Abb. 4.1). Dieser Knopf schaltet die kontinuierliche Ausgabe der zugehörigen Befehle ein und aus. Im eingedrückten Zustand werden die empfangenen Ausgaben untereinander im Textfenster angezeigt. Ist der Knopf ausgeschaltet, wird immer nur die aktuellste Ausgabe am oberen Ende des Textfensters dargestellt. Mit dem „#“-Knopf (Nr. 3 in Abb. 4.1) können die Textfenster schnell geschlossen oder geöffnet werden.

Die Ausgaben der GPSd- und NTP-Daten werden als HTML-Tabelle in den jeweiligen Textfenstern erzeugt. Das ermöglicht die Nutzung verschiedener HTML-Attribute wie z.B. Farbe, Ausrichtung und Textformatierung. So wird beispielsweise der NTP-Offset rot hinterlegt, wenn der Betrag 50ms übersteigt. Gleiches geschieht mit der GPSd-

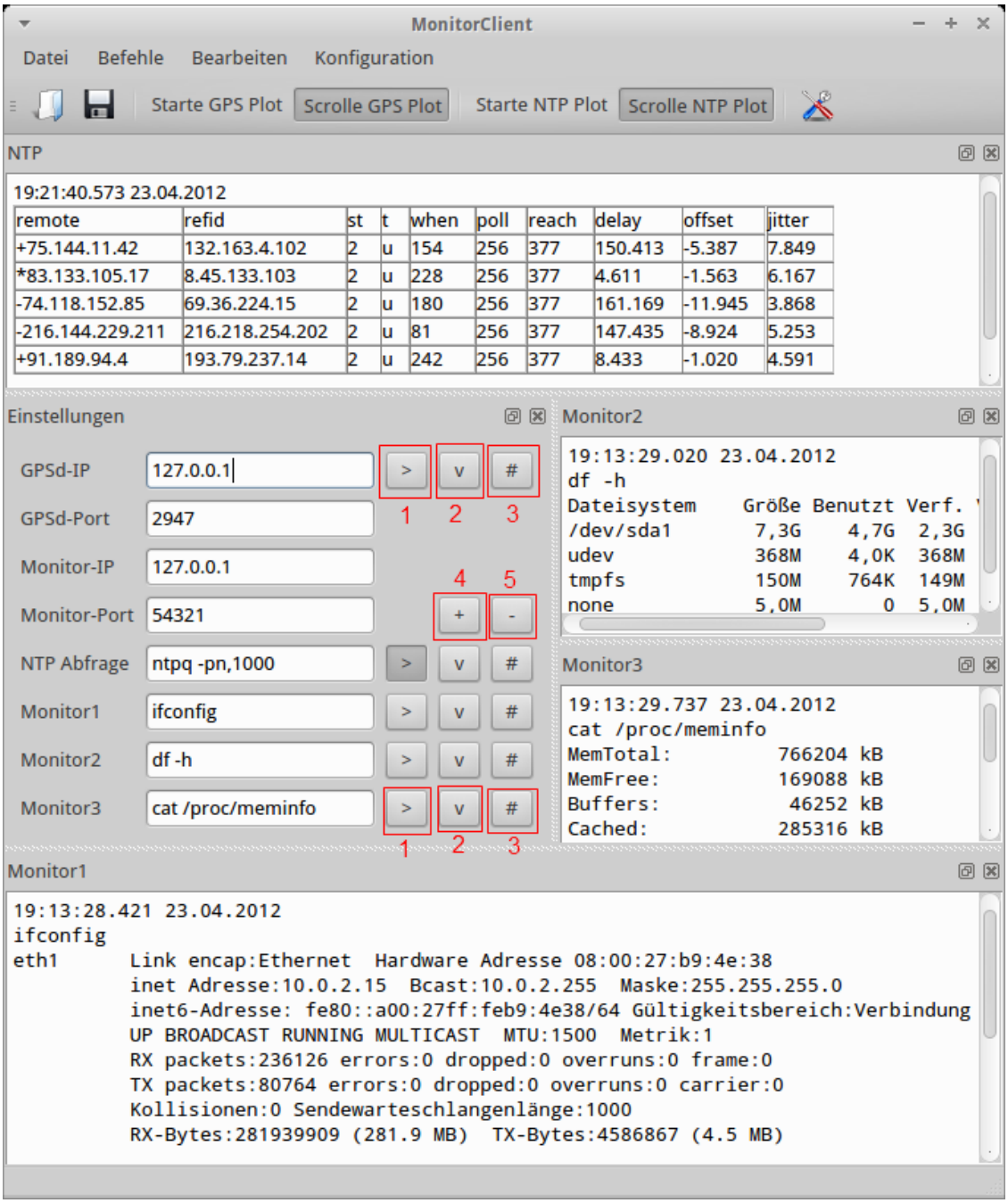

Abbildung 4.1: DockWidgets innerhalb des Hauptfensters

Mode-Tabellenzelle bei einem Wert kleiner als 3, was bedeutet, dass aktuell keine 3D-Positionsbestimmung möglich ist. Außerdem kann die HTML-Tabelle leicht zur weiteren Auswertung durch Kopieren und Einfügen in ein Tabellenkalkulationsprogramm wie LibreOffice Calc oder in ein Textverarbeitungsprogramm wie LibreOffice Writer übernommen werden. Die anderen Textfenster stellen die Befehlsausgaben in reinem Text mit nicht proportionaler Schrift dar, damit durch Leerzeichen formatierte Ausgaben korrekt wiedergegeben werden.

In jedem Textfenster wird über den angeforderten Informationen immer ein Zeit- und Datumstempel angezeigt. Die Zeiten der zwei GPSd-Textausgabefenstern wird aus den GPS-Daten selbst extrahiert. In den anderen Fenstern fügt der Monitor-Client die Empfangszeit der Befehlsausgaben und den abgesendeten Befehl (ohne Pausen- und Wiederholungsangaben) hinzu. Beides ist besonders im Zusammenhang mit der Möglichkeit den Inhalt des Textfensters in dem sich der Textcursor befindet per Menüleiste oder Werkzeugleistenknopf zu speichern interessant, da in der gespeicherten Datei sonst kein Hinweis auf den Befehl zu finden wäre. Gespeicherte Logs können mit dem „Log öffnen“-Befehl geöffnet werden. Die Log-Dateien werden je nach Textformatierung der zugehörigen Fenster als HTML-Dokument oder als einfacher Text gespeichert.

Der Monitor-Client beinhaltet - wie in Abbildung 4.3 gezeigt - ein GPS- und ein NTP-Plotfenster. Zum Starten der Plots drückt man auf die entsprechend bezeichneten Knöpfe. Sind die Plots noch nicht sichtbar, werden sie durch diese Aktion automatisch geöffnet. Achtung: Das Starten der Plots ist erst bei vorhanden GPS- bzw. NTP-Daten möglich. Wurden noch keine Daten abgerufen wird der Klick auf den Starten-Knopf ignoriert.

Bei der Anzeige von Plots ist ein optionales automatisches Scrollen der Zeitachse voreingestellt. Dadurch erscheint der aktuelle Messwert immer am rechten Rand des Graphen und schiebt die Kurve entsprechend der zwischen den Werten verstrichenen Zeit nach links. Ist das Scrollen abgeschaltet, kann die Kurve mit der Maus per Ziehen und Fallen lassen verschoben und mit dem Mausrad die Skalierung verändert werden.

Die Eigenschaften der Plots (zu plottenden Daten, Anzahl der verwendeten Datensätze, Achsenbeschriftung, Anfangsskalierung) können aktuell nur durch Präprozessordirektiven und nicht zur Laufzeit verstellt werden.

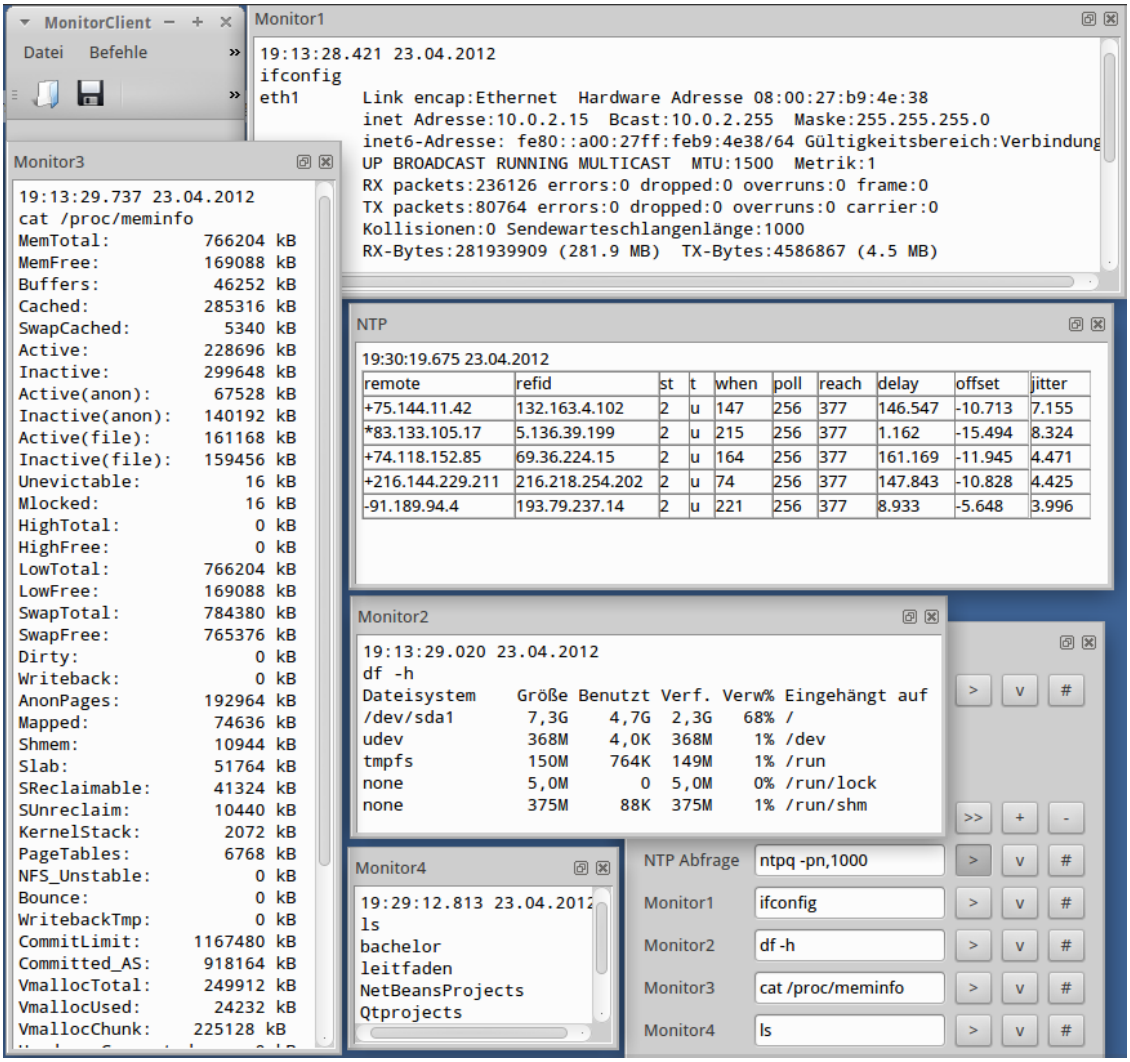

Abbildung 4.2: DockWidgets außerhalb des Hauptfensters

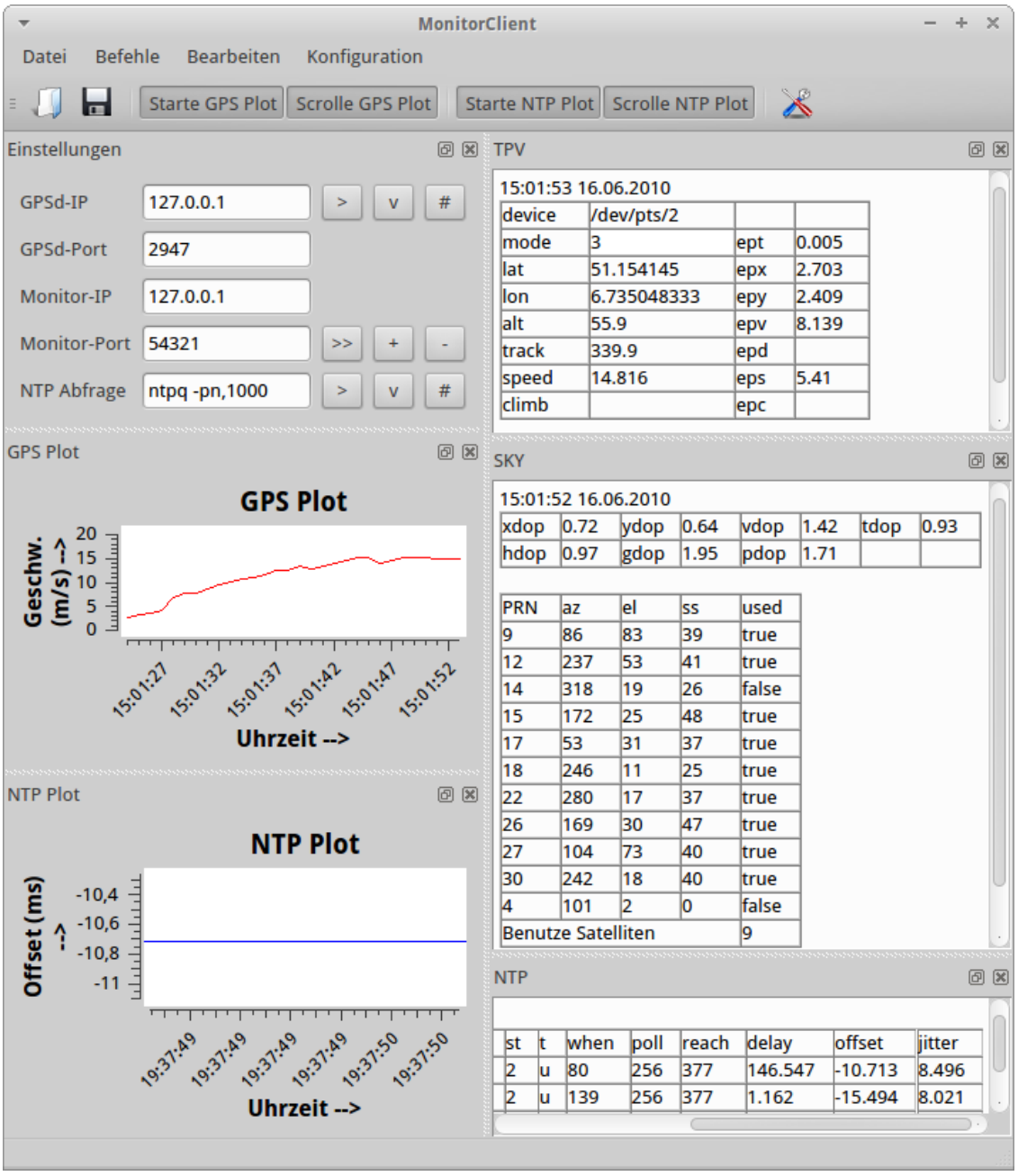

Abbildung 4.3: Hauptfenster mit GPS- und NTP-Plot

Die für das Speichern und Laden verwendete QSettings-Klasse ermöglicht es, die Datei an bis zu vier verschiedenen Orten zu speichern, je nachdem ob die Einstellungen benutzerspezifisch oder systemweit, bzw. anwendungsspezifisch oder organisationsweit gelten sollen. Diese Orte werden beim Starten des Clients in einer bestimmten Reihenfolge durchsucht und die erste gefundene Datei geladen. Auf einem Linuxsystem wäre z.B. folgendes Suchmuster möglich:

1. $HOME/.config/HHU-CN/MonitorClient.conf
2. $HOME/.config/HHU-CN.conf
3. /etc/xdg/HHU-CN/MonitorClient.conf
4. /etc/xdg/HHU-CN.conf

Dadurch kann bei der Installation des Programmes eine Einstellungsdatei auf dem Rechner gespeichert werden, in der beispielsweise die am meisten genutzten Befehle eingetragen sind. Im Gegensatz zu einprogrammierten Voreinstellungen hat dies den Vorteil, dass zur Änderung kein Eingriff in den Sourcecode nötig ist. Stattdessen kann man entweder die Datei direkt im Texteditor ändern oder die Einstellungen in der GUI entsprechend konfigurieren und die beim Beenden automatisch gespeicherte Datei verwenden.

4.2 Monitor-Server

Der Server wurde zunächst in C++ mit Verwendung der ASIO-Bibliothek (Asynchronous I/O) des Boost-Projektes geschrieben, um durch Verwendung von Eventhandlern und nicht-blockierenden Funktionsaufrufe ein effizientes und ressourcenschonendes Programm zu erhalten.

Leider zeigte sich im Laufe der Entwicklung, dass die Nachteile der objektorientierten Programmierung im Hinblick auf den Speicherverbrauch und die CPU-Nutzung die Vorteile überwogen. Deswegen wurde ein komplett neuer Monitor-Server in der Pro-

grammiersprache C entworfen, welcher durch die Bibliothek *pthread* jede Verbindung in einem eigenen Thread verarbeitet.

Wir haben uns bei der Entwicklung gezielt gegen die Verwendung von nur einer TCP-Verbindung zwischen Server und Client entschieden, da bei der Anzeige von N Befehlsausgaben der Ressourcengewinn von (N-1) TCP-Verbindungen gegenüber dem Ressourcenverbrauch von insgesamt 2N+1 Threads in N+1 Prozessen vernachlässigbar klein ist. Des Weiteren muss kein spezielles Netzwerkprotokoll für die Einkapselung der Daten entwickelt und im Server und Client implementiert werden. Gleiches gilt für die Thread- und Prozesssynchronisation die aufgrund der Nutzung von nur einer Netzwerkverbindung berücksichtigt werden müsste. Beides würde zusätzlich die Komplexität der Programme und damit wieder deren Ressourcenverbrauch und zusätzlich noch die Fehleranfälligkeit erhöhen.

Der *Hauptthread* ist nur dafür zuständig einen lauschenden Socket zu erstellen, um dann in einer Endlosschleife auf eine Verbindung zu warten, sie entgegenzunehmen, diese an einen *Workerthread* zu übergeben und wieder auf eine neue Verbindung zu warten. Da die Threads nach dem Schließen oder dem Abbruch der zugehörigen TCP-Verbindung automatisch beendet und deren in Anspruch genommene Ressourcen wieder freigegeben werden, muss der Hauptthread keine Verwaltung der Threads übernehmen oder auf deren Beendigung warten. Dies wird durch das Setzen des Threadattributes *detachable* bei jeder Workerthread Generierung erreicht.

Die Workerthreads entnehmen der Verbindung das Datentupel (siehe Kapitel 3), verarbeiten es und schicken die gewünschten Informationen über die selbe Verbindung zurück. Die dazu notwendigen Daten bekommen die Threads, in dem sie zunächst eine *Pipe* erstellen, dann einen Kindprozess mit *fork()* generieren, den Kindprozess per *execl()* mit einer Shell die den Befehl ausführt ersetzen und die Ausgabe durch die Pipe an den Elternprozess umleiten. Die Pipe wird im Anschluss nach Ende der Ausgabe wieder geschlossen. Der gesamte Vorgang wird bei jeder Wiederholung ausgeführt. Da auf dem Stack jedes Workerthreads nur wenige Daten (ca. 2kB) liegen, wird die Größe des Stackspeichers bei der Erstellung der Threads auf das in Linux erlaubte Minimum von 16kB gesetzt. Das verhindert die Reservierung von unnötig viel virtuellem Speicher.

Damit im Monitor-Client mehrfache Ausgaben desselben Befehls nicht nur hintereinander im zugeteilten Dock-Widget angezeigt werden können, sondern auch eine Ansicht möglich ist, in der die neue Ausgabe die alte überschreibt, muss der Monitor-Server den Anfang jeder Befehlsausgabe signalisieren. Dies wird - um die Implementierung von Server und Client einfach zu halten - nicht Out-of-Band sondern durch ein vorangestelltes 0-Byte realisiert. 0-Bytes sollten in Texten wie Logdateien oder Befehlsausgaben nicht vorkommen, da sie historisch nur als Steuerzeichen für den Test einer Verbindung benutzt wurden und somit heute keinen Zweck erfüllen. Außerdem werden Texte in der Programmiersprache C als 0-Byte terminierte Zeichenarrays gespeichert, weswegen man von dessen Verwendung absehen sollte. Ein 0-Byte kann auch nicht bei der Kodierung in UTF-8 - welche heute in allen Linux-Distributionen voreingestellt ist - entstehen, wenn es nicht schon im ASCII-Text vorhanden ist, da das höchstwertige Bit aller Nicht-ASCII-Zeichen dort immer 1 ist.

Das Restrisiko, dass sich ein doch vorhandenes 0-Byte durch den gepufferten Versand zufällig genau am Anfang einer Übertragung befindet und dadurch ein vorzeitiges Überschreiben auslöst, ist demnach bei Textausgaben als gering zu bewerten. Sollte der Fall dennoch eintreten und es zur Unlesbarkeit einer Befehlsausgabe führen, so kann dies durch Einschalten der kontinuierlichen Ausgabe behoben werden, da dann alle am Anfang befindlichen 0-Bytes ignoriert werden und übertragener Text am Ende angefügt wird.

Sollten allerdings UTF-16 Ausgaben auf diese Weise angezeigt werden, müsste aufgrund der dort eingesetzten Kodierung ein doppeltes 0-Byte als Präfix verwendet werden. Unabhängig davon ist in dem Fall ein Umstellen der Kodierung im Quellcode des Monitor-Client zwingend notwendig, da diese auf UTF-8 eingestellt ist.

Kapitel 5

Evaluierung

Wie bereits in den vorherigen Kapiteln mehrfach erwähnt, wurde bei dem Design und der Implementierung des Monitor-Servers auf einen sehr geringen Ressourcenverbrauch geachtet, damit parallel durchgeführte Messungen möglichst wenig beeinflusst werden. Anhand von Tests, soll nun der tatsächliche Verbrauch abgeschätzt werden. Deswegen wählen wir die dabei verwendeten Kommandos so, dass sie eine reale Nutzung gut simulieren.

Zur Evaluierung des Speicher- und Rechenzeitverbrauchs des Monitor-Servers wird das Programm *top* eingesetzt, welches sowohl den benutzten virtuellen und privaten Speicher, als auch die verbrauchte CPU-Zeit inklusive aller erzeugter und wieder beendeter Kindprozesse anzeigt. Die in Tabelle 5.1 aufgeführten Befehle wurden dreimal 10 Minuten lang jeweils mit 1000ms (Tabelle 5.2) und 100ms Pause (Tabelle 5.3) nach jeder

ntpq -pn	Zeitversatz zu Referenzsystemen
top -b -n1	Ressourcenverbrauch der Prozesse
df -h	Dateisysteme und Speicherplatzverbrauch
cat /proc/swaps	Auslagerungsdatei
cat /proc/meminfo	Arbeitsspeicherverbrauch
ifconfig	Netzwerkadapterkonfiguration
tail /var/log/syslog	Letzte Zeilen der Systemlogdatei

Tabelle 5.1: Ausgewählte Befehle und deren Informationsausgabe

Test	CPU-Zeit (s)	CPU-Nutzung (%)	virt. Speicher (kB)	priv. Speicher (kB)
1	8,40	1,40	4400 - 4404	656
2	8,42	1,40	4400 - 4404	656
3	8,46	1,41	4400 - 4404	656
Ø	8,43	1,40	4402	656

Tabelle 5.2: Benötigte CPU-Zeit nach 10min bei 1000ms Pause nach jeder Befehlsausführung und verwendeter Arbeitsspeicher

Test	CPU-Zeit (s)	CPU-Nutzung (%)	virt. Speicher (kB)	priv. Speicher (kB)
1	26,32	4,39	4400 - 4412	652
2	25,56	4,26	4400 - 4412	652
3	27,18	4,53	4400 - 4412	652
Ø	26,35	4,39	4406	652

Tabelle 5.3: Benötigte CPU-Zeit nach 10min bei 100ms Pause nach jeder Befehlsausführung und verwendeter Arbeitsspeicher

Befehlsausführung auf einem Intel i7-2600 Prozessor mit 3,4 GHz auf nur einem Kern - durch Einstellung der CPU Affinität - gleichzeitig ausgeführt.

Der erste Test mit 1000ms Pause, sollte bereits eine obere Grenze für ein tatsächliches Nutzungsverhalten darstellen. Denn zum einen müssen Befehle nicht immer periodisch abgefragt werden und zum anderen sollten meistens Pausenintervalle von mehr als einer Sekunde ausreichend sein. Der zweite Test soll lediglich zeigen, wie der Monitor-Server in einer Extremsituation skaliert.

In den Tests wurde wie folgt vorgegangen:

1. Monitor-Server starten.

2. top mit folgenden Parameter starten:

   - -b (Textausgabe ohne Formatierung)
   - -d1 (1 Update pro Sekunde)
   - -p <Prozess-ID des Monitor-Servers>

- -S (kumulative Anzeige der CPU-Zeit)
- > monitor.log (Speichern in Logdatei)

3. Ausgewählte Befehle im Monitor-Client zum Zeitpunkt t_0 zeitgleich starten.
4. Nach 10 Minuten Monitor-Server und top beenden.
5. Im Logfile CPU-Zeit zum Zeitpunkt $t_1 = t_0 + 10\text{min}$ ablesen und durchschnittlichen CPU-Nutzung des Monitor-Servers ausrechnen (CPU-Zeit/10min).
6. Im Logfile minimalen und maximalen virtuellen und privaten Speicherverbrauch suchen.

Der Monitor-Server kann problemlos schon vor der Verbindung mit dem Client überwacht werden. Denn in einem vorher ausgeführten Langzeittest zeigte *top* bei einem gestarteten aber nicht verbundenen Monitor-Server nach 180 Minuten konstant 0,00 Sekunden verbrauchte CPU-Zeit an. Dies war zu erwarten, da im Hauptthread die Funktion *accept()* solange blockiert, bis eine TCP-Verbindung zum lauschenden Socket hergestellt wurde.

Aus dem letzten Schritt wird ersichtlich, dass es sich bei den Mittelwerten des virtuellen Speichers nur um Bereichsmittel und keine arithmetischen Mittelwerte wie bei der CPU-Zeit und der CPU-Nutzung handelt. Allerdings kann Aufgrund des kleinen Intervalles und der alternierenden Verteilung im Logfile geschlossen werden, dass der virtuelle Speicherverbrauch bei einer längeren Benutzung nicht signifikant größer oder kleiner werden wird.

Der Verbrauch des privaten Speicher blieb während der jeweiligen Tests konstant und war mit 652kB bzw. 656kB gering und ist somit auch für speicherarme Hardware geeignet.

Die CPU-Nutzung inklusive aller erzeugten Kindprozesse muss wegen des hier verwendeten Prozessors wahrscheinlich mindestens mit einem Faktor 7 multipliziert werden, da bereits die Taktfrequenz 6,8 mal so hoch ist und der neue Prozessor zusätzlich durch

entsprechende Optimierungen im Design, die von Compilern ausgenutzt werden können, schneller arbeitet.

Also sollte im Normalbetrieb mit maximal 10% CPU-Nutzung gerechnet werden. Aus der zweiten Testreihe kann man erkennen, dass ein schnelleres Abfragen aller Informationen zu einer ungefähren Last von umgerechnet 30% führen kann.

Um die Beeinträchtigung der Messungen möglichst klein zu halten, kann die Priorität des Monitor-Servers beim Start mit dem Kommando `nice -n <Priorität> monitor` oder während des Betriebes mit dem Befehl `renice <Priorität> <PID>` vom Client aus geändert werden. <Priorität> ist dabei ein Wert zwischen -20 und 19, wobei größere Zahlen eine niedrigere Priorität bedeuten. Die Prozess-ID <PID> des Servers erhält man ebenfalls über den Monitor-Client, durch den Befehl `ps -e | grep monitor`.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

In dieser Bachelorarbeit wurden zwei Programme und ein Abfrageprotokoll zur Überwachung von Mobilfunkmessungen durch Anzeige von Informationsdaten der Messcomputer entwickelt. Der Monitor-Client stellt mit Hilfe einer grafischen Benutzeroberfläche die vom Monitor-Server per TCP/IP-Verbindung zur Verfügung gestellten Daten dar. Besonderen Wert wurde bei der Implementierung des Servers auf eine sparsame Nutzung von CPU-Zeit und Arbeitsspeicher gelegt. Der Client wurde dagegen im Hinblick auf hohe Flexibilität der Ansicht, einfache Bedienung und Lauffähigkeit auf möglichst vielen Betriebssystemen programmiert.

Das Abfrageprotokoll besteht aus einer menschenlesbaren kommagetrennten Anfrage des Clients bestehend aus dem auszuführenden Befehl, der Pause nach der Befehlsausführung in Millisekunden und der Anzahl der Befehlsausführungen. Die Antwort des Servers ist die Ausgabe des Befehls mit der angeforderten Pause und in der gewünschten Anzahl. Zur Markierung des Anfangs einer Ausgabe wird dieser ein 0-Byte vorangestellt. Dadurch ist das Protokoll universell für alle Textbefehlsausgaben ohne Terminalsteuerbefehle geeignet und muss nicht für neue Befehle angepasst werden.

Die Aufbereitung der Daten als Log, Tabelle oder Plot findet somit ausschließlich im

Client statt, um den Server zu entlasten. Aus dem selben Grund werden Daten, die andere Dienste auf dem Messcomputer per Netzwerkschnittstelle anbieten, direkt vom Client abgerufen. Dies wurde bereits für GPSd implementiert, dessen Daten als Tabelle und Plot angezeigt werden können. Eine solche Visualisierung ist auch für Werte des NTP möglich.

6.2 Ausblick

Aufgrund der begrenzten Zeit war es nicht möglich alle Ideen die im Laufe des Projektes aufkamen zu realisieren. So sind beispielsweise viele Einstellungen der Plot-Fenster des Monitor-Clients nur als Präprozessordirektiven am Anfang des Quelltextes abgelegt und können somit nicht zur Laufzeit verändert werden.

Eine zusätzlich zur einfachen Eingabe vorhandene Auswahlmöglichkeit von voreingestellten Befehlen und Befehlslisten würde die Bedienung weiter vereinfachen. Diese könnten in der Einstellungsdatei gespeichert sein, sodass zur Änderung der voreingestellten Befehle kein Eingriff in den Quelltext nötig ist.

Da in der Einstellungsdatei die Positionen, Größen und Anordnungen des Haupt- und aller Unterfenster beim Beenden automatisch gespeichert werden, ist es denkbar durch manuelles Speichern und Laden von verschiedenen Einstellungsdateien, mehrere Layouts zu erstellen und abzurufen.

Damit der Monitor-Server aus Sicherheitsgründen nicht auf allen Netzwerkschnittstellen Befehle entgegen nimmt, bindet er sich beim Start ohne Parameter an das Loopbackinterface mit der IP-Adresse 127.0.0.1. Dies ist zwar bei der Entwicklung praktisch, aber im realen Einsatz nicht nützlich. Also wäre eine automatische Bindung an die IP-Adresse der Ethernetschnittstelle von Vorteil.

In seiner jetzigen Version nimmt der Server jede Verbindung auf der gebundenen Schnittstelle entgegen. Um unbefugten Zugriff zu verhindern, könnte ein Authentifizierungsmechanismus in beide Programme eingefügt werden.

Literaturverzeichnis

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

[IEE10] IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments. In: *IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009)* (2010), 15, S. 1 –51. `http://dx.doi.org/10.1109/IEEESTD.2010.5514475`. DOI 10.1109/IEEESTD.2010.5514475.

[RBK12] REILIN, Eeli; BARRETO, Luis Gustavo S.; KOCKENTIEDT, Stephen: *qt-json - A simple class for parsing JSON data into a QVariant hierarchy and vice versa (stabile Version vom 05.03.2012)*. Website, 2012. `https://github.com/ereilin/qt-json`; besucht am 21.04.2012.

[RW11] RATHMANN, Uwe; WILGEN, Josef: *Qwt - Qt Widgets for Technical Applications (Version 6.0.1)*. Website, 2011. `http://qwt.sourceforge.net`; besucht am 21.04.2012.

Ehrenwörtliche Erklärung

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

Düsseldorf, 24. April 2012 Daniel Glanert