Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3561

Efficient Positioning and Position Updating for Mobile Devices

Benjamin Gayer

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer/in:	M.Sc. Patrick Baier
Beginn am:	22. August 2013
Beendet am:	21. Februar 2014
CR-Nummer:	C.2.4

Abstract

Die Nutzung ortsbezogener Dienste wird mit der zunehmenden Verbreitung von Smartphones, die ihre Position selbst bestimmen können, immer wichtiger. Es gibt bereits eine Vielzahl solcher Anwendungen, die sich hoher Beliebtheit bei den Nutzern erfreuen. Einige dieser Dienste basieren darauf, dass das mobile Endgerät immer wieder seine Position bestimmt und diese an einen Server schickt. Dabei wird viel Energie für die Nutzung des GPS-Sensors und das Versenden der Positionsdaten über ein drahtloses Netzwerk verbraucht. Dies führt dazu, dass die Akkulaufzeit der Smartphones, auf denen solche Dienste genutzt werden, stark reduziert wird.

Bisher liegt der Schwerpunkt der Forschung darauf, die Nutzung des GPS-Sensors zu minimieren, da die Positionsbestimmung teurer ist als das Versenden der Positionsdaten. Doch auch das Versenden von Positionsdaten verbraucht viel Energie. Baier et al. [BDR13] weisen darauf hin, dass Smartphones über einen eigenen Mechanismus verfügen, um Energie beim Senden von Nachrichten zu sparen. So wird Energie gespart, wenn Nachrichten in kurzen Abständen hintereinander versendet werden. Das Ziel ist also, Positionsdaten zu versenden, kurz nachdem eine andere Nachricht versendet wurde.

Ziel dieser Arbeit ist es, einen Algorithmus zu entwerfen und zu testen, der sowohl die Kosten für Positions-Updates als auch die Kosten für die Nutzung des GPS-Sensors reduziert.

Der hier vorgestellte Algorithmus gehört zur Klasse der distanzbasierten Update-Protokolle und nutzt ein nicht lineares Energiemodell. Mithilfe eines Simulators wird der Algorithmus ausführlich analysiert und zusätzlich auf einem Android-Smartphone getestet. Sowohl die Kostenreduktion für Update-Nachrichten als auch die Kostenreduktion für die Positionsbestimmung werden untersucht. Abhängig von der Häufigkeit, mit der Nachrichten versendet werden, und der Güte der Systemparameter lassen sich, im Vergleich zum Standardalgorithmus, über 50% der gesamten Energiekosten einsparen. Im Durchschnitt werden ca. 30% Einsparung erreicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Aufgrund der hohen Verfügbarkeit von Smartphones, die in der Lage sind, ihre Position zu bestimmen, werden immer mehr standortbezogene Dienste entwickelt. So sind mittlerweile Anwendungen, die nahegelegene Restaurants oder Kinos anzeigen, einem breiten Publikum bekannt. Solche reaktiven Dienste, die nur auf Nachfrage des Nutzers die Position bestimmen, sind weit verbreitet. Doch proaktive Dienste, wie z. B. ein Programm, das anzeigt, welche Freunde sich gerade in der Nähe des Nutzers befinden, sind bisher relativ unbeliebt. Dies liegt allerdings nicht am angebotenen Service, an dem durchaus ein hohes Interesse besteht, sondern an der Tatsache, dass die Nutzung eines solchen Dienstes die Akkulaufzeit erheblich reduziert.

Proaktive Dienste benötigen zu jeder Zeit hinreichend genaue Positionsdaten, daher muss ein mobiles Endgerät immer wieder seine Position bestimmen und diese anschließend an einen Server senden. Das Senden der GPS-Koordinaten an einen Location-Server nennt man Positions-Update. Die dabei versandte Nachricht, die die Position des mobilen Endgeräts und einen Zeitstempel enthält, wird Update-Nachricht genannt. Der Zeitstempel enthält das Datum, an dem die GPS-Messung durchgeführt wurde. Dies ist nötig, da der Zeitpunkt an dem die Nachricht beim Location-Server ankommt, nicht notwendigerweise der gleiche ist, zu dem die Position bestimmt wurde. Die auf diesem Location-Server gespeicherten Positionsdaten können dann von standortabhängigen Diensten genutzt werden. Diesen Sachverhalt verdeutlicht Abbildung 1.1

Sowohl das Messen der aktuellen Position wie auch das Übertragen der Positionsdaten benötigen viel Energie. Daher wurden zahlreiche Update-Protokolle (einen Überblick über diese Protokolle liefert [LR01]) entwickelt, die versuchen, den Energieverbrauch zu minimieren. Dabei wurden drei Typen von Update-Protokollen entwickelt:

Zeitbasierte Update-Protokolle versenden zu bestimmten Zeitpunkten Positions-Updates, distanzbasierte Algorithmen versenden nur dann Updates, wenn eine vorbestimmte Genauigkeit unterschritten wird. Dead reckoning Protokolle versuchen die Bewegungen eines mobilen Endgerätes vorherzusagen, z. B. anhand von Straßenkarten und senden nur dann ein Update, wenn die gemessene Position um einen bestimmten Wert von der berechneten Position abweicht.

Bisher liegt der Schwerpunkt der Forschung darauf, die Kosten für die Positionsbestimmung zu reduzieren, indem die Anzahl der Positions-Messungen minimiert wird. Dabei zeigt sich, dass man Energie nur dann einsparen kann, wenn man eine ungenauere Position verwendet. Man kann also seltener die Position messen oder andere Verfahren als GPS einsetzen, um Positionsdaten zu gewinnen, bzw. diese beiden Möglichkeiten kombinieren. Ein Beispiel für so eine Kombination ist die Arbeit von Chon et al. [CTSC11]. In dieser Arbeit wird das GPS nur verwendet, wenn kein anderes Verfahren eine hinreichend genau Position liefern kann.
Die Verfahren unterscheiden sich also sowohl in den Energiekosten als auch in der Genauigkeit der Positionsdaten.

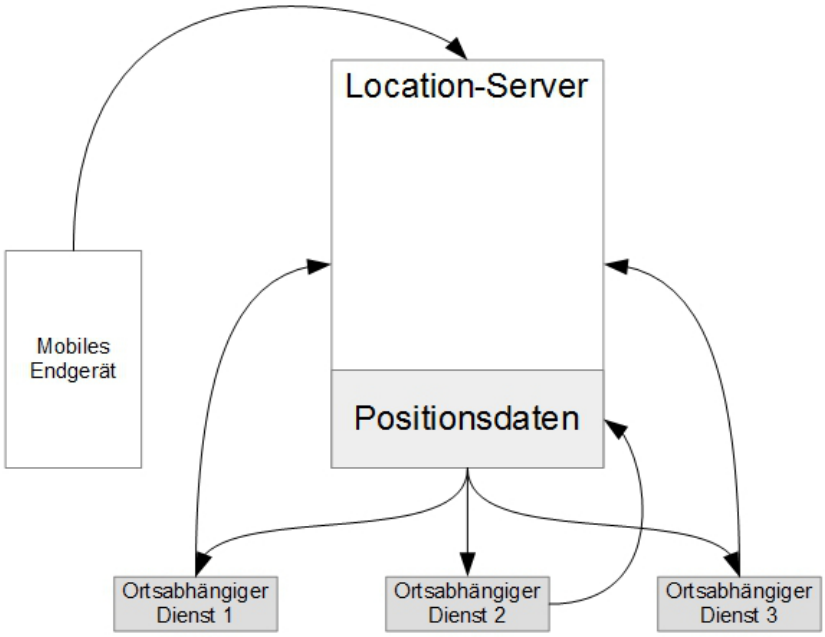


Abbildung 1.1: Positions-Update: Das mobile Endgerät sendet seine Position an einen Location-Server. Jedes Mal wenn der Location-Server eine neue Position erhält, sendet er diese an alle ortsabhängigen Dienste, die sich bei ihm registriert haben.

Doch auch bei der Übertragung von Nachrichten lässt sich Energie sparen, wie Baier et al.[BDR13] zeigen. Die Energie wird eingespart, indem Update-Nachrichten kurz nach Nachrichten versendet werden, die vom mobilen Endgerät regulär versendet werden, also Nachrichten, die nicht zum Update-Protokoll gehören. Es geht darum, dass durch das Ausführen eines Update-Protokolls so wenig wie möglich zusätzliche Energiekosten entstehen. Der Grund dafür, dass sich hierdurch Energie sparen lässt, liegt darin, dass die Sendeeinheit des mobilen Endgeräts über drei verschiedene Zustände verfügt.
Abbildung 1.2 zeigt die drei Energie-Zustände der Sendeeinheit. Falls keine Nachrichten gesendet werden, ist sie abgeschaltet und verbraucht keine Energie. Wenn eine Nachricht versendet wird, geht sie in einen Zustand über, in dem viel Energie verbraucht wird. Nach dem Senden dieser Nachricht bleibt sie noch einige Sekunden in diesem Zustand, falls weitere Nachrichten versendet werden. Anschließend wechselt sie in einen Ruhezustand, in dem weniger Energie benötigt wird. In diesem Zustand verweilt sie wieder mehrere Sekunden, bevor sie sich abschaltet, falls keine weiteren Nachrichten gesendet werden.
Für das Senden einer einzelnen Nachricht fallen daher sowohl Kosten für die Dauer der Übertragung an als auch für die darauf folgende Ruhephase. Werden zwei Nachrichten direkt nacheinander versendet, so bleibt die Ruhephase gleich lang. Daher entsteht pro versendeter Nachricht nur die Hälfte der Kosten für die Ruhephase.
Abbildung 1.3 zeigt die drei Möglichkeiten, wie zwei Nachrichten hintereinander versendet werden können und verdeutlicht, wie dabei Energie gespart werden kann. Der Energieverbrauch beim Senden kann also reduziert werden, wenn Nachrichten vom mobilen Endgerät in Bündeln versendet werden.

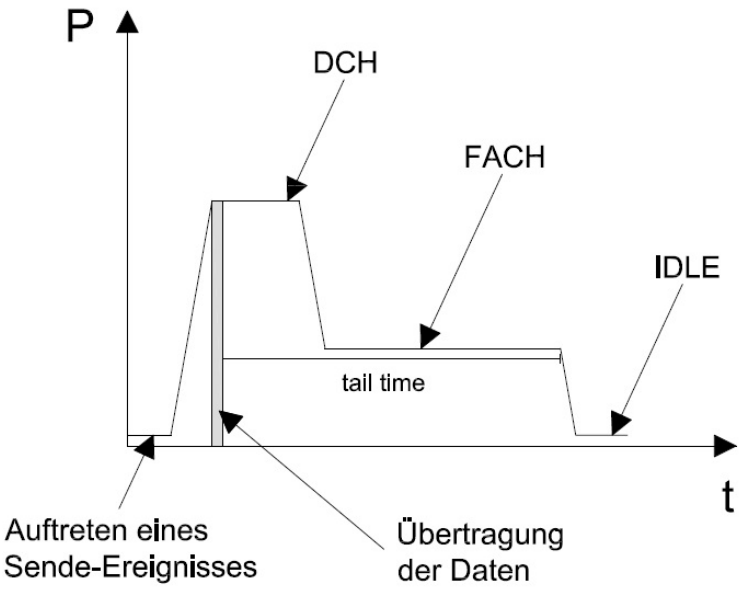


Abbildung 1.2: Energie-Zustände der Sendeeinheit. Die Netzwerkschnittstelle für ein zellulares Netzwerk verfügt über drei Energiezustände:
DCH (Dedicated Channel), FACH (Forward access channel) und IDLE. Der IDLE-Zustand ist der Ruhezustand; in ihm ist der Energieverbrauch am geringsten.
Wird eine Nachricht gesendet, so wechselt die Sendeeinheit in den DCH-Zustand, dieser verbraucht am meisten Energie. Die Sendeeinheit verlässt diesen Zustand nicht direkt nach dem Senden, sondern bleibt noch ein paar Sekunden in diesem Zustand. Anschließend wechselt sie in den FACH-Zustand, welcher weniger Energie benötigt. In diesem Zustand bleibt sie sendebereit und kann, falls eine Nachricht gesendet wird, direkt in den DCH-Zustand wechseln. Wird keine weitere Nachricht gesendet, so wechselt die Sendeeinheit wieder zurück in den IDLE-Zustand.

Diese Technik wird Piggyback-Update (zu deutsch: "Huckepack-Update") genannt.
Der Energiebedarf für das Versenden einer Update-Nachricht ist also nicht immer gleich (linear), sondern hängt davon ab, ob vor oder nach der Update-Nachricht noch weitere Nachrichten versendet werden (nicht linear).

Ziel dieser Arbeit ist es, den Ansatz, Energie mithilfe von Piggyback-Updates einzusparen, mit einem Update-Protokoll zu kombinieren. Dadurch kann nicht nur Energie beim Senden von Update-Nachrichten eingespart werden, sondern auch beim Messen der Positionen. Diese Einsparung wird erreicht, indem der GPS-Sensor weniger genutzt wird, da die Anzahl der notwendigen Messungen verringert werden kann. Der Algorithmus soll mit Hilfe eines Simulators und einer realen Implementierung auf einem Andorid-Smartphone getestet werden. Um dies zu erreichen, wird zuerst einer der drei Typen von Update-Protokollen ausgewählt, anschließend werden verwandte Arbeiten, die einen Bezug zu diesem Update-Protokoll haben, näher betrachtet. Es wird eine dieser verwandten Arbeiten

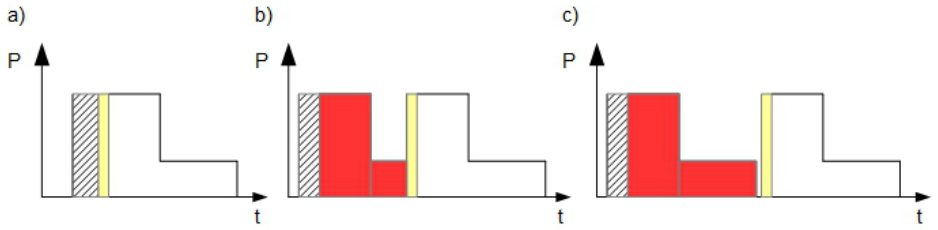


Abbildung 1.3: Energie-Einsparung beim Senden von Nachrichten:
Betrachtet werden Update-Nachrichten (gelb), die nach anderen Nachrichten (grau schraffiert) gesendet werden. Zu sehen sind drei verschiedene Fälle, in denen unterschiedlich viel Energie benötigt wird.
Fall a) maximale Energieeinsparung - Die Update-Nachricht wird unmittelbar nach einer anderen Nachricht versendet.
Fall b) geringe Energieeinsparung - Die Update Nachricht wird kurz vor dem Ende der Standby-Phase der Sendeeinheit gesendet.
Fall c) keine Energieeinsparung - Die Update Nachricht wird nicht in der Standby-Phase gesendet. Die durch die Update-Nachrichten zusätzlich benötigte Energie ist durch die roten und gelben Flächen dargestellt. Dabei repräsentieren die gelben Flächen die Energie, die für das Senden der Nachrichten benötigt wird und die roten Flächen die Energie, die für die zusätzliche Standby-Zeit benötigt wird.

sowie der Standardalgorithmus für das gewählte Update-Protokoll ausgewählt. Der neu entwickelte Algorithmus wird dann mit diesen beiden Ansätzen im Simulator verglichen.

Die vorliegende Diplomarbeit ist wie folgt aufgebaut:
In Kapitel 2 werden Grundlagen und verwandte Arbeiten vorgestellt. Die Grundlagen sind zellulare Netzwerke, GPS, standortbezogene Dienste und Update-Protokolle. Als verwandte Arbeiten werden die Arbeiten von Chon et al. [CTSC11], Farrell et al. [FLR07] und Baier et al. [BDR13] vorgestellt.
Kapitel 3 enthält eine Beschreibung des zugrundeliegenden System Modells, also aller Komponenten, die der in dieser Arbeit beschriebenen Algorithmus verwendet.
In Kapitel 4 wird ein Update-Protokoll gewählt und ein Ansatz aus den verwandten Arbeiten festgelegt, mit dem der neu entwickelte Algorithmus verglichen werden soll. Anschließend wird der Algorithmus im Detail vorgestellt und es werden die Probleme beschrieben, die beim Entwurf des Algorithmus aufgetreten sind.
Die Implementierungen des Simulators und der Android-Applikation werden in Kapitel 5 vorgestellt. Die Ergebnisse der Evaluierung werden in Kapitel 5 präsentiert und analysiert.
Kapitel 6 schließt die Arbeit mit einem Fazit und einem Ausblick auf weiterführende Forschung ab.

2 Grundlagen und verwandte Arbeiten

In diesem Kapitel werden Grundlagen vorgestellt, die für das Verständnis dieser Arbeit erforderlich oder hilfreich sind. Zusätzlich werden verwandte Arbeiten aus dem behandelten Forschungsgebiet vorgestellt, die einen direkten Bezug zur Arbeit haben.
Zellulare Netzwerke werden vorgestellt, da das verwendete Energiemodell nur für diese gilt. Standortbezogene Dienste sind der Hauptgrund für die Entwicklung von Update-Protokollen. Mobile Endgeräte, die standortbezogene Dienste nutzen möchten, müssen ihre Position bestimmen können. GPS ist die am weitesten verbreitete Technologie, um diese Positionsdaten zu ermitteln. Update-Protokolle wurden entwickelt, um die Gewinnung von Positionsdaten und deren Übertragung über ein Netzwerk energieeffizienter zu machen.
Die hier vorgestellten Arbeiten sind mit dem in dieser Arbeit behandelten Themengebiet verwandt und liefern wichtige Grundlagen. SmartDC ist ein weit entwickeltes System, das ein gutes Beispiel dafür ist, wie in der aktuellen Forschung weitere Energieeinsparungen vorgenommen werden. Farrell et al. liefern wichtige Erkenntnisse, um den Energieverbrauch für die Positionsbestimmung in distanzbasierten Update-Protokollen zu senken. Baier et al. zeigen, dass es möglich ist, mit opportunistisch gesendeten Updates Energie beim Übertragen von Nachrichten einzusparen.

2.1 Zellulare Netzwerke

Zellulare Netzwerke sind eine Technologie, um mobile Endgeräte über ein drahtloses Netzwerk zu verbinden. Über sie läuft der Großteil der mobilen Kommunikation, z. B. Telefonate von oder zu einem mobilen Endgerät. Ein Gebiet wird dafür in meist hexagonale Zellen aufgeteilt. Jede dieser Zellen verfügt über eine Basisstation, die über ein verdrahtetes Netzwerk mit anderen Basisstationen verbunden ist.
Ein wichtiger Unterschied zu Technologien wie Bluetooth ist, dass die mobilen Endgeräte nicht direkt miteinander kommunizieren, sondern immer über mindestens eine Basisstation. Dadurch ist es sowohl möglich innerhalb einer Zelle über die Basisstation mit einem anderen mobilen Endgerät zu kommunizieren als auch mit mobilen Endgeräten, die sich in anderen Zellen befinden.
Auch der Wechsel zwischen Zellen ist möglich. Bei mobilen Endgeräten kommt es immer wieder vor, dass ein solches Gerät eine Zelle verlässt und eine andere betritt. Geschieht dies, so muss ein *handover* durchgeführt werden. Dabei meldet sich das mobile Endgerät bei der für die neu betretene Zelle zuständigen Basisstation an. Diese Positionsänderung muss dann im Netzwerk der Basisstationen bekannt gemacht werden, damit das mobile Endgerät auch von anderen Basisstationen gefunden werden kann. Jeder Zelle wird ein bestimmtes Set an Frequenzen zugewiesen, dabei erhalten benachbarte Zellen unterschiedliche Frequenzen, um Interferenzen zu verhindern. Nicht direkt benachbarte Zellen können jedoch die selben Frequenzen verwenden. Dies erhöht die Kapazität des

Netzes im Vergleich zu einem Netz mit nur einer Basisstation, da die selbe Frequenz gleichzeitig für mehrere Verbindungen verwendet werden kann, solange diese in nicht benachbarten Zellen genutzt werden.
Die Größe der Zellen variiert abhängig von der Nutzerzahl, so werden in dicht bewohnten Gebieten kleinere Zellen benötigt, während im ländlichen Raum große Zellen ausreichen. Das liegt daran, dass jeder Zelle nur ein Teil der nutzbaren Frequenzen zugewiesen ist und keine Frequenz innerhalb einer Zelle zur gleichen Zeit von mehreren Nutzern verwendet werden kann.

2.2 GPS

Das Global Positioning System (GPS) ist ein vom U.S.-Militär entwickeltes und betriebenes Satellitennavigationssystem, das es einem Nutzer ermöglicht seinen Standort, gegeben durch Längen- und Breitengrad sowie Höhe über Normal-Null, auf wenige Meter genau zu bestimmen. Das GPS besteht aus 24 Satelliten (seit 2011 27 [GPSc]) und einigen weiteren als Reserve, die die Erde in einer Höhe von 20200 km umkreisen. Die Anordnung dieser 24 Satelliten sorgt dafür, dass von jedem Punkt auf der Erde zu jeder Zeit mindestens 4 Satelliten sichtbar sind. Jeder dieser Satelliten verfügt über mehrere hoch präzise Atomuhren (Arbeitsfrequenz: 10,23 MHz) und sendet periodisch Radio-Signale, die Informationen über seinen Standort und die Sende-Zeit des Signals enthalten. Jeder Satellit überträgt zwei Trägerschwingungen, deren Frequenz Vielfachen der Arbeitsfrequenz der Atomuhren entspricht, vgl. Abbildung 2.2.
Die Frequenz des L1-Signals beträgt 1575,42 MHz (154 * 10,23), auf dieses werden der C/A-Code (Coarse/Acquisition) und der P/Y-Code (Precision/encrypted) moduliert. Der C/A-Code ist eine pseudozufällige Sequenz für die zivile Nutzung, sie wird 1000 Mal in der Sekunde wiederholt und ist für jeden Satelliten einzigartig.
Der P-Code wird verschlüsselt als Y-Code übertragen und militärisch verwendet, daher ist er in dieser Arbeit ohne Bedeutung und wird nicht weiter betrachtet. Der P-Code kann auch unverschlüsselt übertragen werden, aber die Verschlüsselung bringt militärische Vorteile mit sich.
Die Frequenz des L2-Signals beträgt 1227,60 MHz (120 * 10,23), es enthält nur den verschlüsselten P-Code.

Abbildung 2.1 zeigt, wie sich die beiden Signale zusammensetzen.

Um eine Positionsbestimmung durchzuführen, muss ein GPS-Empfänger gültige Informationen über die Bahnen der Satelliten, bzw. deren Aufenthaltsort auf diesen Bahnen besitzen. Diese werden benötigt, um die Satelliten zu finden und die Position zu bestimmen. Zwei Arten von Daten werden dabei unterschieden: Almanach-Daten und Ephemeriden. Diese Daten beschreiben die Bahnen von astronomischen Objekten. Ephemeriden sind dabei wesentlich genauer als Almanach-Daten, aber sie sind weniger lange gültig. Je nach Hersteller des GPS-Empfängers werden diese Daten als unterschiedlich lange gültig erachtet. Ephemeriden sind maximal ca. 4h lang gültig, während Almanach-Daten für mehrere Monate gültig sein können [GPSd].
Je nachdem ob gültige Daten vorhanden sind, dauert die Positionsbestimmung unterschiedlich lange, da gültige Daten erst von einem der GPS-Satelliten heruntergeladen werden müssen. Jeder GPS-Satellit sendet seine eigenen Ephemeriden und Almanach-Daten für jeden anderen GPS-Satelliten.
Einige Hersteller bieten über das Internet spezielle Ephemeriden zum Herunterladen an, die 7 Tage

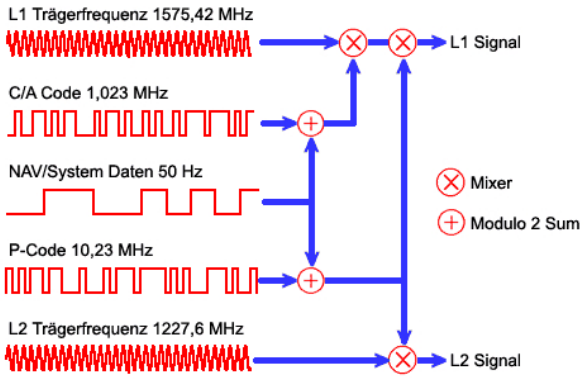


Abbildung 2.1: Aufbau des GPS-Signals. Quelle: [GPSa]

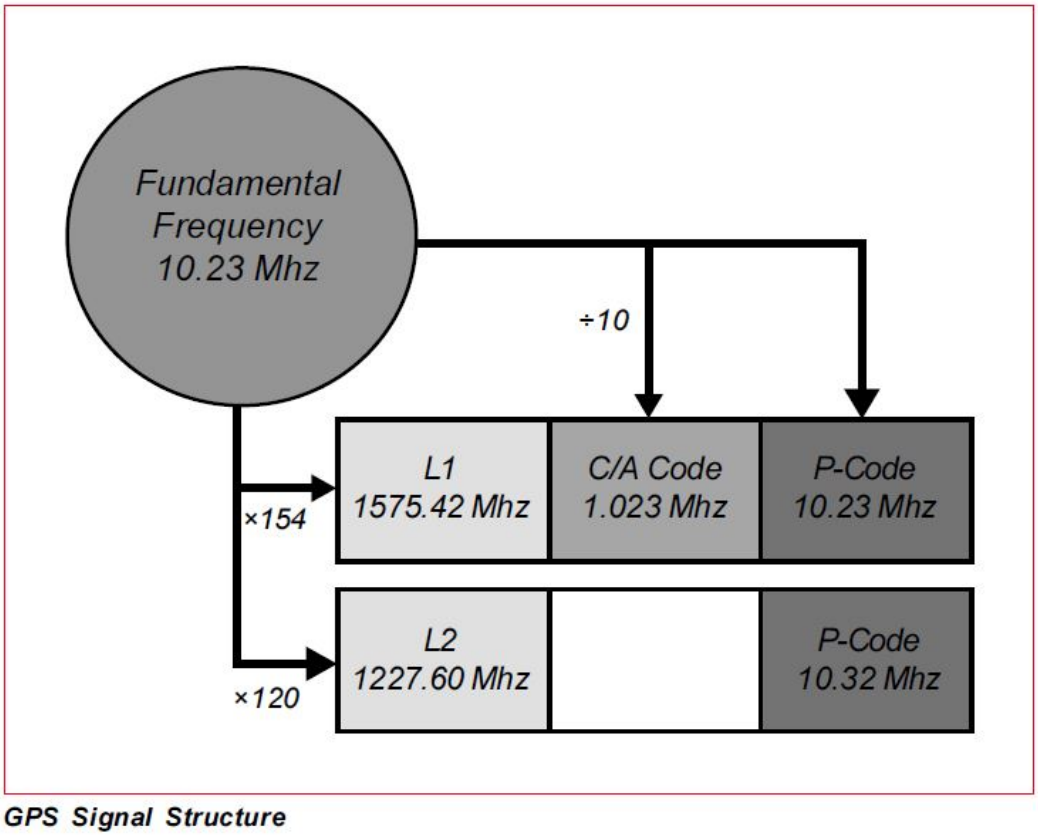


Abbildung 2.2: Struktur des GPS-Signals, Quelle: [GPSb]

gültig sind. Die Verwendung dieser Daten kann die Positionsbestimmung erheblich beschleunigen, da der Download von Ephemeriden ca. eine Minute dauert. Sind gültige Daten vorhanden, so dauert die Positionsbestimmung 5-15 Sekunden [GPSd].
Wenn der GPS-Empfänger während der Messung bewegt wird, so kann sich die Dauer einer Positionsbestimmung ebenfalls verlängern. Besonders bei Flugreisen, bei denen große Distanzen zurückgelegt werden, sind die Ephemeriden am Zielort oft ungültig und der Empfänger benötigt für die erste Positionsbestimmung viel Zeit, da neue Daten heruntergeladen werden müssen.

Um seine Position zu bestimmen, muss der GPS-Empfänger seine Entfernung zu 4 Satelliten bestimmen. Dazu wird ein unverdeckter Blick auf den Himmel und somit auf die Satelliten benötigt. Das ist nötig, weil die Radio-Wellen, die die Satelliten aussenden, sich so ähnlich verhalten wie Licht. Sie durchdringen Glas und Folien, nicht aber Stein.
Für jeden Satelliten ergibt sich eine kugelförmige Oberfläche mit der Entfernung als Radius. Der GPS-Empfänger befindet sich im Schnittpunkt dieser vier Kugel-Oberflächen. Um die Entfernung zu einem Satelliten zu bestimmen, gibt es mehrere Möglichkeiten:
Der *GPS Standard Positioning Service* (SPS) ist frei verfügbar und kann zivil genutzt werden. Die Genauigkeit beträgt ca. +/- 10 m [GPSd].
Der *GPS Precision Positioning Service* (PPS) basiert auf dem verschlüsselten P-Code und kann daher nur von autorisierten Nutzern verwendet werden. Die Präzision dieses Dienstes ist deutlich höher als die des zivilen.
Es gibt auch noch weitere Möglichkeiten, aber diese sind für das Verständnis der Arbeit nicht von Bedeutung und werden daher hier nicht erwähnt.
Im Folgenden wird nur SPS erklärt, da PPS nur autorisierten Nutzern zur Verfügung steht. Um die Entfernung zu einem Satelliten zu bestimmen, wird eines von Newtons Gesetzen der Bewegung verwendet.

$$Weg = Geschwindigkeit * Zeit$$

Die gesuchte Entfernung ist als das Produkt aus der Geschwindigkeit, mit der sich das Radio-Signal des Satelliten ausbreitet, und der Zeit, die es benötigt, um den Empfänger zu erreichen. Die Geschwindigkeit des Radio-Signals ist die Lichtgeschwindigkeit *c*. Die benötigte Zeit wird mit Hilfe des C/A-Codes berechnet. Dazu besitzt ein Empfänger ebenfalls eine Uhr und erzeugt so das gleiche Signal wie der Satellit. Durch die Phasenverschiebung der beiden Signale kann die benötigte Zeit berechnet werden. Die höhere Genauigkeit des PPS kommt zum Teil daher, dass der verwendete P-Code eine höhere Frequenz hat und somit eine präzisere Bestimmung der Zeit zulässt.

Die so berechnete Entfernung wird dann verwendet, um die Position zu berechnen, hierfür werden die exakten Positionen der Satelliten benötigt, also die Ephemeriden.

Wie bereits erwähnt, ist GPS nur bis zu einem gewissen Grad präzise. Dies hat unter anderem folgende Gründe:
Die Lichtgeschwindigkeit ist nur im Vakuum konstant. Ionosphäre und Atmosphäre verlangsamen daher das Radiosignal, was die Berechnung verfälscht.
PPS verwendet sowohl das L1, als auch L2-Signal, um den entstehenden Fehler zu eliminieren. Dies ist möglich, da die Verminderung der Geschwindigkeit von der Frequenz des Signals abhängt.
Sowohl die Uhren der Satelliten als auch die Uhr des Empfängers sind nicht perfekt. Und sie sind auch nicht perfekt synchronisiert. Hier treten auch relativistische Effekte auf, so laufen die Uhren der

Satelliten schneller als die Uhren auf der Erde, wie von Albert Einstein vorhergesagt. Dieser Fehler wird im GPS berücksichtigt und minimiert. Eine weitere Fehlerquelle ist der Mehrwegeempfang. Dabei wird das Signal des Satelliten von glatten Oberflächen in der Nähe des Empfängers reflektiert und erreicht diesen somit auf mehreren Wegen. Dies führt dazu, dass die Präzision von GPS in Häuserschluchten geringer ist [PKG10].

Für die Vermessungstechnik wurden verschiedene Verfahren entwickelt, die es ermöglichen, Positionen auf wenige cm oder mm genau zu bestimmen. Diese Techniken sind allerdings für das Verständnis dieser Arbeit nicht wichtig und werden daher nicht erläutert. Quellen: [GPSc] [GPSa] [GPSd] [GPSb]

2.3 Standortbezogene Dienste

Standortbezogene Dienste, im Englischen als *location based services* (LBS) bezeichnet, sind Dienste, die Informationen bereitstellen, die auf der Position des Nutzers basieren. Sie werden in der Regel mit mobilen Endgeräten genutzt und bieten dann Informationen an, die mit der Position des Nutzers in Bezug stehen.
Es kann grob zwischen reaktiven und proaktiven Diensten unterschieden werden. Reaktive Dienste fragen die Position des Nutzers erst ab, wenn der Nutzer den Service nutzen möchte. Zu dieser Klasse von Diensten gehören z. B. Restaurant-Finder, die dem Nutzer anzeigen, welche Restaurants sich in seiner Nähe befinden.
Proaktive Dienste bestimmen die Position des Nutzers immer wieder neu. Ein Beispiel für einen proaktiven Dienst ist eine Anwendung, die einem Nutzer zeigt, welche von seinen Freunden gerade in seiner Nähe sind. Dafür ist es erforderlich, dass auch die Freunde des Nutzers den gleichen Service nutzen.

Beide Verfahren haben gemeinsam, dass sie die Position des Nutzers benötigen, um ihren Dienst anzubieten. Um die Position eines Nutzers zu bestimmen, können verschiedene Verfahren genutzt werden, die bekanntesten sind GPS und die Bestimmung der Position in zellularen Netzwerken. Bei dem zuletzt genannten Verfahren wird die Tatsche ausgenutzt, dass zellulare Netzwerke protokollieren, in welcher Zelle sich der Nutzer aktuell befindet.
Eine weitere Möglichkeit die Position des Nutzers zu erfahren ist, den Nutzer zu fragen, wo er sich befindet. Oftmals ist der Nutzer in der Lage seine Position auf einer Karte einzutragen. Allerdings wird diese Möglichkeit selten genutzt, da sie eine Interaktion des Nutzers erfordert und daher nur langsam auf sich verändernde Bedingungen reagieren kann.

Weitere Anwendungsbereiche von standortbezogenen Diensten sind:
Nahegelegene Orte, die für den Nutzer von Interesse sind (z. B. Tankstellen, Bankautomaten und Kinos - häufig werden diese mit weiteren Informationen verknüpft, sodass z. B. nach der Tankstelle mit den niedrigsten Preisen gesucht wird.), Stau-Warnungen, Navigation und ortsbezogene Werbung (mit Einkaufsgutscheinen). Ein relativ neuer Trend sind auch standortbasierte Spiele.

Aufgrund der großen Verbreitung von Smartphones mit GPS-Sensoren werden standortbezogene Dienste von immer mehr Menschen genutzt. Doch sowohl die Nutzung von GPS zur Standortbestimmung als auch die Übertragung der Positionsdaten benötigen viel Energie, was die Akkulaufzeiten

verringert und somit der Akzeptanz dieser Dienste schadet. Um den Energiebedarf zu reduzieren, werden die Positionsdaten nicht an jeden Dienst einzeln verschickt, sondern an einen Server, Location-Server genannt, der diese dann den genutzten Diensten zur Verfügung stellt.
Zusätzlich wurden Protokolle entwickelt, die festlegen, wann erneut eine Position gemessen und wann diese versendet werden muss. Diese Update-Protokolle versuchen sowohl die Anzahl der Positionsbestimmung wie auch die Anzahl der Übertragungen zu minimieren. Im Folgenden werden diese Protokolle näher betrachtet.

2.4 Update-Protokolle

Das Sparen von Energie ist bei mobilen Endgeräten äußerst wichtig, da diese nur eine begrenzte Menge an Energie zur Verfügung haben. Durch die starke Verbreitung von Smartphones mit GPS-Sensoren werden standortbasierte Dienste für ein breites Publikum nutzbar und erfreuen sich immer größerer Beliebtheit bei den Nutzern. Die Schattenseite dieser Entwicklung ist der hohe Energiebedarf der GPS-Sensoren und der Energieverbrauch, der beim Senden von Positionsdaten entsteht, was zu einer beträchtlichen Verminderung der Akkulaufzeit führt.

Um den Energiebedarf zu senken, wurden Update-Protokolle entwickelt, die festlegen, wann Positionen bestimmt werden und wann die so gewonnenen Positionsdaten versendet werden. Dabei haben diese Protokolle das Ziel die Anzahl der versendeten Nachrichten und die Anzahl der Positionsbestimmungen so gering wie möglich zu halten, aber dennoch eine bestimmte Genauigkeit der auf dem Location-Server gespeicherten Position zu gewährleisten. Update-Protokolle bestimmen also die Kommunikation zwischen einem mobilen Endgerät und einem Location-Server.
Nach [LR01] können zwei Klassen von Update-Protokollen unterschieden werden.
Zum einen "Querying protocols", bei denen der Location-Server neue Positionsdaten vom mobilen Endgerät anfordert, wenn diese benötigt werden.
Zum anderen "Reporting protocols", bei denen das mobile Endgerät entscheidet, wann eine Position an den Server gesendet wird.

Querying-Protokolle eigenen sich dann, wenn die mobilen Endgeräte wenig Rechenleistung oder wenig Speicherplatz zur Verfügung haben, da alle notwendigen Berechnungen auf dem Server durchgeführt werden und zudem keine Daten auf dem mobilen Endgerät gespeichert werden müssen.
Besonders effektiv sind diese Protokolle, wenn nur sehr selten Positionsupdates nötig sind, da im Gegensatz zu den Reporting-Protokollen keine Daten übertragen werden, wenn es nicht nötig ist.

Reporting-Protokolle erfordern ein komplexeres mobiles Endgerät, da die Berechnungen auf diesem durchgeführt werden. Zudem muss das mobile Endgerät die letzte übertragene Position speichern, um zu wissen, welche Position beim Server gespeichert ist. Ein Vorteil dieses Protokolls ist, dass nur dann Daten übertragen werden, wenn die auf dem Location-Server gespeicherte Position nicht mehr genau genug ist. Wenn sich die Position des mobilen Endgeräts also nur langsam ändert, ist diese Klasse effektiver als die der Querying-Protokolle. Da standortbezogene Dienste meistens von Smartphones aus verwendet werden, sind sowohl Rechenleistung als auch Speicherplatz auf dem mobilen Endgerät verfügbar. Querying-Protokolle eignen sich mehr für den Einsatz in Sensornetzen, in denen die Sensoren in der Regel über eine minimalistische Hardware verfügen.

Im Folgenden werden nur Reporting-Protokolle betrachtet, da diese sich im Bereich der standortbezogenen Dienste durchgesetzt haben.
Nach [LR01] werden vier Klassen von Reporting-Protokollen unterschieden, diese werden nun detailliert betrachtet.
Da Querying-Protokolle für das Verständnis dieser Arbeit unwichtig sind, wird nachfolgend nicht mehr zwischen Querying-Protokollen und Reporting-Protokollen unterschieden und stattdessen der Oberbegriff *Update-Protokoll* verwendet.

2.4.1 Einfache Update-Protokolle

Die einfachste Möglichkeit für ein Update-Protokoll ist es immer dann eine Position an den Location-Server zu senden, wenn eine neue Position vom GPS-Sensor verfügbar ist. Dies sorgt dafür, dass auf dem Location-Server immer die neueste Position verfügbar ist. Allerdings sind heutige GPS-Sensoren in der Lage sekündlich neue Positionen zu bestimmen, was dazu führt, dass sehr viele Updates versendet werden müssen.
Dieser Ansatz ist daher nicht geeignet, um Energie einzusparen und die Akkulaufzeit eines mobilen Endgeräts zu erhöhen.

2.4.2 Zeitbasierte Update-Protokolle

Zeitbasierte Update-Protokolle versenden ihre aktuelle Position periodisch. Dies bedeutet, dass in einem Zeitintervall T eine feste Anzahl von Update-Nachrichten versendet wird - unabhängig davon, ob sich das mobile Endgerät bewegt oder nicht. Die Genauigkeit der auf dem Server gespeicherten Position hängt von der Geschwindigkeit des mobilen Endgeräts und dem gewählten Zeitintervall ab. Ist das Zeitintervall klein gewählt, werden sehr viele Update-Nachrichten versendet und die Position auf dem Server ist relativ genau. Wird das Zeitintervall hingegen groß gewählt, so ist die auf dem Server gespeicherte Position meistens eher ungenau, dafür werden aber weniger Update-Nachrichten versendet und somit wird mehr Energie eingespart. Hier zeigt sich, dass eine Abwägung zwischen Energieeffizienz und Genauigkeit getroffen werden muss. Diese Abwägung muss in allen Update Protokollen getroffen werden und ist in allen Forschungsansätzen, die versuchen Energie zu sparen, zu finden.
Unabhängig davon, wie versucht wird Energie einzusparen. So ist z. B. die Positionsbestimmung mittels GPS teurer als das Abfragen der Position in zellularen Netzwerken, allerdings ist sie auch wesentlich präziser.
Bei diesem Ansatz muss auch die Position periodisch bestimmt werden, damit zu jedem Sendezeitpunkt die aktuelle Position versendet werden kann. Die Kosten für das Versenden von Update-Nachrichten und für Positionsbestimmungen können daher im Voraus berechnet werden.

2.4.3 Distanzbasierte Update-Protokolle

Distanzbasierte Update-Protokolle senden immer dann ein Update, wenn die geographische Entfernung des mobilen Endgeräts zur letzten gesendeten Position einen bestimmten Schwellwert überschreitet. Der Schwellwert definiert also ein kreisförmiges Gebiet um die zuletzt übertragene

Position herum. Ein Update ist immer dann nötig, wenn dieses kreisförmige Gebiet verlassen wird. Wie viele Update-Nachrichten gesendet werden, hängt davon ab, wie schnell und wie häufig sich das mobile Endgerät bewegt.
In der Regel wird eine maximale Geschwindigkeit für das mobile Endgerät angenommen und damit berechnet, wann es erneut nötig ist eine Position zu bestimmen, sodass die Garantie nicht verletzt wird. Zur Berechnung wird dabei die nachfolgende Formel verwendet.

$$T = \frac{Schwellwert - Entfernung\ von\ der\ letzten\ Position}{max.\ Geschwindigkeit}$$

Bei diesem Protokoll ist es im Gegensatz zum zeitbasierten Update-Protokoll nicht möglich im Voraus anzugeben, wie viel Energie benötigt wird.
Dies ist der Fall, da erst nach einer Positionsbestimmung bekannt ist, ob eine Update-Nachricht versendet werden muss. Und auch der zeitliche Abstand von Positions-Messungen kann nicht bestimmt werden, da dieser ebenfalls von der gemessenen Position abhängt.

Um mehr Energie zu sparen, kann auch hier die Genauigkeit der Position reduziert werden, indem statt der maximalen Geschwindigkeit eine durchschnittliche Geschwindigkeit verwendet wird. Das führt dazu, dass gelegentlich der Schwellwert überschritten wird und dies erst später festgestellt wird. Auch hier wird der Trade-off zwischen Genauigkeit und Energiekosten deutlich.

Die große Schwäche dieses Protokolls zeigt sich, wenn ein mobiles Endgerät sich am Rand des durch den Schwellwert definierten Kreises aufhält, diesen aber nicht überschreitet. Es wird kein Update ausgeführt, da der Schwellwert nicht überschritten wird. Allerdings ist dadurch die Zeit bis erneut die Position des mobilen Endgeräts bestimmt werden muss, sehr gering, wodurch in schneller Folge neue Positionsbestimmungen nötig werden. Wie mit diesem Problem umgegangen werden kann, ist z. B. in [FCR07] und [FLR07] beschrieben.
Die zuletzt genannte Arbeit wird später in diesem Kapitel vorgestellt.

2.4.4 Dead reckoning Update-Protokolle

Diese Klasse stellt eine Erweiterung der distanzbasierten Update-Protokolle dar. Der Location-Server versucht die Position des mobilen Endgeräts vorherzusagen. Diese Vorhersage basiert auf dem aktuellen Bewegungszustand (Geschwindigkeit und Richtung) des mobilen Endgeräts. Das mobile Endgerät versucht ebenso seine nächste Position vorherzusagen. Weicht die tatsächlich gemessene Position von der vorhergesagten um einen bestimmten Schwellwert ab, so wird ein Update gesendet. Besonders effektiv ist dieser Ansatz, wenn die Route, auf der sich das mobile Endgerät bewegt, im Voraus bekannt ist. Dies ist z. B. bei einem Lastwagen einer Spedition der Fall.
Es müssen nur Updates gesendet werden, wenn es zu unvorhergesehenen Ereignissen, wie z. B. stockendem Verkehr, Staus oder Streckensperrungen kommt. Positionsbestimmungen werden dann durchgeführt, wenn die Möglichkeit besteht, dass der Schwellwert überschritten wird, also genau wie bei einem distanzbasierten Update-Protokoll.

Neuere Ansätze sind häufig dead reckoning Update-Protokolle, da sich mit diesen am meisten Energie einsparen lässt. So zum Beispiel der Ansatz von Chon et al. [CTSC11], der im Folgenden kurz vorgestellt wird.

2.5 SmartDC

In ihrer Arbeit *Mobility Prediction-based Smartphone Energy Optimization for Everyday Location Monitoring* [CTSC11] stellen Chon et al. das System *SmartDC* vor, das im Folgenden näher beschrieben wird.

Das Ziel von *SmartDC* ist es, für eine begrenzte Menge Energie so genau wie möglich zu bestimmen, wo sich ein Nutzer aufhält. Dafür werden allerdings keine Koordinaten verwendet, sondern sogenannte *meaningful places*, also grobe Angaben, wie z. B. *bei der Abreit* oder *zu Hause*.
Dabei wird ausgenutzt, dass Menschen in ihrem täglichen Leben bestimmten Bewegungsmustern folgen. So besteht der Alltag vieler Erwachsenen daraus, morgens nach dem Aufstehen zur Arbeit zu fahren und dort bis zum Abend zu bleiben. Anschließend fahren sie nach Hause und gehen dann gelegentlich noch in den Sport oder zum Einkaufen. Anhand von Stundenplänen lässt sich also vorhersagen, wann ein Nutzer sich wo aufhält. Dieses Wissen kann für dead rekoning Update-Protokolle genutzt werden, die darauf basieren nur dann Update-Nachrichten zu versenden, wenn die vorhergesagte Position von der gemessenen Position abweicht.

SmartDC läuft auf Smartphones als Hintergrundprozess, die zur Verfügung stehende Energie wird als Prozentsatz der verbleibenden Energie des Akkus festgelegt. Wenn sich ein Nutzer über einen längeren Zeitraum an einer Stelle aufhält, dann erstellt *SmartDC* an dieser Position eine Markierung. Diese Markierung zeigt einen *meaningful place* an und ist mit einer Signatur verknüpft. Die Signatur besteht aus Informationen über die Position, Ankunftszeit, Verweildauer, WLAN-Fingerabdruck und Qualität der Internetverbindung.
Unter einem WLAN-Fingerabdruck versteht man die gesammelten Informationen über MAC-Adressen und Namen (SSID) der verfügbaren WLANs sowie deren Empfangsstärken. Die so erstellten *meaningful places* sind so gut aufgelöst, dass einzelne Räume, in denen sich der Nutzer aufhält, unterschieden werden können.

Um Energie beim Bestimmen der aktuellen Position zu sparen, wird das GPS nur eingesetzt, wenn günstigere Techniken, wie die Bestimmung der Position mittels GSM oder WLAN nicht möglich sind. Dies führt zwar zu ungenaueren Positionsangaben, jedoch reicht die Präzision aus, um zu entscheiden, ob ein Nutzer einen *meaningful place* verlässt.

SmartDC besteht aus 3 Komponenten:

Der *Mobility Learner* hat die Aufgabe Bewegungsprofile zu erstellen. Dazu werden die Möglichkeiten der Positionsbestimmung in drei Level unterteilt. Das erste Level verwendet GSM, um die Position des Nutzers zu bestimmen. Die so gewonnene Postion wird verwendet, um zu prüfen, ob sich der Nutzer auf der vorhergesagten Route bewegt. Nur wenn festgestellt wird, dass sich der Nutzer nicht dort befindet, wo er gemäß der Vorhersage sein soll, wird Level 2 verwendet. Level 2 nutzt WLAN-Signaturen, um festzustellen, ob sich der Nutzer an einem bekannten Ort befindet. Nur wenn der Nutzer sich an einem unbekannten Ort befindet, wird das GPS genutzt.

Der *Mobility Predictor* soll feststellen, wie groß die Verweildauer an einem bestimmten Ort ist und eine Liste von Orten erstellen, die wahrscheinlich als nächstes aufgesucht werden.

Die letzte Komponente ist das *Adaptive Duty Cycling*. Deren Aufgabe besteht darin, die Genauigkeit der beobachteten Positionen zu maximieren. Die Bewegung eines Menschen ist von Natur aus bis zu einem

gewissen Grad zufällig. So gibt es bei der Bewegung eines Menschen immer wieder Abweichungen. Beispielsweise kann es vorkommen, dass ein Arbeiter an einem Arbeitstag nicht zur Arbeit fährt, sondern zu einem Arzt, bei dem er vorher noch nicht war, weil sein Hausarzt im Urlaub ist. Solche Bewegungsabläufe lassen sich nicht vorhersagen, da sie dem Zufall unterliegen.
Um mit diesem Problem umzugehen, wird die Energie, die zur Verfügung steht, in zwei Teile getrennt. Der eine Teil wird für die Vorhersage genutzt, der andere Teil um Anomalien im Bewegungsablauf zu erkennen.
Folgt ein Benutzer 7/10 Mal einem bestimmten Weg, dann wird 3/10 der Energie darauf verwendet Ausnahmen zu finden.

SmartDC reduziert den Energieverbrauch für die Bestimmung des Aufenthaltsortes des Nutzers stark, dafür werden allerdings sehr viele personenbezogene Daten erhoben. Diese Daten werden auf dem Smartphone des Nutzers gespeichert.
Nutzer, die um die Sicherheit ihrer personenbezogenen Daten fürchten, werden schwer davon zu überzeugen sein *SmartDC* zu benutzen.
SmartDC nutzt außerdem nicht die Einsparmöglichkeiten aus, die sich beim Versenden von Update-Nachrichten als Piggyback-Nachrichten ergeben. Daher stellt dieser Ansatz noch keine zufriedenstellende Lösung für die Positionsbestimmung für standortbasierte Dienste dar.

2.6 Early Distance-based Reporting

In ihrer Arbeit *Energy-efficient Tracking of Mobile Objects with Early Distance-based Reporting* stellen Farrell et al. [FLR07] eine Möglichkeit vor, wie distanzbasierte Update-Protokolle erweitert werden können, sodass auch die Kosten für die Positionsbestimmung berücksichtigt werden. Dies wird erreicht, da Updates früher gesendet werden als eigentlich nötig wäre. Diese früher gesendeten Updates werden als *Early-Updates* bezeichnet.
Farrell et al. verwenden ein lineares Energiemodell und betrachten *distance-based reporting* (DBR), also ein distanzbasiertes Update-Protokoll, dass zur Klasse der Reporting-Protokolle gehört.
Ein distanzbasiertes Update-Protokoll sendet immer dann eine Update-Nachricht, wenn die zuletzt gesendete Position von der aktuellen Position um einen bestimmten Schwellwert abweicht. Die Update Bedingung kann auch so gefasst werden, dass der Schwellwert d_{max} nicht überschritten werden darf. Das Update-Protokoll gibt also die Garantie, dass der Abstand zwischen der zuletzt an den Location-Server gesendeten Position und der neu gemessenen Position nie größer als der Schwellwert d_{max} ist.

Dieses Update Protokoll sendet Update-Nachrichten also so spät wie möglich und minimiert dadurch die Anzahl der gesendeten Update-Nachrichten. Farrell et al. zeigen in ihrer Arbeit, dass dieses Vorgehen bei Weitem nicht optimal ist, wenn das GPS zum Bestimmen der Positionen verwendet wird.

Dies liegt daran, dass das mobile Endgerät nach jeder Positionsmessung die Zeit bestimmt, die ohne erneute Positionsmessung verbracht werden kann. Dies ist die Zeit, die das mobile Endgerät mit der maximalen Geschwindigkeit v_{max} benötigt, um den Schwellwert zu erreichen. Also die minimale

Zeit, die das mobile Endgerät benötigt, um die Update-Schwelle zu erreichen. Diese Zeit T_{wait} wird wie folgt berechnet:

$$T_{wait} = \frac{d_{max} - d_r}{v_{max}}$$

Dabei ist d_r der Abstand zwischen der zuletzt gesendeten Position und der aktuellen Position, also die verbleibende Entfernung bis zum Schwellwert.
Die Entscheidung, ob ein Update durchgeführt werden muss, wird anhand der folgenden Formel getroffen:

$$T_{wait} < T_{sense}$$

Dabei ist T_{sense} die Zeit, die benötigt wird, um eine Position zu bestimmen. Für das GPS liegt dieser Wert zwischen 5 und 15 Sekunden, wenn die Bahndaten der Satelliten bekannt sind und nicht erst heruntergeladen werden müssen. Dieser Wert ist also keine Konstante.
Ein Update wird also genau dann ausgeführt, wenn die verbleibende Zeit bis der Schwellwert frühestens erreicht wird, kleiner ist als die Zeit, die benötigt wird, um eine weitere Positionsmessung durchzuführen. Diese Bedingung wird als *Distance-based Update Condition* (DBU-condition) bezeichnet. Falls kein Update erforderlich ist, so wird die Zeit $T_{wait} - T_{sense}$ abgewartet und dann erneut eine Positionsmessung vorgenommen.

Das Problem, das dabei entsteht, ist, dass die Wartezeiten immer kürzer werden, je näher sich das mobile Endgerät an der Update-Schwelle befindet. Bewegt sich das mobile Endgerät also langsam auf den Schwellwert zu oder verweilt nahe dem Schwellwert oder bewegt sich in nahezu konstantem Abstand von dem Schwellwert, so ist die Frequenz der Positionsmessungen sehr hoch. Dies führt zu einem hohen Energieverbrauch durch Positionsbestimmungen, obwohl die Anzahl der Updates minimal ist.
Die Energie, die zwischen zwei Updates verbraucht wird, ist also die Anzahl *n* der Positionsmessungen multipliziert mit der Energie W_s, die für jede Messung benötigt wird. Dadurch ergibt sich die benötigte Energie W für einen Kreislauf aus Positionsmessungen und darauf folgendem Update:

$$W = n * W_s + W_u$$

Die Anzahl der Positionsmessungen pro Kreislauf hat also einen großen Einfluss auf die in diesem Kreislauf benötigte Energie. Die insgesamt benötigte Energie berechnet sich aus der Summe der Energiekosten für jeden Kreislauf.

Der Ansatz von Farrell et al. ist daher der Versuch eine Balance zwischen Updates und Messungen zu erreichen, die zu einem insgesamt minimalen Energieverbrauch führt. Dazu werden zwei Heuristiken vorgestellt, die *Next Fix* (XF) Heuristic und die *Predicted Movement* (PM) Heuristik.

Next Fix:
Dieser Ansatz nutzt die Tatsache aus, dass zu dem Zeitpunkt, an dem entschieden wird, ob ein Update durchgeführt werden soll, sowohl für den Update Fall als auch für den Fall ohne Update, schon bekannt ist, wann die nächste Messung durchgeführt werden muss. Falls kein Update durchgeführt

wird, so wird bis zur nächsten Messung nur die Energie für eine Messung benötigt. Die Energie pro Zeit, also die Leistung, berechnet sich dann nach folgender Formel:

$$P_{no-upd} = \frac{W_s}{T_{wait}} = \frac{v_{max}}{d_{max} - d_r}$$

Falls ein Update durchgeführt wird, entstehen Kosten für das Update und für die nächste Messung. Allerdings vergrößert sich auch die Zeitspanne bis zur nächsten Messung. Diese Zeitspanne ist die maximal mögliche Wartezeit *T*. Die Leistung wird analog zu dem Fall ohne Update berechnet.

$$P_{upd} = \frac{W_U + W_S}{T} = \frac{v_{max}}{d_{max}} * (W_U + W_S)$$

Ein Update sollte also nur dann durchgeführt werden, wenn dadurch das Verhältnis aus Energie und Zeit kleiner ist, als es ohne das Update der Fall ist. Dies führt zur *Energy-based Update Condition* (EBU-condition).

$$d_r > \frac{W_U}{(W_U + W_S)} * d_{max}$$

Diese Gleichung bedeutet, dass ein Update durchgeführt wird, wenn ein bestimmter Prozentsatz des Schwellwertes überschritten ist. Dieser Prozentsatz ist durch $\frac{W_U}{(W_U+W_S)}$ gegeben. Wegen der Bedingung $P_{upd} < P_{no-upd}$ ist bekannt, dass der Energiebedarf während jedes Zyklus, und damit insgesamt, kleiner oder gleich P_{upd} ist. Zudem sind die zusätzlich Berechnungen einfach durchzuführen, so dass sich die Komplexität des Update-Protokolls kaum verändert.

Predicted Movement:
Im Gensatz zur *XF-Heuristik* berücksichtigt die *PM-Heuristik* auch den Energieverbrauch nach der nächsten Positionsbestimmung. Um dies zu erreichen, versucht sie die Bewegungen des mobilen Endgeräts vorherzusagen. Dazu wird nach jeder Positionsmessung aus der zuletzt gemessenen Position und der aktuell gemessenen Position bestimmt, in welche Richtung und mit welcher Geschwindigkeit sich das mobile Endgerät bewegt. Dann wird angenommen, dass sich das mobile Endgerät mit gleichbleibender Geschwindigkeit in die gleiche Richtung weiterbewegt.
Die Berechnung der benötigten Energie ist komplex und für das Verständnis dieser Arbeit nicht erforderlich und wird daher hier nicht weiter behandelt.

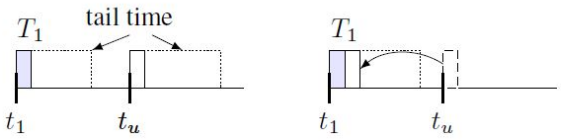
2.7 Opportunistic Position Updates

In ihrer Arbeit *Opportunistic Position Update Protocols for Mobile Devices* [BDR13] weisen Baier et al. darauf hin, dass alle bisherigen Forschungsansätze nicht die Eigenschaften der Netzwerkschnittstelle berücksichtigt haben. Daher haben diese Ansätze alle ein lineares Energiemodell verwendet. Ein lineares Energiemodell nimmt an, dass die Kosten für jedes Positions-Update gleich groß sind.
Neuere Forschungen [HSE07][QWG+10][ZZZL13] zeigen allerdings, dass dies nicht der Fall ist.
Der Grund hierfür liegt darin, dass die Hersteller der Hardware selbst auch versuchen den Energieverbrauch ihrer Produkte zu reduzieren.
Im Detail bedeutet dies, dass die Sendeeinheit nach dem Versenden einer Nachricht nicht direkt

wieder in einen Energiesparmodus zurück geht, sondern noch eine Zeitlang sende-bereit bleibt. Dieser Zeitraum wird *tail time* genannt. Insgesamt verfügt die Sendeeinheit über drei Energiezustände (siehe hierzu Kapitel 1).
Wird eine weitere Nachricht in der *tail time* der vorausgehenden Nachricht gesendet, so reduziert sich die Dauer der Zeit, die insgesamt in der *tail time* verbracht wird. Dadurch reduziert sich die für die beiden Nachrichten benötigte Energie. Daher hängt der Energieverbrauch davon ab, wann zuletzt eine Nachricht über die Netzwerkschnittstelle versendet wurde.

Das Ziel von Baier et al. ist es, die für Positions-Updates benötigte Energie zu minimieren und gleichzeitig die Genauigkeit der auf dem Location-Server gespeicherten Position zu gewährleisten.
So gibt z. B. ein distanzbasiertes Update-Protokoll die Garantie, dass sich das mobile Endgerät innerhalb einer bestimmten Entfernung von der auf dem Location-Server gespeicherten Position aufhält.
Dafür werden Update-Nachrichten in der *tail time* von anderen Nachrichten gesendet. Diese anderen Nachrichten sind Nachrichten, die vom mobilen Endgerät gesendet werden und nicht zum verwendeten Update-Protokoll gehören, sondern von anderen Anwendungen auf dem mobilen Endgerät stammen.
Das Problem dabei ist, dass im Voraus nicht bekannt ist, wann Nachrichten versendet werden.
Wenn eine Nachricht versendet wird, ist es möglich eine Update-Nachricht in deren *tail time* zu senden. So eine früher als nötig gesendete Update-Nachricht wird als *opportunistic update* (opportunistisches Update) bezeichnet. Im Gegensatz dazu sind *forced updates*, Update-Nachrichten, die gesendet werden müssen, um die gegebene Genauigkeit der auf dem Location-Server gespeicherten Position zu gewährleisten. Abbildung 2.3 zeigt jeweils ein Beispiel für ein *forced* Update und ein opportunistisches Update.
Die offene Frage ist, ob mit jeder Nachricht die gesendet wird, opportunistisch eine Update-Nachricht versendet werden soll. Im Extremfall würde man mit jeder Nachricht eine Update-Nachricht senden müssen, was einen großen Overhead bedeutet.

Um opportunistische Updates zu senden, muss das mobile Endgerät genügend Nachrichten von anderen Anwendungen versenden.
Um herauszufinden, wie oft ein mobiles Endgerät Nachrichten versendet, haben Baier et al. im Zuge ihrer Arbeit eine Benutzerstudie durchgeführt, um herauszufinden, wie häufig Nachrichten versendet werden.
Für diese Studie wurde eine Andorid Anwendung programmiert, die jede Millisekunde überprüft, ob Nachrichten empfangen oder versendet wurden. Dafür wurde die *TrafficStats* Klasse der Andorid API verwendet.
Die Zeit, die zwischen zwei aufeinanderfolgenden Übertragungen vergeht, wurde dann von der App auf eine SD-Karte gespeichert.
Die Anwendung wurde während der Studie von 12 Benutzern im Hintergrund ausgeführt, während sie ihr Android Smartphone wie gewohnt weiterverwendet haben. Auf allen Smartphones waren typische Anwendungen, wie web-Browser, E-Mail Clients oder *instant messanger* installiert.
Die so gewonnenen Daten zeigen, dass 89% der gemessenen Zeitintervalle kleiner als eine Sekunde sind. Da es unmöglich ist, opportunistische Updates zu versenden, wenn zu viele sehr große Zweitintervalle vorkommen, wurde zusätzlich die Länge der Zeitintervalle untersucht. Es zeigt sich, dass in 51% der Zeit die Intervalle kleiner als 10 min waren und in 92% der Zeit immer noch kleiner als 40 Minuten.



(a) Forced Update (b) Opportunistic Update

Figure 2: Scenario 1

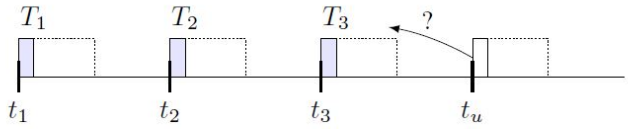


Figure 3: Scenario 2

Abbildung 2.3: Forced Update und opportunistisches Update, Quelle: [BDR13]. Die Abbildung zeigt zwei Szenarien:
In Szenario 1 ist ein Sendeereignis T_1 zu sehen und ein Zeitpunkt t_u, zu dem ein Update ausgeführt werden muss.
In Fall a) wird ein forced Update gesendet.
In Fall b) wird direkt nach dem Versenden der Nachricht T_1 eine Update-Nachricht gesendet. Diese Update-Nachricht ist ein opportunistisches Update, das die Möglichkeit nutzt Energie zu sparen. Dadurch muss zum Zeitpunkt t_u kein Update mehr durchgeführt werden.
Szenario 2 zeigt, dass durch das Versenden eines opportunistischen Updates nach jeder auftretenden Nachricht insgesamt mehr Nachrichten versendet werden, als nötig. Anstatt drei Update-Nachrichten zu senden, würde es reichen ein opportunistisches Update nach der Nachricht T_3 zu versenden.

Um existierende Update-Protokolle um die Funktionalität opportunistischer Updates zu erweitern, haben sich Baier et al. entschieden zusätzlich zu den existierenden Protokollen ein weiteres Protokoll auf auf einem mobilen Endgerät auszuführen. Das zweite Protokoll ist für die opportunistischen Updates zuständig, während das existierende Update-Protokoll die Genauigkeit der auf dem Location-Server gespeicherten Position sicherstellt. Die Schwierigkeit für das zweite Protokoll besteht darin, beim Auftreten eines Sendeereignisses abzuschätzen, ob bis zu dem Zeitpunkt, zu dem ein Update gesendet werden muss, noch weitere Nachrichten gesendet werden. Um dieses Problem zu lösen, verwenden Baier et al. einen Markov-Entscheidungsprozess.

Mit dem so entwickelten Ansatz werden existierende Update-Protokolle getestet. Die Ergebnisse zeigen, dass für zeitbasierte Update-Protokolle im Durchschnitt 70% weniger Energie benötigt wird. Für ein distanzbasiertes Update-Protokoll liegt die Einsparung im Schnitt bei 35%.
Bei dead reckoning Protokollen ist kein Wert angegeben, es wird allerdings darauf hingewiesen, dass diese schon weitgehend optimiert sind und nur wenige Update-Nachrichten versenden. Daher fällt die Einsparung bei diesem Protokoll sehr gering aus.

Eine Schwäche des hier vorgestellten Ansatzes ist, dass nur die Energie für das Senden von Update-Nachrichten reduziert wird. Die Frequenz, mit der Messungen durchgeführt werden, wird nicht optimiert. Hier besteht noch ein großes Einsparpotential, das durch die Verwendung von adaptivem-GPS genutzt werden könnte.

3 System Modell

Das verwendete System Modell besteht aus einem mobilen Endgerät und o. B. d .A. aus nur einem Location-Server. Es ist auch möglich mehrere Location-Server zu verwenden, aber aus Gründen der Komplexität wird in dieser Arbeit nur von einem Location-Server ausgegangen. Das mobile Endgerät sendet seine Position an den Location-Server. Dazu wird ein Netzwerk benötigt. Da das mobile Endgerät mobil ist, wird ein drahtloses Netzwerk verwendet. Nachfolgend werden die einzelnen Komponenten und die an sie gestellten Anforderungen im Detail betrachtet.

3.1 Location-Server

Die Aufgabe des Location-Servers ist es, die Position des mobilen Endgeräts zu speichern und sie allen standortbezogenen Diensten zugänglich zu machen, die der Nutzer des mobilen Endgeräts nutzen möchte.
Um diese Aufgabe zu erfüllen, muss der Location-Server durch ein Netzwerk oder mehrere mit dem mobilen Endgerät verbunden sein. Die Positionsinformationen, die in Form von GPS-Koordinaten vom mobilen Endgerät in Update-Nachrichten gesendet werden, müssen zusammen mit der ebenfalls in der Update-Nachricht enthaltenen Identifikationsnummer des mobilen Endgeräts gespeichert werden. Dies ist nötig, da ein Location-Server auch die Positionen anderer mobiler Endgeräte verwaltet.

3.2 Mobiles Endgerät

In dieser Arbeit wird o. B. d. A. ein Smartphone als mobiles Endgerät genutzt, es sind jedoch auch andere mobile Objekte verwendbar, solange sie die folgenden Eigenschaften erfüllen. Die Gründe für die Wahl eines Smartphones als mobiles Endgerät sind die Verfügbarkeit und Verbreitung von Smartphones.

Um einen ortsabhängigen Dienst nutzen zu können, muss das mobile Endgerät seine Position bestimmen können. In dieser Arbeit wird angenommen, dass das mobile Endgerät hierfür einen GPS-Sensor nutzt. Im allgemeinen Fall kann hierfür aber auch eine andere Technik verwendet werden, z. B. die Bestimmung des Standortes mittels Triangulation von WLAN-Signalstärken. Jedoch liefert ein GPS-Sensor die genauesten Informationen über den Aufenthaltsort des mobilen Endgeräts und ist in einer Vielzahl von Geräten standardmäßig vorhanden. O. B. d. A. wird in dieser Arbeit davon ausgegangen, dass ein GPS-Sensor seine Position exakt bestimmt. Wie mit ungenauen Sensordaten umgegangen werden kann, ist in anderen Arbeiten beschrieben, z. B. in [LWG+09].

Da auf dem mobilen Endgerät ein Algorithmus ausgeführt werden soll, benötigt es einen Prozessor und ausreichend Speicherplatz. Es muss in der Lage sein mathematische Berechnungen durchzuführen und Daten temporär zu speichern.

Um die Positionsdaten an einen Location-Server zu senden, muss das mobile Endgerät über einen Netzwerkzugang verfügen. Für mobile Endgeräte werden zur Übertragung von Daten in der Regel drahtlose Netzwerke verwendet, z. B. das für Mobiltelefone übliche zellulare Netzwerk GSM oder ein WLAN. Diese Arbeit setzt die Verwendung eines zellularen Netzwerks zwingend voraus, da das verwendete Energiemodell nur für zellulare Netzwerke gilt.
Frühere Arbeiten haben oft ein lineares Energiemodell angenommen. Das bedeutet, dass Nachrichten gleicher Größe gleich viel Energie benötigen. Doch neuere Forschungen [HSE07][QWG+10][ZZZL13] haben gezeigt, dass der Energiebedarf nicht-linear ist. Die Kosten für das Übertragen einer Nachricht hängen nicht nur von deren Größe ab, sondern auch davon, wie lange das letzte Sendeereignis her ist. Dieser Zusammenhang ist in Abbildung 1.2 verdeutlicht. In dieser Arbeit wird ein nichtlineares Energiemodell verwendet. Diese Entscheidung ist unersetzlich, da die gesamte Arbeit darauf aufbaut, dass bei einem nicht-linearen Energiemodell Energie gespart werden kann, wenn Nachrichten in Bündeln versendet werden. Der hier vorgestellte Algorithmus reduziert die Kosten für das Senden von Nachrichten dadurch, dass Update-Nachrichten direkt nach anderen Nachrichten versendet werden.

4 Entwurf

Das Ziel dieser Arbeit ist es, einen Algorithmus zu entwickeln, der die Idee von Piggyback-Updates mit existierenden Update-Protokollen kombiniert, um sowohl die Gesamtkosten für Update-Nachrichten als auch die Gesamtkosten für Positionsmessungen zu reduzieren. Piggyback-Updates sind Update-Nachrichten, die in der *tail time* von anderen Nachrichten gesendet werden. Im Gegensatz zu bisherigen Update-Protokollen ist das Ziel nicht länger die Anzahl der Update-Nachrichten, sondern den Energiebedarf zu minimieren.
Die Untersuchungen von Baier et al. [BDR13] weisen darauf hin, dass dies nicht das gleiche ist.

Im Folgenden werden Voruntersuchungen angestellt, um herauszufinden, welche Anforderungen Piggyback-Updates an ein Update-Protokoll stellen und welche Update-Protokolle sich für dieses Vorhaben eignen.

4.1 Vorüberlegungen

Zuerst einmal ist es wichtig zu verstehen, unter welchen Voraussetzungen Piggyback-Updates gesendet werden können. Dieses Wissen wird benötigt, um ein Update-Protokoll auszuwählen, das für die Entwicklung des Algorithmus genutzt werden soll. Dazu werden die Update-Protokolle einzeln betrachtet und anschließend eines ausgewählt.

4.1.1 Bedingungen für ein Piggyback-Update

Die erste Notwendigkeit ist, dass Nachrichten versendet werden, die nicht zum Update Protokoll gehören.
Baier et al. [BDR13] haben bereits gezeigt, dass heutige Smartphones viele Nachrichten versenden, in deren *tail time* Update-Nachrichten versendet werden können. Die erste Voraussetzung ist also erfüllt.
Allerdings zeigen sie auch, dass es immer wieder Zeitabschnitte gibt, in denen keine Nachrichten gesendet werden. Ein Algorithmus, der Piggyback-Updates mit einem Update-Protokoll verbinden will, muss also in der Lage sein, auch ohne Piggyback-Updates Energie einzusparen.

Eine weitere elementare Voraussetzung ist, dass Positionsdaten verfügbar sind, die gesendet werden können.
Diese Positionsdaten müssen vorhanden sein, bevor eine Nachricht gesendet wird, in deren *tail time* eine Update-Nachricht gesendet werden soll. Das liegt daran, dass das Bestimmen einer Position mittels GPS 5-15 Sekunden dauern kann (siehe Kapitel 2.2). Je nach Länge der *tail time*, welche von

Gerät zu Gerät unterschiedlich sein kann, ist dies zu lang, um überhaupt noch in der *tail time* zu senden. Darüber hinaus würde viel Einsparpotential verloren gehen, wenn die Messung der Position nach dem Auftreten einer Nachricht erfolgt, da dann nur ein Bruchteil der *tail time* eingespart werden kann.
Es ist daher unumgänglich, dass die Position, die gesendet werden soll, verfügbar ist, bevor ein Versuch unternommen werden kann ein Piggyback-Update durchzuführen. Diese Position muss neuer sein als die Position, die bereits an den Server gesendet wurde. Es kann zwar die gleiche Position sein, wenn sich der Nutzer nicht bewegt hat, aber sie muss aktueller sein.

Zudem ist es notwendig zu erkennen, wann die Netzwerkschnittstelle Nachrichten versendet, um Piggyback-Updates verschicken zu können. Dazu muss diese überwacht werden. Die für die Überwachung notwendige Energie hängt von der Frequenz ab, mit der abgetastet wird. Die Android API stellt mit der *TrafficStats* Klasse eine Funktion bereit, die es ermöglicht abzufragen, wie viele Bits seit einer bestimmten Zeit versendet wurden. Diese Funktion kann eingesetzt werden, um ein solches Monitoring durchzuführen. Wie groß die Kosten für die Überwachung sind, kann nur die Evaluation eines realen Systems zeigen.

4.1.2 Prüfung der Update-Protokolle auf Eignung

In Kapitel 2 wurden Update-Protokolle vorgestellt, nun werden diese auf ihre Tauglichkeit für Piggyback-Updates hin geprüft. Dazu werden sie einzeln auf ihre Eignung hin untersucht.

Zeitbasierte Update-Protokolle

Zeitbasierte Update Protokolle senden periodisch Update-Nachrichten an einen Location-Server. Wie bereits festgestellt wurde, muss die zu sendende Position ermittelt werden, bevor versucht werden kann ein Piggyback-Update zu senden.
Es wäre also möglich, erst eine Position zu bestimmen und dann zu versuchen sie innerhalb des Zeitintervalls in der *tail time* einer anderen Nachricht zu versenden. Falls dieser Versuch nicht erfolgreich ist, so muss ein *forced* Update gesendet werden. Nach dem Senden einer Update-Nachricht muss dann wieder eine Position gemessen werden.
Der Nachteil dieses Ansatzes ist, dass sich dadurch die Semantik der Update-Nachrichten ändert. Es werden alte Positionen gesendet, auch wenn sich das mobile Objekt mittlerweile an einer ganz anderen Position befindet. Je nach Länge des Zeitintervalls und nach der Geschwindigkeit, mit der sich das mobile Endgerät bewegt, kann diese Position sehr weit von der aktuellen Position abweichen. Ein herkömmliches zeitbasiertes Update-Protokoll (ohne Piggyback-Updates) garantiert, dass die Position auf dem Location-Server maximal so alt ist wie die Länge des Zeitintervalls, nach dem die nächste Position gesendet wird. Diese Garantie kann ein mit Piggyback-Nachrichten erweitertes Update-Protokoll nur dann geben, wenn es häufiger Positionen sendet.

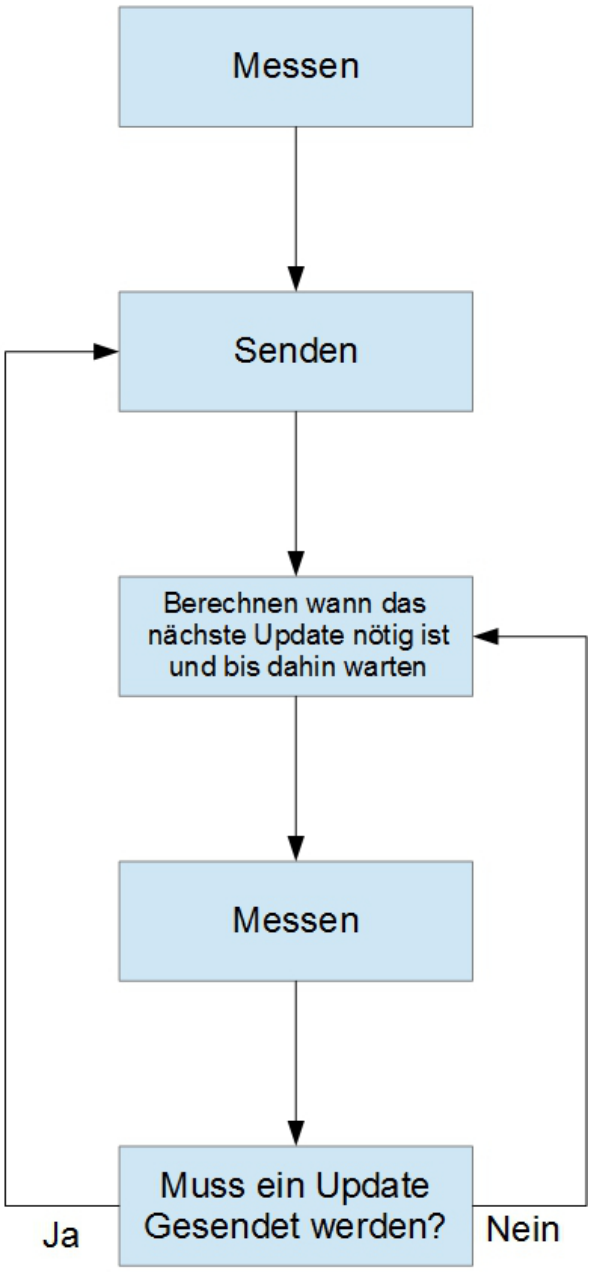


Abbildung 4.1: Flussdiagramm für ein distanzbasiertes Update-Protokoll.

Distanzbasierte Update-Protokolle

Ein distanzbasiertes Update-Protokoll garantiert, dass sich das mobile Endgerät innerhalb eines bestimmten Gebiets, um die zuletzt an den Location-Server gesendete Position herum, aufhält. Dabei ist der Radius dieses kreisförmigen Gebietes der Schwellwert.
Abbildung **??** stellt in einem Flussdiagramm den Ablauf eines distanzbasierten Update-Protokolls dar.
Ein solches Update-Protokoll kann auf zwei Arten mit Piggyback-Updates erweitert werden. Weil für das Senden eines Piggyback-Updates vorher eine Position gemessen werden muss, muss also entweder der Zeitpunkt der Messung verschoben werden oder der Zeitpunkt des Sendens.
Wird der Zeitpunkt des Messens verschoben, so hat dies einen erheblichen Einfluss auf die Energie, die zum Bestimmen einer Position benötigt wird. Wie bereits erwähnt, muss das Protokoll auch damit zurecht kommen, dass es nicht möglich ist ein Piggyback-Update zu senden, falls gerade keine Nachrichten über die Netzwerkschnittstelle versendet werden.
Wird also der Zeitpunkt der Messungen verschoben und es ist nicht möglich ein Piggyback-Update zu senden, so erhöht sich die Frequenz der Messungen, da diese früher als notwendig durchgeführt werden. Dies führt unweigerlich zu einem erhöhten Energieverbrauch, was nicht wünschenswert ist.
Die andere Möglichkeit ist den Sendezeitpunkt zu verschieben. Solange bei der Messung eine Position ermittelt wird, die es nicht erfordert, dass sofort eine Update Nachricht gesendet werden muss, kann ein Early-Update gesendet werden. Die Idee von Early-Updates wurde bereits in Kapitel 2.6 beschrieben.
Early-Updates sind Updates, die gesendet werden, obwohl es noch nicht erforderlich ist ein Update zu senden, um die gegebene Garantie zu gewährleisten. Je näher sich ein mobiles Endgerät am Rand des durch den Schwellwert festgelegten Kreises befindet, desto häufiger muss seine Position bestimmt werden. Durch Early-Updates lässt sich verhindern, dass sich die Frequenz erhöht. Wann genau es sich lohnt ein Early-Update zu senden wurde von Farrell et al. [FLR07] bereits ausführlich untersucht.
Es wäre also möglich Early-Updates als Piggyback-Updates zu senden.

Dead reckoning Update-Protokolle

Da dead rekoning Update-Protokolle im Prinzip nur eine Erweiterung von distanzbasierten Update-Protokollen sind, lässt sich hier die Erweiterung mit Piggyback-Updates auf die gleiche Weise durchführen wie bei distanzbasierten Update-Protokollen.
Die Idee ist zu versuchen Update-Nachrichten früher als nötig zu senden. Falls es nicht möglich ist ein Piggyback-Update zu senden, muss nach wie vor ein forced Update gesendet werden.

Ergebnis

Aufgrund der vorausgegangenen Argumentation kommen nur zwei der drei Update-Protokolle für eine nähere Untersuchung in Frage.
Diese beiden Protokolle haben große Gemeinsamkeiten, jedoch ist das dead reckoning Protokoll komplexer. Ein einfaches Protokoll zu untersuchen und die gewonnenen Ergebnisse auf das komplexere zu übertragen, erscheint dem Autor sinnvoller als das umgekehrte Vorgehen.
Daher fällt die Wahl auf das distanzbasierte Update-Protokoll. Dieses wird also im Folgenden um die

Funktionalität erweitert, Piggyback-Updates als Early-Updates zu versenden. Dadurch soll ein Algorithmus entstehen, der sowohl die Update-Kosten als auch die Kosten für das Messen von Positionen reduziert.

4.2 Piggyback-Update Algorithmus

Die Voruntersuchungen haben dazu geführt, dass die Entscheidung gefallen ist ein distanzbasiertes Update-Protokoll um die Funktionalität der Piggyback-Updates zu erweitern. Ein Blick auf verwandte Arbeiten zeigt, dass Farrell et al. [FLR07] bereits Untersuchungen darüber angestellt haben, wann es sich lohnt Updates früher zu senden als erforderlich. Da Farrell et al. ein lineares Energiemodell verwenden, sind die dort gewonnenen Erkenntnisse nicht direkt verwendbar, da die vorliegende Arbeit auf einem nichtlinearen Energiemodell basiert. Allerdings kann die Arbeit von Farrell et al. zum Vergleich für den in dieser Arbeit entstehenden Algorithmus herangezogen werden.

4.2.1 Der Algorithmus

Wie bereits mehrfach erwähnt, wird das distanzbasierte Update-Protokoll so erweitert, dass es versucht Early-Updates in der *tail time* anderer Nachrichten zu versenden, also als Piggyback-Updates. Abbildung 4.2 zeigt ein Flussdiagramm des so entstandenen Protokolls. Nachfolgend wird dieses Flussdiagramm als Pseudocode dargestellt:

procedure UPDATEPROTOCOL( )
  P ← GETPOSITION( )
  SENDUPDATE(P)
  $P_{transmitted} = P$

  $timeUntilNextMeasurement = \frac{guaranty}{v_{max}} - sensingTime$

  while TRUE **do**
    WAIT(timeUntilNextMeasurement)
    P ← GETPOSITION( )
    if UPDATEDECIDER(P) **then**
      SENDUPDATE(P) // forced update
      $P_{transmitted} = P$

      $timeUntilNextMeasurement = \frac{guaranty - dist(P_{transmitted}, P)}{v_{max}} - sensingTime$

    else if PIGGYBACKUPDATEDECIDER(P) **then**
      TRYTOSENDPIGGYBACKUPDATE(P)
    end if
  end while
end procedure

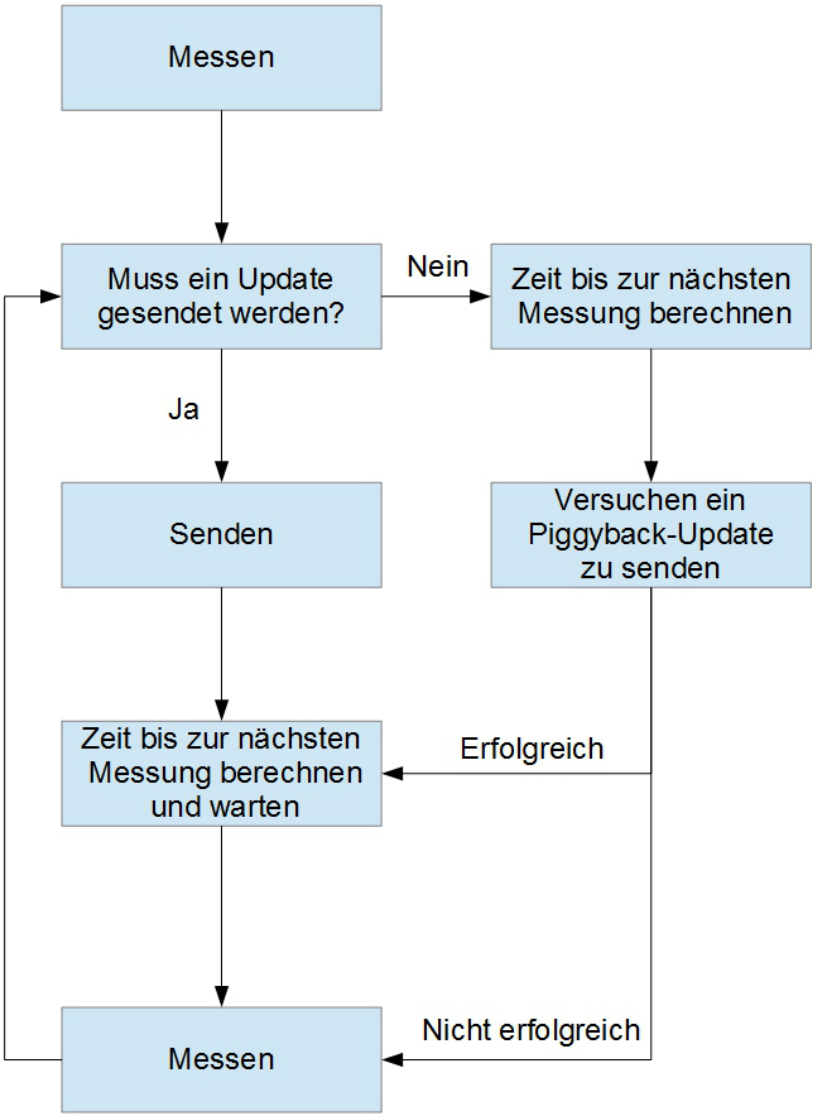


Abbildung 4.2: Flussdiagramm für den Piggyback-Algorithmus.

Wie im Pseudocode beschrieben, wird zuerst eine Positionsmessung durchgeführt und die gemessene Position an den Location-Server gesendet. Dann wird berechnet, wann eine neue Messung erforderlich ist. Anschließend beginnt in einer While-Schleife die Update-Routine. Zuerst wird gewartet, bis die Messung durchgeführt werden muss, dann wird sie durchgeführt. Wenn die gemessene Position verfügbar ist, so muss geprüft werden, ob ein Update notwendig ist, falls ja, wird ein Update ausgeführt. Falls kein Update notwendig ist, so wird geprüft, ob ein Piggyback-Update gesendet werden soll. Welche Bedingung dafür erfüllt sein muss, wird später erläutert. Soll ein Update gesendet werden, so wird dies versucht. Die Einzelheiten dieses Vorgangs werden ebenfalls später betrachtet. Zuletzt wird wieder bestimmt, wann das nächste Update durchgeführt werden muss.

Nun müssen zwei Dinge betrachtet werden:
Wann soll ein Piggyback-Update durchgeführt werden?
Wie genau wird versucht ein Piggyback-Update zu senden?

Die letztgenannte Frage ist einfacher zu beantworten, daher wird sie zuerst betrachtet.

Ein Piggyback-Update soll in der *tail time* einer anderen Nachricht versendet werden, daher ist es nötig die Netzwerkschnittstelle zu überwachen. Dazu wird periodisch geprüft, ob Nachrichten versendet worden sind. In Android kann hierzu die Klasse *TrafficStats* verwendet werden.
Wird festgestellt, dass eine Nachricht versendet wurde, so wird direkt danach ein Positions-Update gesendet. Falls keine Nachricht versendet wird, entspricht die Dauer des Monitorings der Zeit bis zur nächsten geplanten Messung. Falls es nicht möglich ist in dieser Zeit ein Update zu senden, wird die While-Schleife erneut betreten und der Algorithmus startet wieder mit einer Messung.

Wann soll ein Piggyback-Update durchgeführt werden?

Da Piggyback-Updates Early-Updates sind, ist die Antwort auf diese Frage die gleiche, die für Early-Updates bereits von Farrell et al. gefunden wurde: Wenn es sich im Bezug auf den Energieverbrauch lohnt.

Daraus ergibt sich die nächste Frage: Wann lohnt es sich?

Ein Blick in die Arbeit von Farrell et al. zeigt, dass sich diese Frage nur beantworten lässt, wenn weitere Annahmen getroffen werden. Das Problem hierbei ist, dass im Voraus nicht bekannt ist, welche Position als nächstes gemessen wird. Es könnte ja sein, dass sich das mobile Endgerät nach dem Update wieder zur alten Position zurückkehrt. Abbildung 4.3 zeigt diesen Fall. Es würde sich also nicht lohnen ein Update durchzuführen. Andererseits könnte es aber auch sein, dass sich das mobile Endgerät ein Stück weiter von der zuletzt gesendeten Position wegbewegt, aber nur so weit, dass noch kein forced Update durchgeführt werden muss. Das führt dazu, dass sich die Zeit bis zur nächsten Messung reduziert, wodurch sich die Frequenz der Messungen erhöht, wodurch sich wiederum der Energieverbrauch für die Positionsbestimmung erhöht. Ohne weitere Annahmen zu treffen kommt man hier nicht weiter.
Farrell et al. stellen zwei mögliche Annahmen vor, zum einen die Next-Fix-Heuristik und zum anderen die Predicted-Movement-Heuristik.

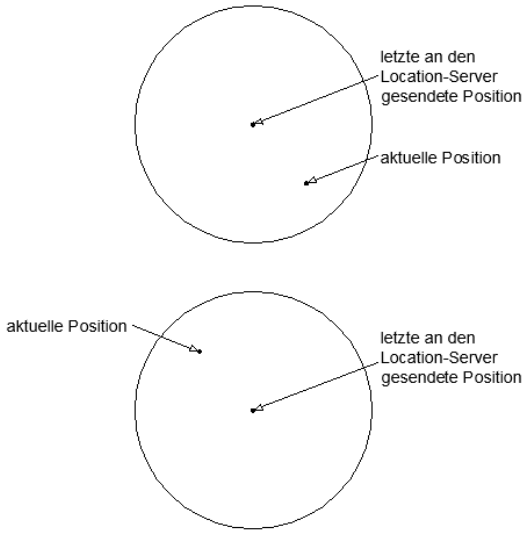


Abbildung 4.3: Beispiel für ein unnötiges Update:
Die Abbildung zeigt oben die Ausgangssituation. Da nicht bekannt ist wohin sich der Nutzer bewegt, wird ein Update durchgeführt. Wie sich im Unteren Bild zeigt war dieses Update unnötig, da der Nutzer sich wieder zu seinem alten Standort begeben hat.

Die Next-Fix-Heuristik betrachtet nur, was im nächsten Schritt passiert, die Predicted-Movement-Heuristik versucht vorherzusagen, welche Positionen die nächsten Messungen ergeben. Da die Next-Fix-Heuristik einfacher ist und die Ergebnisse im Bezug auf den Energieverbrauch nicht wesentlich schlechter sind, wird diese für die folgenden Betrachtungen verwendet.
Farrell et al. bestimmen die Energiekosten für den Fall, dass ein Update ausgeführt wird und für den Fall, dass kein Update ausgeführt wird. Dann vergleichen sie die Kosten. Sind diese für den Fall ohne Update größer, so wird ein Update ausgeführt. Für die Berechnungen wird die folgende Formel verwendet (Für eine Herleitung der Formel siehe Kapitel 2.6).

$$d_r > \frac{W_U}{(W_U + W_S)} * d_{max}$$

Allerdings betrachten Farrell et al. ein lineares Energiemodell, daher sind die Ergebnisse nicht direkt übertragbar. Wenn man diese Formel verwendet und einfach nur die Kosten für ein Update auf die reduzierten Kosten für ein Piggyback-Update setzt, so berücksichtigt man nicht die Kosten, die entstehen, wenn ein forced Update durchgeführt werden muss. Die (zusätzlichen) Kosten für ein Piggyback-Update, das unmittelbar nach einer anderen Nachricht gesendet wird, sind lediglich die Kosten für die Dauer der Übertragung der Update-Nachricht. Es fallen keine Kosten für die *tail time* an.
Bei der Betrachtung der Formel fällt auf, dass die Update-Schwelle angibt, wie viel Prozent der Garantie die Entfernung zwischen der aktuellen Position und der zuletzt an den Location-Server gesendeten Position beträgt.

Welcher Prozentsatz dies für ein nichtlineares Energiemodell mit Piggyback-Updates ist, ist unbekannt. Dieser Prozentwert soll in einer Simulation mit realen Daten ermittelt werden. Je nachdem, wie viel es kostet, ein Piggyback-Update zu senden, ist zu erwarten, dass dieser prozentuale Wert sehr niedrig ausfällt. Um einen ersten Eindruck davon zu erhalten, in welchen Bereich sich dieser Wert befindet, benötigt man ein Energiemodell.

In der Arbeit *EnTracked: Energy-Efficient Robust Position Tracking for Mobile Devices* [KLGT09] ist ein nichtlineares Energiemodell enthalten, das zudem gut untersucht ist. Dieses Energiemodell basiert auf einem Nokia N95 Mobiltelefon.
An dieser Stelle muss darauf hingewiesen werden, dass dieses Energiemodell nur für Nokia N95 Mobiltelefone gültig ist. Andere mobile Endgeräte können einen anderen Energiebedarf und eine andere *tail time* aufweisen.
Die in *EnTracked* verwendeten Werte werden im Kapitel 5 detailliert betrachtet, da sie auch für die Simulation verwendet werden. Hier soll lediglich ein erster Eindruck von der Größe des prozentualen Wertes gewonnen werden.
Der Energiebedarf für die *tail time* berechnet sich wie folgt:

$$TailEnergy = Radio_{active} * Power - off(Radio_{active}) + Radio_{idle} * Power - off(Radio_{idle})$$
$$TailEnergy = 645mW * 5,45s + 466mW * 31,3s = 18101,05mJ$$

Die in der *tail time* benötigte Energie beträgt also ca. 18 Joule. Dem gegenüber steht die Energie, die benötigt wird, um eine Update-Nachricht zu übertragen. Die dafür benötigte Zeit berechnet sich aus der Größe der Update-Nachricht und der Uplink-Geschwindigkeit. Hier werden die gleichen Werte verwendet, die auch von Baier et al. für ihre Arbeit [BDR13] verwendet wurden. Dies sind 384 kB/s für die Uplink-Geschwindigkeit und eine Dateigröße von 0,2 kB.
Daraus ergibt sich der Energiebedarf für das Übertragen einer Update-Nachricht wie folgt:

$$\ddot{U}bertragungdsdauer = \frac{0,2kB}{384kB/s} = 0,00052s$$

$$\begin{aligned}\ddot{U}bertragungsenergie &= \ddot{U}bertragungsdauer * Radio_{active} \\ &= 0,00052s * 645mW = 0,3354mJ\end{aligned}$$

Dieses Ergebnis zeigt, dass das Senden von Piggyback-Updates nahezu kostenlos ist. Daher ist zu erwarten, dass das Resultat der Simulation sein wird, dass es sich immer lohnt zu versuchen ein Piggyback-Update zu senden. Und zwar unabhängig von den Energiekosten für eine Positionsbestimmung.

Der Vollständigkeit halber wird hier noch die Rechnung für die Kosten einer Positionsbestimmung mit dem GPS-Sensor gezeigt.

$$\begin{aligned}GPS - Energie &= Energiekosten * (GPS_{power-off} + sensingTime) \\ &= 324mW * (30\ s + 10\ s) = 12960mJ\end{aligned}$$

Die *sensing time* ist hierbei die Zeit, die der GPS-Sensor benötigt, um eine Position zu bestimmen. Diese Zeit wurde für diese Rechnung auf 10 s festgesetzt. Die Energiekosten für eine Positionsbestimmung sind also etwas niedriger als die für ein Update ($18101,05mJ + 0,3354mJ$). Verglichen mit den Energiekosten für ein Piggyback-Update sind sie allerdings riesig (ca. 38.640 mal höher).

4.2.2 Analyse des Algorithmus

In diesem Abschnitt soll der entstandene Algorithmus analysiert werden um festzustellen, ob er den gestellten Anforderungen genügt. Falls Schwachstellen gefunden werden, so wird versucht diese zu beseitigen.

Energiekosten

Der Algorithmus versucht die Energiekosten zu reduzieren, die beim Senden der Update-Nachrichten entstehen. Dabei werden nicht die Kosten eines einzelnen Updates reduziert, sondern die Gesamtkosten aller Updates. Nach jeder Positionsmessung wird geprüft, ob eine Update-Nachricht versendet werden muss, um die gegebene Garantie zu gewährleisten.
Wenn nach einer Messung kein forced Update gesendet werden muss, versucht der hier beschriebene Algorithmus, abhängig von einem Schwellwert, ein Piggyback-Update zu versenden.
Eine erste Berechnung deutet darauf hin, dass die besten Ergebnisse zu erwarten sind, wenn der Schwellwert nahe bei 0% liegt, also immer versucht wird ein Piggyback-Update zu versenden.
Wie häufig es tatsächlich möglich ist ein Piggyback-Update zu senden, muss die Simulation zeigen.
Wenn es nicht möglich ist Piggyback-Updates zu versenden, verhält sich der Algorithmus genauso wie ein distanzbasiertes Update-Protokoll. Dies hat natürlich den Nachteil, dass die Energiekosten steigen, wenn sich das mobile Endgerät nahe an der Update-Schwelle aufhält.
Da der Algorithmus aber auch Energie einsparen soll, wenn vom mobilen Endgerät keine anderen Nachrichten versendet werden, die für Piggyback-Updates genutzt werden können, ist es nötig diesen zu erweitern.

Erweiterung mit Early-Updates

Eine mögliche Erweiterung ist die Erweiterung um Early-Updates.
Hier ergibt sich wieder die Fragestellung, wann diese gesendet werden sollen. Da diese Frage mit einem nichtlinearen Energiemodell nicht ohne weiteres zu beantworten ist, soll ein geeigneter Wert oder eine Formel im Simulator ermittelt werden. Zu beachten ist, dass die zusätzlichen Early-Updates keine Piggyback-Updates sind und daher die vollen Energiekosten haben. Falls keine Piggyback-Updates möglich sind, verhält sich der Algorithmus mit dieser Modifikation nun genauso wie der Algorithmus von Farrell et al.

Qualität der Position auf dem Location-Server

Eine wichtige Eigenschaft von distanzbasierten Update-Protokollen ist es, dass sie eine bestimmte Qualität der auf dem Location-Server gespeicherten Position garantieren. Diese Garantie ist, dass das mobile Endgerät nie weiter von der auf dem Server gespeicherten Position entfernt ist als um einen gegebenen Schwellwert. Diese Garantie bleibt auch in dem hier vorgestellten Algorithmus bestehen und wird durch die forced Updates gewährleistet. Im Allgemeinen wird die Qualität der auf dem Server gespeicherten Position sogar verbessert, da Early-Updates gesendet werden. Diese Early-Updates werden wenn möglich als Piggyback-Update versendet.

Zusätzliche Beobachtung

Ein weiterer Aspekt ist, dass eine Update-Nachricht auch dann Energie einspart, wenn nach ihrem Versenden eine weitere Nachricht gesendet wird. Dieser Aspekt kann hier allerdings nicht berücksichtigt werden, da nicht im Voraus bekannt ist, ob nach dem Senden einer Update-Nachricht eine weitere Nachricht, die nicht zum Update-Protokoll gehört, versendet wird. Im Simulator hingegen kann dies berücksichtigt werden und der Vollständigkeit halber wird es das auch.

5 Implementierung

Um den in Kapitel 4 beschriebenen Algorithmus zu testen, wurde sowohl ein Simulator implementiert als auch eine Android Applikation entwickelt.

Der Simulator wird dazu genutzt, den Algorithmus an realen Datensätzen zu testen. Als Eingabe erhält er jeweils zwei Dateien.
Zum einen eine Route, die durch eine Abfolge von GPS-Koordinaten gegeben ist, diese wird auch *Position-Trace-File* genannt. Diese Dateien stammen von der Website `http://www.openstreetmap.org/traces` und sind frei verfügbar.
Zum anderen eine Datei, die angibt, zu welchen Zeitpunkten Nachrichten von anderen auf dem mobilen Endgerät ausgeführten Anwendungen versendet werden. Diese Nachrichten gehören nicht zum untersuchten Algorithmus, sondern werden von diesem genutzt, um Update-Nachrichten als Piggyback-Updates zu versenden.
Im Folgenden wird diese Datei *Traffic-Trace-File* genannt.
Für die verschiedenen Simulationen werden insgesamt drei Position-Trace-Files und drei Traffic-Trace-Files verwendet. Diese Traffic-Trace-Files stammen aus der in Kapitel 2.7 vorgestellten Arbeit von Baier et al. [BDR13]. Sie enthält Daten, die im Zuge dieser Arbeit erhoben wurden.
Um die Daten zu gewinnen, wurde vom Autor eine Benutzerstudie durchgeführt, bei der auf den Smartphones der Studienteilnehmer eine App installiert wurde. Diese App prüft jede Millisekunde, ob die Anzahl der versendeten Bytes im Vergleich zur vorherigen Messung gestiegen ist. Ist dies der Fall, so wird eine Zahl in eine Datei geschrieben, die die seit dem letzten Senden vergangene Zeit beschreibt.
Die verwendeten Traffic-Trace-Files sind so modifiziert worden, dass nur Sendeereignisse, die einen Abstand von mindestens einer Sekunde zum vorherigen Sendeereignis haben, enthalten sind.

Die Android Anwendung wird verwendet, um einen Eindruck davon zu erhalten, wie gut sich der Algorithmus in einem realen System implementieren lässt und wie viel Energie für die Überwachung der Netzwerkschnittstelle benötigt wird. Außerdem wird sie dazu genutzt, um festzustellen, wie viel Energie ein Piggyback-Update im Vergleich zu einem normalen Update einspart.

Die Implementierungen dieser Anwendungen werden im Folgenden genauer vorgestellt.

5.1 Simulator

Der Simulator ist in Java implementiert. Das Strukturdiagramm in Abbildung 5.1 zeigt die wichtigsten Klassen, die im Simulator vorhanden sind, diese werden im Folgenden vorgestellt und ihr Zusammenhang erläutert. Der Simulator basiert auf Events, diese werden nach dem Zeitpunkt ihres Auftretens

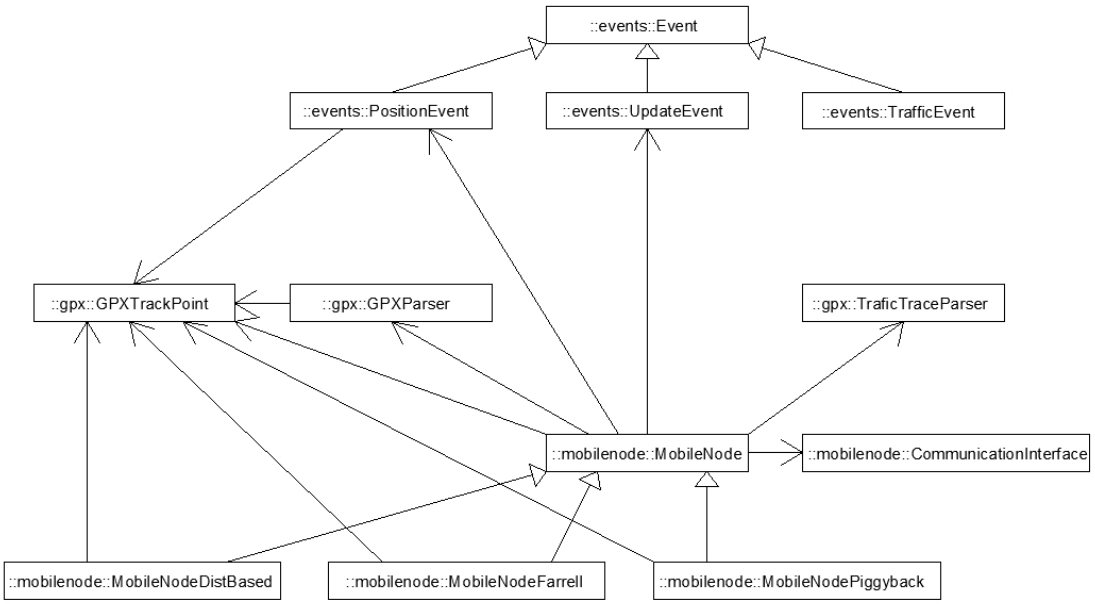


Abbildung 5.1: Das Strukturdiagramm zeigt die wichtigsten Klassen, des Simulators.

geordnet.
Es gibt drei Arten von Events: Position-Events, Traffic-Events und Update-Events.

Position-Events liefern das Ergebnis einer GPS-Messung, also eine GPS-Koordinate mit Längen- und Breitengrad sowie einer Angabe für Zeit und Datum, die zeigt, wann die Messung durchgeführt wurde. Diese Daten werden in sogenannten *Trackpoints* zusammengefasst. Die Klasse *GPXTrackPoint* bietet die hierfür erforderliche Datenstruktur an. Die GPS-Koordinaten stammen aus der Position-Trace-File. Die Daten der Position-Trace-File werden von der Klasse *GPXParser* eingelesen.

Traffic-Events geben Zeitpunkte an, zu denen andere Anwendungen auf dem mobilen Endgerät, auf dem auch der zu testende Algorithmus läuft, Nachrichten versenden. Diese Zeitpunkte stammen aus der Traffic-Trace-File und werden mit der Klasse *TrafficTraceParser* eingelesen.

Update-Events treten immer dann auf, wenn Positionsupdates an den Location-Server versendet werden.

Die Klasse *MobileNode* ist die Basisklasse für alle Update-Protokolle. Jedes Update-Protokoll, das mit dem Simulator getestet werden soll, erweitert diese Basisklasse. Sie bietet für jedes Event eine Funktion an, die aufgerufen wird, wenn das Event auftritt.
Für ein Position-Event wird die Funktion *onPositionEvent()* aufgerufen, für ein Traffic-Event die Funktion *onTrafficEvent()* und für ein Update-Event die Funktion *onUpdateEvent()*. Jeder dieser Funktionen wird das entsprechende Event, und damit die mit dem Event verbundenen Daten, als Parameter übergeben.

Feature	Energiebedarf in [Watt]
GPS	0,324
DCH	0,645
FACH	0,466

Tabelle 5.1: Die Tabelle zeigt die Energiekosten der einzelnen Features.

Das Zentrum des Simulators bildet die Klasse *Simulator*, die die Events in einer *java.util.PriorityQueue*, nach dem Zeitpunkt ihres Auftretens geordnet, speichert. Der Simulator nimmt jeweils das zeitlich nächste Event aus dieser Queue und ruft für den assoziierten MobileNode die Funktion *onEvent()* auf, die wiederum für jedes Event die spezifische Funktion aufruft.

Die Klasse *CommunicationInterface* bietet die nötigen Funktionen, um die Energiekosten berechnen zu können. Dies sind sowohl die Kosten für das Messen der Positionen als auch die Kosten für das Versenden von Update-Nachrichten.

Im Simulator werden drei Update-Protokolle getestet.
Die Klasse *MobileNodeDistBased* enthält den im Kapitel 2.1 beschriebenen Standardalgorithmus für ein distanzbasiertes Update-Protokoll.
Das in Kapitel 2.3 beschriebene Update-Protokoll von Farrell et al.[FLR07] ist in der Klasse *MobileNodeFarrell* implementiert.
Der in Kapitel 4 beschriebene Algorithmus ist in der Klasse *MobileNodePiggyback* implementiert.

Im Folgenden werden ausgewählte Klassen, die zum Verständnis der Evaluation benötigt werden, detaillierter betrachtet. Dies sind die Klasse *CommunicationInterface*, in der die Berechnung der Energiekosten durchführt wird, sowie die drei Update-Protokolle.

5.1.1 CommunicationInterface

Die Klasse *CommunicationInterface* berechnet die Energiekosten, die der im Simulator getestete Algorithmus verursacht.
Es wird zwischen zwei Arten von Kosten unterschieden, zum einen den Kosten, die durch die Nutzung des GPS-Sensors entstehen, zum anderen den Kosten, die durch das Versenden von Update-Nachrichten über die Netzwerkschnittstelle entstehen. Für die Berechnung dieser Kosten wird ein nichtlineares Energiemodell verwendet. Die Werte für dieses Modell stammen aus der Arbeit *Entracked* [KLGT09]. Sie beziehen sich auf ein Nokia N95 und sind nur für ein zellulares Netzwerk gültig. Die verwendeten Werte sind in Tabelle 5.1 und Tabelle 5.2 aufgeführt. Im Folgenden wird die Berechnung dieser Energiekosten im Detail beschrieben.

Feature	Verzögerung in Sekunden
GPS	30,0
DCH	5,45
FACH	31,3

Tabelle 5.2: Die Tabelle zeigt die Verzögerungen der einzelnen Features.

Kosten für GPS-Messungen

Ein mobiles Endgerät mit GPS-Sensor kann diesen nutzen, um seine aktuelle Position abzufragen. Die Nutzung des GPS-Sensors verursacht Kosten in Form von Energie.
Die folgende Formel wird verwendet, um die Energiekosten für eine GPS-Messung zu berechnen.

$$positionEnergy = sensingEnergy * GPS_{powerOff} + sensingEnergy * sensingTime$$

Diese Formel berechnet die Kosten für eine einzelne Positionsmessung. Hierbei ist die *sensingEnergy* die Leistung, die der GPS-Sensor aufnimmt. $GPS_{powerOff}$ ist die Zeit, die der GPS-Sensor nach einer Messung noch betriebsbereit bleibt, bevor er wieder in den Ruhezustand wechselt. Diese Zeitspanne wird im Folgenden auch *Idle-Zeit* genannt. Die *sensingTime* ist die für die Positionsmessung benötigte Zeit.
Wenn eine weitere Messung während der Idle-Zeit der vorangegangenen Messung durchgeführt wird, so reduzieren sich die Kosten für die vorangegangene Messung. Die Kosten sind reduziert, da sich die Idle-Zeit der ersten Messung verkürzt. Die reduzierten Kosten werden mit der nachfolgenden Formel berechnet.

$$positionEnergy = sensingEnergy * (GPS_{powerOff} - timeSinceLastMeasurement) + sensingEnergy * sensingTime$$

Daraus ergibt sich das Problem, dass die Kosten für eine GPS-Messung erst nach dem Verstreichen der Idle-Zeit bekannt sind. Dieses Problem lässt sich durch einen Trick umgehen.
Man kann für eine GPS-Messung zwar nicht unmittelbar die Kosten dieser Messung berechnen, wohl aber die Kosten der vorangegangenen Messung.
Für die erste Messung und für jede Messung, die auf eine Messung folgt, deren Idle-Zeit komplett verstrichen ist, werden die vollen Kosten berechnet. Für alle anderen Messungen werden die Kosten für die vorangegangene Messung berechnet.
Werden *n* Messungen durchgeführt, wobei die Messungen 2 bis *n* jeweils in der Idle-Zeit der vorangehenden Messung durchgeführt werden, so werden für die erste Messung die vollen Kosten berechnet und für alle weiteren Messungen jeweils die Kosten der vorangegangenen Messung. Für die erste Messung werden daher die Kosten für die Messung *n*, also für die letzte Messung, berechnet. Dadurch ergibt sich eine Kette, in der die erste Messung die Kosten der letzten übernimmt, was die Kette schließt und einen Kreis ergibt. Abbildung 5.2 verdeutlicht das Vorgehen bei der Berechnung der Energiekosten. Die Korrektheit dieses Vorgehens liegt darin begründet, dass die Kosten für die ersten *n-1* Messungen jeweils korrekt berechnet werden können. Für die *i-te* Messung gilt, dass ihre Kosten entweder die vollen Kosten sind und von der ersten Messung berechnet wurden oder dass die nachfolgende Messung *i+1* ihre Kosten korrekt berechnet.

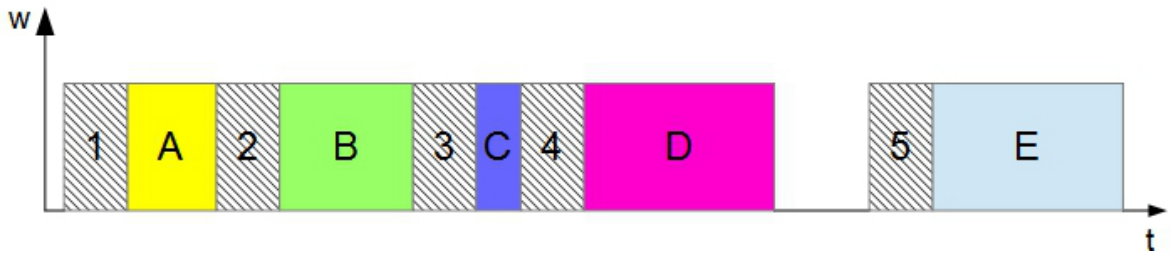


Abbildung 5.2: Kette zur Berechnung der GPS-Kosten:
Jede Messung berechnet die Kosten für die vorangegangene Messung, mit Ausnahme der ersten Messung, diese berechnet die Kosten für das letzte Glied der Kette.
Die Kosten werden daher wie folgt berechnet:
Messung 1 berechnet die Kosten für Messung 4, diese sind durch die Fläche D dargestellt. Die Messungen 2 bis 4 berechnen jeweils die Kosten der vorangegangenen Messungen. D. h. Messung 2 berechnet die Kosten für Messung 1 - also die Fläche A, Messung 3 berechnet B und Messung 4 C.
Die Messung 5 hat keinen Vorgänger, in deren Idle-Zeit sie gesendet wird und auch keinen Nachfolger. Sie ist aber das erste Glied einer (eingliedrigen) Kette und berechnet daher ihre eigenen Kosten E.

Kosten für Positions-Updates

Bei der Berechnung der Kosten für ein Positions-Update ergibt sich das gleiche Problem wie bei der Berechnung der Kosten einer GPS-Messung. Die Kosten können erst beim Auftreten eines weiteren Sendeereignisses oder nach dem Verstreichen der *tail time* berechnet werden. Es kann zwar der gleiche Trick wie bei der Berechnung der Energiekosten für GPS-Messungen verwendet werden, allerdings müssen hier vier Fälle unterschieden werden, da es nicht nur Update-Nachrichten gibt, sondern auch andere Nachrichten, die vom Smartphone versendet werden.
Für die Erklärung der vier Fälle werden die Event-Typen Update-Event, für Update-Nachrichten, und Traffic-Event, für andere Sendeereignisse, verwendet. Berechnet werden die durch das Update-Protokoll **zusätzlich** anfallenden Kosten.

1.) Auf ein Update-Event folgt ein Update-Event . In diesem Fall berechnet das nachfolgende Update die Kosten für das vorhergehende Update. Es wird also genau wie bei den GPS-Messungen eine Kette gebildet, in der der Nachfolger die Kosten für den Vorgänger berechnet.

2.) Auf ein Traffic-Event folgt ein Update-Event. Es können keine Kosten für eine vorangegangene Update-Nachricht berechnet werden, also ist dies der Beginn einer neuen Update-Kette. Theoretisch müssten nun die vollen Update Kosten berechnet werden, allerdings sind die Kosten verringert, da ja in der *tail time* einer anderen Nachricht gesendet wird. Es werden also die reduzierten Kosten berechnet. Dies ist dennoch korrekt, da die andere Nachricht ohne das Update-Protokoll die vollen Kosten für die *tail time* bezahlt hätte, diese Kosten fallen jetzt am Ende der Kette an, sind aber keine zusätzlichen Kosten, die durch das Update-Protokoll verursacht wurden. Daher werden nur die reduzierten Kosten

berechnet und dieser Fall wird daher genauso wie Fall 1 gehandhabt. Fall 2 entspricht also Fall 1 und es muss im Programmcode keine Fallunterscheidung durchgeführt werden.

3.) Auf ein Update-Event folgt ein Traffic-Event. Hier werden die Kosten für den Energiebedarf der vollen *tail time* entfernt und die Kosten für das vorausgehende Update berechnet. Der Grund hierfür ist, dass das vorangegangen Update entweder der Anfang einer neuen Update-Kette ist oder das Ende einer Kette. In beiden Fällen wären die Kosten für dieses Update die Kosten für die ganze *tail time*. Diese werden entfernt und durch die verringerten Kosten ersetzt. Die Kette ist damit beendet.

4.) Auf ein Traffic-Event folgt ein Traffic-Event. Hier fallen keine zusätzlichen, also durch das Update-Protokoll verursachten, Kosten an.

Die Energiekosten setzen sich also aus den Kosten für die GPS-Messungen und den Kosten für das Senden der Positions-Updates an den Location-Server zusammen. Die Kosten für die GPS-Messungen werden bei jedem Auftreten eines Position-Events berechnet. Die Kosten für das Senden von Positions-Updates werden berechnet, wenn entweder ein Update-Event oder ein Traffic-Event auftritt.

5.1.2 MobileNodeDistanceBased

Die Klasse *MobileNodeDistanceBased* implementiert den Standardalgorithmus für ein distanzbasiertes Update-Protokoll (siehe Kapitel 2.4.3). Im Folgenden ist diese Implementierung näher beschrieben.
Wie bereits erwähnt, muss jedes Update-Protokoll die drei Funktionen *onPositionEvent(Event)*, *onUpdateEvent(Event)* und *onTrafficEvent(Event)* implementieren. Als Parameter wird diesen Funktionen jeweils das zugehörige Event übergeben. Der Standardalgorithmus berechnet jedes Mal, wenn er neue Koordinaten erhält, wie viel Zeit verbleibt, bis wieder eine GPS-Messung durchgeführt werden muss. Die Formel um die verbleibende Zeit zu berechnen lautet:

$$T = \frac{Garantie - Entfernung\ von\ der\ letzten\ Position}{max.\ Geschwindigkeit}$$

Hierbei ist die Garantie die maximale Entfernung des mobilen Endgerätes von den letzten an den Location-Server geschickten Koordinaten. Es wird immer dann ein Positions-Update an den Location-Server gesendet, wenn für eine weitere GPS-Messung nicht mehr genug Zeit verbleibt. D. h., wenn die Zeit bis zur nächsten GPS-Messung kleiner ist als die Zeit, die der GPS-Sensor benötigt, um seine Position zu bestimmen.
Die Funktion *onPositionEvent(Event)* muss also die Kosten für die GPS-Messung hinzufügen, die Zeit bis zur nächsten GPS-Messung berechnen und prüfen, ob ein Update ausgeführt werden muss. Je nachdem, ob ein Update durchgeführt werden muss oder nicht, ergeben sich unterschiedliche Zeiten bis zur nächsten GPS-Messung. Daher ist der letzte Aufruf der GPS-Messung das Planen des nächsten Position-Events.
Dazu wird die Zeit bestimmt, zu der die nächste Messung durchgeführt werden muss, um die Garantie zu gewährleisten. Anschließend muss dieser Trackpoint in der Position-Trace-File gesucht werden. Hier tritt ein Problem auf, denn es ist sehr unwahrscheinlich, dass es einen Trackpoint in der Datei gibt, der exakt zu dem gesuchten Zeitpunkt bestimmt wurde.
Daher muss in der Datei nach dem nächsten Trackpoint gesucht werden, dessen Zeit größer ist als die gesuchte. Ist dieser gefunden, so wird zwischen diesem und seinem Vorgänger linear interpoliert.

Wenn ein Trackpoint vorhanden ist, der zur gesuchten Zeit bestimmt wurde, so wird dieser verwendet und es ist keine Interpolation nötig. Die Interpolation führt zu Fehlern in der Simulation. Allerdings sind diese Fehler in den meisten Fällen sehr klein, da in den verwendeten Dateien Trackpoints meistens im Abstand von nur einer Sekunde verfügbar sind. Dennoch treten hin und wieder größere Abstände zwischen den Trackpoints auf.
Die Funktion *onUpdateEvent(Event)* muss lediglich die Funktion der Klasse *CommunicationInterface* aufrufen, die die Kosten für das Update berechnet.
Die Funktion *onTrafficEvent(Event)* teilt der Klasse *CommunicationInterface* mit, dass eine Nachricht gesendet wurde. Die daran anschließenden Berechnungen werden dann dort ausgeführt. Die einzige andere Aufgabe, die die Funktion *onTrafficEvent()* hat, ist die, das nächste Traffic-Event zu planen. Dazu wird der nächste Wert aus der *Traffic-Trace-File* ausgelesen und auf die Zeit des letzten Traffic-Events aufaddiert.

5.1.3 MobileNodeFarrell

Die Unterschiede zum Standardalgorithmus beschränken sich auf die Funktion *onUpdateEvent()*, daher wird nachfolgend nur diese betrachtet.
Der Unterschied zum Standardalgorithmus ist, dass es zusätzlich zu den Updates, die durchgeführt werden müssen, um die Garantie zu gewährleisten, noch eine weitere Art von Updates gibt.
Diese zweite Art von Updates sind die sogenannten Early-Updates, die ausgeführt werden, wenn es sich im Bezug auf die Energie lohnt, ein Update früher zu senden, als für die Gewährleistung der Garantie erforderlich ist.
Early-Updates werden in den Standardalgorithmus integriert, indem hinter der Abfrage für ein forced Update eine weitere Abfrage eingefügt wird, die prüft, ob die Bedingung für ein Early-Update erfüllt ist. Dies ist im nachfolgenden Pseudocode kurz dargestellt.

```
procedure onUpdateEvent(Event)
...
    if forcedUpdateNecessary then
...
    else if earlyUpdateNecessary then
...
    end if
...
end procedure
```

Im von Farrell et al. [FLR07] entwickelten Algorithmus hängt die Entscheidung, ob ein Early-Update gesendet wird oder nicht, davon ab, ob es sich im Bezug auf den Energiebedarf lohnt ein Early-Update zu senden oder nicht. Dazu wird folgende Formel verwendet:

$$d_r > \frac{W_U}{(W_U + W_S)} * d_{max}$$

Hierbei bezeichnen W_U und W_S die Energie, die für ein Update, bzw. für eine Positionsmessung benötigt wird. Die Entfernung der aktuellen Position von der zuletzt an den Location-Server gesendeten

Position ist d_r. Und der Radius des kreisförmigen Gebietes, in dem sich das mobile Endgerät bewegen kann, ohne dass ein forced Update nötig ist, ist d_{max}.
Die Entscheidung ist also davon abhängig, wie viel der maximal zurücklegbaren Strecke, d_{max}, prozentual zurückgelegt wurde.
Da Farrell et al. ein lineares Energiemodell für diese Entscheidung verwenden und in der vorliegenden Arbeit ein nichtlineares Energiemodell verwendet wird, kann die Entscheidung nicht auf die gleiche Art und Weise getroffen werden.
Daher wird der prozentuale Wert variabel gewählt. In den durchgeführten Tests werden dann 101 verschiedene Werte für diesen prozentualer Wert gewählt. Dies sind die Werte von 0%-100%.

5.1.4 MobileNodePiggyback

Im Gegensatz zum in der Klasse *MobileNodeFarrell* verwendeten Algorithmus unterscheidet sich der in Kapitel 4 vorgestellte Algorithmus auch in den Funktionen *onUpdateEvent()* und *onTrafficEvent()* von dem in der Klasse *MobileNodeDistanceBased* verwendeten Algorithmus.
Zusätzlich zu den normalen Updates gibt es hier noch Early-Updates und Piggyback-Updates. Piggyback-Updates sind Update-Nachrichten, die in der *tail time* einer anderen Nachricht gesendet werden. Dafür ist es nötig, die Netzwerkschnittstelle des Smartphones zu überwachen.
Diese Aufgabe wird im Simulator durch die Traffic-Events erfüllt. Ähnlich wie bei Farrells Ansatz wird auch hier die Funktion *onPositionEvent()* durch eine weitere Abfrage ergänzt. Abhängig von einem Schwellwert wird die Entscheidung getroffen, ob versucht werden soll ein Piggyback-Update zu senden oder nicht. Es ist deshalb ein Versuch, da zum Zeitpunkt der Planung nicht klar ist, ob eine Nachricht gesendet wird, in deren *tail time* dann das Update gesendet werden kann.
Genau wie bei den anderen Algorithmen wird auch bei diesem die Zeit berechnet, bis erneut ein Update durchgeführt werden muss, damit die Garantie nicht verletzt wird. Dies geschieht mit der gleichen Formel, die bereits beim Standardalgorithmus vorgestellt wurde. Dies ist auch die Zeit, die für den Versuch ein Piggyback-Update zu senden zur Verfügung steht.
Kann in dieser Zeit kein Piggyback-Update gesendet werden, so verbleiben zwei Möglichkeiten.
Entweder es wird eine GPS-Messung durchgeführt und die Funktion *onPositionEvent()* wird erneut aufgerufen oder es wird wie bei Farrell ein Early-Update durchgeführt.
Diese Entscheidung kann allerdings erst getroffen werden, wenn klar ist, ob ein Piggyback-Update gesendet werden konnte. Daher wird in der Funktion *onPositionEvent()* ein Update-Event sowie ein Position-Event geplant. Hierbei wird das Update-Event auf wenige Sekunden vor dem Positions-Event festgelegt, damit in der Funktion *onUpdateEvent()* entschieden werden kann, ob die GPS-Messung überhaupt durchgeführt werden soll.
Die Funktion *onUpdateEvent()* wird daher um eine Abfrage erweitert, die zwischen notwendigen Updates, solche die die Garantie gewährleisten sollen, und Early-Updates unterscheidet.
Tritt ein Update-Event auf, das zu einem Early-Update gehört, so kann das nächste Position-Event aus der Queue entfernt werden und ein neues Position-Event geplant werden, falls das Update durchgeführt werden soll. Anderenfalls kann das Update-Event ignoriert werden, sodass das wenige Sekunden später geplante Position-Event ausgeführt wird.
Auch die Funktion *onTrafficEvent()* muss geändert werden, denn beim Auftreten eines Traffic-Events muss nun überprüft werden, ob ein Piggyback Update geplant ist oder nicht. Wenn eines geplant wird, so wird dieses durchgeführt. Darüber hinaus wird das nächste geplante Update aus der Queue entfernt

und durch ein neues ersetzt, denn durch das Update ändert sich die Zeit, bis die nächste GPS-Messung nötig ist. Hierbei ist zu beachten, dass aufgrund der Tatsache, dass der Simulator auf Events basiert, die Update-Nachricht direkt nach dem Auftreten des Traffic-Events versendet wird. Also, dass es direkt nach einer vorausgehenden Nachricht versendet wird. Das bedeutet für die Berechnung der Energiekosten, dass nur die Übertragungsdauer der Update-Nachricht in die Kosten einfließt, aber kein Anteil an der *tail time*. Je nach Abtastintervall würde in einer realen Anwendung noch ein kleiner Teil der *tail time* in die Berechnung einfließen. Da dieser aber vermutlich kleiner wäre als eine Zehntel Sekunde kann der Effekt in der Simulation vernachlässigt werden.
Abbildung 5.3 zeigt das Flussdiagramm des hier beschriebenen Algorithmus, dieses wurde bereits in Kapitel 4 gezeigt.

5.2 Android Anwendung

Die in Java geschriebene Android Anwendung bietet eine grafische Benutzeroberfläche, die die Eingabe von zwei Parametern ermöglicht. Abbildung 5.4 zeigt die Benutzeroberfläche.
Der erste Parameter ist die Garantie, also eine Angabe für den Radius eines kreisförmigen Gebiets, in dem sich der Nutzer aufhalten kann, ohne dass eine Update-Nachricht gesendet werden muss.
Der zweite Parameter ist die maximale Geschwindigkeit, mit der das Smartphone bewegt werden soll.
Zusätzlich bietet die grafische Benutzeroberfläche zwei Schaltflächen zum Starten und Stoppen des Update-Protokolls.
Wird die Schaltfläche *START* aktiviert, so wird die *MainActivity* gestartet. Diese startet einen *IntentService*, der das Update-Protokoll in einem neuen *Thread* ausführt. Dadurch läuft das Update-Protokoll im Hintergrund und der Nutzer kann andere Anwendungen benutzen.

Das Update-Protokoll verwendet einen *LocationListener*, um von dem GPS-Sensor Positionsdaten zu erhalten. Sobald Positionsdaten verfügbar sind, ruft dieser die Funktion onPosition() auf. In dieser Funktion ist der in Kapitel 4 beschriebene Algorithmus umgesetzt.
Updates werden unter Verwendung eines *java.net.Socket* versendet. Dazu wurde ein weiteres Socket geschrieben, das auf einem anderen Rechner ausgeführt werden kann. Damit steht ein Location-Server zur Verfügung an den die Positionsdaten gesendet werden können. Der Server ist dabei sehr einfach aufgebaut. Er hat lediglich die Aufgabe die an ihn gesendeten Positionsdaten in eine Datei zu schreiben.
Die für das Senden von Piggyback-Updates nötige Überwachung der Netzwerkschnittstelle erfolgt mit Hilfe der Funktion *android.net.TrafficStats.getMobileTxBytes()*. Diese liefert die Anzahl der seit dem Start des mobilen Endgerät über alle Netzwerke versendeten Bytes.
Die Anzahl der Bytes wird einmal abgerufen und dann wird alle 100 ms überprüft, ob sich dieser Wert geändert hat. Da der Wert monoton wächst, kann dadurch erkannt werden, dass eine Nachricht versendet wurde. Wenn eine Nachricht versendet wurde, wird eine Update-Nachricht über ein java.net.Socket versendet.
Kann kein Piggyback-Update versendet werden, bis die nächste Positionsbestimmung erfolgen muss, so wird geprüft, ob ein Early-Update versendet werden soll.

Das Update-Protokoll kann jederzeit durch das Verwenden der *STOP* Schaltfläche beendet werden.

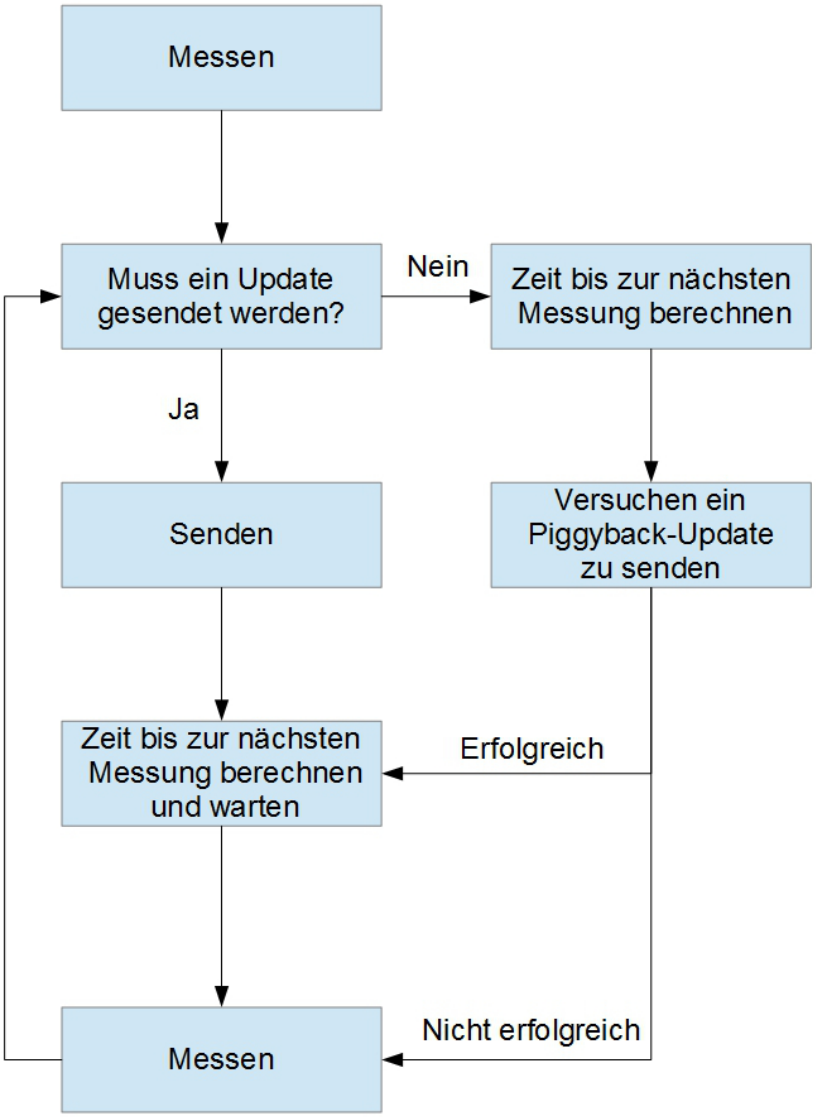


Abbildung 5.3: Flussdiagramm für den Piggyback-Algorithmus.

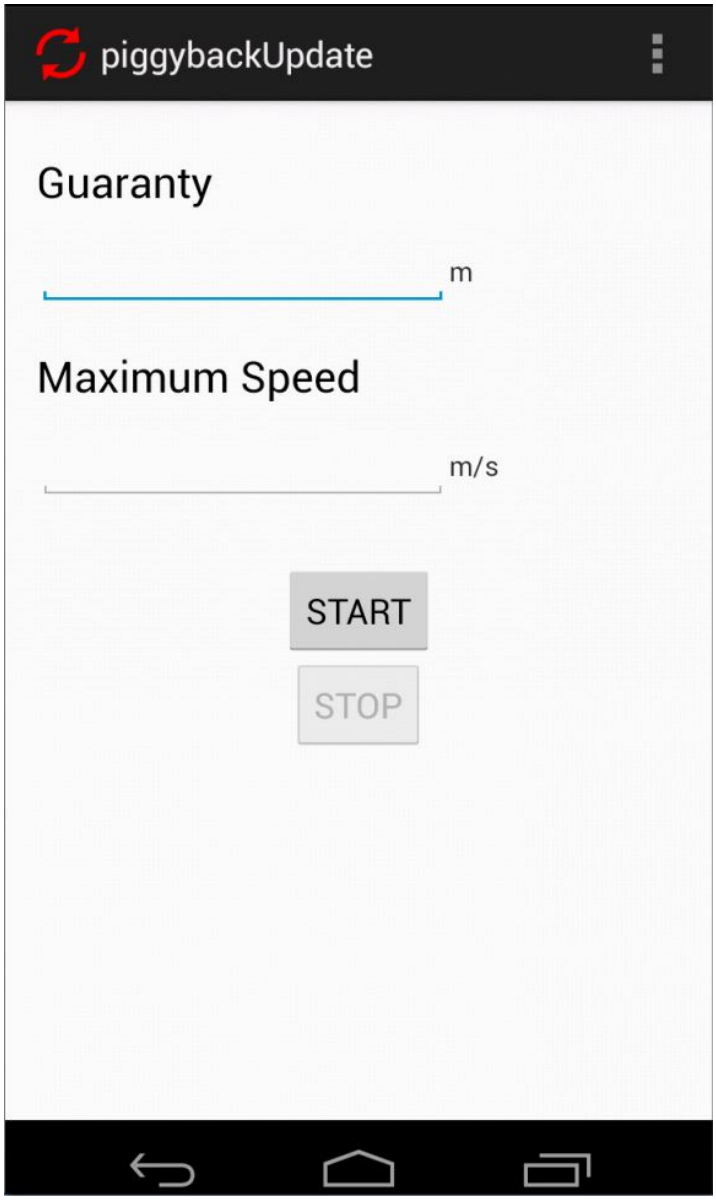


Abbildung 5.4: Die Benutzeroberfläche der Andorid Anwendung.

6 Evaluation

6.1 Simulator

Für die Tests im Simulator werden drei verschiedene Dateien mit GPS-Koordinaten verwendet. Diese Position-Trace-Files stammen von der Website http://www.openstreetmap.org/traces.
Jede der drei Dateien besitzt andere Eigenschaften, d.h. die Routen decken unterschiedlich große Zeitspannen ab, die maximale Geschwindigkeit ist bei jeder Datei anders und das Bewegungsprofil unterscheidet sich ebenso.
Zudem werden drei Dateien verwendet, die Angaben darüber enthalten, wann Sendeereignisse auftreten. Die Herkunft dieser Traffic-Trace-Files ist im Kapitel 5.1 beschrieben.
Vor der Durchführung der Test werden zuerst die verwendeten Dateien analysiert, um die dabei gewonnenen Erkenntnisse später zur Diskussion der Simulationsergebnisse verwenden zu können. Anschließend sind die durchgeführten Tests beschrieben, gefolgt von der Vorstellung und der Diskussion der Ergebnisse.

6.1.1 Voruntersuchungen

In der Voruntersuchung werden die verwendeten Dateien analysiert, um geeignete Werte für die Simulationen zu erhalten und deren Ergebnisse im Anschluss zu erklären.
Der Simulator benötigt für jede Position-Trace-File Informationen über deren Laufzeit, d. h. die Zeitspanne zwischen erster und letzter GPS-Messung, damit die Simulationsdauer festgelegt werden kann. Außerdem benötigen die Algorithmen Angaben über die maximale Geschwindigkeit des mobilen Endgeräts. Die Garantie kann nicht aus den Dateien bestimmt werden, sondern muss festgelegt werden.
Über Traffic-Trace-Files benötigt der Simulator keine Daten, jedoch sind für die Analyse der Ergebnisse die Eigenschaften dieser Dateien wichtig. Das sind Informationen über die Anzahl der gesendeten Nachrichten und die Länge der Abstände zwischen dem Versenden aufeinander folgender Nachrichten.

Position-Trace-Files

Der Zeitpunkt des ersten und des letzten Trackpoints in den Dateien wurde manuell ausgelesen und die abgedeckte Zeitspanne berechnet. In Tabelle 6.1 sind die Laufzeiten der einzelnen Dateien aufgelistet.

Dateiname	Laufzeit
Biketour.gpx	5 h 32 min
45_6h.gpx	5 h 20 min
drive2.gpx	3 h 30 min

Tabelle 6.1: Die Tabelle zeigt die für die Position-Trace-Files ermittelten Laufzeiten.

Dateiname	max. Geschwindigkeit in m/s (3 h)	max. Geschwindigkeit (5 h)
Biketour.gpx	8,8	10,6
45_6h.gpx	43,5	43,5
drive2.gpx	7,3	-

Tabelle 6.2: Die Tabelle zeigt die ermittelten maximalen Geschwindigkeiten der mobilen Endgeräte in den Dateien, für eine Garantie von 1000 m.

Aufgrund dieser Daten wird entschieden, im Simulator die Laufzeiten 3 h und 5 h zu verwenden. Dabei werden 3 h Laufzeit für alle Dateien verwendet, um diese untereinander zu vergleichen. Zusätzlich werden 5 h Laufzeit für die beiden Dateien verwendet, die eine größere Zeitspanne abdecken, damit ein größerer Bereich der Traffic-Trace-Files verwendet werden kann.
Alle GPS-Dateien werden im Simulator ausgewertet, um herauszufinden, was die maximale Geschwindigkeit ist, mit der der GPS-Sensor bewegt wurde. Dazu wird gezählt, wie oft die Garantie für eine gewählte Geschwindigkeitsangabe verletzt wird. Die maximale Geschwindigkeit ist also der Geschwindigkeitswert, für den die Garantie nicht verletzt wird.
Der so ermittelte Wert muss nicht unbedingt der realen maximalen Geschwindigkeit entsprechen, da GPS-Messungen nie exakt sind, sondern immer einen Fehler beinhalten. Je nachdem wie groß dieser Fehler ist, wird die maximale Geschwindigkeit über- oder unterschätzt. Die Geschwindigkeiten wurden in Schritten von 0,1 m/s getestet.

Bei der Durchführung der Analyse fällt auf, dass die maximale Geschwindigkeit, bei der die Garantie nie verletzt wird, sowohl von dem für die Garantie gewählten Wert als auch von der verwendeten Traffic-Trace-File als auch von der Laufzeit abhängig ist.
Für die Datei *drive2.gpx* ergeben sich für eine Garantie von 1000 m bei einer Laufzeit von 3 h Geschwindigkeiten zwischen 7,3 m/s und 12,7 m/s, je nach verwendeter Traffic-Trace-File.
Auffällig ist, dass die Garantie nur ein einziges Mal verletzt wird, wenn die max. Geschwindigkeit auf 7,3 m/s festgelegt wird und dann die anderen Traffic-Trace-Files verwendet werden. Dies deutet darauf hin, dass eine einzelne GPS-Messung eine hohe Ungenauigkeit aufweist. Daher wird entschieden, einzelne Verletzungen zu ignorieren und eine maximale Geschwindigkeit zu verwenden, die je nach verwendeter Traffic-Trace-File, zu maximal einer Verletzung der Garantie führt.
Die so ermittelten Geschwindigkeiten für eine Genauigkeit von 1000 m sind in Tabelle 6.2 für 3 h Laufzeit bzw. 5 h Laufzeit dargestellt.

Dateiname	Anzahl Sendeereignisse (3 h)	Anzahl Sendeereignisse (5 h)
conTimes.csv	125	220
conTimesFrank.csv	311	680
conTimesPatrickNew	513	797

Tabelle 6.3: Die Tabelle zeigt die Anzahl der Sendeereignisse der Traffic-Trace-Files innerhalb von 3 bzw. 5 Stunden.

Traffic-Trace-Files

Für die drei Dateien mit den Kommunikationsdaten werden Histogramme erstellt, um Aussagen darüber treffen zu können, wie groß die Abstände zwischen einzelnen Übertragungen sind und wie häufig welche Abstände vorkommen.

Dabei werden nicht die gesamten Daten aus den Dateien verwendet, sondern nur die Daten, die auch für die Simulationen verwendet werden.
Um zu bestimmen, welche Daten verwendet werden, wird mit dem Simulator getestet, wie viele Übertragungen in der Simulationszeit auftreten. Dafür werden Simulationen mit einer Dauer von 3 und 5 Stunden durchgeführt. Die Anzahl der Übertragungen für jede Datei ist in Tabelle 6.3 dargestellt. Für jede Traffic-Trace-File werden zwei Histogramme erstellt, einmal für 3 Stunden Simulationsdauer und einmal für 5 Stunden.

Dabei werden nur die Werte im Histogramm dargestellt, die während der entsprechenden Simulationsdauer verwendet werden.
Die Ergebnisse dieser Untersuchung sind in den Abbildungen 6.1 dargestellt. In der Datei conTimes haben alle Einträge außer zwei einen Wert der kleiner als 300 ist. Die beiden größeren Werte sind 440 und 1803. Dies gilt für 3 h Laufzeit und für 5 h Laufzeit. In der Datei conTimesFrank sind alle Einträge kleiner als 400, der Großteil sogar kleiner als 125. In der Datei conTimesPatrickNew sind alle Einträge außer 3 kleiner als 400, für eine Simulationsdauer von 5 h sind es vier. Die meisten Einträge haben einen Wert der kleiner als 20 ist. Der größte Wert beträgt 1955.
Die Datei conTimesPatrickNew weist die meisten Einträge auf und fast alle dieser Einträge sind kleiner als 20. D. h. es werden sehr häufig und sehr viele Nachrichten versendet.
In mittleres Kommunikationsaufkommen ist in der Datei canTimesFrank zu finden. Dort gibt es auch keine großen Zeiträume in denen gar keine Kommunikation stattfindet.
Am wenigsten Nachrichten werden in der Datei conTimes versendet, dort gibt es einen großen Zeitraum, in den nicht kommuniziert wird. Die Abstände zwischen einzelnen Sendeereignissen sind deutlich größer als in der Datei conTimesPatrickNew.

6.1.2 Ziele

Ziel der Untersuchungen mit dem Simulator ist es, so viel wie möglich über den Algorithmus zu erfahren. Da beim Entwurf des Algorithmus nicht alle Parameter festgelegt werden konnten, müssen diese zuerst einmal experimentell bestimmt werden.

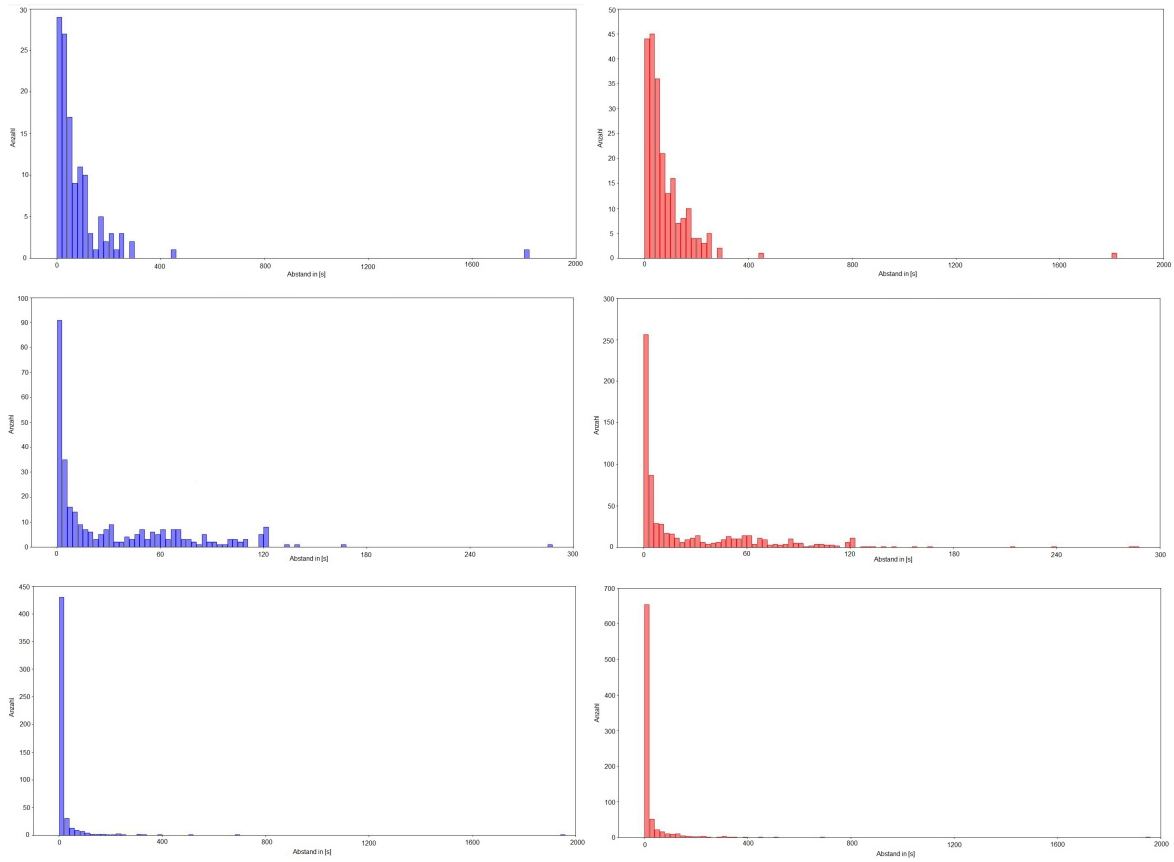


Abbildung 6.1: Die Abbildung zeigt die Histogramm der untersuchten Dateien. Oben sind die Histogramme für die Datei conTimes, in der Mitte die für conTimesFrank und Unten die für conTimesPatrickNew. Links sind jeweils die Histogramme für 3 h Simulationsdauer und rechts die für 5 h.

Schwellwert für Piggyback-Updates

Zuerst wird dafür der Algorithmus ohne die Erweiterung mit Early-Updates getestet, um so herauszufinden, welcher Schwellwert für die Piggyback-Updates gewählt werden soll. Zur Erinnerung: Der Schwellwert gibt prozentual an, wie groß die Entfernung der aktuellen Position zur letzten an den Location-Server gesendeten Position im Verhältnis zur maximal zurücklegbaren Entfernung ist. Die maximal zurücklegbare Entfernung ist dabei die Entfernung, die zurückgelegt werden kann, ohne dass ein forced Update nötig ist.
Der Schwellwert soll dabei so gewählt werden, dass so viel wie möglich Energie bei den Kosten für die Übertragung von Update-Nachrichten eingespart wird.

Schwellwert für Early-Updates

Wenn ein guter Schwellwert für die Piggyback-Updates gefunden ist, soll dieser verwendet werden, um für den erweiterten Algorithmus einen guten Schwellwert für die Early-Updates zu finden. Der Schwellwert für die Early-Updates soll so gewählt werden, dass die insgesamt benötigte Energie so niedrig wie möglich ausfällt.

Vergleich mit anderen Algorithmen

Sind beide Schwellwerte bestimmt, so soll der dadurch vollständige Algorithmus mit anderen Algorithmen verglichen werden.
Der Vergleich ist wichtig, um einen Eindruck davon zu erhalten, wie gut der neu entwickelte Algorithmus wirklich ist. Für die Vergleiche sollen der Standardalgorithmus für distanzbasierte Updates sowie der Update-Algorithmus von Farrell verwendet werden. Dabei wird für den Ansatz von Farrell eine Modifikation vorgenommen, nämlich dass der Schwellwert für die Early-Updates variabel gewählt wird. Diese Modifikation und der Grund für sie sind in Kapitel 5 detaillierter beschrieben.
Gegenstand des Vergleich soll sowohl der absolute Energieverbrauch sein als auch der Energiebedarf für Positionsupdates sowie der Energiebedarf für Positionsmessungen.

Einfluss der Parameter

Da die Effektivität des Algorithmus von mehreren Parametern und den beiden Eingabedateien (Position-Trace-File und Traffic-Trace-File) abhängig ist, muss deren Einfluss ebenfalls berücksichtigt werden.

6.1.3 Test Durchführung

Im Folgenden wird beschrieben, welche Tests durchgeführt werden, um die Ziele der Evaluation zu erreichen. Dabei werden die Werte für alle Parameter angegeben und es wird erwähnt, welche Dateien für den Simulator als Eingabe verwendet werden.

Schwellwert für Piggyback-Updates

Um den Schwellwert für die Piggyback-Updates zu bestimmen, werden zwei der Position-Trace-Files verwendet und jeweils mit den drei Traffic-Trace-Files getestet. Dabei wird die Garantie auf 1000 m und die Simulationszeit auf 3 h festgelegt.
Der Schwellwert für das Piggyback-Update wird für jeden Simulationsdurchlauf anders gewählt. Beginnend mit 0% wird er in jedem weiteren Durchlauf um 1% erhöht, bis er im 101 Durchlauf den Wert 100% erreicht.
Die verwendeten Position-Trace-Files sind 45_6h.gpx und drive2.gpx. Diese Dateien werden aufgrund ihrer stark unterschiedlichen maximalen Geschwindigkeiten gewählt. Dadurch ist die Zeit zwischen zwei Positionsbestimmungen jeweils unterschiedlich. Damit kann ein möglicher Einfluss dieser Zeit

festgestellt werden.
Mit den so gewonnenen Daten soll versucht werden einen Schwellwert zu finden. Falls der Schwellwert von verschiedenen Parametern abhängig ist, soll versucht werden, eine Formel für diesen zu bestimmen. Wird eine Formel gefunden, so soll diese an der dritten Position-Trace-File getestet werden.

Schwellwert für Early-Updates

Sobald ein Schwellwert für die Piggyback-Updates gefunden ist, soll dieser in den mit Early-Updates erweiterten Algorithmus eingesetzt werden. Der erweiterte Algorithmus wird dann mit den Position-Trace-Files Biketour.gpx und drive2.gpx getestet.

Dabei wird der Schwellwert für Early-Updates bei jedem Durchlauf verändert. Beginnend mit 0% wird er in jedem weiteren Durchlauf um 1% erhöht, bis er im 101 Durchlauf den Wert 100% erreicht. Die Garantie wird ebenfalls auf 1000 m festgelegt und auch die Simulationsdauer beträgt 3 h.
Aus den Simulationsergebnissen soll anschließend ein geeigneter Schwellwert abgeleitet werden. Falls dieser von Parametern abhängt, soll versucht werden eine Formel zu finden, die einen möglichst guten Wert liefert.
Falls eine Formel gefunden wird, so soll diese an der dritten Position-Trace-File getestet werden.

Vergleich mit anderen Algorithmen

Für den Vergleich mit anderen Algorithmen soll die modifizierte Version von Farrells Ansatz sowie der hier vorgestellte Algorithmus an allen drei Position-Trace-Files getestet werden. Dazu werden ebenfalls alle drei Traffic-Trace-Files verwendet.

Für die Dateien Biketour.gpx und 45_6h.gpx soll jeweils eine Simulationsdauer von 5 h verwendet werden. Die Datei drive2.gpx wird mit 3 h Simulationsdauer getestet. Die unterschiedliche Simulationsdauer wird gewählt, da die Ergebnisse untereinander nicht mehr verglichen werden müssen, sondern nur noch die verschiedenen Algorithmen miteinander.
Der in Kapitel 4 vorgestellte Algorithmus wird dabei in zwei Varianten getestet. Einmal mit Early-Updates und einmal ohne. Dabei wird als Schwellwert für die Early-Updates der ermittelte Wert verwendet. Im Falle einer Formel wird dieser jeweils gesondert angegeben.
Der Schwellwert für die Piggyback-Updates wird variabel gewählt, ebenso wie der Wert für Early-Updates in der modifizierten Version von Farrells Algorithmus. Dabei werden wieder 101 Durchläufe mit unterschiedlichen Werten durchgeführt.
Die Ergebnisse dieser Simulationen sollen im Bezug auf die insgesamt benötigte Energie, die Energie für Positionsbestimmungen und die Energie für Update-Nachrichten untersucht werden.

6.1.4 Ergebnisse und Diskussion

In diesem Kapitel werden die Ergebnisse der Simulationen dargestellt und diskutiert.

Schwellwert für Piggyback-Updates

In Abbildung 6.2 ist für die Datei drive2.gpx dargestellt, wie viel Energie für Positions-Updates benötigt wird. Hier ist erkennbar, dass sich der in Kapitel 4 vorgestellte Algorithmus für einen Schwellwert nahe an 100% genauso verhält wie der Standardalgorithmus für distanzbasierte Update-Protokolle. Das liegt daran, dass bei 100% gar keine Piggyback-Updates ausgeführt werden und die beiden Algorithmen dadurch identisch sind. Das gleiche ist in Abbildung 6.3 für die Datei 45_6h.gpx dargestellt. Für beide Dateien wurde der in Kapitel 4 vorgestellte Algorithmus ohne Early-Updates verwendet.
Da es mehrere Minima gibt und die Werte so niedrig sind, dass sie durch geringe Schwankungen bereits stark beeinflusst werden, ist ein Schwellwert aus diesen Ergebnissen nicht bestimmbar.
Es kann jedoch gesagt werden, dass mit einem steigenden Schwellwert zunehmend mehr Energie verbraucht wird.

Das die Piggyback-Updates kaum Einfluss haben, wenn die Datei 45_6h.gpx verwendet wird, liegt daran, dass die maximale Geschwindigkeit hier sehr viel größer ist. Dies bewirkt, dass weniger Zeit zur Verfügung steht um ein Piggyback-Update zu versenden.

Außerdem wird an den Schaubildern der Einfluss der Traffic-Trace-Files deutlich. So werden die niedrigsten Energiekosten für die Datei conTimesFrank erzielt. Diese hat, wie die Voruntersuchung gezeigt hat, keine größeren Zeiträume ohne Kommunikation.

Betrachtet man zusätzlich die Anzahl der versendeten Piggyback-Nachrichten, so wird deutlich, dass für einen niedrigen Schwellwert das größte Einsparpotential besteht. Außerdem fällt wieder auf, dass für die Datei conTimesFrank die meisten Piggyback-Updates versendet werden können. In Abbildung 6.4 und 6.5 sind diese Werte veranschaulicht. Hier ist erkennbar, dass für einen Schwellwert von 0% die meisten Piggyback-Updates versendet werden können. Dies ist dadurch erklärbar, dass hier das Zeitfenster, in dem Piggyback-Nachrichten versendet werden können, am größten ist. Daher wird der Schwellwert für die nächste Testreihe auf 0% festgelegt.

Schwellwert für Early-Updates

Abbildung 6.6 zeigt die insgesamt verbrauchte Energie für die Datei trace2.gpx. Für die Simulation wurde der um Early-Updates erweiterte Algorithmus verwendet. Der Schwellwert für Piggyback-Updates liegt bei 0%. Die ermittelten Werte deuten darauf hin, dass ein guter Schwellwert bei ca. 40% liegt.
Abbildung 6.7 liefert ähnliche Ergebnisse für die Datei Biketour.gpx, dort liegt ein guter Schwellwert bei 43%.
Nimmt man die Ergebnisse der Datei 45_6h.gpx hinzu, so sieht man dort einen guten Schwellwert bei

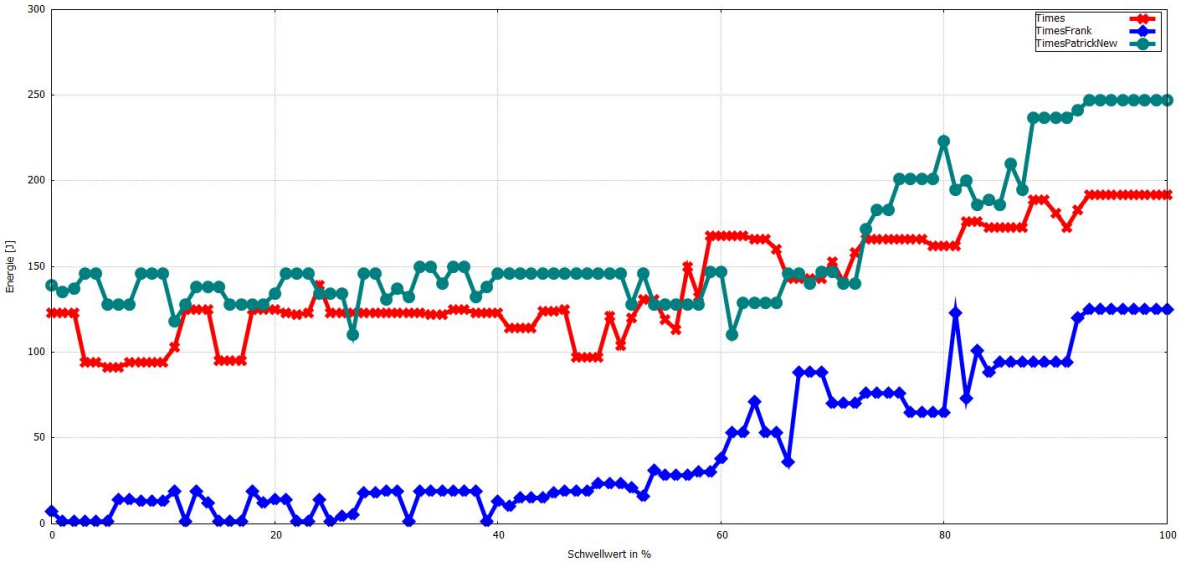


Abbildung 6.2: Die Abbildung zeigt die Kosten für Positions-Updates für die Datei trace2.gpx.

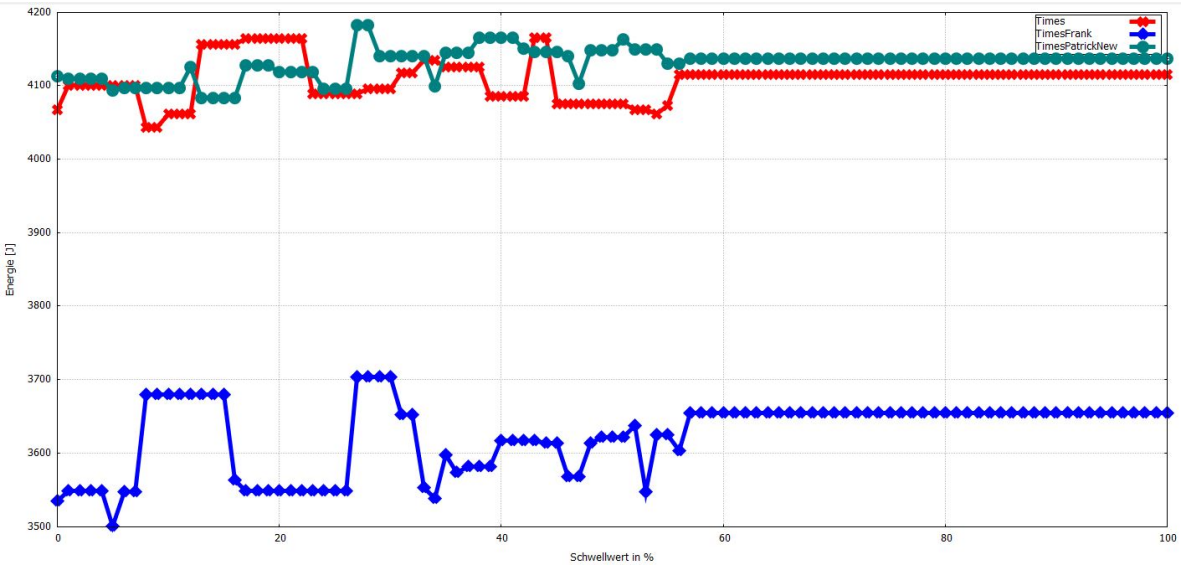


Abbildung 6.3: Die Abbildung zeigt die Kosten für Positions-Updates für die Datei 45_6h.gpx.

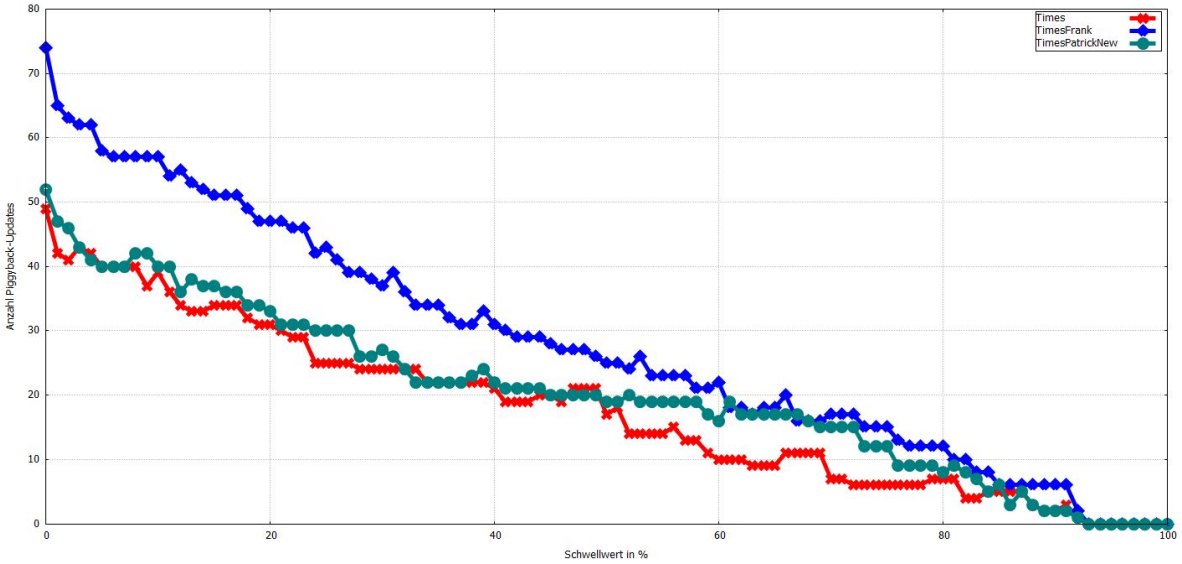


Abbildung 6.4: Die Abbildung zeigt die Anzahl an Piggyback-Updates für die Datei trace2.gpx.

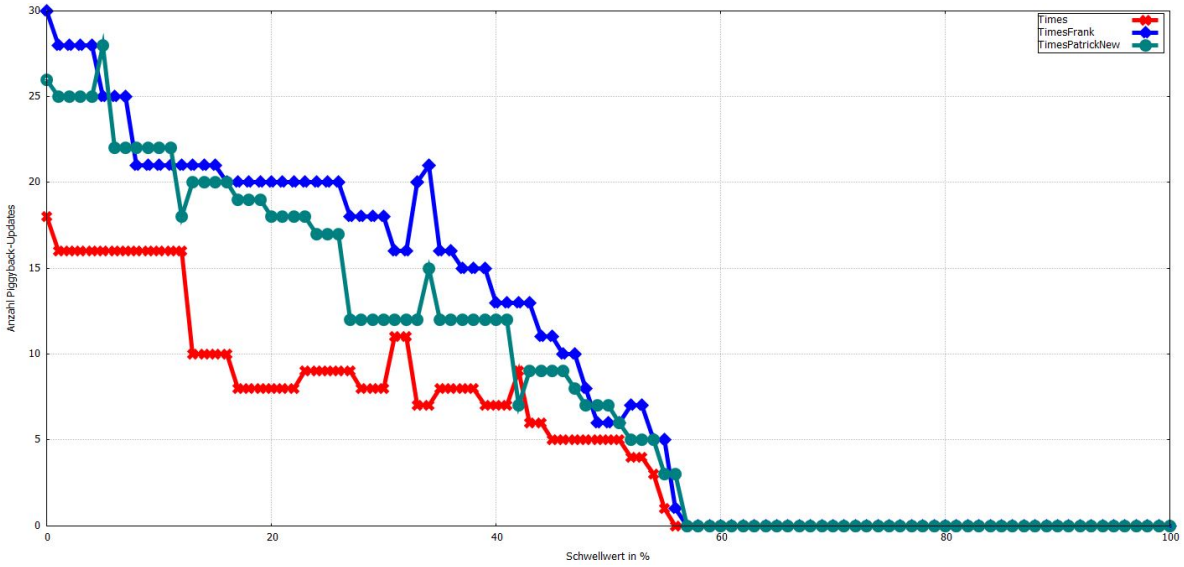


Abbildung 6.5: Die Abbildung zeigt die Anzahl an Piggyback-Updates für die Datei 45_6h.gpx.

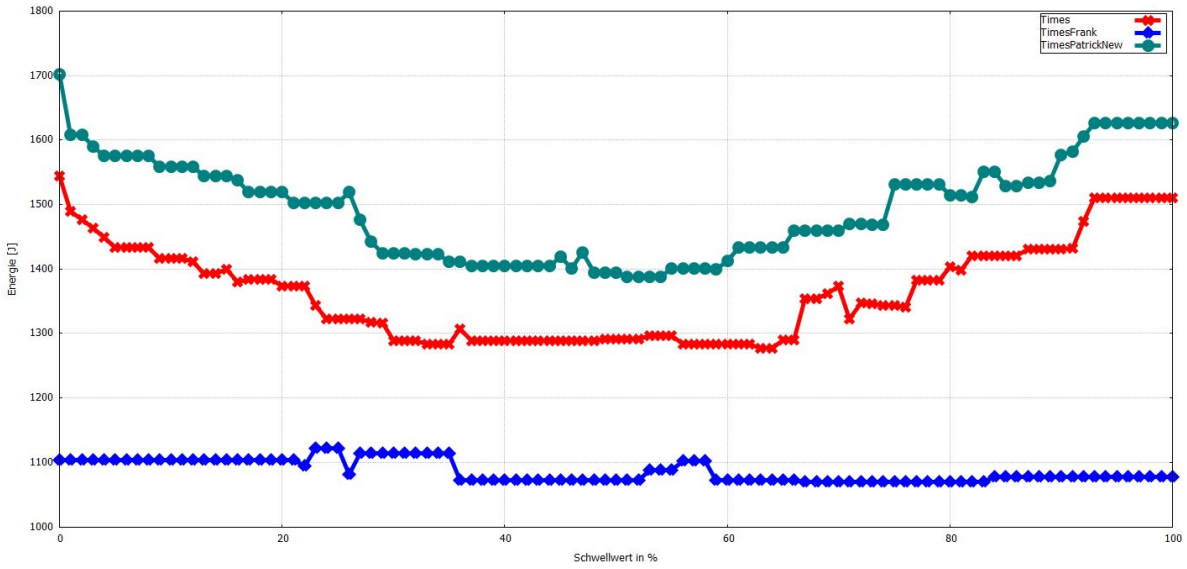


Abbildung 6.6: Die Abbildung zeigt den gesamten Energiebedarf des Algorithmus mit Early-Updates für die Datei trace2.gpx.

57%. Abbildung 6.8 zeigt diese Ergebnisse.
Diese Erkenntnisse deuten auf eine Abhängigkeit des Schwellwertes von einem oder mehreren Parametern hin. Es ist allerdings nicht möglich eine Formel anzugeben, da zu wenig Daten verfügbar sind. Um eine Formel zu finden wäre ein zu großer Aufwand nötig, deshalb wird im folgenden ein Schwellwert von 60% verwendet. Dieser liefert nicht unbedingt optimale Ergebnisse, aber denn noch hinreichend gute.

Vergleich mit anderen Algorithmen

Der Vergleich mit anderen Algorithmen liefert einige interessante Ergebnisse. Diese werden aufgrund der Übersichtlichkeit nur für die Datei Biketour.gpx mit der Traffic-Trace-File conTimesPatrickNew dargestellt.
Abbildung 6.9 zeigt die Gesamtenergie, Abbildung 6.11 zeigt nur die für Update-Nachrichten benötigte Energie und Abbildung 6.10 zeigt nur die für die Positionsmessungen benötigte Energie.
Verglichen werden der Ansatz von Farrell und der hier entwickelte Algorithmus mit einem Schwellwert von 60% für Early-Updates.
Es wird deutlich, dass der hier entwickelte Algorithmus deutlich weniger Energie benötigt, als der Ansatz von Farrell. Auch der Standardalgorithmus für ein distanzbasiertes Update-Protokoll benötigt mehr Energie, der Wert für diesen Entspricht den Werten von Farrell und Piggyback für einen Schwellwert von 100%.
Die Gesamtenergie setzt sich aus der Energie für die Positionsmessungen und der Energie für die Update-Nachrichten zusammen. Betrachtet man die zugehörigen Schaubilder, so ist zu erkennen, dass durch den Schwellwert ein Trade-off zwischen den Energiekosten besteht. Wenn die Kosten

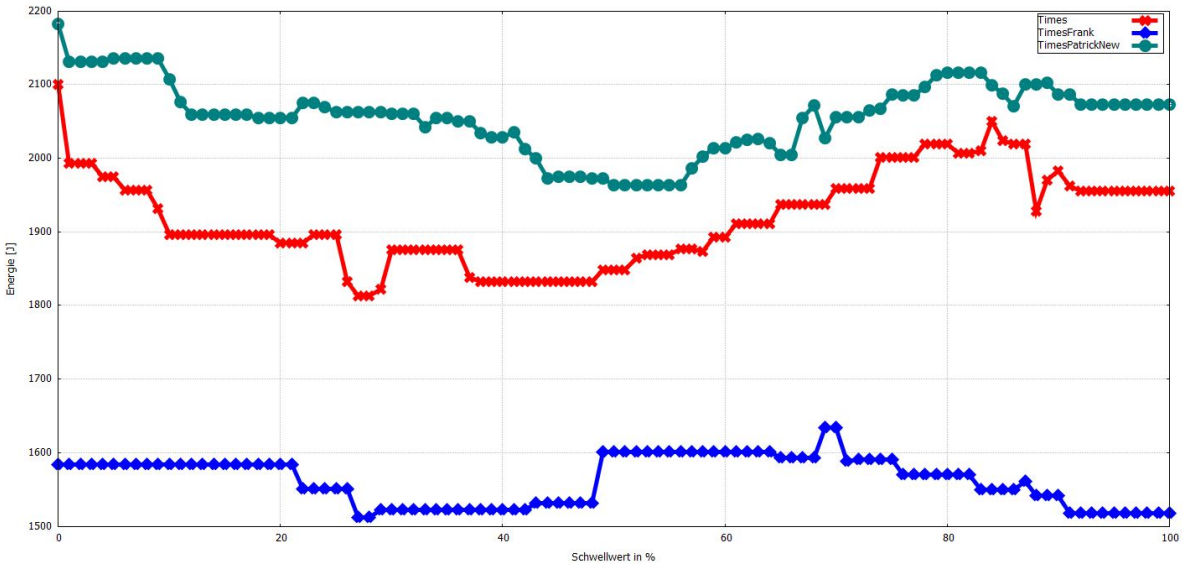


Abbildung 6.7: Die Abbildung zeigt den gesamten Energiebedarf des Algorithmus mit Early-Updates für die Datei Biketour.gpx.

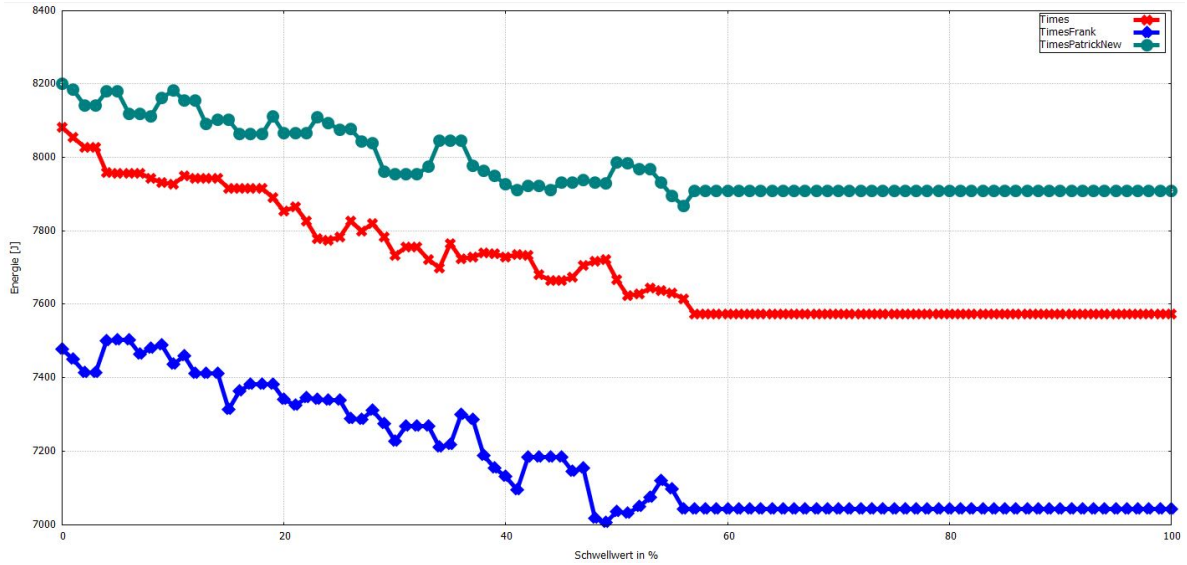


Abbildung 6.8: Die Abbildung zeigt den gesamten Energiebedarf des Algorithmus mit Early-Updates für die Datei 45_6h.gpx.

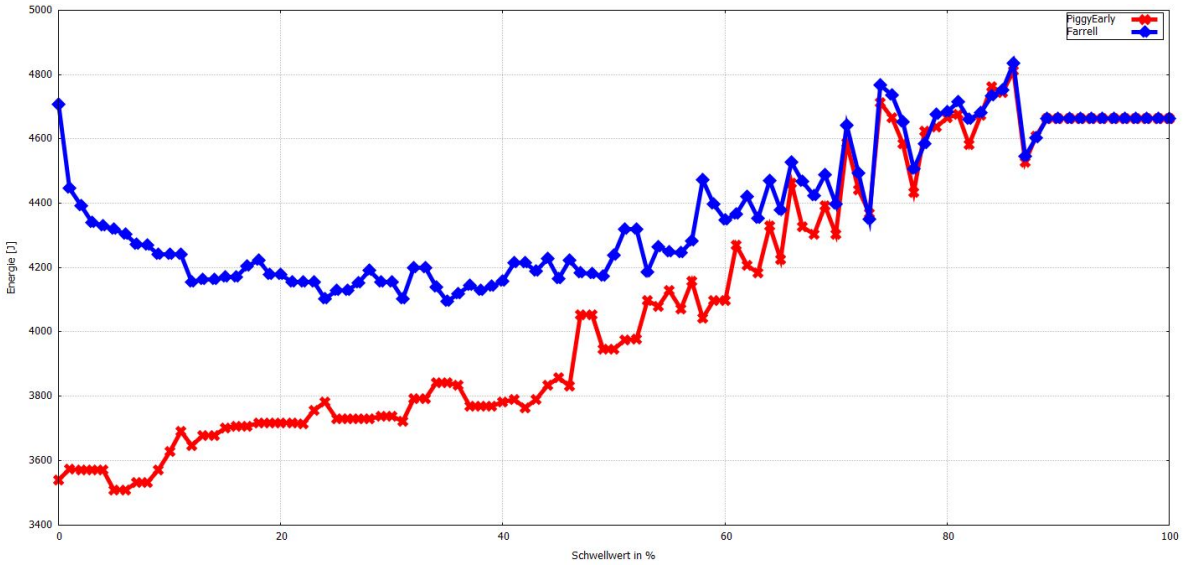


Abbildung 6.9: Die Abbildung zeigt den gesamten Energiebedarf

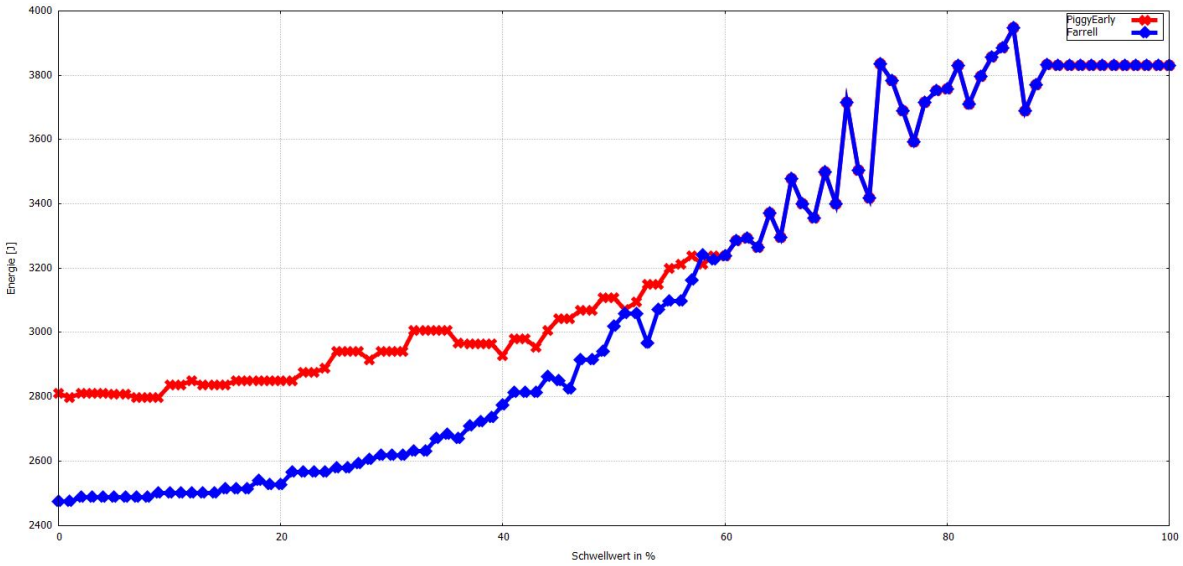


Abbildung 6.10: Die Abbildung zeigt den Energiebedarf für Positionsmessungen.

für Positions-Updates niedrig sind, dann sind die Kosten für die Messungen hoch. Ebenso gilt das Gegenteil.

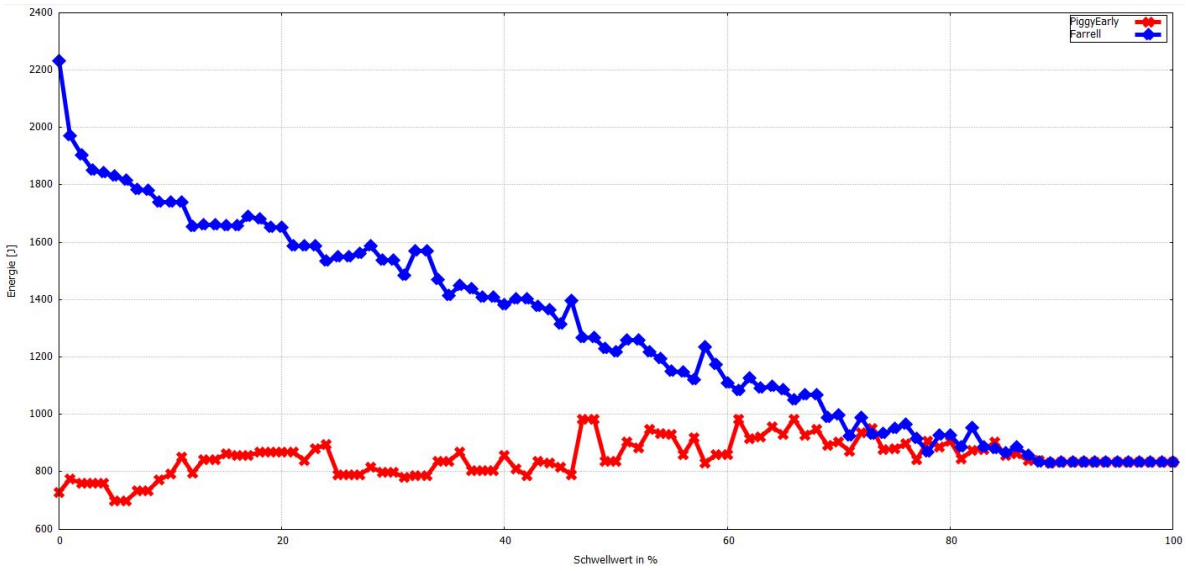


Abbildung 6.11: Die Abbildung zeigt den Energiebedarf für Update-Nachrichten.

6.2 Android Anwendung

Das Messgerät, das zum Bestimmen des Energiebedarfs des Smartphones benötigt wird, ist nicht portabel. Außerdem befindet es sich innerhalb des Informatikgebäudes. Daher sind keine GPS-Messungen mit dem Smartphone möglich.

Es kann jedoch ermittelt werden, wie viel Energie die Überwachung der Netzwerkschnittstelle für verschiedene Abtastfrequenzen benötigt.
Zusätzlich kann bestimmt werden, wie hoch der Energiebedarf für ein forced-Update und ein Piggyback-Update ist.

Für die Tests wird ein Galaxy Nexus verwendet, auf dem die Android Anwendung installiert ist. Das Messgerät wird direkt an den Akku des Smartphones angeschlossen und verbindet den Akku dann mit dem Smartphone. Mit einer Frequenz von 10 Hz werden dann Stromstärke und Spannung gemessen.

6.2.1 Test Durchführung

Um den Energiebedarf des Monitorings zu ermitteln, wird 60 Sekunden lang der Energiebedarf des Smartphones im Standby -Zustand gemessen. Anschließend wird eine modifizierte Version der Anwendung gestartet, die lediglich das Monitoring durchführt. Das Monitoring wird mit vier verschiedene Frequenzen je 60 Sekunden lang durchgeführt. Die verwendeten Frequenzen sind: 1 Hz, 10 Hz, 100 Hz und 1000 Hz. Das entspricht Abtastintervallen von 1000 ms, 100 ms, 10 ms und 1 ms. Bei allen Messungen ist das Display des Smartphones ausgeschaltet.

Abtastintervall in ms	Durchschnittliche Leistung in Watt	Energiebedarf in Joule
1	0,202	12,3
10	0,118	7,3
100	0,058	3,6
1000	0,057	3,4
Standby	0,078	4,7

Tabelle 6.4: Die Tabelle zeigt, wie viel Energie das Monitoring auf einem Galaxy Nexus benötigt

6.2.2 Ergebnisse und Diskussion

Tabelle 6.4 zeigt die Ergebnisse der Messungen des Energiebedarfs für das Monitoring.
Dabei fällt auf, dass der Energiebedarf im Ruhezustand höher ausfällt als für Abtastfrequenzen von 1 Hz und 10 Hz. Woran das genau liegt, ist unbekannt, wurde aber durch mehrfaches Wiederholen der Messungen bestätigt. Es könnte an der Ressourcenzuweisung des Betriebssystems liegen, dies ist allerdings pure Spekulation.
Ohne Zweifel kann jedoch festgestellt werden, dass für Abtastintervalle von 100 ms und 1000 ms keine höheren Kosten als im Ruhezustand auftreten. Diese Art von Monitoring hat somit keinen großen Einfluss auf den Energiebedarf des Algorithmus. Allerdings zeigt sich auch, dass für höhere Abtastfrequenzen die Energiekosten deutlich über dem Energiebedarf im Standby-Modus liegen.
Jedoch ist eine Abtastfrequenz von 10 Hz ausreichend, um Piggyback-Updates in der *tail time* von anderen Nachrichten versenden zu können.

7 Fazit

7.1 Zusammenfassung

In dieser Arbeit wurde ein Algorithmus entwickelt, der sowohl Kosten bei der Nutzung des GPS-Sensors als auch beim Versenden von Update-Nachrichten einspart.
Der Algorithmus gehört zur Klasse der distanzbasierten Update-Protokolle und basiert auf einem nichtlinearen Energiemodell. Das bedeutet, dass die Kosten für das Versenden einer Update Nachricht davon abhängig sind, wann zuletzt eine Nachricht über die Netzwehrschnittstelle versendet wurde. Dieses Wissen wird ausgenutzt, um gezielt Update-Nachrichten genau dann zu versenden, wenn gerade eine andere Anwendung eine Nachricht versendet hat. Eine Implementierung des Algorithmus in Java und ausführliche Tests mit einem Simulator haben ergeben, dass sich schon alleine mit dieser Strategie viel Energie einsparen lässt.
Doch zeigt die Simulation auch, dass der Algorithmus nicht besser als ein Standardalgorithmus für distanzbasierte Update-Protokolle abschneidet, wenn nicht genügend Nachrichten von anderen Anwendungen versendet werden.
Die Analyse der Traffic-Trace-Files hat ergeben, dass es durchaus öfter vorkommt, dass über lange Zeiträume hinweg keine Nachrichten von anderen Anwendungen versendet werden.
Um auch mit dieser Situation klar zu kommen, wurde der Algorithmus erweitert, so dass er Early-Updates senden kann, wenn keine Piggyback-Updates möglich sind. Diese Early-Updates verbrauchen zwar viel mehr Energie als Piggyback-Updates, aber es ist dennoch möglich, Energie mit ihnen einzusparen. Doch auch mit dieser Erweiterung verliert der Algorithmus viel von seinem Einsparpotential, wenn nicht genug Nachrichten von anderen Anwendungen versendet werden.
Die Simulation hat außerdem gezeigt, dass die Zeit zwischen zwei Messungen groß genug sein muss, damit Piggyback-Updates gesendet werden können. Wenn die Geschwindigkeit des mobilen Endgeräts im Vergleich zur gegebenen Garantie so hoch ist, dass nur wenige Sekunden zwischen den Updates vergehen, reicht diese Zeit nicht aus, um Piggyback-Updates zu versenden.
Hier wird dann die Verwendung eines distanzbasierten Update-Protokolls zum Nachteil, denn dadurch steigen die Kosten für Positionsmessungen stark an, da sich deren Frequenz schnell steigert, wenn sich das mobile Endgerät der durch die Garantie festgelegten Grenze nähert.

7.2 Zukünftige Forschung

Auch muss an der Eignung der Traffic-Trace-Files für die im Simulator getesteten Strecken gezweifelt werden, denn die Traffic-Trace-Files basieren auf der Altgasnutzung eines Smartphones. Wenn diese Altgasnutzung darin besteht über Instant-Messanger zu chatten und im Browser zu surfen, dann ist fraglich, ob sie auf eine Radtour übertragbar sind. Denn während einer Radtour gestaltet sich die

intensive Nutzung des Smartphones als schwierig.
Dies zeigt, dass die Ergebnisse aus dem Simulator nicht direkt auf die reale Welt übertragbar sind. Eine ausgiebige Evaluation in der realen Welt ist nötig, um abschließend zu klären, wie effektiv das Update-Protokoll wirklich ist.
Leider war dies im Zuge der Diplomarbeit nicht möglich, auch wenn eine funktionsfähige Android Applikation entwickelt worden ist, da die Messvorrichtung, die benötigt wird, um den Energieverbrauch des Smartphones zu bestimmen, nicht portabel ist.

Der Ansatz kann nicht nur mit einem einfachen distanzbasierten Update-Protokoll genutzt werden, sondern ist universell für jedes Update-Protokoll nutzbar, dass ein zellulares Netzwerk verwendet um Daten zu übertragen. Denkbar wäre es also, die effektivsten Ansätze, die im Moment existieren, mit diesem Ansatz zu erweitern.

Abschließend soll noch einmal das Potential dieses Ansatzes betont werden. Piggyback-Updates, die in der tail time anderer Nachrichten versendet werden, können mit vielen existierenden Ansätzen kombiniert werden.
Dabei beschränkt sich ihr Nutzen nicht nur auf Update-Protokolle.
Vielmehr kann der Ansatz für alle Anwendungen genutzt werden, die Nachrichten über ein zellulares Netzwerk übertragen.
Es wäre auch denkbar Nachrichten, die verzögert werden können, vorsätzlich zu verzögern, um sie dann als Piggyback-Nachrichten zu versenden. Dies könnte für ein Smartphone im Ganzen konzipiert werden, so dass ein umfassendes System entsteht, das beim Senden von beliebigen Nachrichten Energie einspart.

Literaturverzeichnis

[BDR13] P. Baier, F. Dürr, K. Rothermel. Opportunistic Position Update Protocols for Mobile Devices. *ACM*, 2013. (Zitiert auf den Seiten 3, 7, 10, 12, 24, 26, 31, 39 und 43)

[CTSC11] Y. Chon, E. Talipov, H. Shin, H. Cha. Mobility Prediction-based Smartphone Energy Optimization fpr Everyday Location Monitoring. In *SenSys'11*. ACM, 2011. (Zitiert auf den Seiten 9, 12, 20 und 21)

[FCR07] T. Farrell, R. Cheng, K. Rothermel. Energy-Efficient Monitoring of Mobile Objects with Uncertainty-Aware Tolerance. *IEEE*, 2007. (Zitiert auf Seite 20)

[FLR07] T. Farrell, R. Lange, K. Rothermel. Energy-efficient Tracking of Mobile Objects with Early Distance-based Reporting. *IEEE*, 2007. (Zitiert auf den Seiten 12, 20, 22, 34, 35, 45 und 49)

[GPSa] http://www.kowoma.de/gps/Signale.htm. (Zitiert auf den Seiten 7, 15 und 17)

[GPSb] http://www.leica-geosystems.com/downloads123/zz/gps/gps_system500/manuals/GPSBasics_de.pdf. (Zitiert auf den Seiten 7, 15 und 17)

[GPSc] http://www.gps.gov/systems/gps/space/. (Zitiert auf den Seiten 14 und 17)

[GPSd] http://www.pocketgpsworld.com/howgpsworks.php. (Zitiert auf den Seiten 14, 16 und 17)

[HSE07] H. Haverinen, J. Siren, P. Eronen. Energy Consumption of Always-On Applications in WCDMA Networks. In *VTC Spring*, S. 964–968. IEEE, 2007. (Zitiert auf den Seiten 24 und 30)

[KLGT09] M. B. Kjærgaard, J. Langdal, T. Godsk, T. Toftkjær. EnTracked: energy-efficient robust position tracking for mobile devices. In *MobiSys*, S. 221–234. ACM, 2009. (Zitiert auf den Seiten 39 und 45)

[LR01] A. Leonhardi, K. Rothermel. Energy-efficient Tracking of Mobile Objects with Early Distance-based Reporting. In *Cluster Computing*, Band 4, S. 355–367. Springer-Verlag, 2001. The original publication is available at www.springerlink.com: http://www.springerlink.com/content/q2452208kr384402. (Zitiert auf den Seiten 9, 18 und 19)

[LWG+09] R. Lange, H. Weinschrott, L. Geiger, A. Blessing, F. Dürr, K. Rothermel, H. Schütze. On a Generic Uncertainty Model for Position Information. *Springer-Verlag*, 2009. (Zitiert auf Seite 29)

[PKG10] J. Paek, J. Kim, R. Govindan. Energy-Efficient Rate Adaptive GPS-based Positioning for Smartphones. In *MobiSys'10*. ACM, 2010. (Zitiert auf Seite 17)

[QWG+10] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck. Characterizing Radio Resource Allocation for 3G Networks. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, S. 137–150. ACM, 2010. (Zitiert auf den Seiten 24 und 30)

[ZZZL13] D. Zhang, Y. Zhang, Y. Zhou, H. Liu. Leveraging the Tail Time for Saving Energy in Cellular Networks. *IEEE*, 2013. (Zitiert auf den Seiten 24 und 30)

Alle URLs wurden zuletzt am 10. 02. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift