Ein Publish-Subscribe Modem-Informations-Service

Bachelorarbeit

von

Adrian Skuballa

aus

Solingen

vorgelegt am

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

Februar 2012

Betreuer:
Norbert Goebel M. Sc.

Abstract

Die vorliegende Bachelorarbeit befasst sich mit der Entwicklung und Implementierung eines Publish-Subscribe Modem-Informations-Services. Seine Aufgabe ist es, technische Informationen vom Modem und dem infrastrukturbasierten Mobilfunk in Erfahrung zu bringen, um sie anschließend anderen Anwendungen zur Verfügung zu stellen. Diese Anwendungen können den Service in Anspruch nehmen, um z.B. das eigene Netzwerkkommunikationsverhalten anzupassen oder die Charakteristik des Mobilfunknetzes zu untersuchen und zu analysieren. Der hier entwickelte Modem-Informations-Service wird zukünftig der Erforschung des Mobilfunknetzes unter realen Bedingungen dienen. Es sollen die Vor- und Nachteile des Mobilfunks für verschiedene Fahrzeug-zu-Fahrzeug und Fahrzeug-zu-Umgebung Kommunikationslösungen untersucht werden.
Der Modem-Informations-Service bietet die Datenwerte des Modems per Publish-Subscribe an. Dies bedeutet, dass die Anwendungen die benötigten vorab abonnierten Datenwerte automatisch nach einem eingestellten Intervall erhalten. Die Tatsache, dass mehr Datenwerte verlangt werden als das Modem liefern kann, stellte im Rahmen dieser Arbeit eine Herausforderung dar. Als Lösung wurde eine Priorisierung der angeforderten Datenwerte herangezogen. Eine Priorisierung bestimmt, wie oft ein Datenwert bevorzugt wird, wenn zwei oder mehr Datenwerte gleichzeitig benötigt werden. So wird sichergestellt, dass der Service wichtige Daten regelmäßig und zeitnah an die Anwendungen ausliefert. Des Weiteren kann ein Datenwert mit unterschiedlichen Modemabfragen ermittelt werden. Eine Modemabfrage wiederum kann mehrere Datenwerte liefern. Da nicht direkt die Modemabfragen, sondern die Datenwerte priorisiert werden, ist eine Abbildung der Prioritäten der abonnierten Datenwerte auf die Modemabfragen nötig. Problematisch ist ferner, dass ein Datenwert mehrfach und mit unterschiedlichen Prioritäten abonniert werden kann. Zur Lösung wurden fünf Berechnungsverfahren und drei Scheduler entwickelt und implementiert, die eine pünktliche Veröffentlichung von Datenwerten garantieren. Datenwerte mit einer schwachen Priorität werden zwar seltener abgerufen als Datenwerte mit wichtiger Priorität, jedoch nicht gänzlich ausgelassen.
Das Augenmerk bei der Implementierung lag auf der Einhaltung der oben beschriebenen Eigenschaften sowie einer ressourcenschonenden Arbeitsweise, da der Publish-Subscribe Modem-Informations-Service permanent im Hintergrund läuft und über eine lange Zeitspanne in Anspruch genommen wird.

Danksagung

An dieser Stelle möchte ich die Gelegenheit nutzen, um alldenigen, die mir bei der Erstellung meiner Bachelorarbeit zur Seite standen, meinen besten Dank auszusprechen.

Zunächst danke ich Prof. Dr. Martin Mauve für die Herstellung der Rahmenbedingen, die die vorliegende Arbeit erst möglich machten.

Weiterhin möchte ich mich herzlich bei meinem Betreuer Norbert Goebel für die vielen und regelmässigen Diskussionen sowie hilfreichen Anregungen und Tipps bedanken.

Last but not least, danke ich insbesondere auch meiner Familie für ihre ausdauernde Unterstützung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kapitel 1.

Einleitung

1.1. Motivation

Obwohl Personenfahrzeuge in Deutschland immer leistungsfähiger und schneller werden, steigt die Anzahl der Verkehrsopfer nicht proportional zu dieser Entwicklung. Ganz im Gegenteil, seit 1991 sind die Zahlen stetig rückläufig gewesen. Wie das Statistische Bundesamt im Februar 2012 mitteilte, wurden erst 2011 wieder mehr Verkehrstote verzeichnet, was auf die Wetterbedingungen zurückgeführt wird [SBD11]. Zudem stieg die Anzahl in Deutschland registrierter Fahrzeuge kontinuierlich auf mittlerweile fast 50 Mio. an. Restriktivere Verkehrsregeln sowie Blitzeraufstellungen scheinen nicht mehr zu genügen, um die Verkehrsteilnehmer vor Unfällen und ihren Konsequenzen zu schützen [new12].

Sicherheitsmaßnahmen in den Fahrzeugen scheinen hier eine probate Lösung zu sein, um die Zahlen von Verkehrsopfern trotz steigender Verkehrsteilnahme zu reduzieren. Allerdings werden die meisten Sicherheitskonzepte, die aktuell in serienmäßigen Fahrzeugen verkauft werden (z.B. Airbag, ABS), erst im letzten Augenblick oder gar während des Unfalls aktiviert (*Post-Crash Sicherheit*). Die physikalischen Gesetze diktieren es, dass diese Art der Unfallvorbeugung nur bis zu einem bestimmten Punkt die Anzahl der Unfälle vermeiden und Folgen mindern kann. Es ist davon auszugehen, dass in naher Zukunft die Beschaffenheit von Bremsen und Reifen keine großen Entwicklungs-

sprünge machen wird, um ein Auto von 100 km/h innerhalb eines Meters sicher zum Halten zu bringen. Sollte eine solche Entwicklung dennoch gelingen, könnten sehr viele Autounfälle zwar vermieden werden, die Passagiere werden jedoch aufgrund der hohen wirkenden g-Kräften schwere Verletzungen davontragen.

Der Ausbau von *Pre-Crash-Systemen* scheint daher unerlässlich. Im Vordergrund dieser *Pre-Crash-Systeme* müssen die Früherkennung von gefährlichen Situationen sowie die Informationsverteilung an weitere potentiell gefährdete verkehrsteilnehmende Fahrzeuge stehen, um frühzeitig Sicherheitsmaßnahmen einzuleiten. Die Wirksamkeit eines solchen Systems ist wesentlich von einer effizienten Kommunikation mit anderen Fahrzeugen und der Umgebung abhängig. In diesem Bereich konkurrieren zurzeit zwei Techniken: W-LAN 802.11p und der klassische infrastrukturbasierte Mobilfunk. Der Lehrstuhl für Rechnernetze und Kommunikationssysteme der Heinrich-Heine-Universität Düsseldorf erforscht derzeit die Vor- und Nachteile beider Datenübertragungstechniken für den praktischen Einsatz in der Fahrzeug-zu-Fahrzeug (*Car-To-Car*) und Fahrzeug-zu-Umgebung (*Car-To-X*) Kommunikation.

Weil große und aussagekräftige Feldversuche, wie z.B. das Forschungsprojekt sim^{TD} (Sichere Intelligente Mobilität Testfeld Deutschland - [sim12]), aus Kostengründen nur sehr selten durchgeführt werden können, wird in der ersten Entwicklungsphase sehr häufig auf theoretische Modelle und Simulationen zurückgegriffen. Erst wenn eine Idee zur Sicherheits- oder Komfortsteigerung (z.B. Stauvermeidung) alle bekannten Praxissituationen in diversen Simulationen erfolgreich absolviert hat, kann sie Bestandteil eines Praxistests werden. Um Fehlentscheidungen zu vermeiden, ist es notwendig, dass diese Simulationen gewisse Qualitätskriterien, wie Reliabilität und Validität, erfüllen. Dies bedeutet, dass die Umgebungsvariablen der Simulation möglichst realitätsnah und zuverlässig sein müssen, um eine hohe Aussagekraft zu gewährleisten.

In diesem Kontext ist eine Simulation von Datenübertragungstechniken und somit die Vermessung und Analyse des klassischen infrastrukturbasierten Mobilfunks erforderlich. Im Rahmen dieser Bachelorarbeit soll ein Service entworfen und implementiert werden, der die technischen Informationen des aktuellen Zustands der Mobilfunkumgebung und der Eigenschaften des Modems erfasst und den Mess- und Analysesystemen zur Verfügung stellt. Damit die Messdaten dieser Systeme weitestgehend real und repräsentativ

sind, muss der entwickelte Service die Mobilfunkcharakteristik und Modemdaten schnell veröffentlichen (*Publish*). Nur so kann sich die Analysesoftware den neuen Bedingungen umgehend anpassen.

1.2. Aufbau der Arbeit

Der Fokus der vorliegenden Arbeit liegt auf der Entwicklung und Implementierung eines Dienstes, der Modem-Informationen erfasst und automatisch anderen Anwendungen periodisch zur Verfügung stellt.

Allerdings stößt hierbei das im Rahmen dieser Arbeit zur Verfügung gestellte Modem an seine Grenzen, da vom Modem mehr Daten angefordert werden, als dieses in einer bestimmten Zeitspanne liefern kann. Dieses Problem wurde durch die Implementierung drei unterschiedlicher Scheduler gelöst. Die Aufgabe eines Schedulers besteht darin zu entscheiden, welche Modemdaten zuerst abgefragt und veröffentlicht werden, wenn die Hardware-Grenzen des Modems erreicht sind und die geplante Verteilung nicht mehr möglich ist. Vorbereitend werden daher in Kapitel 2 die Grundfunktionalitäten von bekannten Schedulern aus dem Bereich der Netzwerktechnik und Betriebssysteme vorgestellt.

Kapitel 3 gibt einen Überblick über das Design, den Aufbau und wichtige Funktionen des entwickelten Modem-Informations-Services. In diesem Kapitel wird auf die zentralen Aufgaben des Services und die Komponenten und Algorithmen, die zur Lösung ausgesucht und entworfen wurden, eingegangen.

Anschließend werden in Kapitel 4 wichtige Techniken ausführlich präsentiert und ihre Implementierung dargestellt. Dieses Kapitel behandelt die Struktur und Funktionsweise von elementaren Komponenten wie des Netzwerkprotokolls, der Konfigurationsdatei oder der zentralen Algorithmen.

Abschließend fast Kapitel 5 die bei der Entwicklung entstandenen Probleme und ihre Lösungen zusammen und gibt einen Ausblick auf Erweiterungsmöglichkeiten des Modem-Informations-Services.

Kapitel 2.

Verwandte Arbeiten

Der MIS hat die Aufgabe nach dem Publish-Subscribe Ansatz Modeminformationen automatisch zu verteilen. Diese Modeminformationen werden mit dem AT Befehlssatz vom Modem abgefragt. Bei der Entwicklung des MIS sind wir auf das Problem gestoßen, dass zwei AT Befehle zeitgleich ausgeführt werden müssen. Scheduler eine probate Lösung an, weil hierbei AT Befehle sequentiell und nicht parallel ausgeführt werden. Ein Scheduler trifft die Entscheidung, welche Modeminformation zuerst abgerufen wird.

Mit einer steigenden Anzahl von abonnierten Modeminformationen, steigt die Wahrscheinlichkeit, dass zwei oder mehr Informationen gleichzeitig vom Modem abgefragt werden sollen. Falls zu viele Modeminformationen periodisch verlangt werden, können nicht alle angeforderten Daten rechtzeitig vom Modem abgefragt und verteilt werden. Die betroffenen Modeminformationen befinden sich in einer *Kollision*. Im Falle einer Kollision soll der Scheduler wichtige Informationen bevorzugen. Er darf aber unwichtige Daten nicht komplett auslassen.

Im Allgemeinen verteilt ein Scheduler eine bestimmte *Ressource* an verschiedene *Worker*. Typische Beispiele für Worker sind Prozesse oder Threads, die Prozessorzeit (Ressource) beanspruchen, oder Netzwerkpakete, die durch einen bestimmten Link (Ressource) gesendet werden müssen.

Für die Lösung des Problems wurden die folgenden bekannten Scheduler aus den Fachbereichen der Netzwerktechnik und der Betriebssysteme betrachtet:

- **Round Robin**
  Beim Round Robin Scheduling [KR08] werden alle Worker in eine Liste eingefügt. Als erstes erhält der Worker an Position eins die Ressource für eine bestimmte Zeit (*Quantum*). Wenn das Quantum vollständig abgelaufen ist, wird dem ersten Worker die Ressource entzogen und dem nächsten Worker zugeteilt. Nach diesem Muster werden die Worker sequentiell abgearbeitet, bis schließlich dem letzten Worker die Ressource entzogen wird. Sodann erhält wieder der erste Worker die Ressource. Der Round Robin Scheduler teilt jedem Worker dieselbe Menge an Zeit zu, so dass jeder Worker den gleichen Anteil von der Ressource erhält.

- **Earliest Deadline First** (EDF)
  Für den EDF Scheduler [Tan09] muss jeder Worker eine *Deadline* ankündigen, die angibt, bis wann dieser seine Arbeit beendet haben wird. Der EDF Scheduler sortiert alle Worker aufsteigend nach der Deadline und erteilt dem ersten Worker mit der frühesten Deadline die Ressource. Wenn dieser seine Arbeit beendet hat, wird er aus der Liste entfernt, sodass nun dem nächsten Worker die Ressource zugeteilt werden kann. Dieser Scheduler garantiert nicht die Einhaltung der Deadline. Die Deadline dient lediglich der Rangbildung. Eine gleichmäßige Aufteilung der Ressource ist nicht beabsichtigt.

- **First in First out / First come First served** (FIFO)
  Der FIFO Scheduler [KR08] verwaltet die Worker in einer Warteschlange. Die Rangfolge wird durch den Zeitpunkt des Eintreffens der Worker definiert. Der erste Worker erhält die Ressource, neue ankommende Worker werden hinten in die Schlange angestellt. Wenn der erste Worker seine Arbeit beendet hat, erhält der nächste Worker die Ressource. Auch bei diesem Scheduler ist eine gleichmäßige Aufteilung der Ressource nicht beabsichtigt.

- **Fair Share Scheduling**
  Beim Fair Share Scheduling [Tan09] werden die zu verteilenden Ressourcen (z.B. Prozessorzeit an Prozesse) nicht an die Worker an sich, sondern an die Gruppen, denen Worker angehören, verteilt. Es ist nicht das Ziel des Schedulers jedem Worker den gleichen Anteil der Ressource zuzuteilen, sondern jeder Gruppe. Das Entscheidende beim Fair Share Scheduling ist, dass jeder Worker einer Gruppe ange-

hört und dass jede Gruppe den gleichen Anteil der Ressource erhält. Ein klassisches Beispiel stellt das Prozessorscheduling dar. Hierbei werden die Prozesse, die Prozessorzeit benötigen, nach ihren Benutzern gruppiert. Bei n Gruppen erhält jede Gruppe ein n-tel der Prozessorzeit. Wenn weiterhin innerhalb einer Gruppe die Prozessorzeit nach Round Robin verteilt wird, wobei Benutzer A zehn und Benutzer B zwei Prozesse startet, dann erhält ein Prozess von Benutzer B die fünffache Prozessorzeit wie ein Prozess von Benutzer A.

- **Weighted Fair Queuing** (WFQ)
  Der WFQ Scheduler [Tan09] ist eine Erweiterung des Fair Share Schedulers. Der Unterschied besteht darin, dass beim WFQ jede Gruppe eine Priorität besitzt, die über die Größe des Anteils der Ressource entscheidet. Je wichtiger die Priorität, desto größer der Ressourcenanteil. Um den WFQ Scheduler umzusetzen, muss jeder Worker eine eindeutige Eigenschaft haben, die die Priorität und somit die Gruppeneinteilung bestimmt. Die Priorität eines Workers beziehungsweise einer Gruppe wird auch *Prioritätsklasse* genannt.

Wie im Kapitel 3.5.2 beschrieben, ist keiner der vorgestellten Scheduler geeignet das Problem der Kollision gänzlich zu lösen. Um einer Lösung des Kollisionsproblems möglichst nahe zu kommen, wurden Eigenschaften aus mehreren Schedulern übernommen und miteinander kombiniert (vergleiche Kapitel 3.4.2.6 und 3.5).

Kapitel 3.

Design und wichtige Funktionen

Diese Bachelorarbeit befasst sich mit dem Entwurf und der Implementierung eines Publish-Subscribe Modem-Informations-Services (im Folgenden *MIS* genannt), der technische Informationen (im Folgenden *Datenwerte* genannt) von einem Modem abruft, verwaltet und anderen Anwendungen (im Folgenden *Clients* genannt) zur Verfügung stellt. Es ist wesentlich, dass der Client möglichst aktuelle Datenwerte erhält, da er diese benötigt, um sein eigenes Netzwerkkommunikationsverhalten anzupassen oder, um das Netzwerk zu analysieren. Die primäre Aufgabe des MIS besteht darin, angeforderte Datenwerte (im Folgenden *periodische Anfragen* genannt) an die Clients periodisch zu verteilen. Somit agiert der MIS als *Publisher* und der Client als *Subscriber*. Dafür sollen die Datenwerte auf Wunsch vom Client automatisch vom MIS *gepusht* werden. Zusätzlich soll dem Client ein *Pull* einzelner Datenwerten ermöglicht werden.

Eine periodische Anfrage gibt nicht nur den vom Client benötigten Datenwert an, sondern auch den maximalen Zeitabstand in Millisekunden zwischen zwei aktuellen vom MIS veröffentlichten Datenwerten. Dabei sollen verschiedene von einem Client angeforderte Datenwerte unterschiedliche Prioritäten erhalten. Eine Priorisierung ist in diesem Fall erforderlich, da die Hardware des MIS und die Anzahl der Datenabrufe pro Zeiteinheit vom Modem limitiert sind. Die Priorität gibt die Zuverlässigkeit der Pünktlichkeit der periodischen angefragten Daten an. Um die zukünftigen Einsatzszenarien des MIS zu erweitern, ist die gleichzeitige Bedienung mehrerer Clients wünschenswert. Hierbei

kann es sich ergeben, dass ein Datenwert von unterschiedlichen Clients mit unterschiedlichen Prioritäten und Intervallen verlangt wird.

Die Datenabfragen des MIS stellen eine zusätzliche Belastung für das Modem dar. Denn parallel zu seiner primären Aufgabe, Daten senden und empfangen, muss das Modem nun auch auf die Datenanfragen des MIS reagieren und diese bearbeiten. Die Arbeit des MIS kann das Modem belasten und in ihrer Folge die Datenrate der primären Aufgabe verkleinern und ihre Latenz vergrößern. Somit ist es notwendig, dass die Datenabfragen, unter Berücksichtigung und Einhaltung aller periodischer angeforderter Datenwerte, möglichst auf ein notwendiges Minimum reduziert werden. Der MIS soll bei seiner Arbeit möglichst stabil, prioritätszuverlässig und ressourcenschonend für den Prozessor, den Arbeitsspeicher sowie das Modem vorgehen.

Der erste praktische Einsatz des MIS wird, verglichen mit aktuellen Computern, auf sehr schwacher Hardware stattfinden. Er soll auf kleinen portablen Computern arbeiten, die das Mobilfunknetz (GSM, UMTS, LTE) analysieren und testen. Da diese Messhardware auf 256 MB Arbeitsspeicher und einem 500 Mhz Prozessor aufgebaut ist, muss eine schnelle Arbeitsweise des MIS sichergestellt werden. Ein Betrieb auf leistungsstärkerer Hardware ist daher problemlos möglich. Da der MIS ressourcenschonend im Hintergrund laufen und seine Dienste den Analyse- und Testanwendungen anbieten muss, haben wir uns für eine Implementierung in C++ entschieden. Die Anforderung, dass die benötigten Daten und Informationen des Modems mit dem AT Befehlssatz abgefragt werden (siehe Kapitel 3.1), sprach ebenfalls für C++. Der AT Befehlssatz wird per serielle Schnittstelle abgefragt und dies ist in C++ ebenfalls performant möglich. Der schonende Umgang mit den Hardwareressourcen soll sich positiv auf den Stromverbrauch auswirken und längere Akkulaufzeiten zur Folge haben.

Der MIS bietet seine Dienste möglichst vielen Clients an. Hierfür wurde für die Kommunikation zwischen MIS und Clients ein wohldefiniertes Netzwerkprotokoll entworfen, das über TCP/IP angewendet wird. Da für alle gängigen Programmiersprachen eine TCP/IP Socket Implementierung existiert, kann die Programmiersprache für die Clients frei gewählt werden. Der MIS kann somit seine Dienste auch externen Clients anbieten.

3.1. AT Befehlssatz

Der AT Befehlssatz wurde vom US-amerikanischen Hersteller Hayes Communications entwickelt. Dieser Befehlssatz wurde zum ersten Mal in dem Modem *Smartphone 300* implementiert, welches 1981 auf den Markt kam. Die zwei Buchstaben *AT* sind die Abkürzung für ***At**tention* (Achtung). So kann der AT Befehl *ATD01805342020* mit *Attention, **d**ial (wähle) Rufnummer 01805342020* übersetzt werden.

Da ein AT Befehl an eine serielle Schnittstelle gesendet wird, muss das Modem an eine serielle Schnittstelle des Computers angeschlossen oder mit einer virtuellen seriellen Schnittstelle des Modemtreibers verbunden werden. Anschließend kann das Betriebssystem oder ein Programm mit dem Modem über diese serielle Schnittstelle nach den Richtlinien des Befehlssatzes kommunizieren. Dieses einfache Prinzip macht es möglich, dass AT Befehlssatz-konforme Modems von vielen unterschiedlichen Computern unterstützt werden. Dies führte dazu, dass AT Befehlssatz-konforme Modems eine sehr schnelle Verbreitung fanden. Die *International Telecommunication Union* (*ITU*) hat diesen Befehlssatz 1998 in die internationalen Standards *ITU-T V.250* aufgenommen [ITU98]. Im Laufe unserer Arbeit haben wir feststellen müssen, dass sich Hersteller an diesen internationalen Standards nicht konsequent halten, sondern lediglich als Empfehlung auffassen. Für diese Bachelorarbeit standen sechs verschiedene UMTS Modems von den Herstellern *Dell GmbH*, *Huawei Technologies Co*, *Sony Ericsson*, *ZTE Corporation* und *Sierra Wireless* zur Verfügung. In Abbildung 3.1 sind die Modems dargestellt. Nur der letztgenannte Hersteller bietet eine offizielle Dokumentation der unterstützten AT Befehle an [SWA09]. Alle anderen Unternehmen bieten diese Unterlagen nicht an oder haben auf Anfrage eine Zusammenarbeit abgelehnt. Aus diesen Gründen haben wir primär das Modem Model *MC8790* von Sierra Wireless benutzt. Die Modems der anderen Hersteller haben wir nur vereinzelt zur Überprüfung von verschiedenen Funktionen mit rudimentären AT Befehlen herangezogen. Fünf der sechs benutzten Modems sind in Abbildung fig:Modems aufgeführt.

Um einen Datenwert vom Modem zu erfragen, wird ein AT Befehl (*AT Anfrage*) an die zuständige serielle Schnittstelle gesendet. Anschließend erhält man von der seriellen Schnittstelle die *AT Antwort* des Modems. Alle AT Anfragen und AT Antworten werden

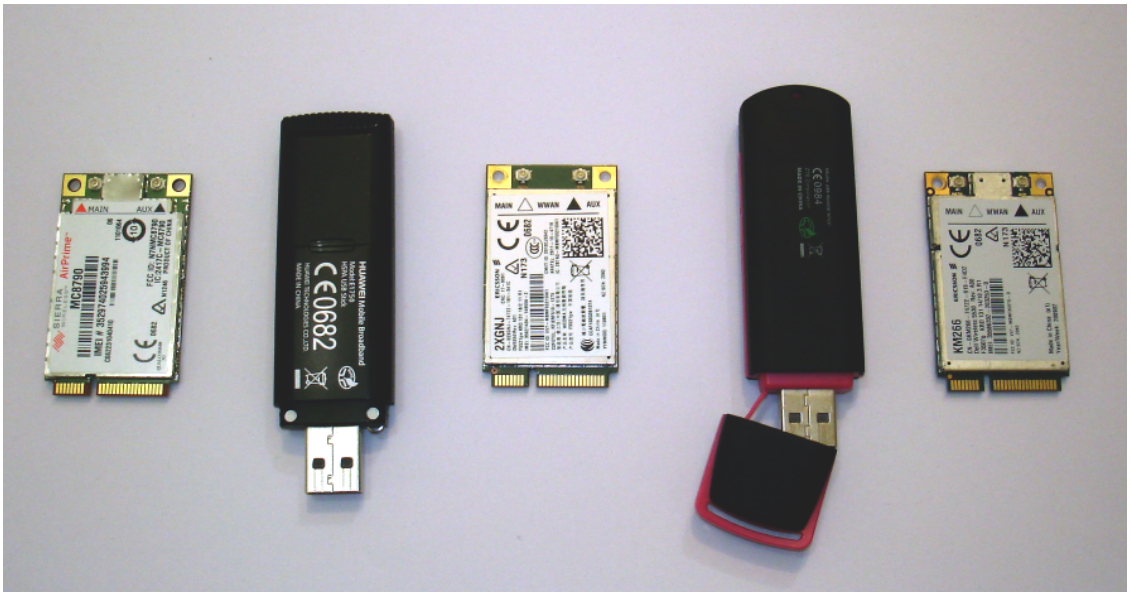

Abbildung 3.1.: v.l.n.r.: Sierra Wireless MC8790, Huawei Technologies Co. E1750, Sony Ericsson F5521gw, ZTE Corporation MF637, Sony Ericsson F3507g

menschenlesbar als ASCII übertragen. Numerische Werte werden entweder im Dezimal-, Oktal- oder Hexadezimalsystem kodiert. Es besteht die Möglichkeit, dass das Parsen der Daten vom MIS übernommen wird. Dies wird, wie in Kapitel 4.6 beschrieben, in der Konfigurationsdatei angegeben.

In der AT Antwort wird zuerst die ursprüngliche Anfrage wiederholt und anschließend folgen die dazugehörigen Daten. Die AT Antworten von erfolgreich verarbeiteten AT Anfragen enden bis auf wenige Ausnahmen mit *OK*. Bei Fehlern endet die AT Antwort mit *ERROR* oder *COMMAND NOT SUPPORT*. Viele AT Befehle geben mehrere Datenwerte in einer AT Antwort zurück. Es ist auch möglich den gleichen Datenwert mit unterschiedlichen AT Anfragen zu erfahren. Weiterhin kann das Modem mit AT Befehlen gesteuert werden (zum Beispiel: Pin Eingaben, Verbindungsaufbau, Ausgabeformat von AT Antworten, Funktionen (de)aktivieren). Viele AT Befehle liefern Informationen nicht nur mit einem einzelnen unabhängigen Datenwert, sondern auch als einen Datensatz, der aus mehreren zusammengehörigen Datenwerten besteht. So liefert der AT Befehl *AT+CSQ* zwei Datenwerte zur Signalqualität, zum einen den *received signal strength* und zum anderen den *channel bit error rate*. Es gibt auch AT Antworten, die aus ei-

ner flexiblen Liste von Datenwerten oder Datensätzen gebildet werden. Die im Rahmen dieser Bachelorarbeit explizit getesteten und implementierten AT Befehle können der Konfigurationsdatei entnommen werden (siehe Kapitel 4.6).

3.2. Netzwerkprotokoll

Die Kommunikation mit dem MIS wird per Netzwerk umgesetzt. Um eine hohe Kompatibilität zu erreichen, wird für die Vermittlungsschicht das *Internet Protocol Version 4* (IPv4) verwendet und, um Datenverlust auszuschließen, wird für die Transportschicht das *Transmission Control Protocol* (TCP) eingesetzt.

Da die eigentlich übertragenen Nachrichten aus kleinen Daten bestehen, wurde auf der Anwendungsschicht ein *in-band* Netzwerkprotokoll mit Baum-Struktur entworfen. Ein *out-of-band* Protokoll würde in diesem Fall die Komplexität nur unnötig erhöhen.

Nachrichtendatenpakete vom Client an den MIS und umgekehrt haben keine feste Länge. Ein Nachrichtenpaket kann aus maximal zwei Teilen bestehen. Der erste Teil (*Header*) gibt den Nachrichtentyp an und der zweite Teil beinhaltet, je nach Nachrichtentyp, die Nutzdaten.

Das erste Byte eines Nachrichten-Headers gibt die Gruppe an, zu der das Nachrichtenpaket gehört. Das nächste Byte gibt den Nachrichtentyp innerhalt der Gruppe an. Jedes weitere Byte des Nachrichten-Headers spezifiziert das Nachrichtenpaket genauer - bis der Nachrichtentyp eindeutig ist. Jedes Byte des Headers, das den Nachrichtentyp angibt, kann man als Knoten und die eigentlichen Nutzdaten als Blatt in einem Baum interpretieren (vergleiche Abbildung 3.2).

In der aktuellen Version des Netzwerkprotokolls werden Nachrichten vom Client an den MIS in fünf Gruppen eingeordnet und haben deshalb im ersten Byte die Werte null (0x00 - System) bis vier (0x04 – Manage Periodically Data Request).

Diese Baumstruktur des Netzwerkprotokolls bietet den Vorteil einer sehr einfachen Erweiterung des Protokolls um neue Nachrichtentypen. Des Weiteren lässt sich so auch

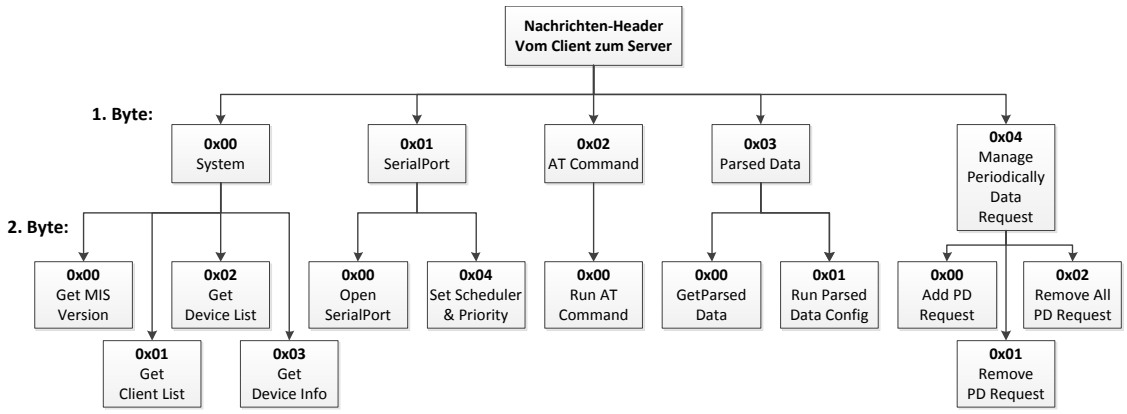

Abbildung 3.2.: Baum-Struktur des Netzwerkprotokolls (Client an MIS).

schnell und logisch analysieren, zu welchem Nachrichtentyp eine Nachricht gehört. Wenn ein Knoten weniger als 256 Kinder aufweist, werden nicht alle Bits des Headers genutzt. Dies kann jedoch vernachlässigt werden, da durch den logischen Aufbau, eine einfache und problemlose Erweiterung sowie eine schnelle Nachrichtenanalyse gewährleistet sind und überwiegen. Da in dem eigentlichen Nutzdatenbereich einer Nachricht nicht nur ASCII Zeichen in Form von Char-Arrays, sondern auch bereits geparste 16 und 32 Bit Datentypen (*Integer, Float*) übertragen werden, wurde die Übertragung dieser Daten auf *Little-Ending* festgelegt.

Die Nachrichten, die ein Client an den MIS senden kann, werden in die folgenden fünf Gruppen eingeordnet:

0x00 System: Erfragt und setzt elementare Eigenschaften des MIS

0x01 AT-SerialPort: Aufforderung einer Verbindung zu einer angegebenen seriellen Schnittstelle/Modem und Einstellung der Prioritäts-Berechnungsverfahren und des Schedulertyps

0x02 AT-Command: Aufforderung eine AT Anfrage an das Modem zu senden, um die originale und unveränderte AT Antwort zu erhalten

0x03 Parsed Data: Erfragt einen bestimmten Datenwert oder (Multi-)Datensatz des Modems und ermöglicht manuelle Konfigurationen

0x04 Manage Periodically Data Request: Ermöglicht das periodische Abonnieren und Abbestellen von Datenwerten und -sätzen

Die Nachrichten, die der MIS an einen Client senden kann, werden in sechs Gruppen eingeordnet. In den ersten fünf Gruppen werden die Pendants zu den Anfragen des Clients verwaltet. In der sechsten Gruppe befinden sich die automatisch vom MIS gesendeten Nachrichten, um die vorab vom Client abonnierten Datenwerte zu veröffentlichen. Die Baum-Struktur des Headers von Nachrichten, die vom MIS an den Client gesendet werden, kann der Abbildung 4.1 entnommen werden.

3.3. Aufgabenteilung im MIS

Die Funktionsweise des MIS wird in die drei Bereiche **Server**, **Client** und **SerialPort** unterteilt. Jede dieser Komponenten wird mit einem eigenen Thread realisiert und im Folgenden beschrieben. Die Abbildung 3.3 skizziert die Funktionsweise des Servers und Client Threads. Eine detaillierte Funktionsweise des SerialPort Threads ist in Abbildung 4.2 dargestellt.

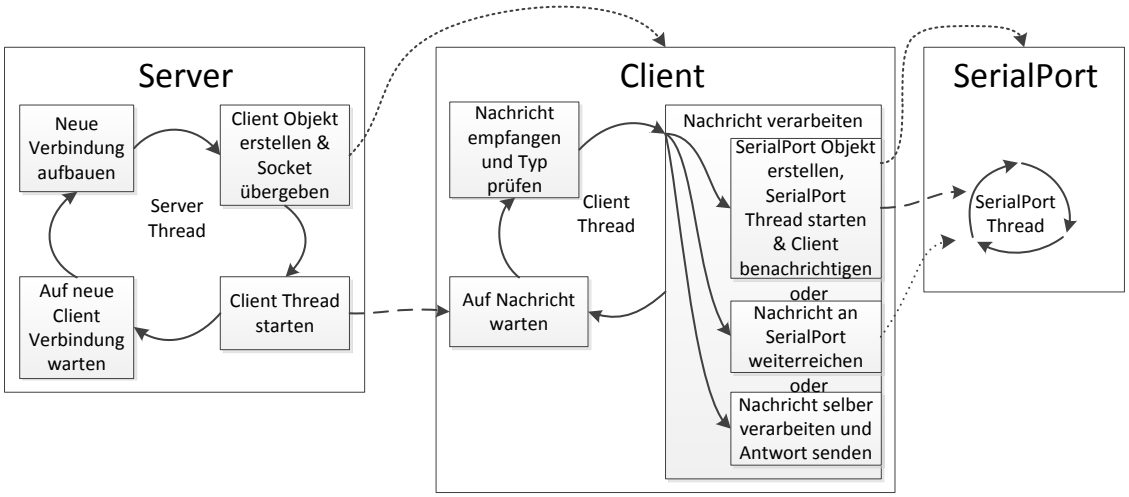

Abbildung 3.3.: Funktionsweise des Servers und Client Threads.

3.3.1. Server

Die Aufgabe des Servers ist es auf einkommende Verbindungen vom Client zu warten. Wenn eine Verbindung zum Client erfolgreich hergestellt wurde, erstellt der Server ein neues Client Objekt und startet den dazugehörigen Client Thread. Nachdem ein neuer Client erfolgreich erstellt und gestartet wurde, wartet der Server auf die nächste Clientverbindung.

3.3.2. Client

Der Client Thread wartet auf die ankommenden Nachrichten vom Client, nimmt diese entgegen und analysiert sie, um sie anschließend zu bearbeiten und bei Bedarf eine Antwortnachricht zu senden.

Ein Client kann auch den MIS beauftragen eine Verbindung zu einem Modem über eine serielle Schnittstelle herzustellen. Wenn der Client Thread diese Aufgabe per Netzwerknachricht erhält, versucht er das *Devicefile* der seriellen Schnittstelle zu öffnen. Anschließend informiert der Client Thread den Client, ob der Versuch erfolgreich war. Beim erfolgreichen Verbindungsaufbau über die serielle Schnittstelle zum Modem wird ein SerialPort Objekt erstellt und der dazugehörige SerialPort Thread gestartet.
Empfangene Nachrichtenpakete, die einmalige Datenwerte vom Modem verlangen oder periodische Anfragen ändern, werden nicht vom Client Thread bearbeitet. Diese Aufgabe reicht der Client Thread an den SerialPort Thread weiter.

3.3.3. SerialPort/Modem

Die komplette Kommunikation mit der seriellen Schnittstelle zum Modem wird nur vom SerialPort Thread durchgeführt. Dieser Thread bearbeitet die weitergeleiteten Aufgaben und bestimmt die dafür optimalen notwendigen AT Befehle, um alle angeforderten Datenwerte rechtzeitig vom Modem zu ermitteln und den Clients mitzuteilen.

Die geplante Vorgehensweise zur Nutzung des MIS sieht vor, dass ein Client zuerst jeden nötigen Datenwert mit einer periodischen Anfrage vom MIS abonniert (*Subscribe*). Im Idealfall empfängt dann der Client die periodisch angeforderten Datenwerte, die vom MIS selbstständig gesendet werden (*Publish*). Auf diese Weise sendet der MIS viel mehr Nachrichten als er empfängt. Aus mancegründen werden die Datenwerte vom SerialPort Thread direkt an die Clients gesendet und nicht den einzelnen Client Threads zum Senden übergeben.

3.3.4. Aufgabenteilung und ihre Konsequenzen

Zusammengefasst wartet der Server Thread auf neue Clients und baut eine Verbindung auf. Der Client Thread empfängt und verarbeitet alle eingehenden Nachrichten, bearbeitet diese und antwortet dem Client. Falls aber die Client Nachricht verlangt, dass ein Datenwert einmalig benötigt oder eine periodische Anfrage geändert wird, so werden diese Informationen an den SerialPort Thread zur Bearbeitung übergeben. Diese Aufgabenteilung macht es möglich, dass mehrere Clients und auch mehrere Modems vom MIS verwaltet und bedient werden.
Die geplante Benutzung und Arbeitsweise beim Praxiseinsatz des MIS sieht vor, dass es wenige Clients geben wird, die die Datenwerte von sehr wenigen Modems für eine längere Zeit abonnieren. Somit werden Client und SerialPort Objekte selten erstellt und die dazugehörigen Threads dementsprechend selten gestartet. Aus diesem Grund haben wir uns gegen ein Socket multiplexing oder ein I/O multiplexing entschieden.

3.4. Konfiguration der Priorität und ihre Auswirkungen

3.4.1. Ermittlung der periodischen Aufgaben für den Scheduler

Es wird erwartet, dass in den meisten Fällen, unabhängig von der Anzahl der Clients, mehrere Datenwerte gleichzeitig beim MIS abonniert werden. Damit der MIS möglichst ressourcenschonend arbeitet, werden alle abonnierten Anfragen mit der geringsten Anzahl an AT Befehlen beantwortet. Es ist die Aufgabe des MIS die dafür notwendigen AT Befehle zu ermitteln. Wie im Kapitel 3.1 bereits erwähnt wurde, ist es möglich, dass ein Datenwert mit mehreren AT Anfragen ermittelt wird und dass eine AT Antwort auch mehrere Datenwerte liefert. Die minimale Menge der AT Befehle wird deshalb mit einer rekursiven Tiefensuche über alle möglichen AT Befehle gefunden.

Aus jedem dieser AT Befehle wird eine *periodische Aufgabe* gebildet. Der SerialPort Thread bearbeitet nicht direkt die periodischen Abfragen, sondern die periodischen Aufgaben. Wenn der SerialPort Thread mit der Bearbeitung einer periodischen Aufgabe nicht anfangen kann, weil er noch mit der letzten Aufgabe beschäftigt ist, kommt es zu einer *temporären Kollision*.

Damit wichtige Datenwerte dennoch pünktlich beim Client ankommen, müssen Clients die periodische Anfrage priorisieren. Wenn es zu einer temporären Kollision von mehreren periodischen Aufgaben kommt, entscheidet die Priorität, welche Aufgabe bearbeitet wird. Somit regelt die Priorität die Wahrscheinlichkeit, wie pünktlich ein Datenwert beim Client ankommt. Bei einer Überbelastung durch zu viele periodische Aufgaben, gibt die Priorität die Verteilung der einzelnen Aufgaben an. Um diese Funktionalität umzusetzen wird ein Scheduler benötigt.

3.4.2. Berechnung der Prioritätsgewichte

Bevor der Scheduler seiner Arbeit nachgehen kann, müssen die Prioritätsgewichte (im Folgenden auch *Gewichte* genannt) der periodischen Anfragen auf die periodischen Aufgaben abgebildet werden. Jede periodische Anfrage wird einer von maximal 255 Prioritätsklassen zugeordnet. Jede Prioritätsklasse hat ein Gewicht, das angibt, wie viele Ressourcen dieser Klasse zur Verfügung stehen. Wenn eine Prioritätsklasse A ein Gewicht von 60 hat, so soll sie die doppelten Ressourcen erhalten, wie eine Prioritätsklasse B mit einem Gewicht von 30. Für den MIS wurden fünf Varianten mit verschiedenen Eigenschaften entworfen, die beschreiben, wie sich das Gewicht der jeweiligen periodischen Aufgabe aus den Gewichten der periodischen Anfragen berechnet.

Um diese fünf Berechnungsverfahren verständlich zu erklären, werden sie in den folgenden Unterkapiteln an einem Beispiel und jeweils einer Abbildung erklärt und abschließend miteinander verglichen.
Um die berechneten Gewichte der periodischen Aufgaben nachzuvollziehen, muss man in den Abbildungen 3.5, 3.6, 3.7, 3.8 und 3.9 die Gewichte, die an den Clients, Prioritätsklassen, Datenwerten und Aufgaben notiert sind, von unten nach oben verfolgen. Die neue primäre Eigenschaft des aktuell besprochenen Berechnungsverfahrens ist in den Abbildungen rot eingefärbt. Eine periodische Aufgabe erhält die Summe aller Anfragen ihrer Datenwerte.

Abbildung 3.4 veranschaulicht die Beziehung zwischen Clients, benutzten Prioritätsklassen, angeforderten Datenwerten und periodischen Aufgaben. Für alle Beispiele gilt die folgende Ausgangssituation:
Wir nehmen an, dass es drei Clients (A, B, C) und drei Prioritätsklassen mit den Gewichten 60, 30 und 10 gibt. Die drei Clients haben acht periodische Anfragen an den MIS gesendet, welche wiederum zu vier periodischen Aufgaben führen.
Client A verlangt mit Priorität 1 die Datenwerte 1.1 und 1.2 und mit Priorität 3 den Datenwert 2.1.
Client B verlangt mit Priorität 1 die Datenwerte 2.1 und 3.1, mit Priorität 2 den Datenwert 3.2 und mit Priorität 3 den Datenwert 3.3.
Client C verlangt nur den Datenwert 4.2 mit Priorität 3.

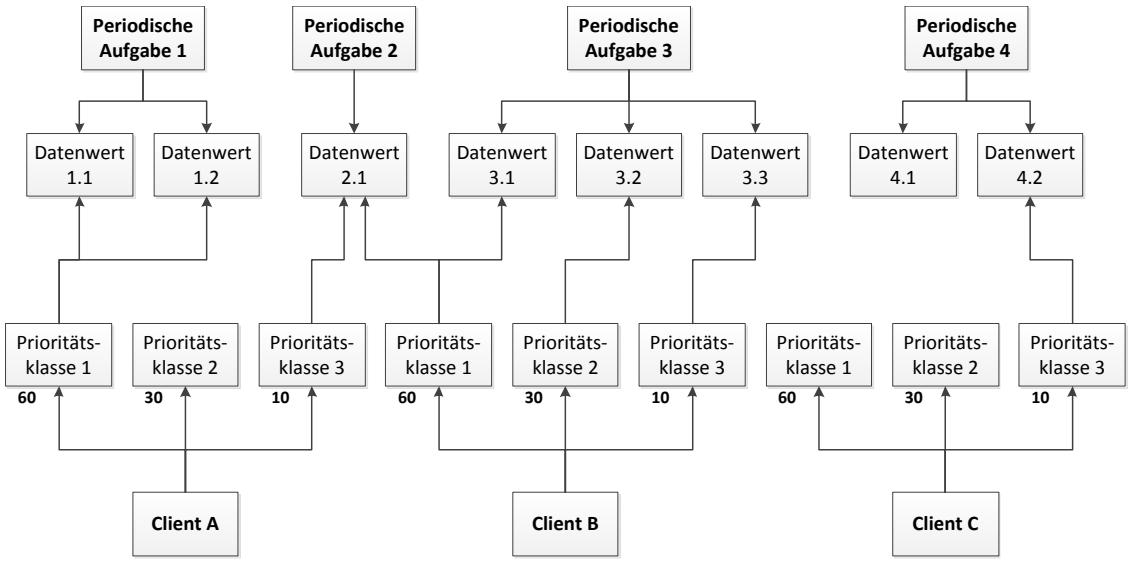

Abbildung 3.4.: Beziehung zwischen Clients, Prioritätsklassen, Datenwerten und AT Befehlen.

3.4.2.1. **Kopie der Prioritätsgewichte** (Prio. kopieren)

Beim ersten Berechnungsverfahren in Abbildung 3.5 wird das Gewicht der Prioritätsklassen an die periodischen angeforderten Datenwerte vererbt. Jede Datenanfrage erhält die gleiche Gewichtung wie die Prioritätsklasse mit der sie angefordert wird.

3.4.2.2. **Teilung der Prioritätsgewichte** (Prio. teilen)

Beim nächsten Berechnungsverfahren in Abbildung 3.6 wird das Gewicht der Prioritätsklassen an die angeforderten Datenwerte verteilt. Wenn ein Client n Anfragen mit der ersten Priorität stellt, dann erhält jede dieser Datenanfragen den n-ten Teil des Gewichts der ersten Priorität.

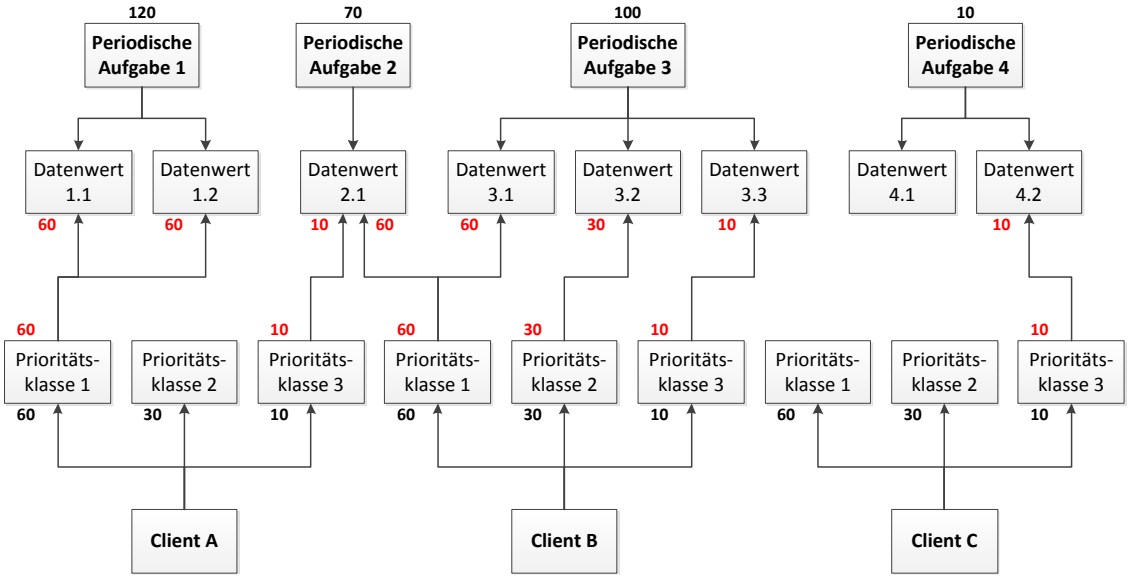

Abbildung 3.5.: Kopie der Gewichte von Prioritätsklassen (Prio. kopieren).

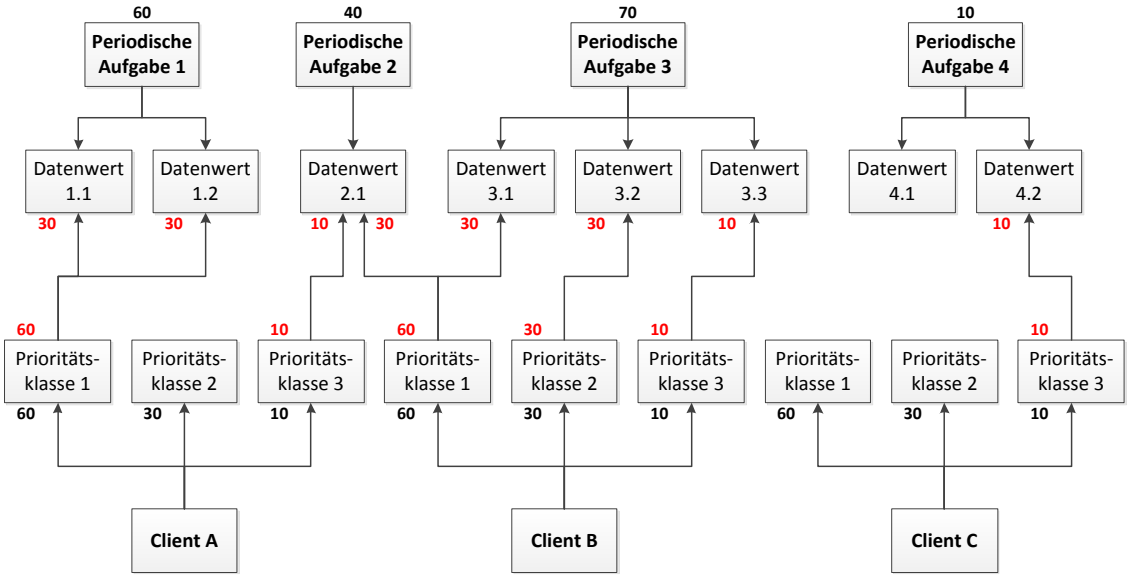

Abbildung 3.6.: Teilung der Gewichte der Prioritätsklasse (Prio. teilen).

3.4.2.3. **Erhalt der Client Ressourcen – Kopie der Prioritätsgewichte** (Client Prio. kopieren)

Das nächste Berechnungsverfahren in Abbildung 3.7 ist eine Erweiterung des ersten Verfahrens aus 3.4.2.1. Bevor das Gewicht der Prioritätsklasse an die Anfragen der Datenwerte kopiert wird, werden die Gewichte der Prioritätsklasse von jedem Client normiert. Dies hat zur Folge, dass ein Client, unabhängig von der Anzahl der periodischen Anfragen und ihrer Prioritäten, ein Minimum an Ressourcen vom MIS zugeteilt bekommt.

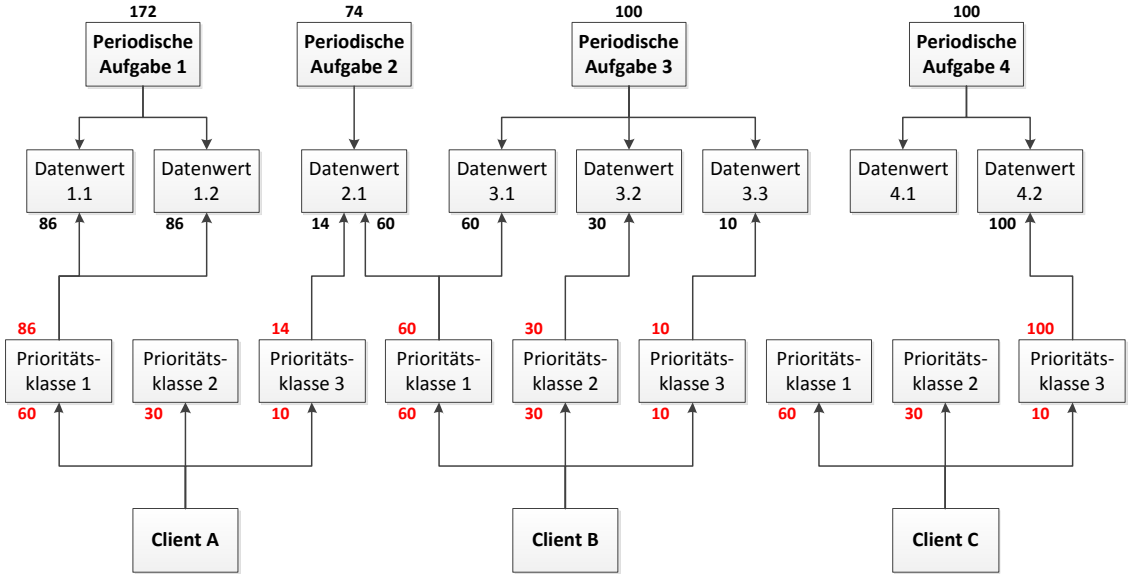

Abbildung 3.7.: Erhalt der Client Ressource und Kopie der Gewichte der Prioritätsklasse (Client Prio. kopieren).

3.4.2.4. **Erhalt der Client Ressourcen – Teilung der Prioritätsgewichte** (Client Prio. teilen)

Das vierte Berechnungsverfahren (Abbildung 3.8) ist eine Erweiterung des zweiten Verfahrens aus Kapitel 3.4.2.2. Bevor das Gewicht der Prioritätsklasse an die Anfragen der Datenwerte verteilt wird, werden, genau wie im vorherigen Verfahren, die Gewichte der

Prioritätsklasse von jedem Client normiert. Dies hat zur Folge, dass jeder Client, unabhängig von der Anzahl der periodischen Anfragen und ihrer Prioritäten, den gleichen Anteil an Ressourcen vom MIS zugeteilt bekommt.

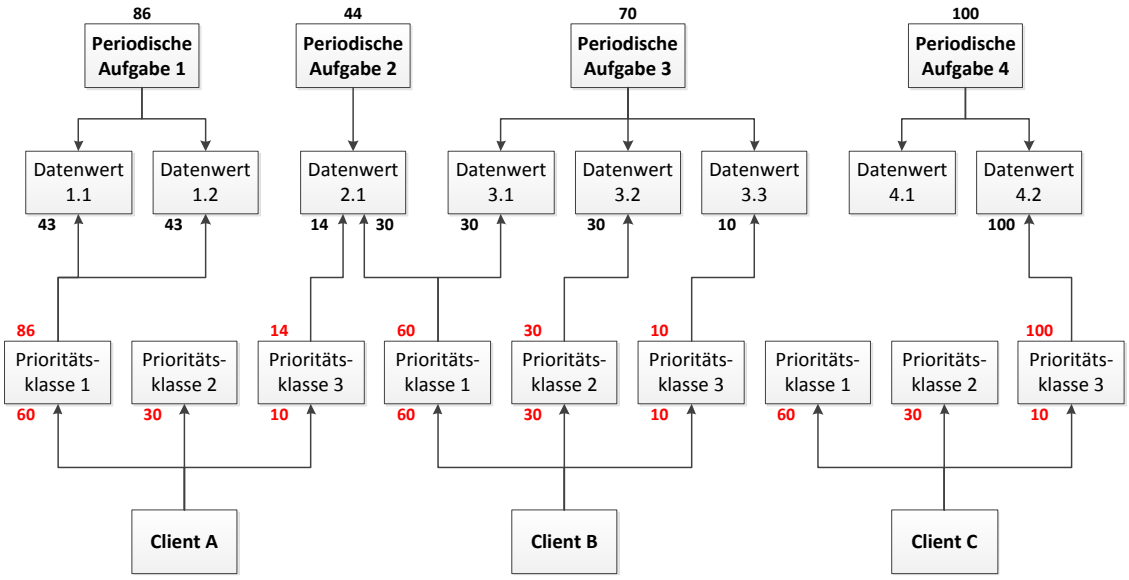

Abbildung 3.8.: Erhalt der Client Ressource und Teilen der Gewichte der Prioritätsklasse (Client Prio. teilen).

3.4.2.5. **Erhalt der Client Ressourcen über alle Datenwerte** (Prio. Datenwert)

Im letzten Berechnungsverfahren wird das Gewicht nicht über die Prioritätsklassen des Clients verteilt, sondern über alle angefragten Datenwerte des Clients (siehe Abbildung 3.9).

3.4.2.6. Berechnungsverfahren im Vergleich

Das Balkendiagramm in Abbildung 3.10 zeigt, dass die fünf Berechnungsverfahren zum Teil sehr unterschiedliche Ergebnisse liefern. Um einen direkten Vergleich der fünf Berechnungsverfahren zu ermöglichen, wurden die Gewichte der AT Befehle von jedem Verfahren auf den Wert 1,0 normiert.

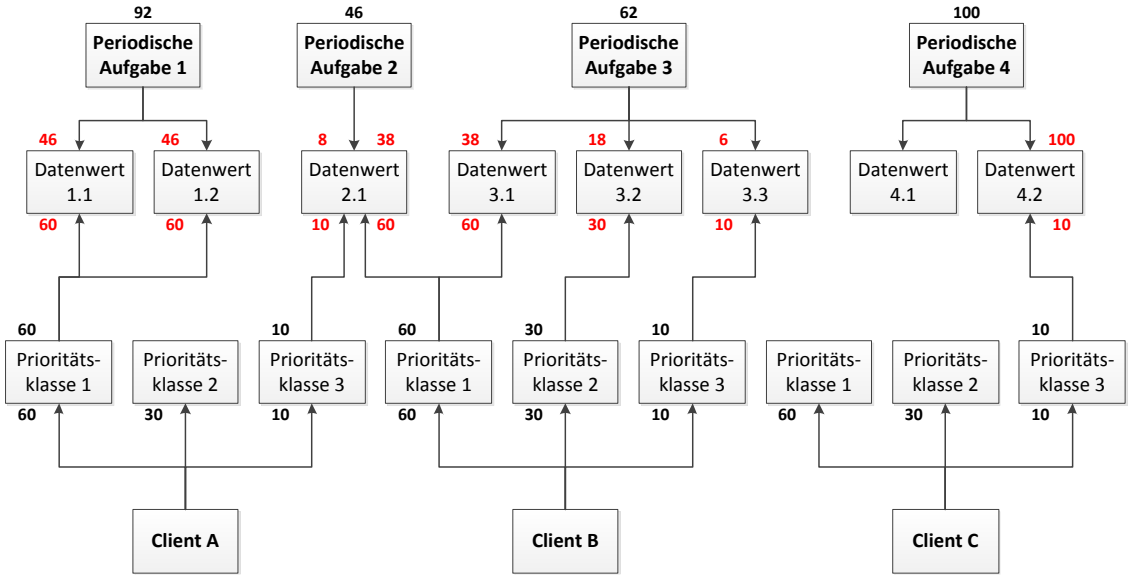

Abbildung 3.9.: Erhalt der Client Ressource über alle periodischen Anfragen (Prio. Datenwert).

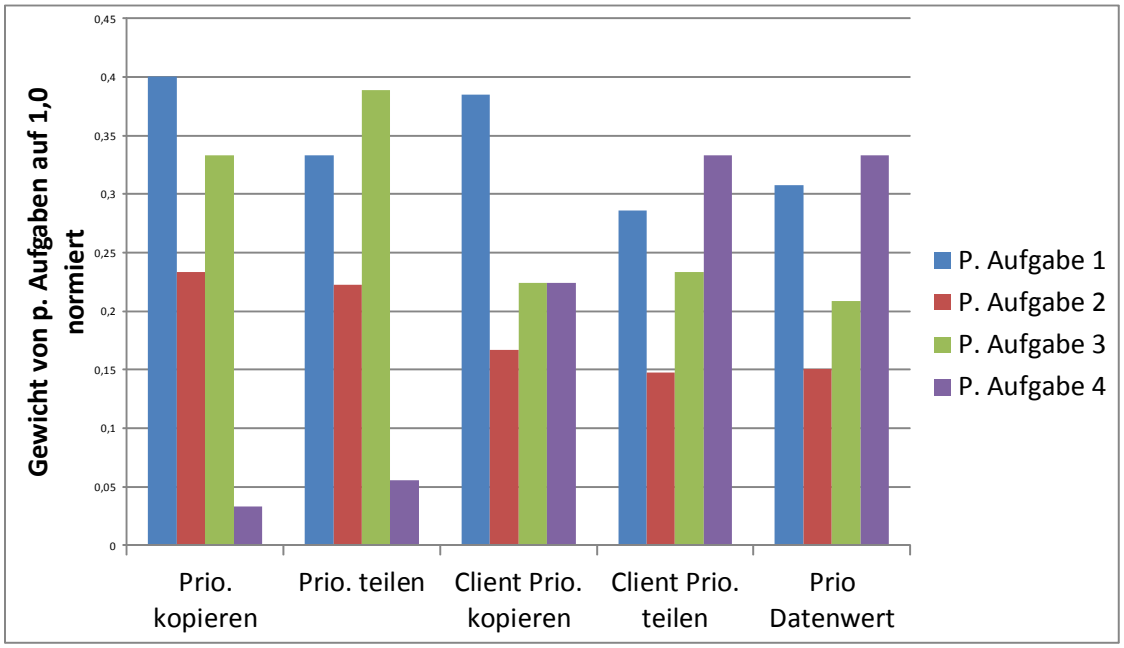

Abbildung 3.10.: Direkter Vergleich der Prioritäts-Berechnungsverfahren der periodischen Aufgaben.

Das erste Verfahren ist für den Fall gedacht, dass nur ein Client existiert und dieser selbst

die Prioritätsgewichte der Anfragen bestimmen möchte.
Mit dem zweiten Verfahren kann ein Client Anfragegruppen verwalten. Eine Anfragegruppe soll unabhängig von der Anzahl ihrer Anfragen ein Gewicht erhalten. Jede Anfragegruppe wird durch eine Prioritätsklasse dargestellt.

Die ersten beiden Verfahren sind primär für den Betrieb mit nur einem Client konzipiert. Denn diese Verfahren erlauben es, dass ein Client mit nur einer periodischen Anfrage von einem zweiten Client mit sehr vielen Anfragen verdrängt wird. Vor allem das erste Verfahren (*Prio. kopieren*) macht dies leicht möglich, da jede weitere Anfrage das ganze Gewicht seiner Prioritätsklasse erhält. Ein Client kann (un-)absichtlich *bösartig* werden, wenn er genügend viele periodische Anfragen stellt, und deshalb einen großen Teil der Ressourcen erhält. (Vergleichbar mit Download-Managern die parallele TCP-Verbindungen nutzen um mehr Bandbreite zu erhalten.)
Im zweiten Verfahren (*Prio. teilen*) sind einem *bösartigen* Client Grenzen auferlegt, weil das Gewicht der Prioritätsklasse geteilt wird. Ein Client kann sich somit nicht gezielt *stärker* machen. Damit eine Verdrängung entsteht, muss sich ein Client aktiv herabsetzen, in dem er nur die schwächste Prioritätsklasse nutzt.

Für den Betrieb von mehreren Clients wurden die nächsten drei Verfahren entworfen. Verfahren Nummer drei (*Client Prio. kopieren*) stellt sicher, dass jeder Client ein Minimum an Ressourcen erhält, indem ein absichtliches oder unabsichtliches Abschwächen verhindert wird. Jedoch schützt es nicht vor *bösartigen* Clients und Verdrängung.

Das Verfahren *Client Prio. teilen* schützt sowohl vor *bösartigen* Clients und Verdrängung, wie auch vor (un-)absichtlicher Abschwächung. Es sichert jedem von n Clients den n-ten Teil der Ressourcen. Die Eigenschaft, dass das Gewicht der Prioritätsklasse geteilt wird, ermöglicht es, dass jeder Client eigene Anfragegruppen erstellen kann.

Das letzte Verfahren (*Prio. Datenwert*) gewährt dem Client die gleiche Freiheit wie das erste Verfahren in Bezug auf die Bestimmung der Gewichte der Anfragen. Im Gegensatz zum ersten Verfahren, schützt es vor Verdrängung, indem jeder Client einen n-ten Teil der Ressourcen erhält.

Diese fünf Prioritäts-Berechnungsverfahren sollen das Benutzen des MIS erleichtern und seine Einsatzmöglichkeiten erhöhen. Dem Anwender wird damit die Möglichkeit gege-

ben, je nach Situation das gewünschte Verfahren mit der Konfigurationsdatei oder nachträglich mit einem Client einzustellen.

3.5. Scheduler

Der Scheduler entscheidet, welche periodische Aufgabe bei einer temporären Kollision als nächste ausgeführt wird, um Datenwerte der AT Antwort an die Clients zu senden. Wenn es zu einer temporären Kollision von mindestens zwei periodischen Aufgaben kommt, soll bei der Auswahl der nächsten auszuführenden Aufgabe ihr Gewicht Berücksichtigung finden.

3.5.1. Eigenschaften der Scheduler

Dem Scheduler müssen die folgenden vier Eigenschaften bekannt sein, damit er seiner Funktion nachgehen kann:

- Anzahl der Prioritätsklassen
- Gewicht jeder Prioritätsklasse
- Berechnungsverfahren des Gewichts
- Schedulertyp

Da eine periodische Aufgabe nicht direkt einer Prioritätsklasse zugeordnet ist, gibt es mehrere Berechnungsverfahren, die definieren, wie das Gewicht der periodischen Aufgabe aus dem Gewicht der periodischen Anfragen ermittelt wird (siehe Kapitel 3.4.2). Die ersten drei Eigenschaften werden für jedes der fünf Berechnungsverfahren benötigt. Der Schedulertyp gibt an, welcher Algorithmus benutzt werden soll, wenn es zu einer temporären Kollision kommt und entschieden werden muss, welche periodische Aufgabe als nächste ausgeführt wird.

Ein Scheduler hat im MIS zwei Aufgaben:

- Vor dem Beginn der Bearbeitung der periodischen Aufgaben werden die Gewichte der Aufgaben nach dem gewählten Berechnungsverfahren ermittelt.

- Vor jeder Bearbeitung einer periodischen Aufgabe wird geprüft, ob eine temporäre Kollision entstehen würde. Im Falle einer Kollision wird dann der entsprechenden Aufgabe (in Abhängigkeit von ihrem Gewicht) der Vorzug zur Bearbeitung erteilt.

3.5.1.1. Anforderungen an einen Scheduler

Wir nehmen an, dass ein Client drei periodische Anfragen mit einem Zeitabstand von einer Millisekunde verlangt. Jeder Datenwert hat eine andere Prioritätsklasse. Da bereits die Modem-Kommunikation und das Parsen eines Datenwertes ca. 3 bis 15 Millisekunden benötigen, ist das Einhalten der vom Client vorgegebenen periodischen Zeitangaben für den MIS unmöglich. Dies hat zur Folge, dass es immer zu Kollisionen kommt, der SerialPort Thread pausenlos Daten vom Modem abfragt und an den Client sendet.

Durch die Priorisierung soll aber erreicht werden, dass ein Datenwert, mit einem hohen Prioritätsgewicht, öfter vom Modem abgefragt und dem Client mitgeteilt wird. Zu beachten ist, dass ein Datenwert und seine periodische Aufgabe mit dem kleinsten Prioritätsgewicht zwar seltener bearbeitet, aber nicht komplett ausgelassen wird und *verhungert*.

Um das Verhungern von periodischen Anfragen bei Überlastung des MIS bzw. des Modems zu verhindern und gleichzeitig Prioritätstreue einzuhalten, wurden ergänzend zu einem angepassten Round Robin Scheduler [Tan09], zwei weitere komplexere Scheduler mit Aging implementiert. Auf die drei Scheduler wird in den nächsten drei Unterkapiteln eingegangen.

Ein Scheduler soll im MIS zwei unterschiedliche Situationen beherrschen können, die gleichzeitig seine Anforderungen beschreiben:

- Wenn es nur wenige periodische Aufgaben gibt, die einen sehr großen Zeitabstand (z.B. > 500ms) aufweisen, so kommt es nur selten zu Kollisionen. In diesem Fall

soll die Funktionalität des Schedulers die Gewichte der periodischen Aufgaben für die Wahrscheinlichkeit einer pünktlichen Bearbeitung heranziehen.

- Wenn es zu sehr vielen oder ununterbrochen zu Kollisionen kommt (*Worst-Case*), dann soll das Gewicht die Anzahl der Ausführungen einer Aufgabe bestimmen.

Ein guter Scheduler geht nicht nur einen Kompromiss ein und kommt beiden Kriterien nur ansatzweise nach, sondern erfüllt beide Kriterien. Im Rahmen dieser Bachelorarbeit wurde ein Scheduler entworfen und implementiert, der beide Eigenschaften erfüllt (invertierter Aging Scheduler – Kapitel 3.5.4).

Anforderungen an einen Scheduler an einem Beispiel:
Wenn drei periodische Aufgaben die Gewichte 60, 30 und 10 haben, so soll in der ersten Situation die erste Aufgabe eine Wahrscheinlichkeit von 60 % haben, dass sie pünktlich bearbeitet wird. Dem folgend soll die zweite Aufgabe mit einer Wahrscheinlichkeit von 30 % und die letzte mit 10 % pünktlich bearbeitet werden.
In der zweiten Situation (*Worst-Case*) soll, wenn der MIS insgesamt 1.000 Aufgaben ausführt, die erste Aufgabe 600 Mal, die zweite 300 Mal und die dritte 100 Mal ausgeführt werden.

Es wurden insgesamt drei Scheduler implementiert, so dass die MIS Funktionalität vergrößert wird und der Anwender nicht nur das Prioritäts-Berechnungsverfahren, sondern auch den Scheduler frei wählen kann.

3.5.2. Bekannte Scheduler

Wir haben folgende bekannte Scheduler verglichen und ihre Eigenschaften und Eignung für den MIS untersucht:

- Fair-Share-Scheduling [Tan09]
  Der Grundgedanke, die Ressourcen fair auf die User (Clients) zu verteilen, wurde bei den Prioritäts-Berechnungsverfahren übernommen und angepasst.

- Earliest Deadline First (EDF) [KR08]
  Ein reiner EDF Scheduler kommt beim MIS nicht in Fragen, da er keine Prioritäten berücksichtig. Für diese Arbeit wurde vom EDF Scheduler lediglich die Eigenschaft der Ermittlung der nächsten periodischen Aufgabe übernommen. Ob diese Aufgabe aber ausgeführt wird, wird in einem zweiten Schritt ermittelt.

- First in First out (FIFO) [KR08]
  Der FIFO Ansatz wiederspricht der primären Funktion des MIS (mit vorgegebenem Zeitabstand Datenwerte periodisch zu verteilen), denn eine Aufgabe mit einer sehr kleinen Frequenz würde, wenn sie an der Reihe wäre, andere Aufgaben mit einer sehr hohen Frequenz blockieren.

- Round Robin [Tan09]
  Ausgeschlossen wurde ebenfalls der unveränderte Round Robin Algorithmus, da dieser ohne Priorität arbeitet und deshalb in diesem Kontext nicht angewendet werden kann. Es besteht die Möglichkeit ein Quantum des Round Robin Schedulers mit einer Ausführung eines AT Befehls gleichzusetzen und die Menge aller AT Befehle, die in einer Kollision betroffen sind, als Warteschlange zu identifizieren. Vorausgesetzt, dass Aufgaben nach der Größe ihrer Gewichte bearbeitet werden, könnte dieser Scheduler eingesetzt werden.
  Wegen seiner übersichtlichen und elementaren Funktionsweise wurde dieser Scheduler mit den beschriebenen Änderungen implementiert.

- Weighted-Fair-Queuing-Algorithmus (WFQ) [KR08]
  Obwohl der WFQ Scheduler durch seine verschiedenen Prioritätsklassen besser geeignet scheint als der Round Robin Algorithmus, wurde dieser nicht umgesetzt. Da eine Implementierung deutlich aufwendiger wäre und dennoch nicht die exakten geforderten Eigenschaften sicherstellen würde (siehe Kapitel 3.5.1.1 - Einfaches Beispiel der zwei Situationen).

3.5.3. Aging Scheduler

Um ein besseres Ergebnis vor allem im Worst-Case Szenario zu erzielen, haben wir uns für einen zusätzlichen Aging Scheduler entschieden.
Bei diesem Scheduler werden die *Gewichtungswerte* aller periodischen Aufgaben, die in einer Kollision betroffen sind, verglichen. Es wird die Aufgabe ausgeführt, deren Gewichtungswert am größten ist. Der Gewichtungswert einer anderen Aufgabe wird um das eigene Gewicht aufaddiert. Der Gewichtungswert des Siegers wird auf sein Gewicht zurückgesetzt.

Um zu überprüfen, ob dieser Scheduler die Vorgaben einhält, wurden mehrere Simulationen durchgeführt. Hierfür wurde der Worst-Case in Form von kontinuierlichen Kollisionen untersucht. Es wurde für verschiedene Anzahlen von Kollisionen gezählt, wie oft eine Aufgabe den höchsten Gewichtungswert hat (Sieger ist). Die Daten der Aufgaben wurden dann auf 100 % normiert und mit den Soll-Werten verglichen. Um die verschiedenen Kollisionsanzahlen zu vergleichen, wurde der maximale Abstand zwischen den Soll- und Ist-Werten ermittelt.
So haben beispielweise die drei Aufgaben mit den Gewichten 40, 20 und 10 folgende Soll-Werte:

$$g_i = \text{Gewicht der i-ten Aufgabe} \quad , \quad \frac{g_i}{\sum_{i=1}^{3} g_i} = s_i = \text{Soll-Wert der i-ten Aufgabe}$$

$$s_1 \approx 57\% \quad , \quad s_2 \approx 28\% \quad , \quad s_3 \approx 17\%$$

Tabelle 3.1 und die Abbildungen 3.11 und 3.12 zeigen, dass die maximale Abweichung zwischen dem Soll- und Ist-Werten mit steigender Anzahl von Kollisionen immer kleiner wird und gegen 0,0 läuft. Dies deutet auf einen sehr gut geeigneten Scheduler Algorithmus.

Wenn man aber die Gewichte auf 70, 25 und 5 ändert, läuft die maximale Abweichung

Tabelle 3.1.: Datenwerte der ersten Simulation des Aging Schedulers.

Anzahl der Kollisionen	Max. Abweichung des Soll- und Ist-Wertes
10	0,0428571428571429
20	0,0357142857142857
30	0,0285714285714286
40	0,0107142857142857
50	0,00857142857142856
75	0,00571428571428567
80	0,00535714285714284
100	0,00857142857142856
200	0,00357143
400	0,00107142857142861
600	0,000714285714285723
1000	0,000714285714285723
2500	0,0001714285714286
5000	0,0001714285714286
10000	0,00007142857142861 11

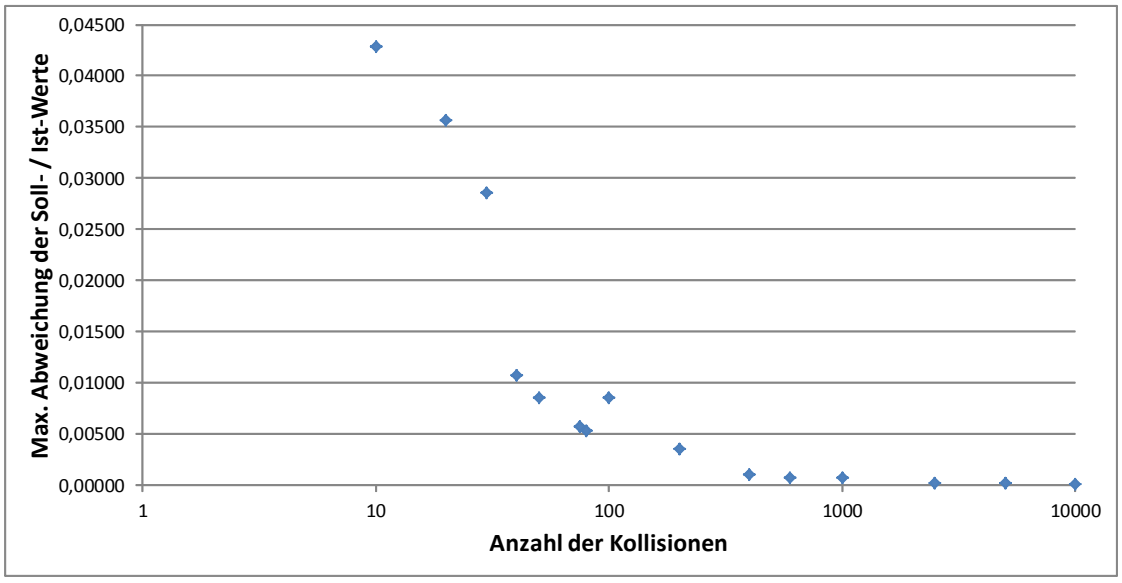

Abbildung 3.11.: Datenwerte der ersten Simulation des Aging Schedulers.

leider nicht mehr gegen 0,0 , sondern gegen 7,5 (siehe Tabelle 3.2 und Abbildung 3.13). Der Grund hierfür liegt in der Tatsache, dass der Gewichtungswert einer Sieger-Aufgabe nach dem Vergleich auf sein Gewicht zurückgesetzt wird. Der Algorithmus berücksich-

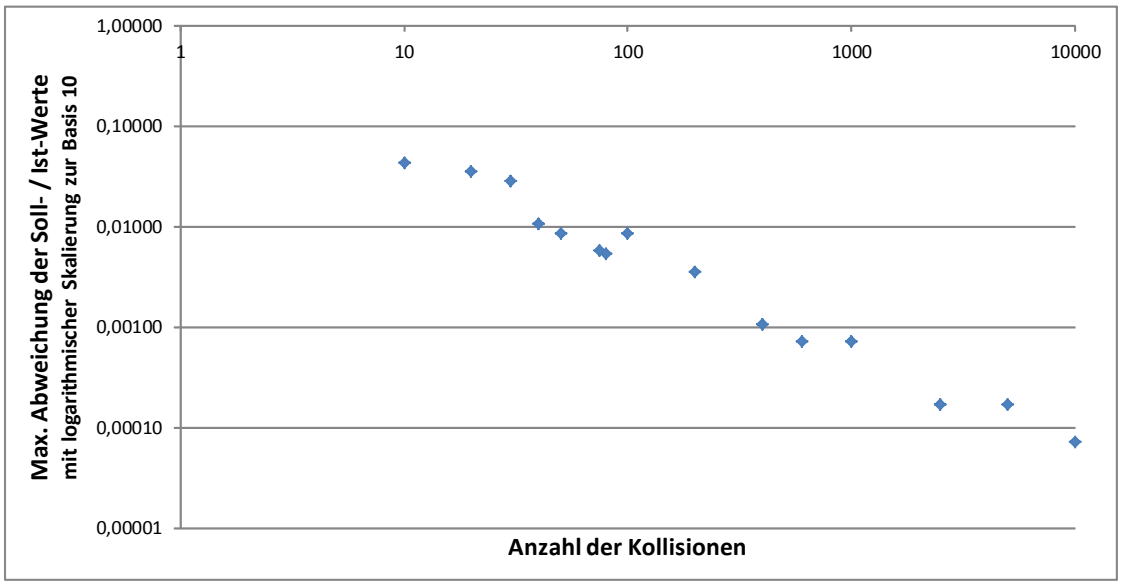

Abbildung 3.12.: Datenwerte der ersten Simulation des Aging Schedulers (log).

tigt nicht, ob ein Sieger wegen einem kleinen oder großen Abstand gewonnen hat. Je nach vorhandenem Gewichtswert kann eine Aufgabe immer mit einem großen Abstand gewinnen. Dieser große Abstand geht sofort verloren, da ihr Gewichtswert auf ihr Gewicht zurückgesetzt wird.

Auch wenn dieser Algorithmus nicht den genauen Soll-Wert ergibt, so liefert er mindestens Ergebnisse in der richtigen Größenordnung. Wenn der Gewichtungswert immer verdoppelt wird, liefert dieser Aging Scheduler exakte Ergebnisse. Da die simulierten Ist-Werte gute bis sehr gute Ergebnisse zeigen, wurde der Aging Scheduler als Alternative zum Round-Robin-Algorithmus im MIS implementiert.

3.5.4. Invertierter Aging Scheduler

Um unabhängig von den Gewichten genaue Ist-Werte zu erhalten, wurde der Aging Scheduler im Rahmen dieser Bachelorarbeit weiterentwickelt. S sollen die maximalen Abweichungen zwischen Soll- und Ist-Werten minimieren werden. Die Aussage, dass dieser verbesserte Scheduler genaue Ist-Werte liefert, wurde durch Simulationen mit zwei, drei und fünf AT Befehlen untermauert. Jede Simulation hatte andere Gewichte (z.B.:

Tabelle 3.2.: Datenwerte der zweiten Simulation des Aging Schedulers.

Anzahl der Kollisionen	max. Abweichung des Soll- und Ist-Wertes
10	0,05
20	0,05
30	0,05
40	0,05
50	0,06
75	0,06
80	0,0625
100	0,07
200	0,07
400	0,0725
600	0,0733333333333333
1000	0,074
2500	0,0748
5000	0,0748
10000	0,0749

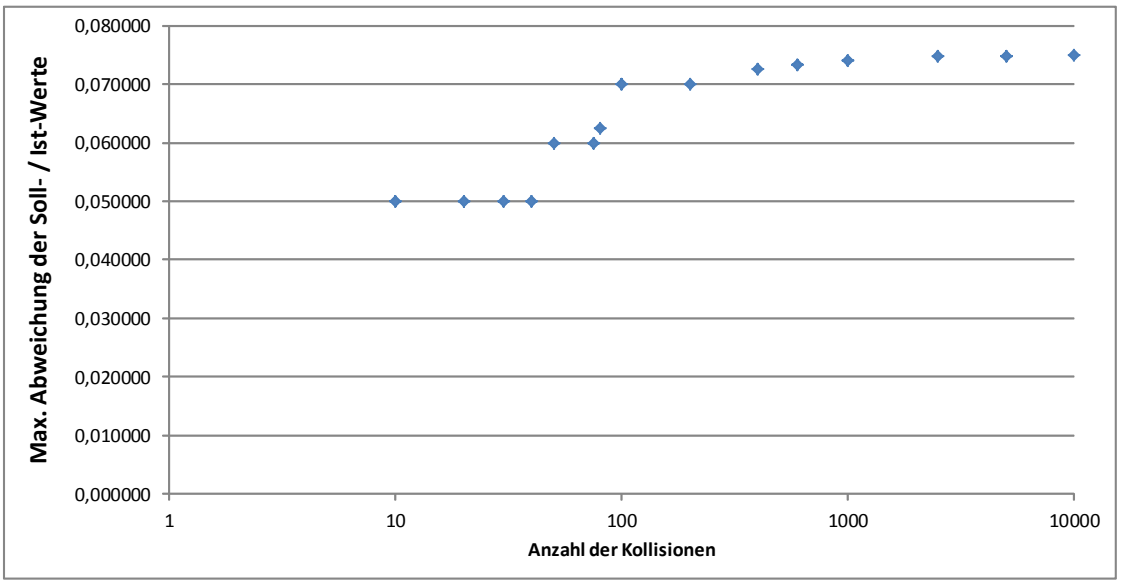

Abbildung 3.13.: Datenwerte der zweiten Simulation des Aging Schedulers.

Primzahlen) für die periodischen Aufgaben, über 65.000 Kollisionen und eine maximale Abweichung zwischen Soll- und Ist-Werten von weniger als 0,0001. Wenn die Anzahl der Kollisionen ein Vielfaches aller Gewichte ist, so beträgt die maximale Abweichung zwischen Soll- und Ist-Werten immer genau 0,0.
Die genauen Ergebnisse begründen sich dadurch, dass dieser Scheduler, im Vergleich zum vorherigem Aging Scheduler, den Gewichtungswert des Siegers nicht zurücksetzt und damit gegebenenfalls einen großen Abstand zu den anderen Gewichtungswerten zunichtemacht.

3.5.4.1. Funktionsweise des invertierten Aging Schedulers

Zuerst müssen die Gewichte der periodischen Aufgaben invertiert werden:

$$g_i = \text{Gewicht von der i-ten Aufgabe}$$

$$ig_i = 1/g_i = \text{invertiertes Gewicht von der i-ten Aufgabe}$$

Die drei Gewichte 70 , 25 und 5 ergeben folgende invertierte Gewichte:

$$ig_1 = 1/70 \approx 0,0142 \quad , \quad ig_1 = 1/25 = 0,04 \quad , \quad ig_1 = 1/5 = 0,2$$

Bei einer Kollision wird die Aufgabe mit dem größten Gewichtungswert ausgeführt. Von dem Gewichtswert des Siegers wird das eigene invertierte Gewicht subtrahiert. Zu den Gewichtungswerten der anderen Aufgaben wird das invertierte Gewicht des Siegers addiert. Bei diesem Algorithmus steigen die Gewichtungswerte immer weiter an. Wie im Kapitel 4.4 gezeigt wird, lässt sich dies aber sehr einfach und schnell korrigieren.

3.6. Konfigurationsdatei

Beim Entwerfen und Programmieren des MIS wurde besonders viel Wert auf einen modularen Aufbau gelegt, um die Erweiterung des MIS um noch nicht bekannte Funktionen jederzeit zu ermöglichen. Um Implementierungsarbeiten zu minimieren, wurde der MIS um die Funktionalität einer Konfigurationsdatei erweitert.
Der MIS entnimmt beim Starten aus der Konfigurationsdatei den TCP Listener Port für den Server, das Standard Prioritäts-Berechnungsverfahren, den Schedulertyp und alle Informationen zu Modems, AT Befehlen und Datenwerten. Der Anwender kann somit ohne Änderungen des Quellcodes den MIS an zukünftige beziehungsweise ihm noch nicht bekannte Modems (z.B. LTE) anpassen.

Die Bedeutung und genaue Funktionsweise aller 27 Konfigurationsbefehle sind in Kapitel 4.6 im Detail erklärt. Die Konfigurationsdatei soll dem Anwender ermöglichen, Standardwerte zu setzen und den MIS um weitere Modems und AT Befehle zu erweitern.

Kapitel 4.

Implementierung

In Kapitel 3 wurden grundlegende Techniken und die allgemeine Funktionsweise des MIS vorgestellt und erklärt. Dieses Kapitel wird einen tieferen Einblick in die relevanten Aspekte der Implementierung geben. Sodann wird im Unterkapitel 4.7 ein MIS Client näher vorgestellt, der im Rahmen dieser Bachelorarbeit entwickelt wurde.

Bei der Implementierung wurde ein hoher Wert auf einen geringen Verbrauch sowie eine geringe Belastung von Arbeitsspeicher und Prozessor gelegt. Deshalb ist es möglich, dass der MIS über viele Stunden von mehreren Clients mit unterschiedlicher Verbindungsdauer und sehr vielen abwechselnden, einmaligen und periodischen Anfragen belastet wird und dennoch konstant lediglich ca. 0,25 Megabyte Arbeitsspeicher verbraucht.

4.1. Programmiersprache und Programmierkonventionen

Wie im vorherigen Kapitel bereits erwähnt, haben wir uns aus Performancegründen für eine Implementierung in C++ entschieden. Um eine größtmögliche Kompatibilität zu gewährleisten, wird der MIS in C++ nach der Version *ISO/IEC 14882:2003* [cpp03] (*C++03*) ohne Zusatz-Bibliotheken wie *Boost* [boo05] oder *tr1* [Aus05] implementiert.

Obwohl der C++ Standard *ISO/IEC 14882:2011* [cpp11], besser bekannt unter den Bezeichnungen *C++0x* und *C++11*, neuer und benutzerfreundlicher ist, haben wir uns gegen seine Nutzung entschieden, da er nur in sehr wenigen Systemen und Compilern vollständig in einer Release Version implementiert ist.
C++ ist eine objektorientierte Programmiersprache. Der allgemeinbevorzugte Programmierstil sieht eine Kapselung von sensiblen Daten und Funktionen vor. Dennoch haben wir im kompletten MIS auf Getter- und Setter-Methoden verzichtet, da dies die Zugriffszeit verlängert und die Performance verschlechtert. Das Deklarieren von lokalen Variablen belastet den Prozessor zum Teil enorm. Deshalb haben wir auf eine häufige und in Schleifen durchgeführte Deklaration von lokalen Variablen möglichst verzichtet. Die Vererbungsmöglichkeit von Klassen wurde gezielt nicht in Anspruch genommen. Diese Entscheidungen wurden zugunsten von Leistungssteigerungen des MIS getroffen. Bei der Anwendung von Getter- und Setter-Methoden und vieler Variablendeklarationen sind die Performanceverluste im Gegensatz zu der Klassenvererbung sofort ersichtlich. Ein Zugriff auf Daten einer vererbten Klasse verursacht stets zur Laufzeit eine teure Typumwandlung, beziehungsweise eine Typüberprüfung. Dies ist im Quellcode nicht sichtbar, da dieser notwendige Mechanismus vom Compiler eingebaut wird.
Es werden nur triviale Datentypen als Kopie zwischen Objekten und Funktionen übergeben. Auf alle anderen Datenstrukturen und Objekte wird primär mit Zeigern zugegriffen, um den Arbeitsspeicherverbrauch und die Prozessorbelastung (große Datenblöcke kopieren) zu minimieren.
Kritische Programmabschnitte werden wegen dem Einsatz von Multi-Threading mit Mutexen und Conditionsvariablen geschützt.

4.2. Thread-Implementierung

Wie im Kapitel 4.1 beschrieben, haben wir uns für *C++03* ohne Zusatz-Bibliotheken entschieden. Jedoch wird in dieser C++ Version noch kein benutzerfreundliches Multi-Threading für Klassen und Objekte angeboten. Wir haben deshalb das objektorientierte Multi-Threading mit einem gängigen Workaround implementiert. Die Objekte der Klassentypen NetServerCl, NetClientCl und SerialPortCl haben einen eigenen Thread, der jeweils mit einem Posix Thread (*Pthread*) umgesetzt wird. Für jede dieser Klassen wur-

den drei Methoden geschrieben, die einen Thread starten. Die drei Methoden werden im Folgenden anhand des Server Threads erklärt.

startServerThread
Diese Methode wird von außen aufgerufen. Mit *pthread_create(&(ServerThreadPo->thread), NULL, &threadStart, this)* wird in ihr ein neuer Pthread erstellt. Seine Identifikationsnummer wird in die Adresse *ServerThreadPo->thread* gespeichert und mit der Einstiegsmethode threadStart gestartet.
Der zweite Parameter erwartet eine Adresse auf ein *Union* vom Typ *pthread_att_t*. Diese Union gibt die Eigenschaften des Threads an. Beim Server Thread wird als zweiter Parameter *NULL* angegeben, damit der Thread mit Standardeigenschaften erstellt wird. Dies ist wichtig, damit der Haupt-Thread, der das MIS Objekt erstellt und startet, auf den Server Thread mit *pthread_join* warten kann. Damit wird verhindert, dass er nicht zum Ende der main-Methode läuft und anschließend das ganze Programm mit allen Threads beendet. Beim Client und SerialPort Thread wird als zweiter Parameter die Adresse einer Union übergeben. Diese Union wird vorab mit *pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED)* angepasst. Die Folge ist, dass die Client und SerialPort Threads nach Beendigung ihrer Arbeit ihre Ressourcen sofort vollständig freigeben und nicht auf den Haupt-Thread mit *pthread_join* warten.
Mit dem vierten Parameter können Parameter für die Einstiegsmethode übergeben werden. Es wird immer ein Pointer auf das neu erstellte Objekt, dessen Thread gestartet wird, übergeben.

threadStart
Das eigentliche Ziel ist, dass der neue Thread seine Arbeit in einer Methode der neu erstellten Klasseninstanz beginnt. Jedoch verlangt *pthread_create* einen statischen Einstiegspunkt. Die Methode *threadStart* wird als Hilfsmittel eines statischen Einstiegspunktes benutzt und ruft dann mithilfe des Pointers, der auf die neu erstellte Klasseninstanz zeigt und als Parameter übergeben wurde, die eigentliche objektbezogene Methode auf (*threadRun*).

threadRun
In dieser Methode fängt der Thread mit seiner klassentypspezifischen Arbeit an. In allen drei Variationen dieser Methode (Server, Client und SerialPort) befindet sind eine

Schleife für die primäre Arbeit. Der Server Thread empfängt Client-Verbindungen, der Client Thread empfängt Nachrichtendatenpakete und der SerialPort Thread bearbeitet periodische oder vom Client Thread überreichte Aufgaben.

4.3. Netzwerkprotokoll

Die Zielvorgabe des Netzwerkprotokolls besteht in einer schnellen Analyse einer Netzwerknachricht, einem logischen Aufbau und einer leichten Erweiterungsmöglichkeit. Beim entworfenen Netzwerkprotokoll basiert der Header einer Nachricht auf einer Baum-Struktur und kann aktuell 52 verschiedene Netzwerknachrichtentypen identifizieren. Die Zuordnung der Nachrichtentypen in Gruppen und Untergruppen können den Abbildungen 3.2 und 4.1 entnommen werden. Die Tabellen 4.1, A.1, A.2, A.3, A.4 und A.5 erklären den Aufbau und die Bedeutung von jedem Nachrichtentyp.

Welche Informationen die Tabellen beinhalten und wie sie zu lesen sind, soll an dieser Stelle anhand von Tabelle 4.1 exemplarisch genauer erläutert werden. Als weitere Verständnishilfe zum Aufbau des Headers werden zusammengehörige Begriffe und Werte in dieser Tabelle mit der gleichen Farbe gekennzeichnet. Der grundlegende Aufbau ist in allen Tabellen identisch und kann auf die anderen fünf Tabellen übertragen werden. Die Kenntnis der Übertragungsrichtung der Nachricht ist erforderlich, um den Nachrichtentyp eindeutig zu bestimmen, da am Header nicht zu erkennen ist, ob eine Nachricht zum MIS oder zum Client gesendet wird.

Die Nachrichten der Gruppe *System* sind in der Tabelle 4.1 eingeordnet und haben im ersten Byte des Headers den Hexadezimalwert *0x00*. Für die Bezeichnung *System* und den Wert des ersten Bytes wurde in der Tabelle die gemeinsame Farbe Rot gewählt. Die Gruppe *System* hat vier Untergruppen, die man am zweiten Byte des Headers ablesen kann. Die Bezeichnung jeder Untergruppe und des dazugehörigen Bytes haben dieselbe Farbe. Zum Beispiel ist die Bezeichnung der Untergruppe *Get Client List* in der Farbe Grün. Die dazugehörigen Bytes im Header sind deshalb ebenfalls grün markiert. Jede Untergruppe hat eine Anfrage, die der Client an den MIS sendet. In Abhängigkeit vom Typ der Anfrage kann der MIS mit verschiedenen Nachrichten antworten. Wenn der

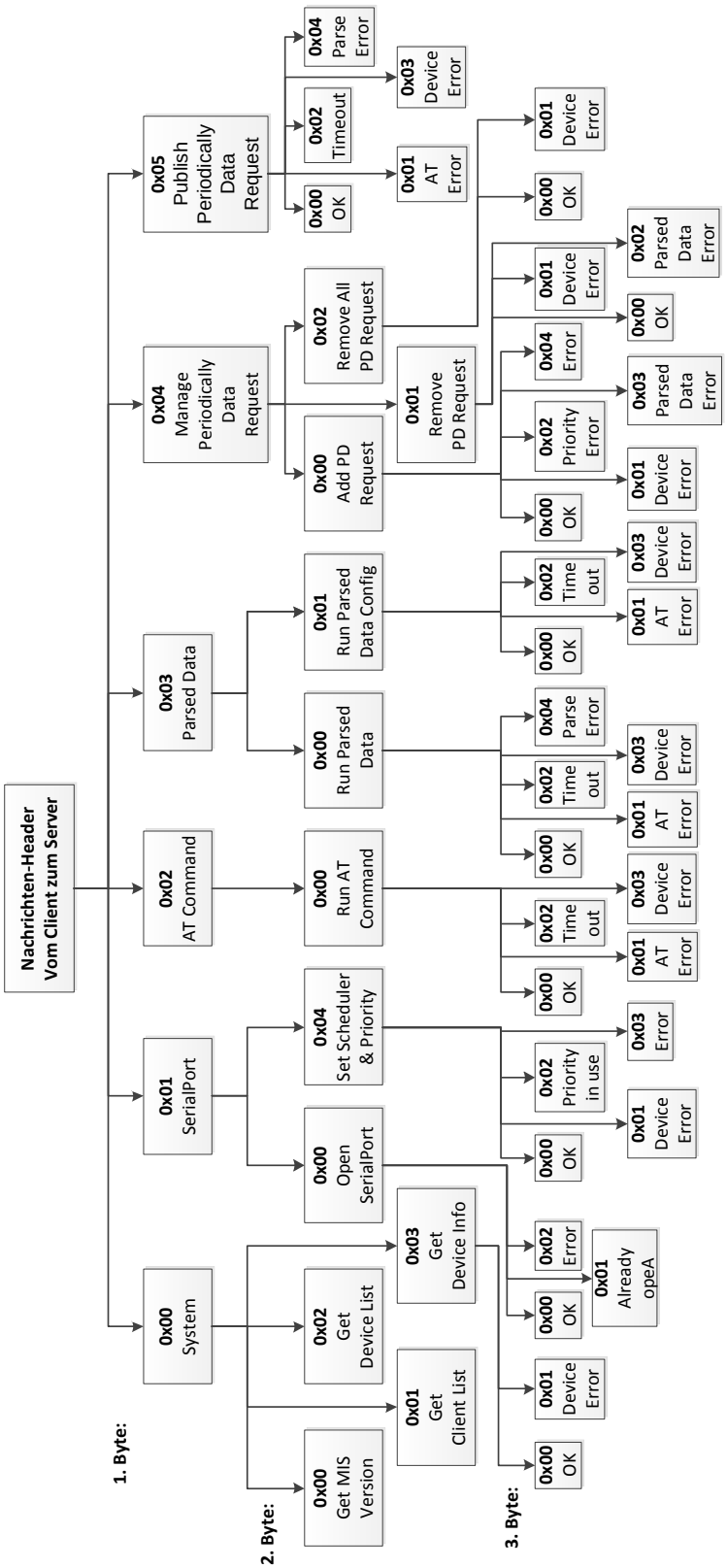

Abbildung 4.1.: Baum-Struktur des Netzwerkprotokolls (MIS an Client).

Tabelle 4.1.: Aufbau der Netzwerknachrichten (System).

System

Get MIS Version

Anfrage	0x00	0x00		
OK	0x00	0x00	l(a)	a

Get Client List

Anfrage	0x00	0x01				
OK	0x00	0x01	b	l(c)	c	...

Get Device List

Anfrage	0x00	0x02				
OK	0x00	0x02	d	l(e)	e	...

Get Device Info

Anfrage	0x00	0x03	f											
OK	0x00	0x03	0x00	f	g	h	l(i)	i	...	j	m	l(n)	n	...
Device Error	0x00	0x03	0x01	f										

Variable = Beschreibung [Größe in Byte]
l(x) = Länge von Zeichenkette x [1 Byte]
a = MIS Versioninfo [l(a) Byte]
b = Anzahl Clients [1 Byte]
c = Client Info [l(c) Byte]
d = Anzahl Devices (SerielPorts/Modem) [1 Byte]
e = Deviceinfo [l(e) Byte]
f = Device Index [1 Byte]
g = Anzahl Datenwerte [2 Byte]
h = Datenwerttyp [1 Byte]
i = Watenwertname [l(i) Byte]
j = Anzahl Datenwert-Konfigurationsbefehle [2 Byte]
m = Datenwerttyp [1 Byte]
n = Konfigurationsbefehl-Name [l(n) Byte]

MIS mit unterschiedlichen Nachrichten antworten kann, so ist der Header der Antworten um ein Byte länger, als der Header der Anfrage. Die verschiedenen Antworttypen auf eine Anfrage werden am neuen Byte erkannt. Die Bezeichnung des Antworttyps und

des passenden Wertes im Header haben dieselbe Farbe. Da die ersten drei Anfragen der *System-Tabelle* nur eine Antwort ermöglichen, ist der Header dieser Anfragen und Antworten gleich groß.

Falls der MIS die Anfrage erfolgreich verarbeiten kann, so wird eine Antwort mit *OK* deklariert. Wenn es zu einem Fehler kommt, wird die Fehlerart dem Client durch den Antworttyp mitgeteilt.

Bevor eine Zeichenkette versendet wird, werden zuerst ihre Länge und dann die Zeichen selbst gesendet. Somit wird der Empfänger im Vorfeld über die Länge einer Zeichenkette informiert und muss nicht die Zeichenkette nach einem Terminalsymbol untersuchen. Die Datenwerttypen *Float* und *Integer* sind vier Byte groß und werden in der *Little-Ending* Kodierung gesendet.

Die drei Punkte (...) in einer Tabellenzelle deuten darauf hin, dass die Angaben zuvor (z.B. *Device Info* in der Antwort auf *Get Device List*) sich mehrfach wiederholen können. Wie oft sie sich wiederholen, wird stets vorab angegeben (in diesem Beispiel im dritten Byte = d).

4.4. Priorität und Scheduler im SerialPort Thread

Im Kapitel 3.4.2 und 3.5 wurden die fünf Prioritäts-Berechnungsverfahren und drei Scheduler vorgestellt und verglichen. Alle Komponenten wurden im SerialPort Thread implementiert. Wie der Client und Server Thread, so arbeitet auch der SerialPort Thread seine Arbeit in einer großen Schleife ab. Abbildung 4.2 beschreibt die Funktionsweise des SerialPort Threads. Ein SerialPort Objekt hat eine *First In – First Out* Aufgabenschlage. In dieser Warteschlage kann der Client Thread Aufgaben hinzufügen. Eine Aufgabe hat einen Typ, der angibt, was der SerialPort Thread machen soll. Des Weiteren befinden sich in dieser Aufgabe alle nötigen Informationen. Es gibt fünf externe Aufgabentypen, die der Client Thread aus einer Netzwerknachricht erstellt und an den SerialPort weiterreicht. Falls es keine externen Aufgaben gibt und die Warteschlage leer ist, wird überprüft, ob eine periodische Aufgabe existiert, die aktuell oder zukünftig bearbeitet werden

muss. Dies wird mit dem sechsten Typ (*TaskDataType_PeriodicallyTask*) gekennzeichnet.

Es gibt folgende Aufgabentypen für den SerialPort Thread:

- **TaskDataType_ATCommand:** Ausführung einer gelieferten AT Anfrage (Char-Array) und Zurücksenden der unveränderten AT Antwort (Char-Array) an den Client.
- **TaskDataType_ATDeviceValue:** Abfrage eines bestimmten Datenwertes und Senden an den Client.
- **TaskDataType_ATDeviceConfig:** Ausführung eines bestimmten AT Befehls und Senden der unveränderten AT Antwort (Char-Array) an den Client
- **TaskDataType_addPeriodicallyDataRequest:** Einreichen einer neuen periodischen Anfrage oder Änderung der Prioritätsklasse oder des Intervalls einer Anfrage (*Subscribe*).
- **TaskDataType_removePeriodicallyDataRequest:** Entfernen einer oder aller periodischen Anfragen.
- **TaskDataType_PeriodicallyTask:** Bearbeiten einer periodischen Aufgabe (*Publish*).

An dieser Stelle werden die Arbeitsschritte des SerialPort Threads erläutert.
Die Arbeitsschritte sind in Abbildung 4.2 dargestellt. Die vier großen Rauten bilden Entscheidungsinstanzen ab, die den aktuellen Zustand überprüfen und die anstehenden Arbeitsschritte bestimmen. Die Rechtecke konkretisieren die zu leistenden Arbeitsschritte. Ein SerialPort hat einen Status, der nach einem erfolgreichen Erstellen des SerialPort Objektes und dem Starten des dazugehörigen Threads auf *SerialPortStatus_OK* gestellt wird. Wenn es zu einem kritischen Fehler kommt (z.B. Schreib-/Lesefehler des Devicefiles), ändert sich dieser Status und die Schleife wird verlassen. Im Flussdiagramm ist dies als kleine Raute dargestellt. Aus Abbildung 4.2 wird ersichtlich, dass externe Aufgaben immer bevorzugt werden. Dies könnte für einen *Denial of Service* Angriff ausgenutzt

werden. Ein Client müsste hierfür ununterbrochen Nachrichten senden, die eine externe Aufgabe verursachen (z.B. einmaliges Abfragen eines Datenwertes). Dies halten wir aber bei den aktuell geplanten Einsatzgebieten für unwahrscheinlich.

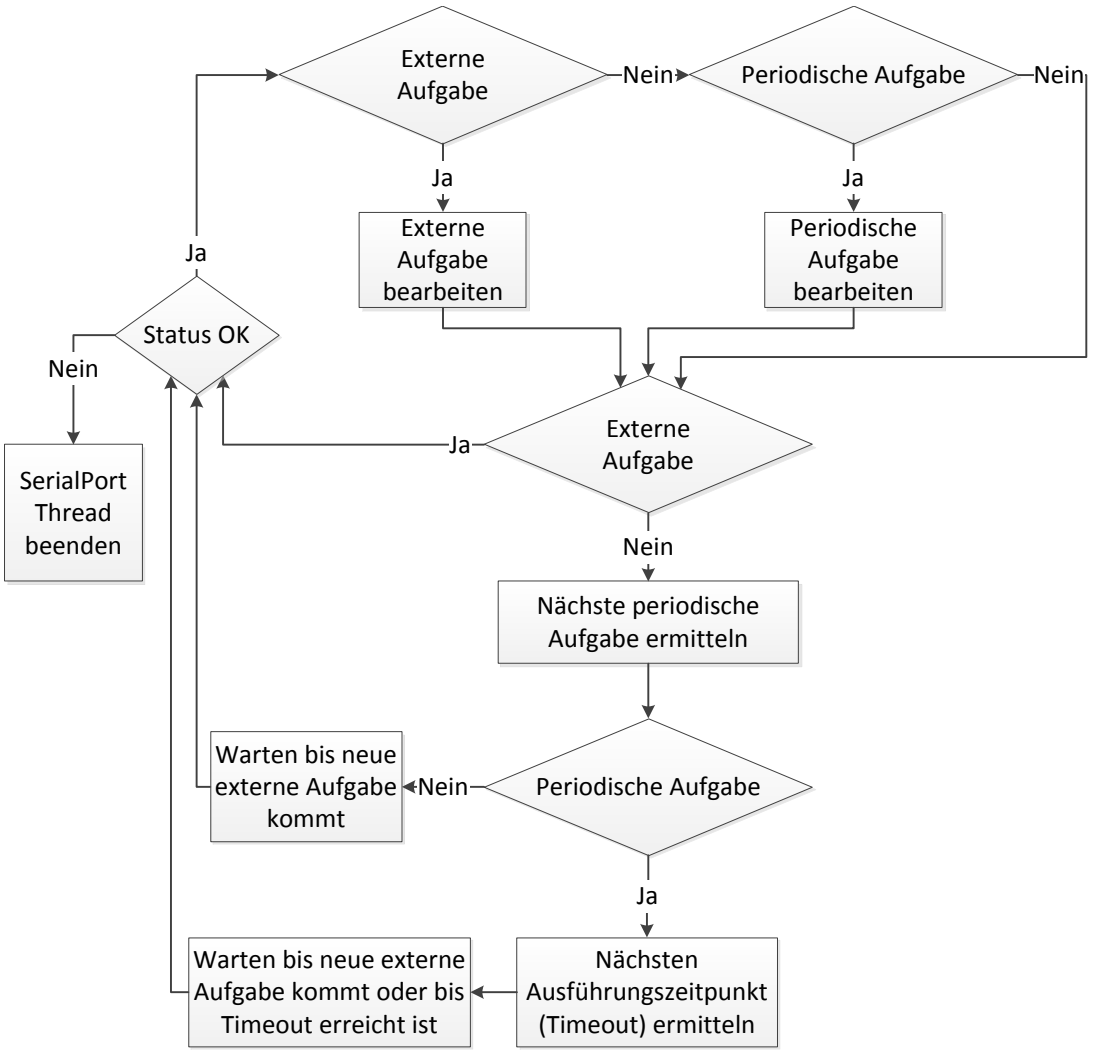

Abbildung 4.2.: Flussdiagramm zur Funktionsweise eines Modem Threads.

Wenn eine externe Aufgabe eine periodische Anfrage hinzufügt, entfernt oder die Prioritätsklasse ändert, wird die rekursive Tiefensuche nach der minimalen Menge der periodischen Aufgaben und der Berechnung ihrer Gewichte nach einem der fünf Prioritäts-Berechnungsverfahren immer vom SerialPort Thread durchgeführt.

Der eingestellte Scheduler Algorithmus wird stets bei der Ermittlung der nächsten periodischen Aufgabe benutzt. Er ermittelt die nächste periodische Aufgabe und ihre Arbeitszeit, die in der Konfigurationsdatei eingegeben wird (siehe Kapitel 4.6). Um alle Aufgaben zu erhalten, die in einer Kollision betroffen sind, werden anschließend alle Aufgaben gesucht, die während der Arbeitszeit gestartet werden müssten. Dies ist möglich, da die Ausführungszeiten der periodischen Aufgaben und ihre Arbeitsdauer bekannt sind. Anschließend wird der Gewichtungswert aller periodischen Aufgaben verglichen, um den Sieger zu finden. Dieser wird zwischengespeichert und anschließend werden alle Gewichtungswerte je nach Schedulertyp erhöht oder verringert.

Beim Schedulertyp *invertierter Aging Scheduler* werden die Gewichtungswerte immer größer, weil sie öfter erhöht wie verringert werden (vergleiche Kapitel 3.5.4). Einem Überlauf der Gewichtungswert-Variable wird mit einem ressourcenschonenden Variablenvergleich und mit sehr seltenen und ebenfalls ressourcenschonenden Subtraktionen vorgebeugt: Wenn der Gewichtungswert des Siegers größer als 1.000.000 ist, dann werden alle Gewichtungswerte um 1.000.000 verkleinert.

4.5. Vermeidung eines Speicherlecks durch Aufräumarbeiten

Im Kapitel 3.3.2 und Abbildung 3.3 wird die allgemeine Arbeitsweise des Client Threads - Empfangen, Verarbeiten oder Weiterreichen von Nachrichten - beschrieben. Die Baum-Struktur des Headers einer Netzwerknachricht erlaubt eine effiziente Analyse des Headers mit mehreren *switch-case-Anweisungen*, um den Nachrichtentyp zu ermitteln. Anschließend können die Nutzdaten geladen werden. Aus diesen Daten ergibt sich eine Aufgabe, die direkt vom Client Thread bearbeitet wird oder die er an den SerialPort Thread weiterreicht. Wenn die Verbindung zu einem Client beendet wird, springt sein Thread aus seiner primären Arbeitsschleife und gibt seine eigenen Thread Ressourcen frei, um Speicherlecks (*memory leaks*) zu verhindern. Als letzte Aufgabe, vor dem Beenden, soll ein Client Thread prüfen, ob es noch verbundene Clients gibt. Wenn dies nicht der Fall ist, gibt er ihren Speicher frei und beendet alle SerialPort Threads, indem er den

Status des SerialPorts auf *ClientStatus_Closed* setzt und sie aufweckt, da sie auf eine externe Aufgabe warten könnten (vergleiche Abbildung 4.2). Anschließend schließt er das *Devicefile* der seriellen Schnittstelle und gibt auch diesen Speicher frei. Ungenutzte SerialPort Objekte werden nicht sofort geschlossen und freigegen, weil ihre Existenz einem Client bekannt sein kann (Netzwerknachricht: Get Device List) und er ihre Dienste verlangen könnte.
Ein SerialPort Thread überprüft beim Senden der Daten an die Clients ebenfalls ihren Status und reagiert mit Aufräumarbeiten, um gegebenenfalls Speicher freizugeben. Wenn eine Verbindung zu einem Client getrennt wurde, bevor er seine periodischen Anfragen entfernt hat, werden diese ermittelt, ihre Ressourcen freigegeben und neue, aufgrund weniger periodischer Anfragen, effizientere periodische Aufgaben erstellt.

4.6. Konfigurationsdatei

Die Konfigurationsdatei ist eine ASCII UTF-8 Textdatei, die zeilenweise vom MIS von oben nach unten gelesen wird. Der Speicherort der Datei wird dem MIS beim Start mit den Parametern *-c Dateispeicherort* mitgeteilt. Damit die Verwaltung der Konfigurationen von mehreren Modems leichter und praktischer ist, liest der MIS nicht nur die angegebene Datei ein, sondern auch alle Dateien, die sich im gleichen Ordner befinden. Der Anwender kann die Konfigurationen eines jeden Modems somit übersichtlich in einer eigenen Datei pflegen. Der MIS erkennt eine Konfigurationsdatei an den ersten acht Bytes:

0x6D, 0x69, 0x73, 0x63, 0x6F, 0x6E, 0x66, 0x69, 0x67

Dies bedeutet nach ASCII Kodierung in UTF-8 *misconfig* und verhindert das Laden von falschen Dateien. Wenn es zu einem Syntaxfehler kommt und eine Zeile nicht geparst werden kann, wird die fehlerhafte Zeile mit einer Meldung ausgegeben und der MIS beendet.

Ein Konfigurationsbefehl (im Folgenden *Konfigbefehl* genannt) besteht aus einem vorgegebenen Attributnamen und dem dazugehörigen Wert. In einer Zeile darf maximal ein Konfigbefehl stehen. Alle Attributnamen und Werte werden nur mit einem Gleichheitszeichen getrennt. Beispielsweise hat der Konfigbefehl *at-c_worktime=20* den Attributnamen *at-c_worktime* und den Wert *20*. Sehr oft wird eine Zeichenkette als Wert

angegeben. Diese wird in Anführungszeichen eingebettet. Die ASCII-Zeichen *new line*, *carriage return* und *backslash* werden nach der C++ Kodierung mit \n, \r und \\ in die Zeichenkette eines Wertes eingegeben. Zeilen, die mit einem Rautezeichen (#) anfangen, können für Kommentare genutzt werden.

Alle 28 Konfigurationsbefehle sind in der Konfigurationsdatei, die während der Bachelorarbeit entstanden ist (*config.txt*), zu entnehmen. Diese Befehle werden mit Kommentaren in der Konfigurationsdatei erläutert oder in den Kapiteln 4.6.1, 4.6.2, 4.6.3 und 4.6.4 erklärt. Wenn der MIS ein *Devicefile* öffnet um ein Modem anzusprechen, wird zuerst überprüft, welche Modemkonfiguration die Passende ist. Dies wird mit den zwei Konfigurationsbefehlen *identity_at-command* und *identity_value* ermöglicht. Wenn ein Modem neu ist und noch keine eigene Konfiguration hat, so wird eine Standardkonfiguration verwendet (siehe beigelegte Konfigurationsdatei).

4.6.1. Hierarchische Anordnung der Konfigurationsbefehle

Alle Konfigbefehle, die ein Modem und seine AT Befehle sowie Datenwerte beschreiben, sind hierarchisch angeordnet. Dies soll an einem Beispiel erläutert werden.

Beispiel hierarchische Anordnung:
Ein Modem besitzt mehrere AT Befehle. Der AT Befehl ist dem Modem untergeordnet. Wenn ein neuer AT Befehl mit dem Konfigbefehl *at-command="AT+GMI"* definiert wird, dann wird er demjenigen Modem zugeordnet, das zuletzt in der Konfigurationsdatei mit dem Konfigbefehl *deviceconfigname="Eindeutige Modembezeichnung"* definiert wurde.

In Tabelle B.1 sind alle Abhängigkeiten aufgeführt. Durch diese hierarchische Anordnung entsteht eine Baum-Struktur. Denn ein Konfigbefehl kann mehreren Konfigbefehlen übergeordnet, aber auch gleichzeitig selbst einem anderen Konfigbefehl untergeordnet sein. Dem Anwender ist freigestellt die Konfigbefehle mit beliebig vielen Leerzeichen und Tabulatoreinrückungen anzufangen, um die hierarchische Anordnung im Text der Konfigurationsdatei visuell übersichtlich darzustellen.

4.6.2. Beispiel für Konfigurationsbefehle

Das folgende Beispiel erklärt wie ein Modem mit einem AT Befehl und einem Datenwert definiert wird.

Listing 4.1: Konfigurationsbefehle für Modem, AT Befehl und Datenwert.

```
1  deviceconfigname="Sierra Wireless, Incorporated | MC8790 | K2_0_7_35AP"
2
3      end_signal_ok="OK\n\n"
4      end_signal_error="ERROR\n\n"
5      end_signal_error="COMMAND NOT SUPPORT\n\n"
6
7      identity_at-command="AT+GMI"
8          identity_value="AT+GMI\n\n\nSierra Wireless, Incorporated\n\n\n\nOK\n\n"
9
10     identity_at-command="AT+GMM"
11         identity_value="AT+GMM\n\n\nMC8790\n\n\n\nOK\n\n"
12
13     identity_at-command="AT+GMR"
14         identity_value="AT+GMR\n\n\nK2_0_7_35AP C:/WS/FW/K2_0_7_35AP/MSM6290/SRC 2010/03/04 17:37\n\nOK\n\n"
15
16     at-command="AT+GMI"
17         at-c_worktime=20
18         atc_value_name="Manufacturer"
19             atc_v_type=string
20             atc_v_beforestring="AT+GMI\n\n\n"
21             atc_v_beforecount=0
22             atc_v_endstring="\n\n"
23             atc_v_endcount=0
```

In der ersten Zeile wird ein neues Modem definiert. Der Wert dieses Befehls ist frei wählbar und gibt die Beschreibung des Modems an. Die Beschreibung wird einem Client mitgeteilt, wenn ein Modem beziehungsweise SerialPort erfolgreich geöffnet wurde oder wenn die Netzwerkanfrage *Get Device List* genutzt wird.
Die nächsten drei Konfigbefehle definieren das Standardende einer erfolgreichen oder fehlerbehafteten AT Antwort. Diese Befehle sind multipel und können deshalb mehrfach eingegeben werden (siehe Zeile 4 und 5). Sie sind dem Befehl *deviceconfigname="..."* untergeordnet und deshalb definieren ihre Werte die Standard-*End-Signale* des Modems aus Zeile eins. Mit der Hilfe dieser End-Signale wird das Ende einer AT Anfrage verglichen und gefunden, um zu entscheiden, ob die AT Anfrage vom Modem erfolgreich oder fehlerhaft verarbeitet wurde.
Die sechs Konfigbefehle in den Zeilen 7 bis 14 sind drei *Konfigbefehl-Paare*. Ein Paar besteht aus den Befehlen *identity_at-command* und *identity_value*. Mit ihnen wird die Identität eines Modems geprüft. Hierfür wird der Wert des Konfigbefehls *identity_at-command* als AT Anfrage ans Modem gesendet. Anschließend wird die AT Antwort des

Modems mit dem Wert von *identity_value* verglichen. Wenn alle (in diesem Beispiel *drei*) vom Modem erhaltenen AT Antworten genau übereinstimmen, wird das Modem als passend zu dieser Konfiguration erkannt. Wird keine Konfiguration zu einem Modem gefunden, so wird die Standardkonfiguration angewendet. Sie wird mit *deviceconfigname="default"* eingeführt (siehe *config.txt*).
Der Konfigbefehl in Zeile 16 ist dem Modem (erste Zeile) zugeordnet und definiert einen neuen AT Befehl. Der Wert des Befehls ist die AT Anfrage, die an das Modem gesendet wird (*AT+GMI*). In Zeile 17 wird die Ausführungszeit des AT Befehls angegeben. Sie wird vom Scheduler für die *temporäre Kollisionserkennung* benutzt (vergleiche Kapitel 3.4.1). Der Konfigbefehl in Zeile 18 ist dem AT Befehl aus Zeile 16 zugeordnet und definiert einen neuen Datenwert, den der AT Befehl liefert. Falls ein Datenwert von mehreren AT Befehlen geliefert wird, so hat er bei seiner Definition immer denselben Namen, der im Wert angegeben wird (*Manufacturer*). Die nächsten fünf Zeilen sind dem Datenwert (Zeile 18) zugeordnet und beschreiben den Datenwerttyp und seine Position in der AT Antwort. Es gibt die fünf folgenden unterschiedlichen Typen um einen Datenwert festzulegen:

- **string:** Eine Zeichenkette (Char-Array)
- **float:** Eine Gleitkommazahl (z.B. 12.35)
- **defloat:** Eine Gleitkommazahl, bei der Nachkommastellen mit einem Komma gekennzeichnet werden (z.B. 12,35)
- **int:** Eine 32 Bit Integer (z.B. 12)
- **octalint32:** Eine 32 Bit Integer im Oktalsystem (z.B. 0o14 = 12 im Dezimalsystem)
- **hexint32:** Eine 32 Bit Integer im Hexadezimalsystem (z.B. 0xC = 12 im Dezimalsystem)

Die Befehle in Zeile 19 und 20 beschreiben die Zeichenkette, die direkt vor dem Datenwert in der AT Antwort steht. Die *count*-Angabe gibt den Index an, da die Zeichenkette mehrfach in der AT Antwort stehen kann.

Einfaches Beispiel: Es wird die Zeichenkette *Katze* in der AT Antwort *abc Hund abc Haus abc Katze abc Auto OK* gesucht. Mit den Befehlen *atc_v_beforestring=“abc “* und *atc_v_beforecount=2* kann der gesuchte Wert gefunden werden.
Datenwerte vom Type *string* benötigen zur Startposition auch eine Endposition. Diese Position wird ebenfalls mit einer Zeichenkette und einem Index angegeben. Die Suche nach der richtigen Endposition beginnt ab dem Datenwert. Im aufgeführten einfachen Beispiel wäre die Endposition das *0-te Leerzeichen* nach der Startposition:
atc_v_endstring=“ “ und *atc_v_endcount=0*

4.6.3. AT Konfigbefehl

Wie in Kapitel 3.1 erwähnt, gibt es auch AT Befehle, die vor ihrem ersten Aufruf durch einen anderen AT Befehl konfiguriert werden müssen (im Folgenden *AT Konfigbefehl* genannt). Ein AT Konfigbefehl wird z.B. mit *at-command-config=“AT+CQI=1“* definiert. Wenn ein AT Befehl einen AT Konfigbefehl benötigt, werden bei seiner Definition die AT Konfigbefehle, die eine bestimmte Eigenschaft aktivieren und deaktivieren, mit einem Konfigurationsbefehl angegeben (z.B. *at-c_config_once_begin=“Enable Network Registration“*). Um einen AT Konfigbefehl mehrfach benutzen zu können, wird er vor seiner ersten Benutzung definiert. Die Definition des AT Befehls verweist auf ihn mit seinem Namen. Der Definitionsaufbau des AT Konfigbefehls ist dem eines AT Befehls sehr ähnlich und wird in der beigelegten Konfigurationsdatei mit Kommentaren erklärt.

4.6.4. MultiDataSet

Ein *MultiDataSet* ist ein Datenwertsatz, der aus mehreren Datenwerten bestehen kann. Des Weiteren kann dieser Datenwertsatz auch mehrfach in einer AT Antwort vorhanden sein, so dass eine Liste von Datenwertsätzen gebildet wird. Ein MultiDataSet wird in der Konfigurationsdatei mit dem Befehl *atc_mdataset_name*, der sich auf der gleichen hierarchischen Ebene wie *atc_value_name* befindet, angegeben. Einem MultiDataSet sind die Datenwerte mit *atc_mds_value_name* untergeordnet. Die Definition dieser *Multi-Datenwerte* ist identisch mit der Defintion der normalen Datenwerte. Zur Unter-

scheidung ist der Attributname nur leicht angepasst (z.B. *atc_v_type* - *atc_mds_v_type*). Wegen der möglichen Auflistung des Datenwertsatzes entfällt der *beforecount*-Index.

Das folgende Listing 4.2 soll den Aufbau der Konfigbefehle eines MultiDataSets veranschaulichen. Für einen direkten Vergleich wird in diesem Beispiel auch ein normaler Datenwert (*Integer in Hexadezimalsystem*) definiert.

Listing 4.2: Konfigurationsbefehle für MultiDataSet.

```
1   at-command="AT+USET?0"
2       at-c_worktime=20
3       atc_value_name="WCDMA Active Set Count"
4           atc_v_type=hexint32
5           atc_v_beforestring="Count: "
6           atc_v_beforecount=0
7       atc_mdataset_name="WCDMA Active Set"
8           atc_mds_value_name="WCDMA Sync Neighbour Set PSC"
9               atc_mds_v_type=string
10              atc_mds_v_beforestring="PSC: "
11              atc_mds_v_endstring="\n\n"
12              atc_mds_v_endcount=0
13          atc_mds_value_name="WCDMA Sync Neighbour Set SSC"
14              atc_mds_v_type=string
15              atc_mds_v_beforestring="SSC: "
16              atc_mds_v_endstring="\n\n"
17              atc_mds_v_endcount=0
```

In den ersten beiden Zeilen wird ein AT Befehl definiert. Die nächste Zeile führt einen normalen Datenwert ein. Zeilen vier bis sechs beschreiben seine Position in der AT Antwort. In Zeile sieben wird ein MultiDataSet definiert. Die darauffolgenden fünf Zeilen definieren einen Datenwert des MultiDataSets. Seine Konfigbefehle sind identisch zu den Konfigbefehlen eines normalen Datenwertes aufgebaut. Seine Attributnamen haben zur Unterscheidung den Zusatz *_mds* in der Mitte. Einen *beforecount* Konfigbefehl gibt es nicht. Der MIS ermittelt mit dem ersten Datenwert des MultiDataSet die Vorkommenshäufigkeit des Datenwertsatzes in der AT Antwort.

4.7. MIS Client

Parallel zu dem MIS wurde ein MIS Client in Java entwickelt. Diese Anwendung ermöglicht es, alle Funktionen des MIS mit einer Benutzeroberfläche zu testen. Sie besteht primär aus zwei Java Klassen:

MIS_GUI: In dieser Datei befindet sich der Code für die Benutzeroberfläche mit den dazugehörigen Event-Methoden. Die Benutzeroberfläche wurde mithilfe von NetBeans erstellt.

MIS_Server: Die eigentliche Kommunikation zum in C++ programmierten MIS findet in dieser Java Klasse statt.

Die MIS_Server Klasse funktioniert eigenständig und kann problemlos in andere Java Programme integriert werden. Dies soll den Praxiseinsatz des MIS vereinfachen und beschleunigen.

Hierfür bietet ein MIS_Server Objekt öffentliche Methoden an, um bestimmte Netzwerknachrichten an den MIS zu senden. Jedem Nachrichtentyp ist eine Methode zugeordnet. Die Nutzdaten der Nachricht werden der Methode als Parameter übergeben.
Beispiel: Die Methode *public boolean runATDeviceValue(int DeviceIndex, int ValueIndex)* fordert einen Datenwert einmalig an. Wenn die Nachricht erfolgreich abgesendet wurde, liefert sie *true* zurück.

Der MIS_Server hat einen eigenen Thread, der die ankommenden Nachrichten empfängt. Er legt ihre Daten in dafür vorgesehene Objekte ab. Die Objekte werden mit einem *PropertyChangeEvent* an beliebig viele Interessenten veröffentlicht.

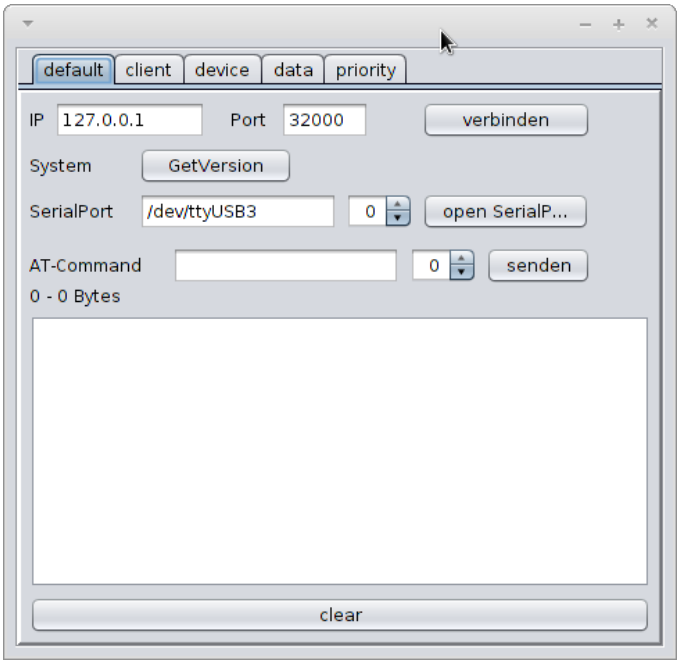

Abbildung 4.3.: Screenshot des MIS Client (default Einstellungen).

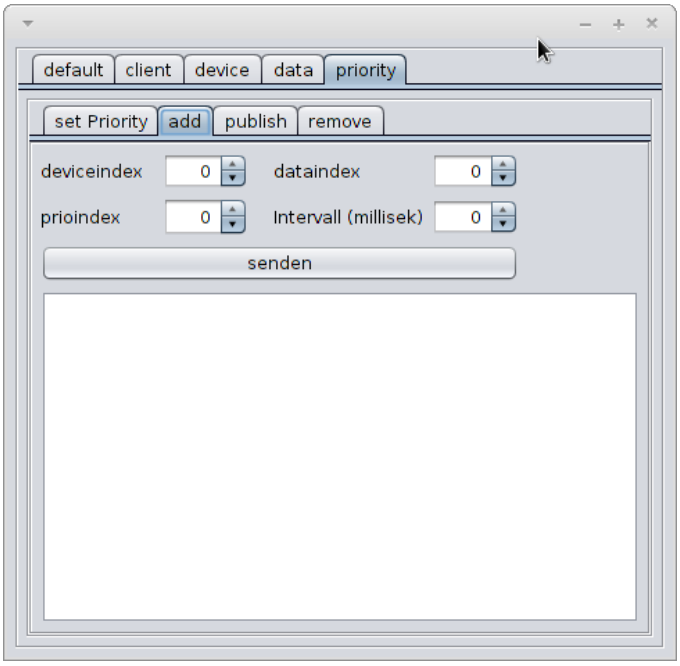

Abbildung 4.4.: Screenshot des MIS Client (Datenwert abonnieren).

Kapitel 5.

Evaluierung

5.1. Testumgebung und Ausgangssituation

Dieses Kapitel präsentiert die Ergebnisse der Testung und Evaluierung der Funktionsweise des MIS. Hierfür wurden 21 Messreihen durchgeführt. Als Basis für die Evaluierung wurde ein Notebook mit 8 GB Arbeitsspeicher und einem Intel i5-2540M Prozessor (Dual-Core mit Hyper-Threading) benutzt. Beim Prozessor wurden im Bios die Stromsparfunktionen deaktiviert, so dass er einen konstanten Takt von 2597,137 MHz hatte. Als Betriebssystem wurde ein LinuxMint 12 (lisa) in 64 Bit mit einem Linux Kernel der Version 3.0.0-12 generic und der Benutzeroberfläche GNOME 3.2.1 (Classic - No effects) verwendet. Der MIS wurd für 32 Bit kompiliert. Auf dem System liefen bis auf den Modem-Informations-Service und den MIS Client keine weiteren Anwendungen. Der Arbeitsspeicher war während aller Messungen mit circa 2,5 GB von 8 GB belegt. Es wurden die Zeitintervalle zwischen zwei beim Client empfangenen Datenwerten gemessen. Da die *PropertyChangeEvent* Funktion im Client ein zusätzliches Delay verursacht, wurde sie zum Weiterreichen von empfangenen Daten nicht genutzt. Stattdessen hat der zuständige Thread beim Client nach dem vollständigen Empfang eines Datenwertes einen Zeitstempel (in Millisekunden) abgelegt und sofort auf den nächsten Datenwert gewartet. Erst nach dem Empfang von 10.000, 100.000 beziehungsweise 1.000.000 Datenwerten wurden die Messdaten zur Analyse ausgegeben.

5.2. Evaluierung der Testumgebung

Vor der Durchführung der eigentlichen Messungen mit echten Datenwerten vom Modem, wurde die Exaktheit der Testumgebung geprüft. Hierfür wurde der MIS wie folgt minimal angepasst.
Die Quellcodepassagen, in denen eine AT Anfrage an das Modem gesendet und auf eine AT Antwort gewartet wird, wurden auskommentiert. Stattdessen wurde eine fest einprogrammierte AT Antwort genutzt, so dass bei Tests mit diesen Anpassungen kein Delay durch die Arbeitszeit des Modems entstand. Alle anderen im MIS stattfindenden Vorarbeiten (z.B. nächste periodische Aufgabe ermitteln) und Nacharbeiten (z.B. Parsen der AT Antwort) wurden unverändert durchgeführt.

Zuerst wurde eine Messung mit einem Intervall von 0 Millisekunden durchgeführt, so dass die maximale Geschwindigkeit der Testumgebung ermittelt wurde. Tabelle 5.1 zeigt, dass 99,5 % der empfangenen Datenwerte einen Abstand von unter einer Millisekunde hatten. Bei diesem Test wurde mit dem Linux-Befehl *top* eine Prozessorauslastung von 100 % beim MIS Client und eine Prozessorauslastung von 70 % beim MIS gemessen. Diese Werte zeigen, dass der MIS Client mit dem Empfangen und Vermessen der Datenwerte überlastet war.

Tabelle 5.1.: Performancetest mit Intervall von 0 ms.

Intervall in ms	Anzahl Datenswerte
0	995.231
1	4.769
> 1	0

Anschließend wurde der gleiche Test mit nur 100.000 empfangenen Datenwerten und einem vom Client geforderten Zeitintervall von 20 Millisekunden durchgeführt. Wie Tabelle 5.2 zeigt, sind die Daten zwar nicht mehr ganz exakt, aber dennoch gut. Denn ca. 95 % der gemessenen Intervalle entsprechen dem geforderten Wert. Bei dieser Messung hatten der MIS und MIS Client eine Prozessorauslastung von maximal 1 %.

Wir vermuten, dass die Gründe für dieses etwas schlechtere Ergebnis teilweise auf die ungenauen Messungen des Clients in Millisekunden zurückzuführen sind. Des Weiteren

Tabelle 5.2.: Performancetest mit Intervall von 20 ms.

Intervall in ms	Anzahl Datenswerte
< 20	0
20	94.897
21	5.103
> 21	0

legt sich der SerialPort Thread im MIS zwischen zwei Publish-Vorgängen *schlafen* und muss zum angegebenen Zeitpunkt vom Betriebssystem *aufgeweckt* werden. Das *Schlafenlegen* und *Aufwecken* verbrauchen Zeit und können sich folglich negativ auf die Messresultate auswirken.

Diese beiden Messungen zeigen, dass die Testumgebung und der MIS in der Lage sind, gute und erwartungstreue Ergebnisse für die Messeinheit in Millisekunden zu liefern.

5.3. Zuverlässigkeit und Arbeitszeit des Modems

Mit den nächsten Messreihen wurde die Zuverlässigkeit der Arbeitszeit (im Folgenden *WorkTime* genannt) des Modems getestet. Hierfür wurden die Änderungen am MIS, die im vorherigen Kapitel 5.2 beschrieben sind, rückgängig gemacht.

Um die Zuverlässigkeit des Modems zu testen, wurden sieben Datenwerte 10.000 Mal mit einem eingestellten Intervall von 20 ms beim Client empfangen. Alle sieben Datenwerte wurden einzeln und nacheinander gemessen. Die vom Client gemessenen Intervalle können in einer *Cumulative distribution function* (*CDF*) in Abbildung 5.1 abgelesen werden. Diese Abbildung zeigt, dass der MIS die Datenwerte primär in zwei unterschiedlichen Intervallen veröffentlicht. WEiterhin geht aus der Abbildung hervor, dass die Arbeitszeit des Modems für die Bearbeitung eines AT Befehls zwei Zeitspannen aufweist.

Wir haben eine Gleichverteilung der Intervalle und somit nur eine Steigung in den Grafen erwartet. Die zwei starken Steigungen der Grafen in Abbildung 5.1 zwischen den

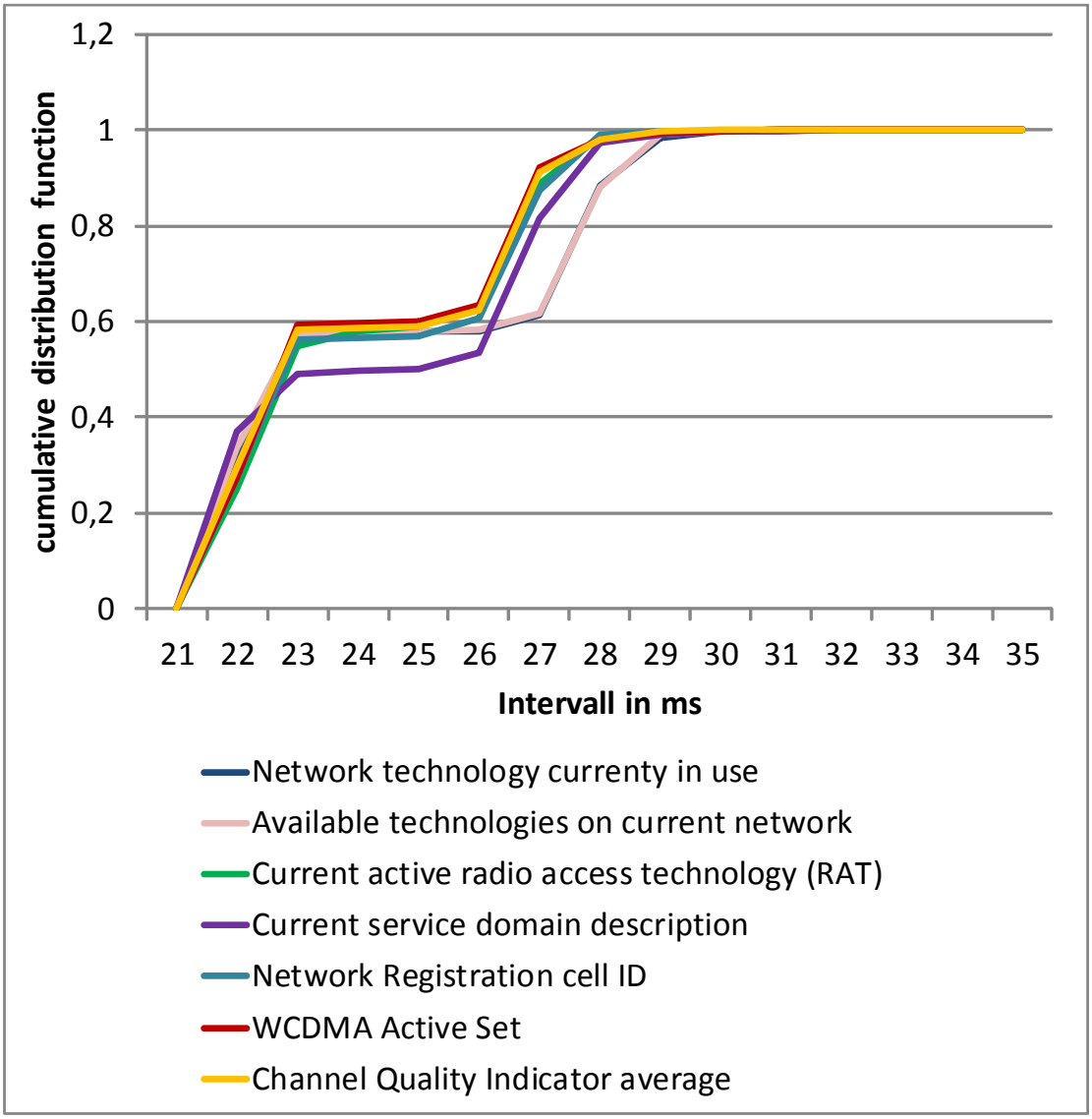

Abbildung 5.1.: CDF für Intervalle von empfangenen Datenwerten.

Werten 21 bis 23 und 26 bis 28 wurden nicht erwartet. Um diese Ergebnisse weiter zu untersuchen, wurden weitere Messungen des Modems durchgeführt.

Bei der nächsten Messreihe wurde nicht das Intervall der beim MIS Client empfangenen Datenwerte in Millisekunden gemessen, sondern die reine Arbeitszeit des Modems vom MIS in Mikrosekunden. Abbildung 5.2 bestätigt die Stufenbildung von Abbildung 5.1. Zusätzlich lassen die genaueren Messungen feinere Stufen erkennen. Dies deutet darauf

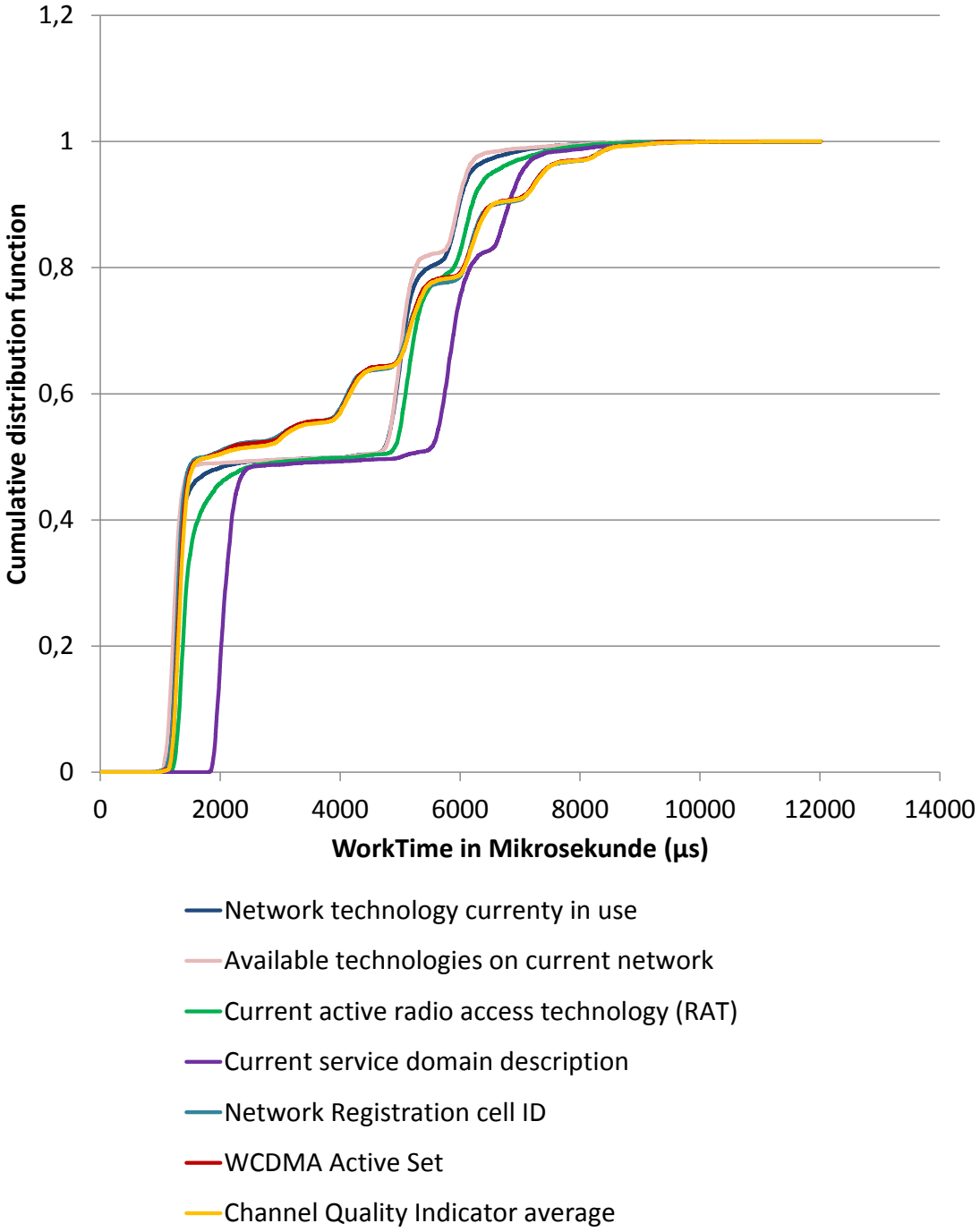

Abbildung 5.2.: CDF für WorkTimes des Modems in Mikrosekunden.

hin, dass die virtuelle serielle Schnittstelle und/oder der USB Link und/oder das Modem die Daten nur nach bestimmten Intervallen bearbeiten oder weiterleiten. Dies verleitet zu der Hypothese, dass der MIS eine erwartungstreue Ressourcenverteilung mit steigender Kollisionsanzahl und steigender periodischer Veröffentlichung einhält (Kollision, siehe Kapitel 4.4).

5.4. Praxisrelevante Tests und Messungen

In den nächsten fünf praxisrelevanten Tests und Messungen werden gleichzeitig mehrere Datenwerte mit unterschiedlichen Prioritätsklassen beim MIS abonniert. Es werden drei Prioritätsklassen mit den Gewichten 70, 25 und 5, das Prioritäts-Berechnungsverfahren *Prio. kopieren* (Kapitel 3.4.2.1) und invertierte Aging Scheduler angewendet. In einem Test wurde zusätzlich der Aging Scheduler zum Vergleich herangezogen. Die Datenwerte werden so gewählt, dass jeder Datenwert eine eigene periodische Aufgabe bildet. Diese drei gewählten Eigenschaften haben zur Folge, dass die periodischen Aufgaben die Gewichte ihrer Datenwerte erhalten. Bei diesem Test subtrahiert der MIS vom angegeben Intervall die WorkTime. Auf diese Weise soll erreicht werden, dass ein Client das Intervall ohne Berücksichtigung der WorkTime auswählen kann. Die unzuverlässige WorkTime des Modems (siehe Ergebnisse aus Kapitel 5.3) lässt für diese Messreihe stetig bessere Ergebnisse bei einer steigenden Kollisionsanzahl erwarten. Nach dem *Gesetz der großen Zahlen* ist zu erwarten, dass sich die wiederholenden unzuverlässigen WorkTimes der einzelnen AT Befehle ausbalancieren. Deshalb wurden die Werte für die folgenden Messungen erst nach mindestens 10.000 veröffentlichten Datenwerten entnommen.

Beim ersten Test werden drei Datenwerte mit jeweils unterschiedlichen Prioritätsklassen (im Folgenden PK genannt) und einem Intervall von 25 ms vom MIS abonniert. Die Daten in Tabelle 5.3 zeigen, dass der MIS seine Aufgabe gut bis sehr gut erfüllt, wenn man bei der Interpretation die Ergebnisse aus Kapitel 5.3 berücksichtigt. Denn Datenwert A und B werden nach dem angegebenen Intervall und Datenwert C trotz eines deutlich kleineren Prioritätsgewichts mit nur 3 ms Verspätung veröffentlicht.

Tabelle 5.3.: Test 1: 3 Datenwerte, Intervall von 25 ms, invertierter Aging Scheduler.

Datenwerte pro Sekunde (gesamt)	Ø Intervall in ms		
	Datenwert A mit PK 1 (Gewicht 70)	Datenwert B mit PK 2 (Gewicht 25)	Datenwert C mit PK 3 (Gewicht 5)
114	25	25	28

Datenwert A = Network technology currently in use
Datenwert B = Available technologies on current network
Datenwert C = Current active radio access technology (RAT)

Um ein *Worst-Case Szenario* zu erstellen, wird der erste Test zwei Mal mit einem Intervall von einer Millisekunde wiederholt, so dass konstant Kollisionen evoziert werden und die vom Client gewünschten Intervalle nicht eingehalten werden können. Die gemessenen Daten werden in Tabelle 5.4 und Tabelle 5.5 dargestellt. Sie zeigen, dass in einem realitätsnahen Praxisfall, wie in Kapitel 3.5.3 durch Simulationen erprobt, der Aging Scheduler nur befriedigende, der invertierte Aging Scheduler jedoch bessere Ergebnisse liefert. Tabelle 5.5 zeigt eine deutliche Fairness-Verbesserung des invertierten Aging Schedulers.

Tabelle 5.4.: Test 2: 3 Datenwerte, Intervall von 1 ms, Aging Scheduler.

	Datenwert A mit PK 1 (Gewicht 70)	Datenwert B mit PK 2 (Gewicht 25)	Datenwert C mit PK 3 (Gewicht 5)
Ø Intervall in ms	8	15	77
Empfangene Datenwerte pro Sekunde	125,0	66,6	12,9
Ist-Wert der prozentualen Ressourcenverteilung	61,0 %	32,5 %	6,3 %
Soll-Wert der prozentualen Ressourcenverteilung	70 %	25 %	5 %

Im vierten Test wird ein Worst-Case Szenario, nicht durch ein zu kleines Intervall, sondern durch die Anzahl der Abonnements hervorgerufen. Insgesamt werden sieben Datenwerte mit jeweils einem Intervall von 25 ms abonniert. Hierbei fordert der Client einen Datenwert mit Prioritätsklasse 1 (PK1 - Gewicht 70), drei Datenwerte mit Prioritätsklas-

Tabelle 5.5.: Test 3: 3 Datenwerte, Intervall von 1 ms, invertierter Aging Scheduler.

	Datenwert A mit PK 1 (Gewicht 70)	Datenwert B mit PK 2 (Gewicht 25)	Datenwert C mit PK 3 (Gewicht 5)
Ø Intervall in ms	6	18	90
Empfangene Datenwerte pro Sekunde	166,6	55,5	11,1
Ist-Wert der prozentualen Ressourcenverteilung	71,4 %	23,8 %	4,7 %
Soll-Wert der prozentualen Ressourcenverteilung	70 %	25 %	5 %

se 2 (PK2 - Gewicht 25) und drei Datenwerte mit Prioritätsklasse 3 (PK3 - Gewicht 5) an.

Tabelle 5.6.: Test 4: 7 Datenwerte, Intervall von 25 ms, invertierter Aging Scheduler.

	A mit PK 1	B mit PK 2	C mit PK 2	D mit PK 2	E mit PK 3	F mit PK 3	G mit PK 3
Ø Intervall in ms	27	29	30	30	42	42	42
Empfangene Datenwerte pro Sekunde	37	34	33	33	23	23	23

Datenwert A = Network technology currently in use
Datenwert B = Available technologies on current network
Datenwert C = Current service domain description
Datenwert D = Network Registration cell ID
Datenwert E = Current active radio access technology (RAT)
Datenwert F = WCDMA Active Set
Datenwert G = Channel Quality Indicator totalvalidsamples

Bei einem Intervall von 25 ms sollte jeder Datenwert 40 Mal pro Sekunde veröffentlicht werden. Wenn bei einem Intervall von 25 ms sieben Datenwerte abonniert werden, befindet sich das Modem in einer Überlastungssituation. Die Ergebnisse in Tabelle 5.6 zeigen, dass der MIS seine Aufgaben, das heißt Datenwerte mit Berücksichtigung ihrer Priorität beim Modem abfragen und veröffentlichen, gut ausführt. Denn Datenwert A wird nur

Tabelle 5.7.: Test 5: 7 Datenwerte, Intervall von 1 ms, invertierter Aging Scheduler.

	A mit PK 1	B mit PK 2	C mit PK 2	D mit PK 2	E mit PK 3	F mit PK 3	G mit PK 3
Ø Intervall in ms	11	30	30	30	153	154	154
Empfangene Datenwerte pro Sekunde	90,9	33,3	33,3	33,3	6,5	6,4	6,4
Ist-Werte % Ressourcenverteilung	43,2 %	15,8 %	15,8 %	15,8 %	3,1 %	3,1 %	3,1 %
Soll-Werte % Ressourcenverteilung	43,7 %	15,6 %	15,6 %	15,6 %	3,1 %	3,1 %	3,1 %

Datenwert A = Network technology currently in use
Datenwert B = Available technologies on current network
Datenwert C = Current service domain description
Datenwert D = Network Registration cell ID
Datenwert E = Current active radio access technology (RAT)
Datenwert F = WCDMA Active Set
Datenwert G = Channel Quality Indicator totalvalidsamples

drei Mal pro Sekunde weniger ausgeliefert wie erwartet. Die Ressourcen verteilen sich auf alle sieben Datenwerte mit Berücksichtigung ihrer Prioritäten.

Im letzten Test wurde die Messreihe des vierten Tests mit einem Intervall von nur 1 ms repliziert. Die Resultate können Tabelle 5.7) entnommen werden. Der maximale Abstand zwischen Ist- und Soll-Werten ist sehr klein, da es kontinuierlich zu Kollisionen kommt, die alle periodischen Aufgaben betreffen. Somit kann der MIS nach den Vorgaben des invertierten Aging Schedulers die Ressourcen komplett und eigenständig verteilen. Bei der Messreihe mit einem Intervall von 25 ms (Test 4) war dies nicht möglich, da das große Intervall den MIS für jeden Datenwert *ausgebremst* hat.
Tabelle 5.7 zeigt, dass, wenn der MIS durch den Intervall nicht *ausgebremst* wird, er sehr gute Ist-Werte mit dem invertierten Aging Scheduler liefert.

Kapitel 6.

Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde ein Service entwickelt, der Modeminformationen (im Folgenden Datenwerte genannt) abruft und anderen Anwendungen (im Folgenden Clients genannt) nach dem Publish-Subscribe Verfahren zur Verfügung stellt. Die Kommunikation zwischen dem Modem-Informations-Service (im Folgenden MIS genannt) und den Clients wird nach einem entworfenen und erweiterbaren Netzwerk-Protokoll mit Baum-Struktur über IP/TCP durchgeführt. Jeder Client kann eigenständig bestimmen, wie häufig und welche Modemdaten er erhält.

Da das Modem die Datenwerte sequentiell und mit nur einer begrenzten maximalen Geschwindigkeit liefern kann, besteht die Gefahr einer Überlastung des Modems durch zu viele angeforderte periodische Datenwerte. Dies bedeutet, dass nicht alle Datenwerte rechtzeitig abgefragt und verteilt werden können. Dieses Worst-Case Szenario zögert der MIS möglichst lange hinaus, indem er die Anzahl der Datenabrufe beim Modem minimiert und einen Modemabruf für mehrere Datenwerte und Clients verwendet. Eine Prioritäts-Funktion wurde implementiert, um eine Worst-Case Situation zu kontrollieren. Die Clients geben jedem angefragten Datenwert eine Priorität, die der Service bei seiner Arbeit berücksichtigt. Um das Hinauszögern und Kontrollieren des Worst-Case Szenarios umzusetzen, wurden die folgenden Techniken entworfen und implementiert:

- Die benötigten Daten werden nicht einzeln und direkt beim Modem abgefragt, sondern durch einen AT Befehl. Ein AT Befehl kann mehrere Datenwerte liefern

und ein Datenwert kann wiederum durch verschiedene AT Befehle in Erfahrung gebracht werden. Deshalb wird mit einer rekursiven Tiefensuche die minimale Menge der AT Befehle ermittelt, die alle benötigten Datenwerte liefert. Dieses Vorgehen ermöglicht den Erhalt vieler Datenwerte durch nur einen Modemabruf. Anschließend werden diese Datenwerte an alle Abonnenten (Clients) veröffentlicht. Dadurch wird die Anzahl der Modemabrufe minimiert und in dessen Folge auch die Modembelastung verringert.

- Die Prioritäten der Datenanfragen werden auf die AT Befehle abgebildet. Hierfür wurden zwei Prioritäts-Berechnungsverfahren für den Single-Client und drei für den Multi-Client Betrieb entworfen. Diese Verfahren bestimmen den Arbeitsumfang, den der MIS für die Anfragen eines Clients investiert. Des Weiteren kann der Client Anfragen in Gruppen organisieren und die Aufteilung der Prioritäten der einzelnen Anfragen dem Service überlassen.

- Jeder AT Befehl erhält durch das vom Anwender gewählte Prioritäts-Berechnungsverfahren ein Prioritätsgewicht. Sodann verteilt der MIS die begrenzten Modemressourcen an die AT Befehle entsprechend ihrer Prioritätsgewichte.
  Um die Ressourcen des MIS und Modems möglichst genau auf die eingestellten und berechneten Prioritätsgewichte der AT Befehle zu verteilen, wurden ein Round Robin Scheduler, ein Aging Scheduler und ein weiterentwickelter *invertierter* Aging Scheduler implementiert. Der Round Robin Scheduler eignet sich primär für Situationen mit wenigen periodischen Modemabrufen, die besonders große Zeitintervalle haben. Der Aging Scheduler ist zusätzlich in Worst-Case Situationen einzusetzen, wenn die AT Befehle zueinander das halbe oder doppelte Prioritätsgewicht haben. Der *invertierte* Aging Scheduler geht einen Schritt weiter. Er erlaubt eine komplett freie Wahl der Prioritätsgewichte und ist somit für alle Situationen geeignet. Diese Schlussfolgerungen werden durch Testergebnisse, die im Rahmen einer Evaluierung des MIS entstanden sind, bestätigt.

Der Anwender kann durch die Wahl von geeigneten Berechnungsverfahren und Schedulern eine optimale Passung zwischen MIS und den äußeren Bedingungen herbeiführen.

Bei der Implementierung des MIS wurde besonders viel Wert auf eine ressourcenschonende Funktionsweise und Erweiterbarkeit gelegt. Durch eine strikte Aufgabenteilung im MIS, die durch unterschiedliche Objektklassen und Threads realisiert ist, wird die gleichzeitige Verwaltung und Bedienung von mehreren Modems und Clients ermöglicht.

Um häufige Änderungen am Programmiercode und einen benutzerfreundlichen Praxiseinsatz zu gewährleisten, wurde der MIS um eine Konfigurationsdatei erweitert. Die Konfigurationsdatei hat einen logischen Aufbau und kann vom Anwender mit einem Texteditor bearbeitet werden. Der MIS erhält aus dieser Datei alle nötigen Informationen, um ein Modem zu bedienen und seine Datenwerte zu ermitteln. Somit ist es möglich ohne Änderungen des Quellcodes den MIS an zukünftige und ihm noch nicht bekannte Modems (z.B. LTE) anzupassen.

6.1. Ausblick

Während der Erstellung dieser Bachelorarbeit sind weitere Aufgaben aufgekommen, die bisher aufgrund zeitlicher Vorgaben noch nicht im MIS implementiert werden konnten. Dadurch ist zwar die Funktionalität des MIS nicht beeinträchtigt, sie werden aber an dieser Stelle dennoch erwähnt, um in weitere Arbeiten und Erweiterungen Berücksichtigung zu finden.

In den drei Monaten Bearbeitungszeit ist es circa sechs Mal vorgekommen, dass das Modem die Kodierung der AT Antwort minimal verändert hat. Eine Zeile mit ASCII Zeichen in der AT Antwort endet für gewöhnlich mit zwei new lines (\n\n). Ganz selten ist es aber vorgekommen, dass das Modem nach dem Starten des Computers eine Zeile mit einem carriage return und einem new line (\r\n) beendet hat. Dieser *Kodierungswechsel* ist seitens des Herstellers bisher weder dokumentiert noch konfigurierbar. Um die ursprüngliche Kodierung zu erhalten, muss der Anwender den Computer für mindestens eine Minute komplett ausschalten, oder den Akku kurzzeitig entfernen. Wir vermuten deshalb einen eventuellen Fehler in der Firmware, der durch eine Reinitialisierung, die durch das komplette Ausschalten erzwungen wird, automatisch gelöst wird. Ferner bietet sich für dieses Problem an, eine zweite Konfiguration für dieses Modem

anzulegen, in der die minimal veränderte Kodierung berücksichtigt wird. Eine benutzerfreundlichere Lösung wäre es, diese systematisch unterschiedliche Kodierung mit in die Modem Konfiguration aufzunehmen, damit sie automatisch vom MIS berücksichtigt wird. Dieser Ansatz wurde bereits diskutiert, jedoch noch nicht implementiert, da der Kodierungswechsel die bisherige Arbeit kaum beeinträchtigt hat.

Bei der Gestaltung der Konfigurationsdatei wurde auf Übersichtlichkeit geachtet. Die meisten Erweiterungen lassen sich durch simples Kopieren, Einfügen und minimales Anpassen umsetzen. Gleichzeitig steckt auch hier eine potentielle Fehlerquelle, die in Zukunft durch einen GUI-basierten Editor für Konfigurationsdateien beseitigt werden könnte.

Die serielle Schnittstelle und der AT Befehlssatz sind die primären Bremsen und Flaschenhälse des MIS. Es wäre nützlich noch andere Schnittstellen und Protokolle in Anspruch zu nehmen, um noch schneller und weitere Informationen vom Modem zu erhalten (z.B. CnS von Sierra Wireless). Für die Zukunft empfiehlt sich eine engere Kooperation mit den Herstellern (vergleiche Kapitel 3.1).

Anhang A.

Netzwerkprotokoll

Tabelle A.1.: Aufbau der Netzwerknachrichten (SerialPort).

SerialPort

Open SerialPort

Anfrage	0x01	0x00	a	l(b)	b				
OK	0x01	0x00	0x00	a	l(b)	b	l(c)	c	d
Already open	0x01	0x00	0x01	a	l(b)	b			
Error	0x01	0x00	0x02	a	l(b)	b			

Set Scheduler & Priority

Anfrage	0x01	0x01	d	e	f	g	h	...	
OK	0x01	0x01	0x00	d	e	f	g	h	...
Device Error	0x01	0x01	0x01	d	e	f	g	h	...
Priority in use	0x01	0x01	0x02	d	e	f	g	h	...
Error	0x01	0x01	0x03	i	s	p	c	g	...

Variable = Beschreibung [Größe in Byte]
l(x) = Länge von Zeichenkette x [1 Byte]
a = SerialPort Timeout [1 Byte]
b = Pfand der Gerätedatei [l(b) Byte]
c = Bezeichnung der Gerätekonfiguration [l(c) Byte]
d = Device Index [1 Byte]
e = Schedulertyp [1 Byte]
f = Prioritäts-Berechnungsverfahren [1 Byte]
g = Anzahl von Prioritätsklassen [2 Byte]
h = Gewicht einer Prioritätsklasse [1 Byte]

Tabelle A.2.: Aufbau der Netzwerknachrichten (AT Command).

AT Command

Run AT Command

Anfrage	0x02	a	l(b)	b				
OK	0x02	0x00	c	a	l(b)	b	l(d)	d
AT Error	0x02	0x01	c	a	l(b)	b	l(d)	d
Timeout	0x02	0x02	c	a	l(b)	b	l(d)	d
Device Error	0x02	0x03	c	a	l(b)	b	l(d)	d

Variable = Beschreibung [Größe in Byte]
l(x) = Länge von Zeichenkette x [1 Byte]
a = Device Index [1 Byte]
b = AT Anfrage [l(b) Byte]
c = Zeitstempel [8 Byte]
d = AT Antwort [8 Byte]

Tabelle A.3.: Aufbau der Netzwerknachrichten (Parsed Data).

Parsed Data

Get Parsed Data

Anfrage	0x03	0x00	a	b					
OK	0x03	0x00	0x00	c	d	a	b	e	f
AT Error	0x03	0x00	0x01	c	a	b	e		
Timeout	0x03	0x00	0x02	c	a	b	e		
Device Error	0x03	0x00	0x03	c	a	b	e		
Parse Error	0x03	0x00	0x04	c	a	b	e		

Run Parsed Data Config

Anfrage	0x03	0x01	a	g				
OK	0x03	0x01	0x00	c	a	g	h	i
AT Error	0x03	0x01	0x01	a	g	h	i	
Timeout	0x03	0x01	0x02	a	g	h	i	
Device Error	0x03	0x01	0x03	a	g	h	i	

Variable = Beschreibung [Größe in Byte]
l(x) = Länge von Zeichenkette x [1 Byte]
a = Device Index [1 Byte]
b = Datenwert Index [2 Byte]
c = Zeitstempel [8 Byte]
d = Arbeitszeit in ms vom MIS [8 Byte]
e = Datenwerttyp [1 Byte]
f = Datenwert in Abhängigkeit vom Datenwerttyp [x Byte]
g = AT Konfigbefehl Index [2]
h = Datenwerttyp der AT Antwort vom AT Konfigbefehl (string) [1 Byte]
i = AT Antwort des Modems (string) [x Byte]

Tabelle A.4.: Aufbau der Netzwerknachrichten (Manage Periodically Data Request).

Manage Periodically Data Request

Add Periodically Data Request

Anfrage	0x04	0x00	a	b	c	d		
OK	0x04	0x00	0x00	a	b	c	d	
Device Error	0x04	0x00	0x01	a	b	c	d	
Priorität Error	0x04	0x00	0x02	a	b	c	d	
Parsed Data Error	0x04	0x00	0x03	a	b	c	d	
Error	0x04	0x00	0x04	a	b	c	d	

Remove Periodically Data Request

Anfrage	0x04	0x01	a	c		
OK	0x04	0x01	0x00	a	c	e
Device Error	0x04	0x01	0x01	a	c	
Parsed Data Error	0x04	0x01	0x02	a	c	

Remove All Periodically Data Request

Anfrage	0x04	0x01	a		
OK	0x04	0x02	0x00	a	e
Device Error	0x04	0x02	0x01	a	

Variable = Beschreibung [Größe in Byte]
l(x) = Länge von Zeichenkette x [1 Byte]
a = Device Index [1 Byte]
b = Prioritätsklasse Index [1 Byte]
c = Datenwert Index [2 Byte]
d = Zeitintervall in ms [2 Byte]
e = Anzahl gelöschter periodischer Anfragen [2 Byte]

Tabelle A.5.: Aufbau der Netzwerknachrichten (Publish Periodically Data Request).

Publish Periodically Data Request

OK	0x05	0x00	a	b	c	d	e
AT Error	0x05	0x01	a	b	c	d	
Timeout	0x05	0x02	a	b	c	d	
Device Error	0x05	0x03	a	b	c	d	
Parse Error	0x05	0x04	a	b	c	d	

Variable = Beschreibung [Größe in Byte]
l(x) = Länge vom Zeichenkette x [1 Byte]
a = Zeitstempel [8 Byte]
b = Device Index [1 Byte]
c = Datenwert Index [2 Byte]
d = Datenwerttyp [1 Byte]
e = Datenwert in Abhängigkeit vom Datenwerttyp [x Byte]

Anhang B.

Konfigurationsdatei

Tabelle B.1.: Hierarchische Anordnung der Konfigurationsbefehle (Attributnamen).

Übergeordnet	Untergeordnet
deviceconfigname	end_signal_ok
deviceconfigname	end_signal_error
deviceconfigname	identity_at-command
deviceconfigname	at-command
deviceconfigname	at-command-config
at-command	at-c_name
at-command	at-c_worktime
at-command	atc_value_name
at-command	at-c_config_once_begin
at-command	at-c_config_once_end
atc_value_name	atc_v_type
atc_value_name	atc_v_beforestring
atc_value_name	atc_v_beforecount
atc_value_name	atc_v_endstring
atc_value_name	atc_v_endcount
identity_at-command	identity_value
at-command-config	at-c-c_name
at-command-config	at-c-c_answer
at-command	atc_mdataset_name
atc_mdataset_name	atc_mds_value_name
atc_mds_value_name	atc_mds_v_type
atc_mds_value_name	atc_mds_v_beforestring
atc_mds_value_name	atc_mds_v_endstring
atc_mds_value_name	atc_mds_v_endcount

Anhang C.

Datenträger

Der in dieser Bachelorarbeit entwickelte und implementierte Publish-Subscribe Modem-Informations-Service und MIS-Client werden mit dieser DVD-R mitgeliefert. Des Weiteren befinden sich die entworfene und benutzte Konfigurationsdatei sowie die vorliegende schriftliche Ausarbeitung ebenfalls auf diesem Datenträger.

Literaturverzeichnis

[Aus05] AUSTERN, Matt: *C++ Technical Report 1 (TR1)*. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1836.pdf, 2005.

[boo05] *Boost C++ Libraries*. http://www.boost.org/, 2005.

[cpp03] *C++ Standard ISO/IEC 14882:2003*. http://cs.nyu.edu/courses/summer11/G22.2110-001/documents/c++2003std.pdf, 2003.

[cpp11] *C++ Standard ISO/IEC 14882:2011*. http://www.iso.org/iso/iso_catalogue/catalogue_tc/
catalogue_detail.htm?csnumber=50372, 2011.

[ITU98] *ITU-T V.250*. http://www.itu.int/rec/T-REC-V.250/, 1998.

[KR08] KUROSE, James F.; ROSS, Keith W.: *Computernetzwerke - der Top-Down-Ansatz (4. Aufl.)*. Pearson Studium, 2008. 687 ff S. ISBN 978–3–8273–7330–4

[new12] *news - Mehr Verkehrsopfer durch schönes Wetter*. http://www.news.de/auto/855275459/verkehrstote-3991-tote-2011-ist-das-wetter-schuld/1/, 2012.

[SBD11] *Statistisches Bundesamt Deutschland*. http://www.destatis.de/jetspeed/portal/cms/Sites/destatis/Internet/DE/Navigation/Statistiken/Verkehr/Verkehrsunfaelle/Verkehrsunfaelle.psml, 2011.

[sim12] *Sichere Intelligente Mobilität Testfeld Deutschland*. www.simtd.de, 2012.

[SWA09] *Sierra Wireless AT Command Reference*. siptune.net/sm/docs/sierra-at-commands-292.pdf, 2009.

[Tan09] TANENBAUM, Andrew S.: *Moderne Betriebssysteme (3. Aufl.)*. Pearson Studium, 2009. 200 ff S. ISBN 978–3–8273–7342–7

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, 29. Februar 2012 Adrian Skuballa