Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3318

Location Update Algorithms for Position Sharing

Simon Haenle

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	M. Sc. Pavel Skvorzov
begonnen am:	16. April 2012
beendet am:	16. Oktober 2012
CR-Klassifikation:	C.2.4, H.2.0, H2.8, H.3.5

Kurzfassung

Mit der immer zunehmenden Anzahl an Smartphones innerhalb der letzten Jahre hat auch deren Einfluss auf unser tägliches Leben immer mehr zugenommen. Unter anderem wurden auch die sogenannten Location Based Services immer beliebter. Diese Location Based Services benötigen die Position eines mobilen Benutzers und finden dann z. B. das nächste Restaurant. Natürlich ist die Sicherheit der Benutzerdaten hierbei ein wichtiges Thema. Die Daten sind meistens auf Location Servern gespeichert, und wenn der Anbieter des Servers kompromittiert oder unsicher ist, können die Daten verschlampt oder gestohlen werden. Aus diesem Grund wurden viele Ansätze zum Verschleiern der tatsächlichen Benutzerposition vorgeschlagen, so auch von Rothermel, Dürr und Skvortsov. Diese Diplomarbeit stellt nun neue Techniken vor, um den eben genannten Basis-Ansatz zum Verschleiern von Benutzerpositionen weiter zu verbessern. Das Hauptziel ist dabei, die Anzahl der Update-Nachrichten zu reduzieren, die vom mobilen Objekt an die Location Server geschickt werden, um die Position des Benutzers aktuell zu halten. Die vorgeschlagenen Techniken werden analysiert, und das entwickelte System wird simuliert, um die Kommunikation zwischen dem mobilen Objekt und den Location Servern, sowie die Kommunikation zwischen den Location Servern und den Location Based Services auszuwerten. Hierbei werden echte Benutzer-Trajektorien aus dem GeoLife-Datensatz verwendet. Die Evaluierung zeigt, dass die Nachrichtenzahl in manchen Fällen um über 90% reduziert werden kann. Hierbei findet jedoch im Hinblick auf den Basis-Ansatz weder ein Verlust der Sicherheit noch der Genauigkeit statt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Algorithmen

1 Einleitung

Smartphones sind aus dem heutigen Leben nicht mehr wegzudenken. Die immer besser werdende Rechenleistung und technische Ausstattung dieser Geräte eröffnet neue Möglichkeiten für alle Arten von Anwendungen (die sogenannten „Apps“). Auch ist es zum Beispiel mittels GPS (Global Positioning Service) möglich, die aktuelle geografische Position des Benutzers festzustellen. So wurden auch sogenannte LBS (Location Based Services), die auf die Positionsdaten des Benutzers zugreifen und z. B. den nächsten umliegenden Supermarkt anzeigen (Point of Interest Search) oder eine Benachrichtigung schicken, sobald sich ein Freund in der Umgebung aufhält, innerhalb der letzten Jahre immer populärer. Auch soziale Netzwerke wie z. B. Facebook bieten solche Dienste an, damit entsprechende Benutzer besuchte Orte markieren können (Ich war mit Person X und Person Y an Ort Z).
Da aber nicht jeder Benutzer möchte, dass seine Daten veröffentlicht oder an Dritte weitergegeben werden, kommen Sicherheitsfragen auf. Die mobilen Geräte schicken die Positionsdaten meistens nicht direkt an die LBS, sondern speichern die Positionsdaten auf sogenannten Location-Servern (LS) zwischen, auf die die LBS Zugriff haben. Ist nun einer der Anbieter des Location-Servers kompromittiert und verkauft Daten an Dritte, oder leicht angreifbar, kann es passieren, dass Positionsdaten gestohlen oder „verschlampt“ werden. Deshalb dürfte der Location-Server die exakte Benutzerposition eigentlich zu keinem Zeitpunkt kennen, da er in diesem Sinne einen Single Point of Failure darstellt.
Ein weiteres wichtiges Forschungsgebiet in diesem Umfeld beschäftigt sich neben der Erhöhung der Sicherheit der Benutzerdaten auch mit der Senkung des Kommunikationsaufwands zwischen dem mobilen Gerät und dem Location-Server. Wenn man z. B. alle 5 Sekunden eine Update-Nachricht, die die aktuelle Benutzerposition beinhaltet, an den Location-Server sendet, sich der Benutzer in der Zwischenzeit aber gar nicht oder nur wenig bewegt hat, so war die Update-Nachricht eigentlich überflüssig und hat lediglich zur Erhöhung der Kommunikationskosten beigetragen.
Je nachdem, welcher LBS benutzt wird, ändern sich auch die Anforderungen an die Genauigkeit der Positionsdaten: So reicht es bei manchen LBS wie z. B. bei einem Dienst, der das nächstgelegene Restaurant findet, nur die ungefähre Benutzerposition preiszugeben (Ich befinde mich innerhalb des Kreises mit dem Radius X, finde das nächste Restaurant innerhalb dieses Kreises). Ein Navigationssystem hingegen benötigt jedoch stets aktuelle Positionsupdates um eine angemessene Service-Qualität anbieten zu können, und muss daher zum einen die exakte Benutzerposition kennen, sowie in kürzeren Zeiträumen mit Update-Nachrichten versorgt werden, um diese Genauigkeit beizubehalten.
Diese beiden Beispieldienste lassen sich in die Kategorien „Sporadic Updates“ und „On-the-fly tracking“ einordnen. Abbildung 1.1 veranschaulicht die beiden Fälle grafisch.

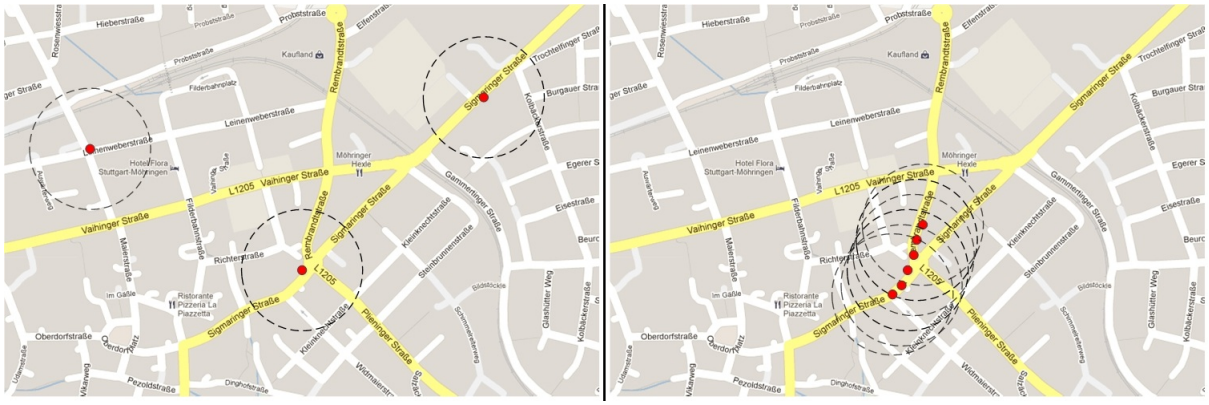


Abbildung 1.1: Sporadic Position Updates (links) und On-the-fly tracking (rechts)

Beim on-the-fly tracking gibt es die verschiedensten Update-Strategien, welche ausführlich in Kapitel 2 vorgestellt werden. Die meisten dieser Strategien zielen darauf ab, die Anzahl an benötigten Update-Nachrichten möglichst gering zu halten, um die Kommunikation zwischen dem mobilen Objekt(MO) und dem Location-Server zu schonen. So kann beispielsweise ein Zeitintervall t_Δ festgelegt werden, nach dessen Ablauf eine Update-Nachricht geschickt wird. Auch kann eine maximale Distanz d festgelegt werden, nach deren Überschreitung durch den Benutzer eine Update-Nachricht geschickt wird. Jede dieser Strategien hat ihre Vor- und Nachteile und meistens muss man einen Mittelweg zwischen hinreichender Genauigkeit der Positionsdaten und geringer Anzahl an Update-Nachrichten finden.
Es gibt wie bereits erwähnt viele Ansätze, die unter anderem darauf abzielen, die Sicherheit der Benutzerdaten zu erhöhen oder den Kommunikationsaufwand zwischen dem mobilen Objekt und dem Location-Server zu senken. In Kapitel 2 werden einige dieser Ansätze vorgestellt.
Als Grundlage dieser Diplomarbeit dient ein solcher Ansatz, nämlich [DSR11], der in Kapitel 3 ausführlich vorgestellt wird. Das Hauptziel dieser Diplomarbeit ist es, diesen Ansatz so zu verbessern, dass die Kommunikationskosten zwischen dem mobilen Objekt und den Location-Servern gesenkt wird, und so insbesondere unnötige Update-Nachrichten vermieden werden. Da das Hauptthema dieser Diplomarbeit allerdings nicht die Sicherheit ist, sondern die Reduktion von Update-Nachrichten, werden Sicherheitsprobleme nur am Rande diskutiert.

1.1 Gliederung

Diese Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen und Verwandte Arbeiten: Hier werden verwandte Arbeiten vorgestellt und somit auch grundlegenden Begriffe eingeführt.

Kapitel 3 – Basisansatz: Hier wird im Detail die Arbeit vorgestellt, welche dieser Diplomarbeit als Grundlage dient.

Kapitel 4 – Ansätze zur Reduktion des Kommunikationsaufwandes: Hier werden Verbesserungsvorschläge für den in Kapitel 3 Ansatz vorgestellt und diskutiert.

Kapitel 5 – Analyse und Auswertung der Ansätze zur Reduktion des Kommunikationsaufwandes: Hier werden die in Kapitel 4 vorgestellten Verbesserungsvorschläge analysiert und ausgewertet.

Kapitel 6 – Evaluierung: Hier wird das entwickelte System simuliert und die Ergebnisse werden vorgestellt und diskutiert.

Kapitel 7 – Zusammenfassung und Ausblick: Hier werden die Ergebnisse der Arbeit zusammengefasst und Anknüpfungspunkte vorgestellt.

2 Grundlagen und Verwandte Arbeiten

In diesem Kapitel werden grundlegende Begriffe und Techniken beschrieben, die für das Verständnis dieser Arbeit essentiell sind. Im Anschluss wird ein Überblick über verwandte Arbeiten, sowie ein abschließendes Fazit gegeben.

2.1 Grundlagen

2.1.1 Überblick über Location-Update-Protokolle

Zunächst sollen die verschiedenen Arten von Update-Protokollen vorgestellt werden. Ein guter Überblick über die verschiedenen Location-Update-Protokolle wird in [LR01] gegeben: Abbildung 2.1 stellt die verschiedenen Protokollarten dar, die sich in in die folgenden drei

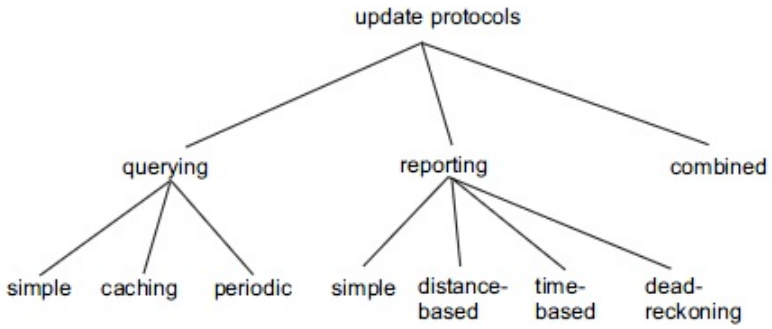


Abbildung 2.1: Überblick über die verschiedenen Protokollarten (Quelle: [LR01])

Oberkategorien einordnen lassen: „Querying Protocols“, in denen der Server entscheidet, wann er ein Location-Update der Quelle haben möchte, „Reporting Protocols“, bei denen die Entscheidung eines Location-Updates allein bei der Quelle liegt, sowie ein „Combined Protocol“, das die beiden zuvor genannten Protokolle miteinander vereint. Des Weiteren wird auf die einzelnen Klassen eingegangen, in die man die genannten Protokollarten einteilen kann. Die Querying-Protokolle lassen sich klassifizieren in *simple*, *caching* und *periodic*.

In *Simple*-Protokollen verlangt der Server immer dann ein Location-Update von der Quelle, wenn er die aktuelle Position z. B. für eine Anwendung benötigt. Dies versichert zwar einerseits, dass die Position immer akkurat ist, kann jedoch zu einer hohen Anzahl an Update-Nachrichten führen, wenn die Information häufig benötigt wird.

In *Caching*-Protokollen, die eine Optimierung der Simple-Protokolle darstellen, speichert der Server stets eine Kopie der letzten übermittelten Positionsdaten. Wird nun die Position der Quelle benötigt, nimmt der Server eine Abschätzung de Genauigkeit vor. Bleibt das Ergebnis unter einer vorher festgelegten Schranke ϵ, so leitet der Server die gespeicherte Information weiter, andernfalls fordert er ein Update seitens der Quelle an. Die Abschätzung der Genauigkeit kann sowohl pessimistisch als auch optimistisch erfolgen: bei der pessimistischen Variante wird berechnet, wie weit sich die Quelle seit dem letzten Update mit ihrer maximalen Geschwindigkeit bewegt haben könnte (die maximale Geschwindigkeit muss natürlich vorher festgelegt werden), bei der optimistischen Variante hingegen wird von der Durchschnittsgeschwindigkeit der Quelle ausgegangen. Somit kann bei der optimistischen Variante der Fall eintreten, dass die Positionsabweichung zwischen der letzten gespeicherten Position und der tatsächlichen Position die Schranke ϵ überschreitet (nämlich dann, wenn sich die Quelle seit dem letzten Update mit einer größeren Geschwindigkeit als der Durchschnittsgeschwindigkeit bewegt hat). Es ist offensichtlich, dass bei der optimistischen Variante weniger Update-Nachrichten gesendet werden müssen als bei der pessimistischen Variante.

Periodic-Protokolle zeichnen sich dadurch aus, dass der Server immer nach einem festgelegten Zeitintervall ein Location-Update von der Quelle anfordert. Aufgrund der starken Ähnlichkeit zum time-based Reporting-Protokoll wird an dieser Stelle nicht näher auf die Eigenschaften des Protokolls eingegangen.

Die Reporting Protokolle lassen sich klassifizieren in *simple*, *time-based*, *distance-based* und *Dead-Reckoning*.
In *Simple*-Protokollen wird einfach immer dann upgedatet, wenn ein Sensor-System eine Änderung der Position festgestellt hat. Dies kann (z. B. bei einem sehr genauen Sensor-System) zu einer sehr hohen Anzahl an Update-Nachrichten führen.

In *time-based* Protokollen wird immer nach einem festgelegten Zeitintervall T eine Update-Nachricht an den Server gesendet. Die Genauigkeit der Positionsdaten hängt hierbei von der Geschwindigkeit der Quelle ab: Bewegt sich die Quelle mit einer hohen Geschwindigkeit bei gleichzeitigem hohen Zeitintervall T, so kann die tatsächliche Position der Quelle stark von der zuletzt auf dem Server gespeicherten Position abweichen.

In *distance-based* Protokollen wird immer dann eine Update-Nachricht der Quelle gesendet, wenn die geografische Distanz seit dem letzten Update einen festgelegten Grenzwert D überschreitet. Dieses Protokoll eignet sich also hervorragend für mobile Objekte, die sich langsam oder kaum bewegen.

Die *Dead-Reckoning* Protokolle stellen eine Optimierung der distance-based Protokolle dar, indem der Server die aktuelle Position der Quelle anhand ihrer letzten Position, ihrer Geschwindigkeit und ihrer Bewegungsrichtung vorausberechnet. Die Quelle berechnet diese Position ebenfalls, und sendet eine Update-Nachricht, wenn die geografische Differenz zwischen der tatsächlichen Position und der berechneten Position größer als eine festgelegte

Schranke ϵ ist. Dieses Protokoll funktioniert gut, sofern sich die Quelle mit einer konstanten Geschwindigkeit in eine festgelegte Richtung bewegt, oder wenn das Ziel der Quelle bekannt ist.

Das von Leonhardi und Rothermel vorgeschlagene *combined*-Protokoll, das die Ideen der distance-based Reporting-Protokolle und die der Querying-Protokolle vereint, funktioniert wie folgt: der Grundansatz ist derselbe wie bei den distance-based Protokollen, jedoch kann der Server im Falle ungenügender Genauigkeit hinsichtlich der letzten gespeicherten Position jederzeit eine Update-Nachricht anfordern. Um die Anzahl an Update-Nachrichten zu minimieren, kann der Grenzwert D (aus dem distance-based Protokoll) entsprechend der Mobilität der Quelle dynamisch angepasst werden.

Des Weiteren wird auf das Verhalten der Protokolle im Falle eines Verbindungsabbruchs zwischen der Quelle und dem Server eingegangen, und es wird ein analytischer Vergleich der Protokolle hinsichtlich Genauigkeit und Anzahl an Update-Nachrichten durchgeführt. Die Ergebnisse des analytischen Vergleichs lassen sich wie folgt zusammenfassen: generell lässt sich sagen, dass ein distance-based Protokoll besser abschneidet als ein time-based Protokoll, und ein optimistisches caching Querying-Protokoll besser als ein pessimistisches. Die Anzahl an benötigten Update-Nachrichten erhöht sich bei den Querying-Protokollen mit der Anzahl der Anfragen, deshalb sind die Querying-Protokolle bei einer niedrigen Query-Anzahl besser als die Reporting-Protokolle. Wenn die Query-Rate hoch ist, ist das distance-based Reporting Protokoll besser als das pessimistische caching Querying-Protokoll. Zur Genauigkeit wurden folgende Aussagen getroffen: Das pessimistische Querying-Protokoll kann eine festgelegte Schranke ϵ unabhängig von der Geschwindigkeit der Quelle immer einhalten. Das optimistische Querying-Protokoll hingegen ist mit einer ständigen Durchschnitts-Ungenauigkeit behaftet. Das time-based Reporting Protokoll besitzt eine kleine Durchschnitts-Ungenauigkeit für niedrige Bewegungsgeschwindigkeiten, wohingegen das distance-based Reporting Protokoll eine konstante Durchschnitts-Ungenauigkeit für alle Geschwindigkeiten besitzt. Im Allgemeinen sind die Reporting-Protokolle nicht geeignet, wenn der Client flexibel die Schranke ϵ bestimmen kann.
Die Performance des combined-Protokolls hängt von dem Wert ab, der bestimmt, ob die letzte auf dem Server gespeicherte Position upgedatet wird, sowie von der Query-Rate. Bei hoher Query-Rate ist es weniger effizient als das distance-based Reporting Protokoll, bei niedriger Query-Rate hingegen benötigt es weitaus weniger Update-Nachrichten. Im Gegensatz zu den Reporting-Protokollen unterstützt es die Einhaltung einer flexibel festgelegten Schranke ϵ für die Genauigkeit.
Zuletzt wurde eine Simulation durchgeführt, in der mit mehreren GPS-Trajektorien, welche bei Autofahrten in und um Stuttgart herum aufgezeichnet wurden, die oben genannten Protokolle getestet wurden. Die getroffenen Aussagen des analytischen Vergleichs der Protokolle wurden hierdurch bestätigt, auch wenn einige Messwerte leicht von den getroffenen Aussagen abwichen.

2.1.2 Überblick über Dead-Reckoning Protokolle

In [LNR02] erhält man einen guten Überblick über die verschiedenen Dead-Reckoning Protokolle: ein Dead-Reckoning Protokoll ist ein Protokoll, das versucht, die Position des mobilen Objekts (der Quelle) anhand seiner letzten gespeicherten Position, der Geschwindigkeit des mobilen Objekts, und der Bewegungsrichtung des mobilen Objekts zu berechnen. Das mobile Objekt führt diese Berechnung ebenfalls durch, und sendet eine Update-Nachricht, sofern die berechnete Position mehr als um eine vorher festgelegte Schranke ϵ abweicht.
Eine Übersicht über die verschiedenen Dead-Reckoning Protokolle gibt Abbildung 2.2:

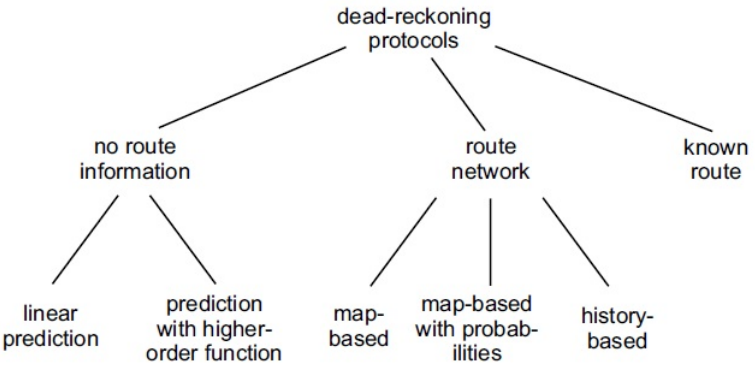


Abbildung 2.2: Übersicht über die verschiedenen Dead-Reckoning Protokolle (Quelle: [LNR02])

Die Protokolle lassen sich wie folgt klassifizieren:
Linear predicition: Das einfachste denkbare Dead-Reckoning Protokoll. Es wird von einer linearen Bewegung des mobilen Objekts ausgegangen. Es ist einfach zu implementieren und benötigt in vielen Fällen (z. B. Auto auf der Autobahn) weitaus weniger Update-Nachrichten als die bereits zuvor erwähnten Update-Protokolle.

Higher-order prediciton function: Hier ist das Protokoll dazu in der Lage, auch nichtlineare Bewegungen des mobilen Objekts, wie z. B. Kurven oder Splines, vorauszuberechnen. Ebenfalls kann die Geschwindigkeit des mobilen Objekts anhand der gemessenen Beschleunigung vorausberechnet werden.

Map-based Dead-Reckoning: Oft bewegt sich ein mobiles Objekt entlang eines Straßennetzwerks, wie z. B. ein Auto, das durch die Stadt fährt. Die map-based Dead-Reckoning Protokolle versuchen nun, die Position des mobilen Objekts auf eine Karte der Umgebung abzubilden. Solche Karten kann man z. B. aus Navigationssystemen extrahieren. Das Protokoll muss nun an jeder Kreuzung entscheiden, in welche Richtung sich das mobile Objekt am wahrscheinlichsten bewegen wird.

Map-based with probability information: Diese Verbesserung des map-based Dead-Reckoning Protokolls benutzt Wahrscheinlichkeitsinformationen an den Kreuzungen. Dies kann

Benutzer-unabhängig erfolgen (wie viel Prozent aller Benutzer wählen an Kreuzung X die Abzweigung Y), oder auch Benutzer-spezifisch (Wie oft wählt Benutzer X an Kreuzung Y die Abzweigung Z). Natürlich ist die Beschaffung dieser Wahrscheinlichkeitsinformationen mit erheblichem Aufwand verbunden.

History-based Dead-Reckoning: In dieser Variante wird über einen längeren Zeitraum hinweg ein Bewegungsprofil des Benutzers aufgezeichnet. Da viele Benutzer einen geregelten Tagesablauf haben (z. B. morgens zur Arbeit fahren und abends wieder nach Hause), ist die Performanz dieser Variante äquivalent zum "map-based Dead-Reckoning with probability information". Ebenfalls kann die Erstellung des Profils Benutzer-spezifisch oder Benutzer-unabhängig erfolgen.

Dead-Reckoning with known route: Ist die Route des mobilen Objekts erkannt, muss lediglich die Geschwindigkeit des mobilen Objekts beachtet werden, da das Protokoll weiß, wann der Benutzer welche Abzweigung wählen wird. In diesem Fall verhält sich das Protokoll wie das map-based Dead-Reckoning mit optimaler Vorhersage (die vorausberechnete Route des Benutzers stimmt mit der tatsächlichen Route überein).

2.1.3 Lineares Dead-Reckoning

Positionsberechnung

Da lineares Dead-Reckoning ein wichtiger Ansatz zur Einsparung von Update Nachrichten ist, soll zunächst ein Grundverständnis darüber aufgebaut werden, wie lineares Dead-Reckoning funktioniert und wie ein entsprechender Algorithmus aussehen könnte. Da von einer linearen Bewegung des mobilen Objekts ausgegangen wird, dient als Grundformel der Berechnung einer zurückgelegten Strecke s mit der Bewegungsgeschwindigkeit v innerhalb der Zeit t:

(2.1) $s = v * t$

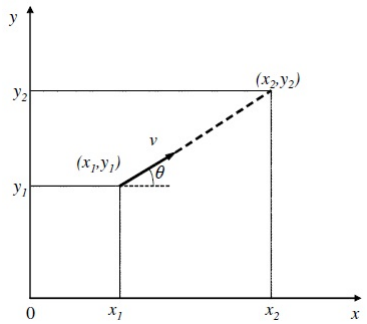


Abbildung 2.3: Euklidische Distanz zwischen zwei Punkten (Quelle: [SYZ05])

Überträgt man dies auf den $\mathbb{R}^2$, so lässt sich die Strecke s zwischen zwei Punkten $p_i = (x_i,y_i)$ und $p_{i-1}= (x_{i-1},y_{i-1})$ mithilfe des Satzes von Pythagoras anhand folgender Formel berechnen (zur Veranschaulichung siehe Abbildung 2.3):

$$s(p_{i-1}, p_i) = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{2.2}$$

Anhand der zurückgelegten Strecke des mobilen Objekts zwischen Zeitpunkt t_i und t_{i-1} lässt sich dann die Geschwindigkeit des mobilen Objekts innerhalb dieser Zeit berechnen:

$$v = \frac{s}{(t_i - t_{i-1})} \tag{2.3}$$

Würde sich [LDR11] lediglich auf Mobiltelefone beschränken, könnte man sich die Geschwindigkeitsberechnung sparen, da moderne Mobiltelefone meistens einen eingebauten Geschwindigkeitsmesser besitzen. Da aber im Allgemeinen lediglich von mobilen Objekten ausgegangen wird, ist die Geschwindigkeit v des mobilen Objekts zunächst unbekannt. Die Berechnung von v nach Formel 2.3 kann also erst erfolgen, wenn mindestens zwei frühere Positionen $p_i = (x_i,y_i)$ und $p_{i-1} = (x_{i-1},x_{i-1})$ des mobilen Objekts bekannt sind. Für die Berechnung der darauffolgenden Position $p_{i+1} = (x_{i+1},y_{i+1})$ wird dann jeweils die Geschwindigkeit verwendet, die das mobile Objekt während der Bewegung von $p_{i-1} = (x_{i-1},y_{i-1})$ nach $p_i = (x_i,y_i)$ besessen hat.
Zur Vorausberechnung einer Position $p_{i+1} = (x_{i+1},y_{i+1})$ muss nun noch der Winkel θ in die Rechnung mit einbezogen werden. Möchte man von der zurückgelegten Strecke $s = (p_i, p_{i+1})$ den Anteil s_x in X-Richtung bestimmen, so gilt:

$$s_x = \cos(\theta) * s(p_i, p_{i+1}) \tag{2.4}$$

und analog gilt für den Anteil s_y in Y-Richtung:

$$s_y = \sin(\theta) * s(p_i, p_{i+1}) \tag{2.5}$$

Die Strecke $s(p_i, p_{i+1})$ wird wie folgt berechnet:

$$s(p_i, p_{i+1}) = v * (t_{i+1} - t_i) \tag{2.6}$$

Die Geschwindigkeit v ist, wie bereits erwähnt, die Geschwindigkeit, die das mobile Objekt während des Zurücklegens der Strecke $s = (p_{i-1}, p_i)$ besessen hat. Der Zeitpunkt t_{i+1} ist der erwartete Ankunftszeitpunkt, an dem das mobile Objekt an Position $p_{i+1} = (x_{i+1}, y_{i+1})$ ankommt. Er dient gleichzeitig als Tuning-Parameter für den Algorithmus. Bei kleineren t_{i+1} sind logischerweise die Berechnungsintervalle kürzer, und das mobile Objekt vergleicht seine tatsächliche Position öfters mit der berechneten Position. Als Folge werden häufiger Update-Nachrichten gesendet, allerdings ist auch die Genauigkeit der vorausberechneten Position höher. Bei größeren t_{i+1} verhält sich das Ganze genau andersrum und es werden weniger Update-Nachrichten gesendet, was allerdings durch eine kleinere Genauigkeit der vorausberechneten Position erkauft wird.
Die Formel zur Berechnung von θ lässt sich mittels trigonometrischer Grundlagenrechnung wie folgt herleiten:

$$\cos(\theta) = \frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)} \tag{2.7}$$

Auflösen nach θ ergibt

$$\theta = \arccos\left(\frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)}\right) \tag{2.8}$$

Man beachte, dass Formel 2.8 nur auf positive Argumente anwendbar ist. Für negative Argumente gilt:

$$\theta = \pi - \arccos\left(\frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)}\right) \tag{2.9}$$

Die Formeln 2.8 und 2.9 lassen sich also wie folgt zusammenfassen:

$$\theta = \begin{cases} \arccos\left(\frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)}\right), & \text{falls } \frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)} > 0 \\ \pi - \arccos\left(\frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)}\right), & \text{falls} \frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)} < 0 \end{cases} \tag{2.10}$$

Für die Berechnung von θ gilt analog dieselbe Annahme wie zur Berechnung von v (θ ist der Winkel, den das mobile Objekt beim Zurücklegen der Strecke $s = (p_{i-1}, p_i)$ mit der X-Achse eingeschlossen hat).
Hat man also Kenntnis von mindestens zwei Positionen p_i = (x_i,y_i) und p_{i-1} = (x_{i-1},x_{i-1}) des mobilen Objekts inklusive den Zeitstempeln t_i und t_{i-1}, so lassen sich die Koordinaten einer weiteren Position p_{i+1} = (x_{i+1},y_{i+1}) wie folgt vorausberechnen:

(2.11)

$$X_{i+1} = \begin{cases} x_i + \cos\left(\arccos\left(\frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)}\right)\right) * \frac{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}}{(t_i - t_{i-1})} * (t_{i+1} - t_i), & \text{falls } \frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)} > 0 \\ x_i + \cos\left(\pi - \arccos\left(\frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)}\right)\right) * \frac{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}}{(t_i - t_{i-1})} * (t_{i+1} - t_i), & \text{falls} \frac{xi - x_{i-1}}{s(p_{i-1}, p_i)} < 0 \end{cases}$$

und analog

(2.12)

$$Y_{i+1} = \begin{cases} y_i + \sin\left(\arccos\left(\frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)}\right)\right) * \frac{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}}{(t_i - t_{i-1})} * (t_{i+1} - t_i), & \text{falls } \frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)} > 0 \\ y_i + \sin\left(\pi - \arccos\left(\frac{x_i - x_{i-1}}{s(p_{i-1}, p_i)}\right)\right) * \frac{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}}{(t_i - t_{i-1})} * (t_{i+1} - t_i), & \text{falls} \frac{xi - x_{i-1}}{s(p_{i-1}, p_i)} < 0 \end{cases}$$

Die Formeln 2.11 und 2.12 entstehen durch jeweiliges Einsetzen von 2.2, 2.3, 2.6 und 2.10 in 2.4 und 2.5. Des Weiteren wird jeweils die X- bzw. Y-Koordinate der Vorgängerposition addiert.

Algorithmus

Da nun bekannt ist, wie eine Position vorausberechnet werden kann und welche Parameter dafür notwendig sind, kann ein einfacher linearer Dead-Reckoning Algorithmus implementiert werden. Algorithmus 2.1, der als Beispiel für einen einfachen linearen Dead-Reckoning Algorithmus dienen soll, erhält als Parameter die beiden Vorgängerpositionen p_i und p_{i-1},

Algorithmus 2.1 Linearer Dead-Reckoning Algorithmus (client)

```
1: function LINEAR_DEAD_RECKONING(p_i, p_{i-1}, t_i, t_{i-1}, update_interval)
2:     while (execute == 1) do
3:         t_{i+1} ← actual_time + update_interval
4:         p_{i+1} ← calculate_next_position(p_i, p_{i-1}, t_i, t_{i-1}, t_{i+1})
5:         if (system_time == t_{i+1}) then
6:             s_1 ← sense_position()
7:             if (|s_1 − p_{i+1}| > ε) then
8:                 send_update_message(s_1)
9:             end if
10:        end if
11:    end while
12: end function
```

die dazugehörigen Zeitstempel t_i und t_{i-1}, sowie die Variable update_interval, die vom Benutzer als Update-Frequenz bestimmt werden kann. Der Algorithmus wird solange ausgeführt, bis der Benutzer ihn abschaltet (1). Im ersten Schritt (2) wird nun der nächste Zeitpunkt für die zu vorauszuberechnende Position festgelegt. Zunächst wird die Position p_{i+1}, an dem sich das mobilen Objekts zum Zeitpunkt t_{i+1} befinden wird, gemäß der Formeln 2.11 und 2.12 vorausberechnet (3). Wenn dann der Zeitpunkt t_{i+1} erreicht ist, misst das mobile Gerät seine aktuelle Position mittels GPS (4). In (5) wird dann verglichen, ob die vorausberechnete Position mehr als ϵ von der tatsächlichen Position abweicht. ϵ ist vom Benutzer vorgegeben und gibt an, wie weit die tatsächliche Position maximal von der aktuellen Position abweichen darf. Die Abweichung wird in (7) berechnet, und wenn die Schranke ϵ überschritten wird, wird eine Update-Nachricht mit der durch GPS bestimmten Position an den Server geschickt (8).

Algorithmus 2.2 Linearer Dead-Reckoning Algorithmus (server)

```
1: function LINEAR_DEAD_RECKONING(p_i, p_{i-1}, t_i, t_{i-1}, t_{i+1})
2:     while execute == 1 do
3:         p_{i+1} ← calculate_next_position(p_i, p_{i-1}, t_i, t_{i-1}, t_{i+1})
4:         if (system_time == t_{i+1}) then
5:             p_{i-1} ← p_i;  p_i ← p_{i+1};
6:         end if
7:     end while
8: end function
```

Auf dem Server läuft eine leicht abgewandelte Version des Algorithmus, nämlich Algorithmus 2.2. Der Server berechnet die Position des mobilen Objekts zum Zeitpunkt t_{i+1}, (3) und sobald dieser Zeitpunkt erreicht ist (4), wird die vorausberechnete Position p_{i+1} gespeichert, und die letzten beiden Positionen werden dementsprechend aktualisiert (5). Erhält der Server eine Update-Nachricht vom Client, wird die laufende Berechnung abgebrochen und die zuletzt berechnete Position durch die Position aus der Update-Nachricht ersetzt.

Die Idee des Dead-Reckoning Ansatzes ist es also, durch Vorausberechnungen der zukünftigen Benutzerposition seitens des Servers Update-Nachrichten einzusparen. Dabei wird anhand der Geschwindigkeit v, bereits bekannten Positionen p_{i-1} und p_i des Benutzers, sowie einem linearen Bewegungsmodell die Position zu einem in der Zukunft liegenden Zeitpunkt t_{i+1} vorausberechnet. Das mobile Objekt führt die Berechnung ebenfalls durch, und bei einer Abweichung der beiden Positionen, die größer als ein vom Benutzer bestimmten Toleranzwert ϵ, wird eine Update-Nachricht mit einer korrigierten Position geschickt (veranschaulicht durch Abbildung 2.4). Dead-Reckoning macht somit nur bei on-the-fly tracking Sinn.

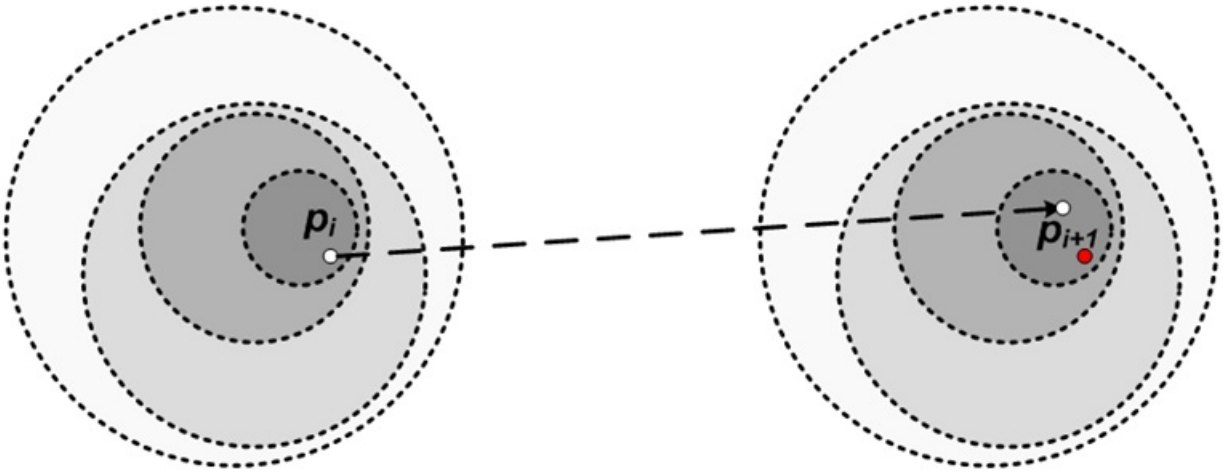


Abbildung 2.4: Dead-Reckoning Strategie: Der Server versucht, die Position des Benutzers vorauszuberechnen (Punkt, auf den die Pfeilspitze zeigt). Dabei können Abweichungen zur tatsächlichen Benutzerposition p_{i+1} entstehen (Quelle: [Skv12])

Natürlich ist die Vorausberechnung nicht zu 100% genau, da beispielsweise die Geschwindigkeit des Benutzers nur eine Abschätzung ist und sehr wahrscheinlich nicht konstant bleibt. Zudem wird ein Benutzer sich sehr wahrscheinlich nie vollkommen linear bewegen. Die dadurch entstehenden Abweichungen von der tatsächlichen Benutzerposition widersprechen dem Prinzip von [DSR11], da der Ansatz garantiert, dass die exakte Benutzerposition zu jedem Zeitpunkt bekannt ist.
Ebenfalls lassen sich dadurch im Vergleich zu den in Kapitel 4 vorgestellten Verfahren nicht wirklich Update-Nachrichten einsparen, da im Korrekturfall so oder so eine Update-Nachricht pro Server geschickt werden muss, zudem würden bereits kleinste Berechnungsfehler das Share-Set inkonsistent machen. Zu guter Letzt kommt hinzu, dass Dead-Reckoning für *einen* Location-Server ausgelegt ist und nicht für n Location Server. Das Verfahren müsste also erheblich modifiziert werden, damit es effektiv anwendbar wäre.
Da die oben genannten Gründe allesamt gegen Dead-Reckoning sprechen, wird der Dead-Reckoning Ansatz an dieser Stelle nicht weiter analysiert.

2.2 Verwandte Arbeiten

2.2.1 Map-based Dead-Reckoning

In [LNR02] wurde ein Map-based Dead-Reckoning Algorithmus implementiert, der anschließend durch Simulationen und echten GPS-Daten ausgewertet wurde: Der Algorithmus benutzt eine Karte, die aus einem Navigationssystem eines Autos extrahiert wurde. Die Karte wird als eine Art Graph interpretiert, wobei jede Kreuzung einen Knoten darstellt und die Verbindung zwischen den Kreuzungen eine Kante (siehe Abbildung 2.5).
Kurvige Stücke werden durch sogenannte „shape points“ in mehrere lineare Stücke aufge-

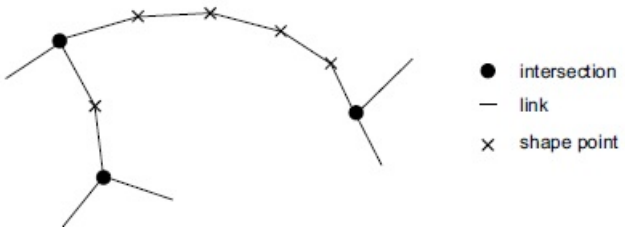


Abbildung 2.5: Beispiel zur Interpretation der Karteninformation(Quelle: [LNR02])

teilt. Da jeder GPS-Sensor mit einer gewissen Ungenauigkeit behaftet ist, muss diese vom Algorithmus berücksichtigt werden. So existiert um jede Kante herum ein Radius (siehe Abbildung 2.6) und sofern sich der Benutzer innerhalb dieses Radius befindet, wird seine Position auf die nächstmögliche Position auf der Kante abgebildet. Befindet sich der Benutzer

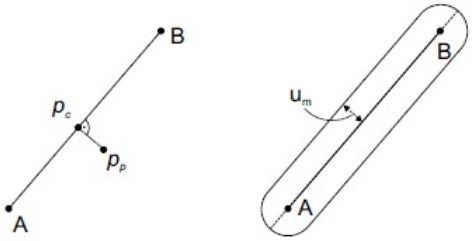


Abbildung 2.6: Abbilden einer Position P_p auf einen Punkt P_c auf der Kante (links), Festlegung des Radius U_m (rechts)(Quelle: [LNR02])

außerhalb dieses Radius, so kann seine Position nicht auf die Kante abgebildet werden. Es wird entweder forward-tracking oder backward-tracking eingesetzt, um dies trotzdem zu erreichen.

Forward-tracking: Die letzte bekannte Position war beispielsweise auf der Kante von Knoten A zu Knoten B. Hat der Benutzer den Punkt B bereits erreicht, wird nun die Distanz der letzten gemessenen Position des Benutzers zu allen ausgehenden Kanten von B gemessen, und die nächstmögliche Kante wird als aktuelle Kante gewählt.

Backward-tracking: Hat der Benutzer den Punkt B nicht erreicht, wird davon ausgegangen, dass zuvor die falsche Kante gewählt wurde. Es wird nun die vorherige Kreuzung herangezogen (und wenn nötig, die davor usw.), und jede ausgehende Kante überprüft.

Konnte sowohl mit forward- als auch mit backward-tracking keine passende Kante gefunden werden, so schickt das mobile Objekt eine Update-Nachricht mit einer leeren Kante. In diesem Fall wird das linear-prediction Protokoll aktiviert. Es wird von einer linearen Bewegung des Benutzers ausgegangen, und bei jeder Abzweigung wird nun diejenige Kante ausgewählt, die mit der vorherigen Kante den kleinsten Winkel einschließt.
Der Algorithmus wurde anschließend in verschiedenen Situationen (Autobahn, Verkehr innerhalb der Stadt, Fußgänger) mit einem GPS-Empfänger getestet. Zum Vergleich wurden ebenfalls ein linear-prediction Algorithmus sowie ein distance-based Reporting Algorithmus implementiert. Die Ergebnisse zeigen, dass der linear-prediciton Algorithmus bereits bis zu 83% weniger Update-Nachrichten als der distance-based Reporting Algorithmus benötigt (in diesem Fall die Autobahnfahrt) und der map-based Dead-Reckoning Algorithmus die Anzahl an benötigten Update-Nachrichten nochmals um 60% reduziert. Ebenfalls erlaubt der map-based Dead-Reckoning Algorithmus eine kleinere Schranke ϵ, da er die Bewegung des mobilen Objekts besser vorhersagen kann.

Dieser Dead-Reckoning Ansatz erzielt zwar eine deutliche Reduktion an Update-Nachrichten und entlastet somit die Kommunikation zwischen dem mobilen Objekt und dem Location-Server. Allerdings bietet er keine Möglichkeit, die tatsächliche Position des Benutzers zu verschleiern und besitzt zudem das Problem, dass der Server, der die Vorausberechnungen durchführt, ein Single Point of Failure ist. Es muss grundsätzlich davon ausgegangen werden, dass der Server vertrauenswürdig, also „trusted“ ist, da er die genaue Position des Benutzers kennt, und im Falle eines kompromittierten oder unsicheren Servers die Sicherheit der Benutzerdaten gefährdet ist.

2.2.2 Connection-preserving Dead-Reckoning

Lange, Dürr und Rothermel stellen in [LDR08] einen effizienten, Dead-Reckoning basierten Algorithmus vor, der dazu in der Lage ist, komplette Benutzer-Trajektorien zu erfassen und in einer MOD (Moving objects database) zu speichern. MODs sind Datenbanken, die Benutzer-Trajektorien anhand von Polylines, welche aus Vertices bestehen, abspeichern. Die Vertices stellen dabei erfasste Positionen des Benutzers dar. Das Ziel des Algorithmus ist es mit in Echtzeit gemessenen und gesendeten Positionsdaten mit möglichst wenigen Vertices auszukommen, was sich mit dem Begriff “Online Trajectory Reduction“ beschreiben lässt. Ebenfalls soll es eine Schranke ϵ geben, die angibt, wie weit die vorausberechnete Position maximal von der Polyline, die die tatsächlichen Bewegung des Benutzers beschreibt, abweichen darf. Die Dead-Reckoning Strategie geht dabei von linearen Bewegungen aus, und benutzt daher ein lineares Berechnungsmodell. Der Algorithmus besitzt zudem eine Sensing-History, die alle Positionen speichert, die seit der letzten Update-Nachricht aufgezeichnet wurden.
Eine Update-Nachricht wird nur dann gesendet, wenn entweder die LDR-Condition oder

die Section-Condition verletzt wird (oder beide).

LDR-Condition: Die tatsächliche momentane Position darf nicht mehr als ϵ von der Position abweichen, die anhand der zuletzt gespeicherten Benutzerposition (und der damaligen aufgezeichneten Geschwindigkeit) vorausberechnet wurde.

Section-Condition: Der Abstand aller Positionen, die in der Sensing-History gespeichert sind, zu dem Linienstück, das die Position des letzten Updates mit der aktuellen Position verbindet, darf nicht mehr als ϵ betragen.

Erhält die MOD eine Update-Nachricht, ersetzt sie die letzte geschätzte Position mit der korrigierten Position aus der Update-Nachricht.
Das Hauptproblem des Algorithmus ist Speicherverbrauch, da die Sensing-History beliebig groß werden kann. Aus diesem Grund haben Lange, Dürr und Rothermel eine verbesserte Version ihres CDR-Algorithmus entwickelt, den sie CDR^m nennen. Die Idee dahinter ist, dass es einen bestimmten Zeitpunkt gibt, ab dem eine Position aus der Sensing-History nicht die Section-Condition verletzen kann, ohne dass die tatsächliche momentane Position die LDR-Condition verletzt. Ab diesem Zeitpunkt kann man also die betroffenen Punkte aus der Sensing-History entfernen. Die Speicherplatzbeschränkung des Algorithmus wird allerdings erst dadurch erzielt, dass es eine Variable gibt, die eine Abstandsgrenze definiert. Kann aufgrund der Platzbeschränkung eine Position nicht mehr in der Sensing-History gespeichert werden, so wird stattdessen die Grenze angepasst. Die Grenze wird so gewählt, dass wenn der Abstand der aktuellen Position und der vorausberechneten Position kleiner ist, keiner der Punkte aus der Sensing-History die Section-Condition verletzen kann.
Beide Algorithmen wurden durch Experimente (Autofahrt, Fahrradfahrt, Spaziergang) mit GPS-Traces ausgewertet. Ebenfalls wurden die beiden Algorithmen mit einem bereits existierenden Ansatz für Online Trajectory Reduction, sowie einer offline-Version verglichen. Die Ergebnisse zeigen, dass die beiden Algorithmen ca. 30% weniger Vertices als der andere online-Ansatz benötigen. Die offline-Version benötigt wie erwartet weniger Vertices und wird daher als Referenz benutzt. Wird die Schranke ϵ erhöht, steigt die Reduktion der Vertices der beiden Algorithmen im Vergleich zum bereits existierenden online-Ansatz sogar auf bis zu 40% an.

Dieser Dead-Reckoning Ansatz erzielt eine ähnliche Verbesserung im Hinblick auf die Update-Nachrichten wie der vorige. Allerdings besitzt er auch exakt dieselben Probleme.

2.2.3 Feeling-based Location Privacy Protection for Location-based Services

Xu und Cai präsentieren in [XC09] einen Ansatz, der die bestehenden k-anonymity Ansätze benutzerfreundlicher gestalten soll. K-anonymity ist ein bekanntes und oft genutztes Verfahren zur Verschleierung von Positionsdaten. Der Benutzer eines LBS gibt dabei eine Zahl k an, die seiner gewünschten Sicherheitsstufe entspricht. Es wird nun nicht die genaue Position an das LBS bzw. den LS übermittelt, sondern es wird eine sogenannte „cloaking Area“ berechnet. Die cloaking Area ist ein möglichst kleines Gebiet, in dem sich mindestens

k andere Benutzer befinden. Somit kann im Falle des Verlusts der Positionsdaten an Dritte verhindert werden, dass der Benutzer enttarnt wird. Im Gegensatz dazu gibt der Benutzer im Falle des Feeling-based Ansatzes eine Public Region, wie z. B. ein Einkaufszentrum anstatt einer Zahl *k* an. Dort werden dann sogenannte „Footprints“ von anderen Benutzern gesammelt und anhand dieser Footprints wird basierend auf dem Konzept der Entropie (es wird nicht nur die Anzahl der Benutzer innerhalb einer Public Region gezählt, sondern auch die Häufigkeit ihrer Besuche in der spezifizierten Region) ein Wert der Popularität dieser Public Region errechnet.
Die entsprechende Netzwerk-Domäne wird hierbei rekursiv in Zellen analog zu einem Quad-Tree eingeteilt. Es entsteht eine Art Pyramiden-Struktur, die in Abbildung 2.7 zu sehen ist. Der oberste Level der Pyramide enthält eine einzelne Zelle, die die gesamte

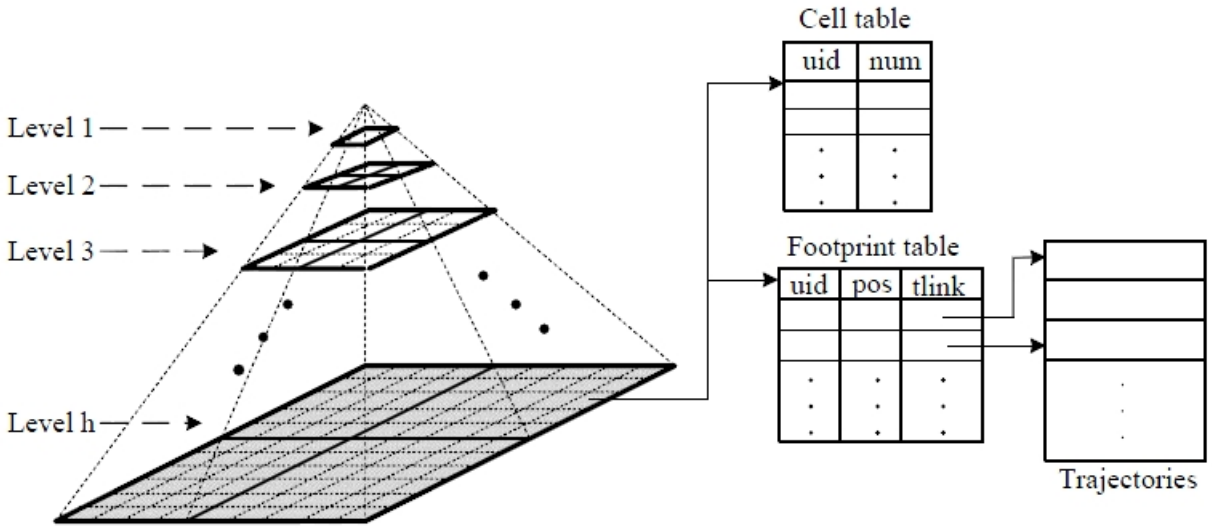


Abbildung 2.7: Datenstruktur und Netzwerk-Domänen Einteilung in Zellen(Quelle: [XCo9])

Netzwerk-Domäne enthält, die Level darunter enthalten jeweils immer mehr Zellen, die die Netzwerk-Domäne jeweils immer feiner aufteilen. Die minimale Zellgröße beträgt $100m^2$.
Soll nun ein Location Update durchgeführt werden, muss der errechnete Wert der Popularität des preiszugebenden Standorts gleich hoch oder höher sein, als der der spezifizierten Public Region. Erreicht wird diese Bedingung durch sogenannte „Cloaking-Boxes“: Bei jedem Update wird eine Cloaking-Box errechnet, die mindestens denselben Wert der Popularität besitzt, wie die zuvor spezifizierte Public Region, und dabei so klein wie nötig ist. Hierzu durchläuft der Algorithmus von unten nach oben die Datenstruktur, um die kleinstmögliche Cloaking-Box zu finden, die die Bedingung erfüllt.
Der LBS erhält dann nicht die exakte Benutzerposition, sondern lediglich eine Cloaking-Box, welche die Benutzerposition enthält. Ebenfalls muss der Benutzer seine Zielroute nicht vorher spezifizieren, da die Cloaking-Boxes on-the-fly berechnet werden.

Der Feeling-based Ansatz löst zwar das Problem der Location-Privacy, da die Position

des Benutzers verschleiert wird, jedoch wird dort ebenfalls von einem trusted Location-Server ausgegangen, also einem Server, auf dem die Positionsdaten eines Benutzers sicher sind. In der Realität kann aber jeder Server/Anbieter unsicher oder kompromittiert sein. Des Weiteren werden keinerlei Bemühungen angestellt, um Update-Nachrichten einzusparen und somit Kommunikationskosten zu reduzieren.

2.2.4 In-Device Spatial Cloaking for Mobile User Privacy Assisted by the Cloud

Wang und Wang stellen in [WW10] ein System vor, das ähnlich wie der Feeling-Based Ansatz von Xu und Cai ebenfalls auf k-anonymity basiert. Sie bemängeln, dass die meisten k-anonymity Ansätze einen zentralen Server benutzen, um eine Cloaking-Area zu erzeugen. Dieser Server kann als Single Point of Failure angesehen werden, da er zu viel Informationen besitzt. Wird er kompromittiert oder gehackt, ist die Sicherheit der Positionsdaten in Gefahr. Daher wird die Cloaking-Area in ihrem System „In-Device“, also direkt auf dem Client-Gerät erzeugt. Dabei wird eine Server-Cloud benutzt, die Statistiken von anderen LBS-Benutzern besitzt (z. B. „Wie viele Benutzer sind gerade in Zelle X?“). Die Annahme ist, dass die Server „semi-honest“ sind, was heißt, dass sie zwar die bestmöglichen und korrekten Informationen liefern, aber dennoch kompromittiert sein könnten. Die Infrastruktur ist auf Abbildung 2.8 zu sehen.

Anders als bei anderen Cloaking-Algorithmen, die Bottom-Up funktionieren (also zunächst

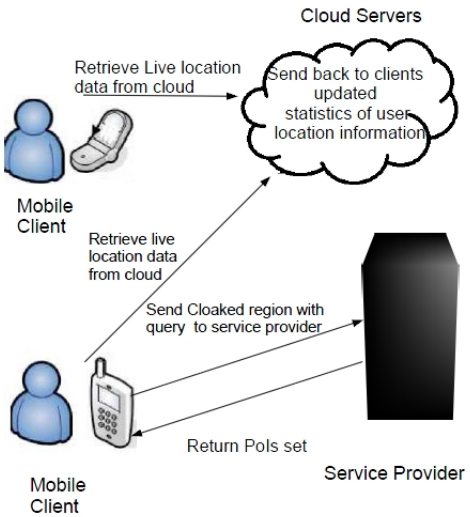


Abbildung 2.8: Infrastruktur des In-Device Spatial Cloaking Ansatzes(Quelle: [WW10])

auf der untersten Zellenebene nach Benutzern suchen), arbeitet der Algorithmus von Wang und Wang Top-Down (da von semi-honest Servern ausgegangen wird, und der Angreifer beim Bottom-Up Ansatz die Zelle, in der sich der Benutzer befindet, durch Heuristiken erahnen könnte). Der Algorithmus fängt also bei der Wurzel an (die auf jeden Fall mehr als k andere Benutzer enthält), und arbeitet sich dann Stückweise nach unten durch. Hierbei

wird immer überprüft, ob die Zelle, die eine Ebene tiefer ist und die Benutzerposition enthält, die k-anonymity Bedingung erfüllt. Ist dies nicht der Fall, wird überprüft, ob die Zelle vereint mit ihren horizontalen und vertikalen Nachbarn die Bedingung erfüllt. Der Algorithmus arbeitet sich nun soweit nach unten durch, bis die kleinste mögliche Zelle (oder Vereinigung von Zellen) gefunden wurde, die die tatsächliche Benutzerposition enthält und die k-anonymity Bedingung erfüllt.
Um die Kommunikation mit der Cloud gering zu halten, wird in einer verbesserten Version nicht immer von der Wurzel ausgegangen, sondern von Zellen, die in einer tieferen Ebene liegen. Hierzu werden die min- und max-Werte von Zellen gespeichert, die angeben, wie viele Benutzer sich bisher mindestens und maximal in der Zelle aufgehalten haben.
Experimentelle Ergebnisse haben zudem gezeigt, dass das System in der Praxis gut funktioniert: Es besitzt niedrige Kommunikationskosten und eine hohe Service-Qualität.

Der In-Device Spatial Cloaking Ansatz geht nicht von einem trusted Location-Server aus und nimmt dieses Problem somit effektiv in Angriff. Jedoch basiert der Ansatz auf k-anonymity, was den Benutzer dazu auffordert, eine Zahl anzugeben, die sein Privacy-Level angibt (was nicht sehr intuitiv ist) und nicht allzu viele Möglichkeiten zulässt, um Update-Nachrichten einzusparen. Ebenfalls ist gegenüber dem LBS keine exakte Angabe der Benutzerposition möglich, sondern nur ein Gebiet, in dem sich der Benutzer befindet, da jedes Update die k-anonymity Bedingung erfüllen muss. Benötigt ein LBS aber die genaue Benutzerposition, ist die Service-Qualität erheblich eingeschränkt.

2.2.5 Advanced Query Evaluation Techniques for Preserving Privacy and Efficiency of Mobile Objects

Ramya, Priyanka, Anusha, Brahmini, Lakshmi und Abraham stellen in [RPA+12] einen Ansatz vor, der darauf abzielt, Benutzerpositionen zu verschleiern, und dabei die Kosten möglichst gering zu halten. Dies wird dadurch erzielt, dass sowohl die drahtlosen Kommunikationskosten für die Location-Updates, als auch die Auswertungskosten für die Query auf dem Location-Server minimiert werden. Das zugrundeliegende Systemmodell ist in Abbildung 2.9 dargestellt: Jederzeit können Application Server (damit sind eigentlich LBS gemeint) eine Query registrieren (1). Wenn ein Objekt ein Location-Update sendet (2), identifiziert der „Query Processor“ die Anfragen, die von diesem Update beeinflusst werden und wertet diese Anhand des „Object Index“ erneut aus. Die neuen Ergebnisse werden dann an die jeweiligen Application Server gesendet. Dann errechnet der „Location Manager“ eine neue „safe region“ für das mobile Objekt (4) und sendet diese an das Objekt (5).
Es wird im Anschluss ein „SpaceTwist“ Algorithmus vorgestellt, der auf dem Client ausgeführt wird, und die k nächsten Nachbarn vom Server anfordert (hier ist die Anwendung einer Point of Interest Search gemeint, die k nächsten Nachbarn können z. B. die 4 nächsten italienischen Restaurants sein). Dabei wird nicht die exakte Benutzerposition geschickt, sondern eine sogenannte Ankerposition, die sich in der Nähe befindet. Falls der Server kompromittiert ist, hat er also keine Kenntnis von der exakten Benutzerposition. Zunächst wird ein Max-Heap initialisiert, der die k nächsten Objekte, die bisher gefunden wurden, abspeichert. Anhand dieses Max-Heaps werden nun der „supply space“ und der „demand

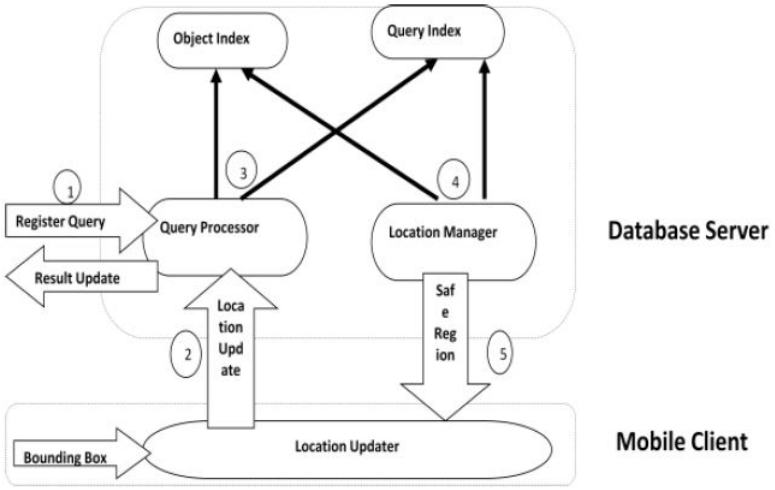


Abbildung 2.9: Systemmodell des PAM-Frameworks(Quelle: [WW10])

space“ gebildet. Der demand space ist ein Kreis um die Benutzerposition mit Radius γ (γ ist die maximale Distanz im Max-Heap), der supply space ist ein Kreis um die Ankerposition mit Radius τ (τ ist die größte Distanz von der Ankerposition zu einem bisher gefundenen Objekt). Der Alorithmus (der zu komplex ist, um ihn kurz vorzustellen) findet nun die k nächsten Nachbarn (Points of Interest), und sendet diese an das mobile Objekt.
Ebenfalls wird noch ein „granular search“ Algorithmus vorgestellt (der auf dem Server ausgeführt wird), der anhand einer vom Benutzer spezifizierten Fehlergrenze nur einen Teil der k nächsten Nachbarn zurückliefert, um somit die Suche zu beschleunigen und Kommunikationskosten einzusparen.
Abschließend wird noch ein dritter Algorithmus namens „SCUBA“ vorgestellt, der es ermöglicht, kontinuierliche Anfragen auf mobilen Objekten auszuwerten. SCUBA nutzt dabei aus, dass mobile Objekte und Anfragen aufgrund gemeinsamer Eigenschaften (z. B. Geschwindigkeit, Bewegungsrichtung, Abstand zueinander, usw.) in Gruppen eingeteilt werden können. Der Algorithmus selbst ist ebenfalls zu komplex um hier kurz vorgestellt werden zu können. SCUBA ist besser als traditionelle grid-based Ansätze, wo jedes mobile Objekt einzeln ausgewertet wird.

Dieser Ansatz, der sporadische Updates anhand einer falschen Benutzerposition (Ankerposition) auswertet, verhindert zwar, dass ein eventuell kompromittierter Server Kenntnis von der exakten Benutzerposition erlangt, bietet aber in diesem Sinne eine schlechte Service-Qualität an, da eigentlich Points of Interest in der Nähe der Ankerposition gesucht werden. Will man bessere Ergebnisse haben, muss die Ankerposition näher an der exakten Position liegen, was wiederum die Sicherheit einschränkt. Außerdem wird eine Nachrichteneinsparung lediglich dadurch erzielt, dass vom Benutzer eine Fehlergrenze angegeben wird und nur ein Teil der angeforderten k Points of Interest zurückgeliefert werden.

2.2.6 Fazit

Die vorgestellten verwandten Arbeiten decken viele Aspekte im Gebiet der Optimierung im Bezug auf die Benutzung von Location-based Services ab. Während die Arbeiten zwar gute Ergebnisse erzielen, schafft es jedoch keine, die Forschungsgebiete Location-Privacy und Reduktion von Update-Nachrichten zu vereinen.
Diese Diplomarbeit löst alle genannten Probleme, indem die Problemfelder effektiv miteinander vereint werden. Wie in [WW10] wird die Verschleierung der Benutzerposition auf dem Client selbst durchgeführt, um einen zentralen Server, der eventuell kompromittiert sein könnte, zu vermeiden. Allerdings basiert die Verschleierung nicht auf k-anonymity, was viele Einschränkungen aufhebt und ein neues Konzept ermöglicht: Der Benutzer gibt lediglich einen Radius in Metern an, innerhalb dieses Radius ist seine Position verschleiert. Der Verschleierungs-Algorithmus wählt nun eine zufällige Benutzerposition innerhalb des angegebenen Radius aus, und zerlegt die Benutzerposition in mehrere Vektoren, die aufaddiert von der zufällig gewählten auf die genaue Benutzerposition zeigen. Die falsche Benutzerposition (auch Master-Share genannt) wird nun zusammen mit den Verschiebungs-Vektoren an die Location-Server übermittelt, so dass jeder Server einen unterschiedlichen Vektor speichert. So kann grundsätzlich von untrusted Location-Servern ausgegangen werden, da die Location-Server voneinander unabhängig sind, und sich somit nicht kennen. Benötigt ein LBS nun die Position des Benutzers, so kann er beliebig viele Vektoren von den Location-Servern anfragen, je nachdem, welche Genauigkeit benötigt wird. Mit jedem angefragten Vektor kann er den Radius, in dem sich der Benutzer aufhält, verringern und die Vektoren Stück für Stück aneinander hängen, bis er die exakte Benutzerposition kennt. Die Service-Qualität des LBS ist also zu keiner Zeit eingeschränkt und die Benutzerposition ist jederzeit sicher. Ebenfalls werden die benötigten Update-Nachrichten deutlich reduziert.
Das entwickelte System basiert auf dem ausführlich in Kapitel 3 vorgestellten Location-Privacy Ansatz, der so verbessert wurde, dass in manchen Fällen über 90% an Update-Nachrichten eingespart werden können, und das ohne Verlust an Location-Privacy oder Genauigkeit.
Im nächsten Kapitel wird der Basis-Ansatz, auf dem diese Diplomarbeit aufbaut, ausführlich vorgestellt.

3 Basisansatz

Diese Diplomarbeit baut hauptsächlich auf der Arbeit [DSR11] von Rothermel, Skvortsov und Dürr auf. Der dort vorgestellte Ansatz soll weiterentwickelt werden, sodass unnötige Updates vermieden und die Kommunikationskosten reduziert werden. Da diese Grundlagen essentiell sind, wird in diesem Kapitel den Basis-Ansatz aus [DSR11] ausführlich vorgestellt.

3.1 Idee

Location-based services (LBS) sind Dienste, die die Position des Benutzers benötigen, um z. B. anzuzeigen, ob sich ein Freund in der Nähe befindet oder wo das nächste Kino ist. Diese Dienste werden hauptsächlich von mobilen Geräten wie Smartphones, PDA's oder Laptops/Tablets benutzt, die Positionsdaten des Benutzers werden dabei üblicherweise auf Location-Servern des Dienstanbieters gespeichert. Ist nun ein Server oder Dienstanbieter kompromittiert, kann die Information missbraucht werden. Ebenfalls können solche Server angegriffen werden, und Dritte können in Besitz der Information gelangen.
Rothermel et al. stellen einige verwandte Arbeiten vor, die versuchen, eben diese Probleme zu umgehen. So z. B. der k-anonymity Ansatz, bei dem die Positionsdaten des Benutzers zusammen mit Daten von anderen Benutzern anonymisiert werden. Dieser Ansatz setzt allerdings einen zuverlässigen Server für die Anonymisierung voraus.
Unter anderem wird auch der „dummies approach" genannt, bei dem unter die echten Positionsdaten zusätzlich falsche Daten gemischt werden. Diese Methode setzt zwar nicht zwingend einen zuverlässigen Server voraus, jedoch erfordert sie hohe Kommunikationskosten und bietet nur wenig Sicherheit, da Angreifer die falschen Positionen herausfiltern können.
Die Hauptidee ist daher ein „spatial obfuscation"-Ansatz. Der LBS erhält nicht die korrekten Benutzerpositionen, sondern nur ungefähre Positionen. Diese Ungenauigkeit kann durch den Benutzer angepasst werden. Um die Sicherheit weiter zu erhöhen, werden die Benutzerpositionen aufgeteilt, und jeder Teil wird auf einem Server eines anderen Anbieters gespeichert, um einen Single Point of Failure zu vermeiden.

3.2 System-Modell

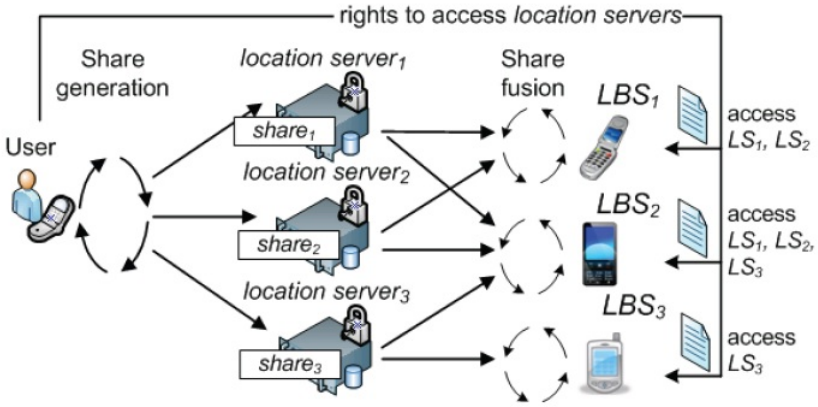


Abbildung 3.1: System-Modell (Quelle: [DSR11])

Das System-Modell lässt sich am besten anhand von Abbildung 3.1 beschreiben.
Das mobile Objekt (MO) ermittelt die Position des Benutzers via GPS. Auf dem mobilen Gerät läuft eine sogenannte „share generation"-Komponente, die die aktuelle Position in mehrere Vektoren aufteilt und diese an die verschiedenen Location-Server sendet.
Die Location-Server (LS) speichern die Positionen der mobilen Objekte. Die Server stellen die Positionsdaten für die Location-based Services über Anfragen bereit, sofern der Benutzer dies erlaubt. Der Benutzer kann genau spezifizieren, welcher Location-based Service auf welche Location-Server Zugriff erhält.
Location-based services (LBS) benötigen die Position des Benutzers, um ihren Service anzubieten. Je genauer die Position, desto höher ist die Qualität des Ergebnisses. Ist dem Benutzer die Verschleierung seiner Position wichtiger als das Ergebnis des Services, so muss er Abstriche bei der Service-Qualität in Kauf nehmen. Ebenfalls läuft beim LBS eine sogenannte „share fusion"-Komponente, die die einzelnen Vektoren so zusammensetzt, dass jeder hinzugefügte Vektor die Genauigkeit der entstehenden Position erhöht. Besitzt das LBS Kenntnis von allen Vektoren, kann es die exakte Benutzerposition rekonstruieren.

3.3 Formale Grundlagen

Eine Position π mit einer gewissen Präzision ist definiert durch eine Kreisfläche mit Radius r, wobei $prec(\pi) = r$. Formal kann das Problem der share-generation als Funktion beschrieben werden, die eine Position π auf eine Menge S = $\vec{s_1}, ..., \vec{s_n}$ von n Teilpositionen abbildet:

(3.1) $generate(\pi, n, \phi_{min}) = (s_{master}, \vec{s_1}, ..., \vec{s_n})$

Diese Funktion erhält zusätzlich den Parameter ϕ_{min}, der die Präzision des sogenannten „Master share“ s_{master} definiert. s_{master} ist die Teilposition mit der gröbsten Präzision ϕ_{min}. Teilpositionen aus der Menge S werden „Refinement shares“ genannt. Durch das Zusammensetzen von S_{master} und einer Teilmenge S′ ⊆ S erhält man eine verfeinerte Position mit einer höheren Präzision:

(3.2) $fuse(s_{master}, S' \subseteq S) = \pi' \; with \; prec(\pi) \leq prec(\pi')$

3.4 Erzeugung der Teilpositionen

Der Share-generation Algorithmus erzeugt eine Menge aus Verschiebungs-Vektoren. Formal gesehen ist die Menge eine symmetrische Permutations-Gruppe G(S) = { $G_1(S), ..., G_m(S)$}: S $\rightarrow S$, sodass für ein gegebenes $\pi = p_n$ mit $\vec{p_n} + G(\sum \vec{s_i}) = \vec{p_0}$ folgende Bedingungen erfüllt sein müssen:

(3.3) $\vec{p_i} = \vec{p_{i-1}} + \vec{s_i}$

Die erste Bedingung (Gleichung 3.3) definiert, dass die Verschiebungs-Vektoren addiert werden, um inkrementell verfeinerte Benutzerpositionen zu erhalten.

(3.4) $p_n \in c_i$

Die zweite Bedingung (Gleichung 3.4) versichert, dass die exakte Benutzerposition p_n sich immer innerhalb jedes verfeinerten Kreises c_i befinden muss.

(3.5) $c_i \in c_{i-1}$

Die dritte Bedingung (Gleichung 3.5) versichert, dass jeder verfeinerte Kreis *ci* vollständig im vorherigen (gröberen) Kreis c_{i-1} enthalten sein muss (siehe Abbildung 3.2). Ebenfalls wird garantiert, dass jeder Verfeinerungs-Schritt die Genauigkeit um den Wert Δ_ϕ erhöht. Die maximale Länge eines Vektors ist durch $|\vec{s_i}|_{max} = \Delta_\phi = r_0/n$ beschränkt, da der Verschleierungs-Faktor durch jeden hinzugefügten Vektor konstant sein soll. Im Folgenden werden zwei unterschiedliche Ansätze zum Erzeugen der Teilpositionen vorgestellt.

3.4.1 a-posteriori Ansatz

Algorithmus 3.1 erhält als Eingabe den Radius r_0 des initialen Kreises, die Gesamtanzahl an Teilpositionen n, die exakte Benutzerposition $\pi = p_n$ und die maximale Vektorenlänge Δ_ϕ. Die Verschiebungs-Vektoren werden in (3) so gewählt, dass sowohl die Richtung, in die die Vektoren zeigen, als auch die Länge der Vektoren, zufällig gewählt werden. Die Länge ist dennoch uniform. In (4) werden dann alle Vektoren an die exakte Benutzerposition p_n gehängt und so wird der Mittelpunkt p_0 des Initial-Kreises c_0 a-posteriori bestimmt.

Algorithmus 3.1 a-posteriori Algorithmus

1: **function** GEN_N_SHARES_A_POSTERIORI(r_0,n,p_n,Δ_ϕ)
2: **for** ($i = 1 \rightarrow n$) **do**
3: **select randomly** s_i **with** $|\vec{s_i}| \leq \Delta_\phi$
4: $\vec{p_0} \leftarrow \vec{p_n}$ - $\sum_{i=1}^{n} \vec{s_i}$
5: **end for**
6: **return** S[0...n]
7: **end function**

3.4.2 a-priori Ansatz

Der a-priori Ansatz unterscheidet sich zum a-posteriori Ansatz darin, dass zuerst der Mittelpunkt p_0 des Initial-Kreises c_0 festgelegt wird, und dann zufällig Verschiebungs-Vektoren erzeugt werden.
Algorithmus 3.2 erhält dieselben Parameter wie Algorithmus 3.1. Zunächst wird der Mit-

Algorithmus 3.2 a-priori Algorithmus

1: **function** GEN_N_SHARES_A_PRIORI(r_0,n, p_n, Δ_ϕ)
2: **select randomly** p_0 **such that** *distance* $(p_0, p_n) \leq r_0$
3: **while** (*distance*$(p_0 + \sum_{i=1}^{n-1} \vec{s_i}, \vec{p_n}) > \Delta_\phi$)) **do**
4: **for** ($i = 1 \rightarrow n - 1$) **do**
5: **select rand.** $\vec{s_i}$ **with** $|\vec{s_i}| \leq \Delta_\phi$ **such that** $p_n \in c_i$
6: **end for**
7: **end while**
8: $\vec{s_n} \leftarrow \vec{p_n} - (\vec{p_0} + \sum_{i=1}^{n-1} \vec{s_i})$
9: **return** S[0...n]
10: **end function**

telpunkt p_0 des Initial-Kreises c_0 zufällig so gewählt, dass die exakte Benutzerposition $\pi = p_n$ innerhalb des Kreises liegen (2). Die Einschränkung in (3) garantiert, dass die erzeugten Vektoren den Kreismittelpunkt p_0 mit der exakten Benutzerposition p_n verbinden können (6). In (4) und (5) werden analog zum a-posteriori Ansatz die Verschiebungs-Vektoren zufällig gewählt.

3.5 Zusammensetzen der Teilpositionen

Jede Teilposition ist als Vektor definiert, sodass die Konkatenation aller Vektoren auf die genaue Benutzerposition zeigt. Eine Teilmenge der Vektoren ergibt somit lediglich eine verschleierte Position, also ein Kreis mit einem bestimmten Radius, in dem sich der Benutzer aufhält.
Algorithmus 3.3 bekommt als Eingabe die Gesamtanzahl an Teilpositionen n, den Master-Share c_0, sowie die einzelnen Teilpositionen $\vec{s_1}$ bis $\vec{s_k}$ in Form von Verschiebungs-Vektoren,

Algorithmus 3.3 Zusammensetzen der Teilpositionen

```
1: function FUSE_K_SHARES(n,c_0,s_1...s_k)
2:     Δ_φ ← r_0 / n
3:     p ← p_0; r ← r_0
4:     for (i = 1 → k) do
5:         p ← p + s_i;  r ← r − Δ_φ
6:     end for
7: return c_k = {p, r}
8: end function
```

die er zusammensetzen soll. Zunächst wird die Verringerung des Radius berechnet (2). Beginnend vom Mittelpunkt des Ursprungskreises c_0 (3) werden Stück für Stück für k Teilpositionen (4) die Vektoren konkateniert (und somit jeweils der Kreismittelpunkt verschoben) und der Radius verringert (5). Das Ergebnis ist der Kreis c_k (7). Abbildung 3.2 veranschaulicht diesen Vorgang.

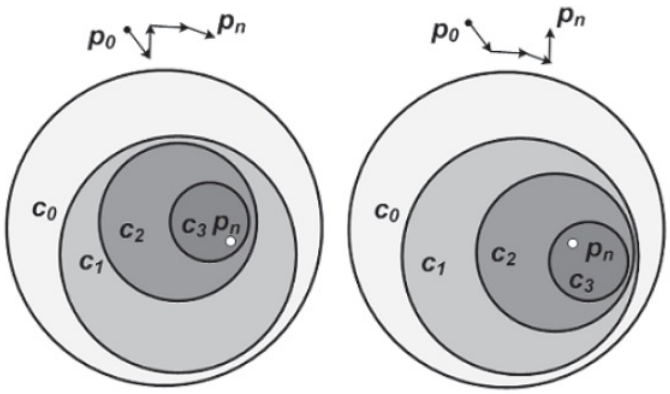


Abbildung 3.2: Grafische Veranschaulichung des Zusammensetzens der Teilpositionen (Quelle: [DSR11])

3.6 Sicherheitsanalyse

Die Sicherheitsanalyse ist für diese Diplomarbeit nicht allzu relevant, daher wird sie kürzer ausfallen als die anderen Kapitel. Es wurde versucht, mithilfe von Monte-Carlo Simulationen und analytischen Methoden (Faltung) die pdf (probability distribution function) der Benutzerposition in Erfahrung zu bringen. Besitzt ein Angreifer Kenntnis von dieser Funktion, weiß er, mit welcher Wahrscheinlichkeit der Benutzer sich an welcher Stelle innerhalb des Kreises aufhält. Hierzu wurden die beiden Share-generation Algorithmen separat ausgewertet. Das Problem des a-priori Algorithmus ist, dass die Wahrscheinlichkeitsverteilung der Benutzerposition π innerhalb des Kreises c_0 nicht uniform ist, da es wahrscheinlicher ist,

dass sich der Endpunkt aller konkatenierten Vektoren in der Nähe des Kreismittelpunkts befindet. Der a-posteriori Algorithmus hat das Problem, dass die Benutzerposition π umso vorhersagbarer wird, desto mehr Teilpositionen man bereits kennt. Dies rührt daher, dass die Verschiebungs-Vektoren so gewählt werden müssen, dass die Konsistenz-Einschränkungen erfüllt werden müssen, was dazu führt, dass sowohl die Länge als auch die Richtung der Vektoren miteinander korrelieren.
Der a-priori Algorithmus ist vorteilhaft in Situationen, in denen der Angreifer lediglich wenig Teilpositionen kennt, der a-posteriori Algorithmus hingegen bietet in diesem Fall weniger Sicherheit, da sich der Benutzer wahrscheinlich Nahe des Mittelpunktes von c_0 aufhält. Kennt ein Angreifer viele Teilpositionen, bietet der a-posteriori Algorithmus mehr Sicherheit.
Ebenfalls hängt die Sicherheit des a-priori Algorithmus stark von der konkreten Vektoren-Teilmenge ab, wohingegen die Sicherheit des a-posteriori Algorithmus lediglich von n, k und der Größe von c_o abhängt.

3.7 Performanceanalyse

Der Ansatz erfordert zusätzliche Rechenleistung auf mobilen Geräten, da bei jedem Positionsupdate Teilpositionen sowohl erzeugt als auch wieder zusammengeführt werden müssen. Außerdem sind die Kommunikationskosten ebenfalls hoch, da mehrere Teilpositionen an mehrere Server übermittelt werden müssen.
Auf einem HTC Desire HD Gerät wurde die Rechenzeit gemessen, die für das Erzeugen der Teilpositionen benötigt wird. Die Ergebnisse zeigen, dass der a-posteriori Algorithmus unabhängig von der Gesamtanzahl an zu erzeugenden Teilpositionen stets schneller ist als der a-priori Algorithmus. Dies liegt daran, dass letzterer mehr Vektoren-Mengen durchsuchen muss, um denjenigen Vektor zu finden, der p_0 und p_n verbindet. Bei einer Update-Rate von 1 HZ kann der a-posteriori Algorithmus in Echtzeit über 64 Teilpositionen erzeugen kann, der a-priori Algorithmus 8.
Die Kommunikationskosten können gering gehalten werden, wenn die Update-Nachrichten (die aus der user id, die 32 bytes groß ist, und dem Bewegungsvektor, der 8 bytes groß ist, bestehen) über UDP in einem Binärformat gesendet werden. Auf diese Weise kostet die Übertragung von 8 Teilpositionen nur lediglich 550 bytes. Leider nutzen viele LBS wie z.b. Google Latitude HTTP-Requests und JSON oder XML. Die Übertragung einer einzelnen Update-Nachricht mit HTTP und JSON ist ungefähr 8-mal so groß wie mit UDP und Binärformat.
Im nächsten Kapitel werden Ansätze zur Verbesserung der in diesem Kapitel vorgestellten Arbeit präsentiert.

4 Ansätze zur Reduktion des Kommunikationsaufwandes

In diesem Kapitel werden diverse Ansätze vorgestellt, die den in [DSR11] vorgestellten Ansatz verbessern sollen, so dass hauptsächlich Kommunikationskosten reduziert werden. Insbesondere sollen unnötige Update-Nachrichten vermieden werden. Zunächst wird das Problem der Nachrichtenoptimierung formal definiert.

4.1 Nachrichtenoptimierungsproblem

Sei S_{basic} die Nachrichtenmenge, die der in Kapitel 3 vorgestellte Basis-Ansatz für eine gegebene Benutzer-Trajektorie T_i erzeugt. Sei S_{opt} die Nachrichtenmenge, die von der optimierten Strategie für dieselbe Trajektorie erzeugt wird und weiter sei T_{Σ_2} die Menge aller Benutzer-Trajektorien mit mindestens zwei Benutzerpositionen. Es muss nun folgendes gelten:

(4.1) $\forall_{T_i \in T_{\Sigma_2}} | min(|S_{opt}|, |S_{basic}|) = S_{opt}$

Der optimierte Ansatz soll also für jede mögliche Benutzer-Trajektorie aus T_{Σ_2} weniger Nachrichten als der Basis-Ansatz erzeugen.
Die Nachrichtenreduktion darf jedoch nicht durch einen Verlust der Präzision erreicht werden. Sei $\pi(P_i)$ die Präzision der Position P_i (gemäß der Definition aus Kapitel 3).
Es muss nun pro Update-Nachricht gelten:

(4.2) $\forall_{P_i \in S_{opt}} \exists_{P_j \in S_{basic}} | \pi(P_i) >= \pi(P_j)$

Das heißt, die Präzision π jeder Position P_i, die durch Vektoren aus S_{opt} beschrieben wird, muss mindestens genauso hoch sein, wie die Präzision der entsprechenden Position P_j, die durch Vektoren aus S_{basic} beschrieben wird. Es darf insbesondere kein Präzisionsverlust stattfinden.
Zusammenfassen lässt sich das Nachrichtenoptimierungsproblem also folgendermaßen: Für eine gegebene Benutzer-Trajektorie aus T_{Σ_2} ist es das Ziel, so wenig Nachrichten wie möglich zu schicken, insbesondere weniger Nachrichten als der Basis-Ansatz (gemäß Formel 4.1). Eine wichtige Einschränkung dabei ist, dass die Präzision jeder Position P_i der Benutzer-Trajektorie, die durch Vektoren mit dem optimierten Ansatz beschrieben werden, mindestens genau so hoch sein muss, wie wenn die entsprechende Position mit Vektoren durch den Basis-Ansatz beschrieben werden würde (siehe Formel 4.2). Da diese Einschränkung unbedingt eingehalten werden muss, kann dies bedeuten, dass durch den optimierten Ansatz in

manchen Fällen nur eine geringe oder sogar gar keine Nachrichtenreduktion erreicht werden kann.
S_{opt} stellt hierbei die optimale Lösung des Optimierungsproblems dar, weil bereits eine Nachricht weniger ($S_{opt} - 1$) zu einem Verlust der Genauigkeit führen würde, und damit das Share-Set inkonsistent wäre. Eine Nachricht mehr ($S_{opt} + 1$) wäre im Sinne der Nachrichtenoptimierung überflüssig, da S_{opt} bereits die oben genannten Bedingungen erfüllt.

Im Anschluss wird der erste Ansatz zur Nachrichtenreduktion vorgestellt, der dann angewendet werden kann, wenn sich ein Benutzer zwischen zwei Updates gar nicht, kaum, oder mit niedriger Geschwindigkeit bewegt hat (also wenn die Update-Frequenz niedrig ist, wie z. B. beim on-the-fly Tracking). Auch bietet sich der erste Ansatz an, wenn der Radius des Master-Shares eher groß ist.

4.2 Verbesserung des share-generation Algorithmus (Fall 1)

4.2.1 Idee

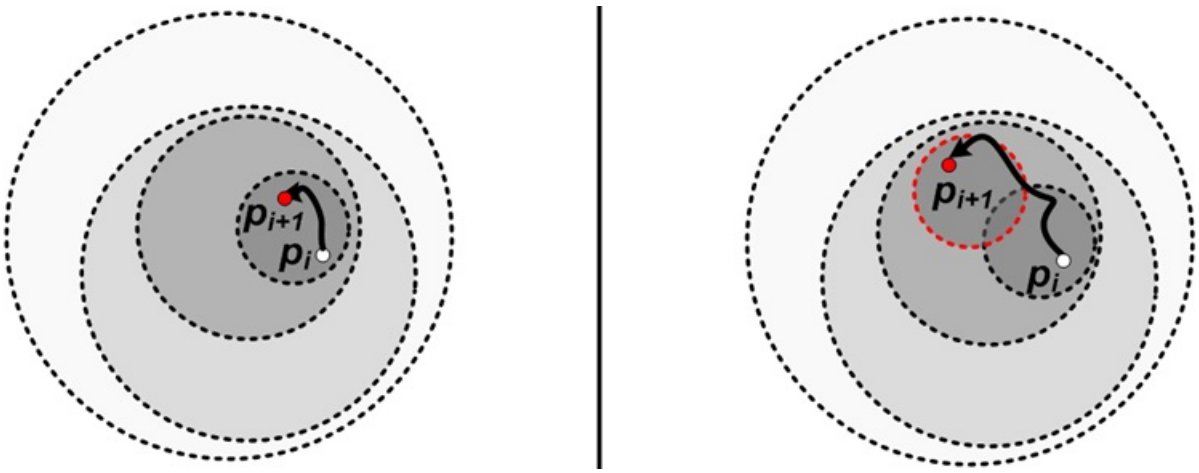


Abbildung 4.1: Einsparung von Update-Nachrichten: Benutzer befindet sich nach einem Update noch im kleinsten Teilkreis (links), Benutzer befindet sich nach einem Update noch im zweitkleinsten Teilkreis (rechts) (Quelle: [Skv12])

Abbildung 4.1 zeigt ein Beispiel, in dem der erste Ansatz (im folgenden auch Fall 1 bezeichnet) eine wesentliche Reduktion von Update-Nachrichten erreichen kann: Der Benutzer hat sich zwischen zwei Updates so wenig bewegt, dass er sich immer noch innerhalb des ersten (bzw. zweiten) Teilkreises befindet. Eigentlich muss jetzt nur der innerste Share (bzw. die inneren beiden Shares) neu berechnet und an die entsprechenden Server gesendet werden, da die Vektoren eine relative Positionsangabe darstellen und somit die neu berechneten Vektoren einfach an die alten angehängt werden können. Die gesamte Kette zeigt weiterhin auf die exakte Benutzerposition, somit ist die Präzision genauso hoch, wie wenn man alle

Vektoren neu berechnet hätte (also so wie es im Basis-Ansatz getan worden wäre).
Ein anderer möglicher Ansatz um solche Update-Nachrichten zu verhindern, wird in [KrLGTr09] vorgestellt. Dort wird anhand eines im mobilen Geräts verbauten Geschwindigkeitsmessers die Geschwindigkeit des mobilen Objekts gemessen. Ist sie Null oder nahe Null, werden der UMTS-Empfänger, sowie das GPS-System heruntergefahren um Akkuleistung zu sparen. Da sich allerdings [DSR11] nicht nur auf Mobiltelefone beschränkt, kann nicht immer davon ausgegangen werden, dass ein Geschwindigkeitsmesser vorhanden ist.
Der genaue Ablauf in Fall 1 sieht folgendermaßen aus:
Muss eine Update-Nachricht geschickt werden, so wird zunächst anhand der zuletzt erzeugten Verschiebungs-Vektoren überprüft, ob sich der Benutzer immer noch im kleinstmöglichen Teilkreis c_n befindet, dessen Mittelpunkt p_m sich aus der Subtraktion des Verschiebungs-Vektors ergibt, der auf die letzte Benutzerposition zeigt. Man kann anhand folgender Ungleichung herausfinden, ob sich der Benutzer noch innerhalb des Teilkreises befindet:

$$|p_n - p_m| < r_n \tag{4.3}$$

Hierbei stellt p_n die aktuelle Position des Benutzers dar, p_m ist der Mittelpunkt des zu überprüfenden Teilkreises c_n (in diesem Fall der kleinste) und r_n dessen Radius. Ist die Bedingung erfüllt, muss lediglich der Vektor neu erzeugt werden, der vom Mittelpunkt p_m des Teilkreises c_n auf die neue Benutzerposition p_n zeigt (Siehe Abbildung 4.1). Die restlichen Verschiebungs-Vektoren können beibehalten werden und müssen weder neu berechnet und insbesondere nicht in der Update-Nachricht mitgeschickt werden. Befindet sich der Benutzer allerdings nicht mehr innerhalb des kleinstmöglichen Teilkreises c_n, wird überprüft ob er sich innerhalb des zweitkleinsten Teilkreises c_{n-1} befindet, usw.. In diesem Fall müssen die Verschiebungs-Vektoren ab dem Mittelpunkt des Teilkreises, in dem sich der Benutzer befindet, bis zur neuen Benutzerposition p_n neu erzeugt und an die jeweiligen Location-Server gesendet werden. Befindet sich der Benutzer außerhalb des größten Teilkreises, also des Master-Shares, kann Fall 1 nicht angewendet werden.
Abbildung 4.2 veranschaulicht den eben beschriebenen Vorgang nochmals anhand eines Beispiels.

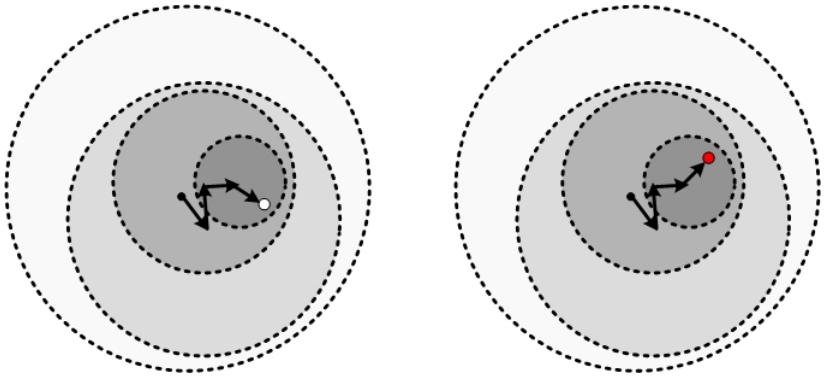


Abbildung 4.2: Nachrichtenoptimierung in Fall 1: Für die neue Benutzerposition (rot) muss lediglich der letzte Vektor neu erzeugt werden. Er kann dann einfach an die bereits vorhandenen Vektoren angehängt werden.

4.2.2 Verbesserter share-generation Algorithmus (Fall 1)

Der nun vorgestellte verbesserte Share-generation Algorithmus kommt in speziellen Fällen mit deutlich weniger Update-Nachrichten aus, als der in [DSR11] vorgestellte Algorithmus.

Algorithmus 4.1 Verbesserter share-generation Algorithmus (Fall 1)

1: **function** GEN_MINIMUM_OF_N_SHARES(p_0,r_0,p_n,p_{n-1},$S_{old}[0...n]$,Δ_ϕ)
2: $p_m \leftarrow p_{n-1} - S[n-1]$
3: $i \leftarrow n-1$
4: $r_n \leftarrow \Delta_\phi$
5: **while** ($dist(p_n, p_m) > r_n$) **do**
6: $i--$
7: $p_m \leftarrow p_m - S[i]$
8: $r_n \leftarrow r_n + \Delta_\phi$
9: **end while**
10: **while** ($distance(\vec{p_m} + \sum_{j=i}^{n-2} \vec{s_j}, \vec{p_n}) > \Delta_\phi$)) **do**
11: **for** ($j = i \rightarrow n-1$) **do**
12: **select randomly** s_j **with** $|\vec{s_j}| \leq \Delta_\phi$ **such that** $p_n \in c_i$
13: **end for**
14: **end while**
15: $\vec{s_{n-1}} \leftarrow \vec{p_n} - (\vec{p_m} + \sum_{j=i}^{n-2} \vec{s_j})$
16: **return** S[*i...n-1*]
17: **end function**

Die Prüfung der Bedingung, ob Fall 1 wirklich angewendet werden kann (Benutzer befindet sich mindestens innerhalb des Teilkreises des Master-Shares), wird bereits vor Aufruf des Algorithmus durchgeführt. Der Algorithmus bekommt als Parameter den Mittelpunkt p_0 des zuvor erzeugten Master-Share Kreises c_0, dessen Radius r_0, die neue Position des Benutzers p_n, die vorherige Position des Benutzers p_{n-1}, die alten bereits erzeugten Verschiebungs-Vektoren $S_{old}[0...N]$ sowie den Faktor Δ_ϕ, der die Dekrementierung des Radius festlegt.
Es wird nun zunächst der Mittelpunkt p_m des kleinsten Teilkreises initialisiert (2). Ebenfalls wird eine integer-Variable i, die zu Zählzwecken dient, mit $n-1$ initialisiert, und die Variable r_n, die als Maß für die Entfernung dient, mit Δ_ϕ initialisiert. (3+4). Dann wird anhand der while-Schleife in (5) der Mittelpunkt p_m des kleinsten Teilkreises bestimmt, in dem sich der Benutzer momentan befindet (also in dem sich die neue Benutzerposition p_n befindet). Der Algorithmus arbeitet in (10) - (15) analog zum a-priori Algorithmus und findet Verschiebungs-Vektoren, so dass diese p_m und p_n miteinander verbinden. Die Schleife in (10) läuft bis zum Index $n-2$, da sie bis zum vorletzten Vektor laufen muss (und $n-1$ bereits der letzte Vektor ist). Der letzte neu zu berechnende Verschiebungs-Vektor ergibt sich dann aus der Subtraktion des Vektors von der neuen Benutzerposition, der sich aus der Addition von p_m und allen danach neu berechneten Verschiebungs-Vektoren ergibt (15). Dann werden lediglich die neu berechneten Vektoren in (17) an die entsprechenden Server gesendet.
Im Anschluss wird der zweite Ansatz zur Nachrichtenreduktion vorgestellt, der dann angewendet werden kann, wenn sich ein Benutzer zwischen zwei Updates viel bzw. mit

hoher Geschwindigkeit bewegt hat (also wenn die Update-Frequenz hoch ist, wie z. B. bei sporadic Updates). Auch bietet sich der zweite Ansatz an, wenn der Radius des Master-Shares eher klein ist.

4.3 Verbesserung des Share-generation Algorithmus (Fall 2)

4.3.1 Idee

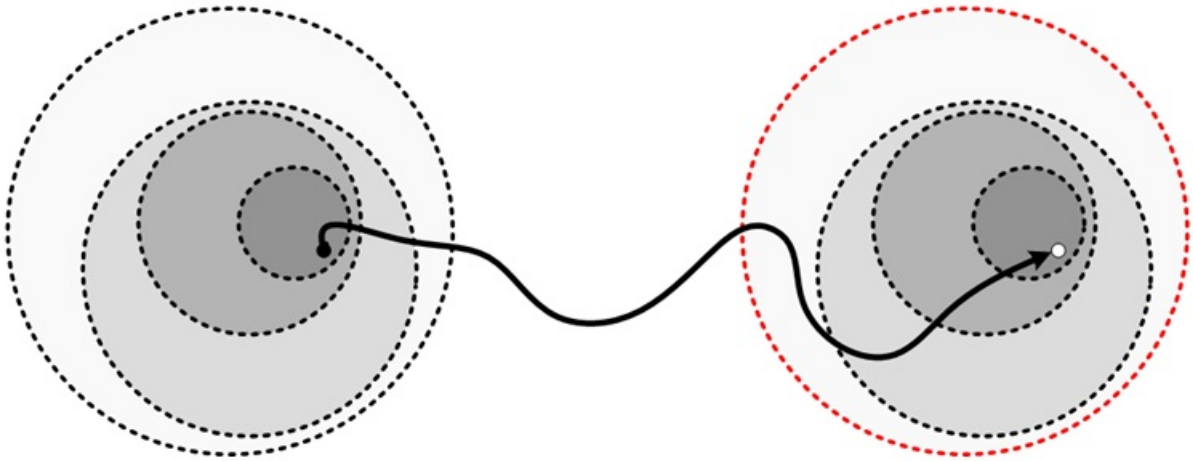


Abbildung 4.3: Einsparung von Update-Nachrichten: Ist die neue Benutzerposition ausreichend von der alten entfernt, muss lediglich der Master-Share (roter Kreis) neu gesendet werden (Quelle: [Skv12])

Befindet sich der Benutzer nach einem Update außerhalb des größten Teilkreises, also dem Master-Share, müssen bisher alle Shares neu erzeugt und an die entsprechenden Location-Server gesendet werden. Dies kann ebenfalls vermieden werden, indem in diesem Fall lediglich der Master-Share neu berechnet und gesendet wird (siehe Abbildung 4.3 zur Veranschaulichung). Da die Refinement-Shares keine absolute Position angeben, sondern relativ sind, können diese einfach wie gehabt an den Mittelpunkt des neu berechneten Master-Share Kreises angehängt werden. Dadurch bleibt die Anordnung der Teilkreise dieselbe. Des Weiteren bleibt auch die Präzision genauso hoch, wie sie beim Basis-Ansatz wäre, da die komplette Vektorenkette auf die exakte Benutzer-Position zeigt.
Dieser Ansatz kann allerdings nur benutzt werden, wenn folgende Bedingung erfüllt ist:

$$|p_n - p_{n-1}| > 4 * r_0 \tag{4.4}$$

Dann kann garantiert werden, dass sich die Teilkreise der Master-Shares (und eventuell auch Teilkreise der Refinement-Shares) nicht überschneiden, da die Benutzerpositionen p_n und

p_{n-1} im worst-case jeweils am gegenüberliegenden Rand der beiden Kreise sein könnten. Da dies aber sehr unwahrscheinlich ist, kann bei einer Nichterfüllung der Bedingung 4.4 immer noch die folgende Bedingung zutreffen:

(4.5) $|c_n - c_{n-1}| > 2 * r_0$

Wobei c_n den Mittelpunkt des Teilkreises des Master-Shares der aktuellen Position kennzeichnet, und c_{n-1} den Mittelpunkt des Teilkreises des Master-Shares der alten Benutzerposition. Der Fall, in dem sich zwei Kreise der Master-Shares überschneiden, sollte aus Sicherheitsgründen immer vermieden werden, da ein Angreifer in diesem Fall anhand der Geschwindigkeit des Benutzers die Position auf bestimmte Gebiete einschränken kann (maximum-velocity Attack, später mehr dazu).

4.3.2 Verbesserter Share-generation Algorithmus (Fall 2)

Der nun vorgestellte verbesserte Share-generation Algorithmus kommt in speziellen Fällen mit deutlich weniger Update-Nachrichten aus, als der in [DSR11] vorgestellte Algorithmus.

Algorithmus 4.2 Verbesserter Share-generation Algorithmus (Fall 2)

```
1: function GEN_MINIMUM_OF_N_SHARES(r_0,n,c_1)
2:     Δ_φ ← r_0 / n
3:     S_0 ← random master-share; c_0 ← centerOf(S_0)
4:     while (dist(c_0, c_1) >= Δ_φ) do
5:         S_0 ← random master-share; c_0 ← centerOf(S_0)
6:     end while
7: return S_0
8: end function
```

Die Prüfung der Bedingungen in den Gleichungen 4.4 und 4.5 werden bereits vor Aufruf des Algorithmus durchgeführt. Der Algorithmus erhält als Parameter den Radius r_0 des Master-Share Teilkreises, sowie den Mittelpunkt c_1 des nächstkleineren bereits vorhandenen Shares S_1 und die Anzahl an Location-Servern n. Es wird zunächst in (2) die Variable Δ_ϕ initialisiert, die die Verkleinerung des Radius pro Verfeinerungs-Schritt angibt. Dann wird durch Zufall ein neuer Master-Share berechnet(3). Da jeder Share den nächstkleineren Share komplett enthalten muss, muss nur überprüft werden, ob der neu erzeugte Master-Share den nächstkleineren Share enthält (4), da dieser bereits alle anderen Shares enthält. In (7) wird der neue Master-Share an den entsprechenden Location-Server gesendet.

Anschließend wird darauf eingegangen, was passiert, wenn keiner der beiden vorgestellten Ansätze angewendet werden kann. Zunächst wird erklärt, warum sich die Teilkreise der Shares über mehrere Updates hinweg nicht überschneiden dürfen.

4.4 Verbesserung des Share-generation Algorithmus (Fall 3)

4.4.1 Maximum Movement Boundary

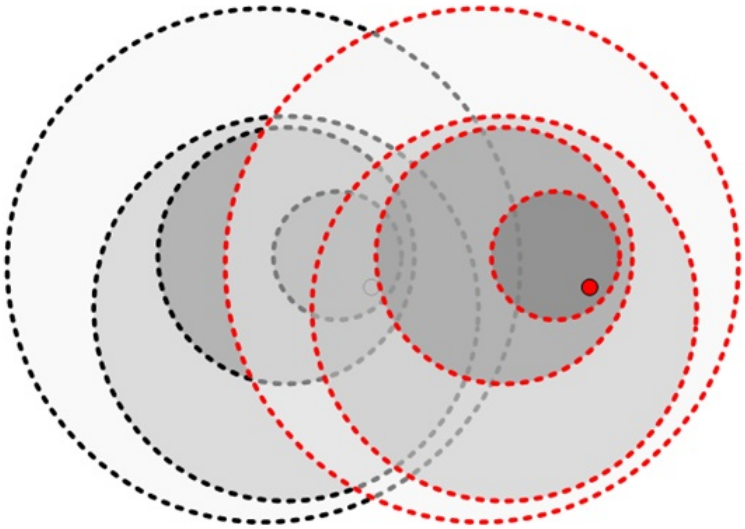


Abbildung 4.4: Fall 3, der Benutzer befindet sich außerhalb des Teilkreises des Master-Shares und die Teilkreise der Master-Shares überschneiden sich. (Quelle: [Skv12])

Wie bereits erwähnt, darf man aus Sicherheitsgründen im Falle einer Überschneidung der Teilkreise (siehe Abbildung 4.4) keine Shares neu schicken, bis die Überschneidung nicht mehr vorhanden ist. Der Grund ist das Prinzip der „Maximum Movement Boundary“: Bewegt sich ein mobiles Objekt linear mit der Geschwindigkeit v, so kann es in der Zeit t maximal die Strecke s zurücklegen, die sich aus

$$s = v * t \tag{4.6}$$

ergibt. Dies bietet auch die Grundlage für eine „Maximum Velocity Attack“: Kommt ein Angreifer beispielsweise in Besitz zweier aufeinanderfolgend gesendeten Master-Shares S_i und S_{i+1}, so kennt er auch die Zeitpunkte t_1 und t_2, zu denen die Shares gesendet wurden. Besitzt er zusätzlich noch Informationen über das mobile Objekt (z. B. weiß er, dass die Person zu Fuß unterwegs ist und kann die maximale Bewegungsgeschwindigkeit v_{max} auf 10 m/s abschätzen), kann er anhand von Formel 4.6 die maximale Strecke s_{max} berechnen, die der Benutzer in der Zeitspanne

$$t = t_2 - t_1 \tag{4.7}$$

zurückgelegt hat. Nun kann er ausgehend von dem Mittelpunkt des Teilkreises des Master-Shares S_i einen weiteren Kreis bestimmen, der den Radius s_{max} besitzt. Der Benutzer befindet sich mit hoher Wahrscheinlichkeit innerhalb der Schnittfläche des Kreises von Master-Share s_{i+1} und des Kreises mit Radius s_{max} (Siehe Abbildung 4.5).

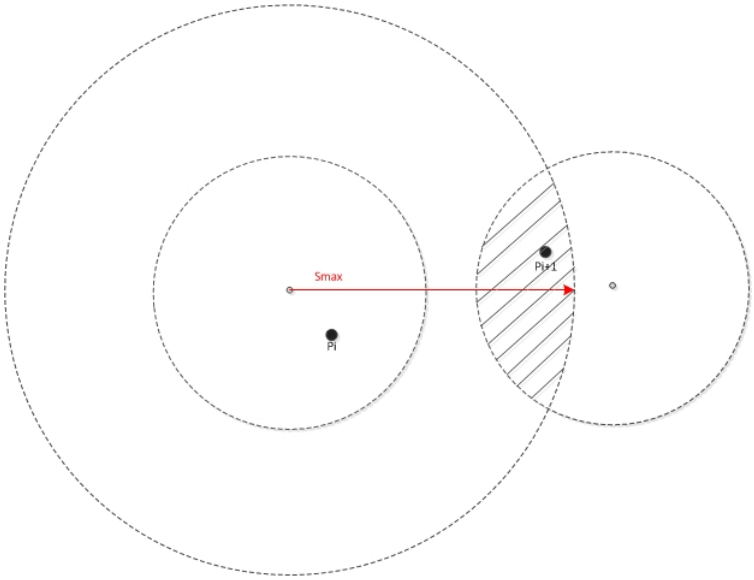


Abbildung 4.5: Veranschaulichung einer Maximum Velocity Attack: Anhand der maximal zurückgelegten Strecke s_{max} kann ein Angreifer die mögliche Fläche, in der sich der Benutzer aufhält (in der Abbildung schraffiert), eingrenzen

Die Größe des Kreises des Master-Shares S_{i+1} trägt also in diesem Fall nur wenig zur erhöhten Sicherheit bei.
Es gibt Möglichkeiten, um solche Attacken zu vermeiden. So erwähnen [WDR12] in ihrem Paper die Möglichkeit, die Update-Nachrichten so zu verzögern, dass innerhalb der Zeitspanne zwischen den Update-Nachrichten der Benutzer jeden möglichen Punkt im Teilkreis des Master-Shares S_{i+1} erreichen kann. Bezieht man dies auf Abbildung 4.4, so müsste der Kreis mit Radius s_{max} den kompletten Teilkreis des Master-Shares S_{i+1} einschließen. Je nach Geschwindigkeit des Benutzers und Größe der Kreise kann dies allerdings zu inakzeptablen Zeiträumen zwischen den Update-Nachrichten führen. Die Service-Qualität des LBS würde dadurch deutlich eingeschränkt. Beträgt z. B. der Radius des Teilkreises des Master-Shares 10 km, und der Benutzer bewegt sich mit einer maximalen Geschwindigkeit von 10 km/h, so müsste man um eine Maximum-Velocity Attack zu verhindern, die Update-Nachricht erst nach einer Stunde schicken.
Das wäre allerdings unpraktikabel und schränkt die Service-Qualität der LBS erheblich ein. Eine weitere Möglichkeit wäre eine Verringerung des Radius des Teilkreises des Master-Shares, um Überschneidungen zu verhindern. Dies würde die Sicherheit der Benutzerposition allerdings sehr stark beeinträchtigen. Deshalb stellt Fall 3 den absoluten worst-case dar, und führt dazu, dass alle Verschiebungs-Vektoren neu erzeugt werden müssen und an alle Location-Server (sowie an alle LBS) erneut gesendet werden müssen. In Fall 3 werden also genauso viele Nachrichten produziert wie im Basis-Ansatz.

Im nächsten Kapitel werden die hier vorgestellten Ansätze zur Nachrichtenoptimierung mathematisch analysiert, ausgewertet und miteinander verglichen.

5 Analyse und Auswertung der Ansätze zur Reduktion des Kommunikationsaufwandes

In diesem Kapitel werden die in 4 vorgestellten Ansätze analysiert, ausgewertet und diskutiert sowie ein abschließendes Fazit gegeben. Zunächst erfolgt jedoch ein kurzes Kapitel, das Klarheit im Bezug auf die Nomenklatur von Shares und Teilkreisen schaffen soll.

5.1 Bezeichnung von Shares und Teilkreisen

Da die Nomenklatur im Bezug auf Shares und Teilkreise zur Verwirrung führen kann, wird nochmals ausdrücklich darauf hingewiesen, dass bei den Share-Sets der Master-Share immer durch S_0 bezeichnet wird, und dann jeder Share, der vektoriell aufaddiert wird, fortlaufend um eins inkrementiert wird. Daher beginnt das Share-Set immer bei S_0 und endet immer bei S_{n-1}.
Bei der Nummerierung der Teilkreise, in denen sich der Benutzer befinden kann, ist es genau umgekehrt. Der innerste Teilkreis (dessen Mittelpunkt S_{n-1} ist), wird also beginnend mit der Nummer Eins nummeriert, und fortlaufend nach außen um eins inkrementiert. Abbildung 5.1 veranschaulicht dies.

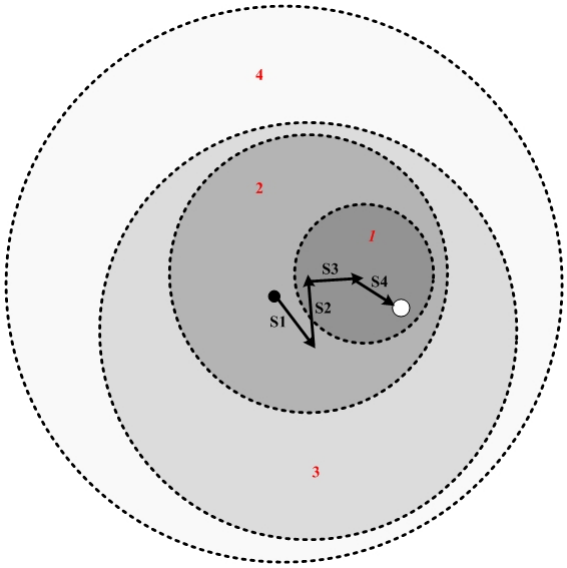


Abbildung 5.1: Bezeichnung von Shares (Schwarz) von und Teilkreisen (Rot)

Im Anschluss werden jeweils die Kommunikationskosten zwischen dem mobilen Objekt und den Location-Servern für Fall 1 und Fall 2 ausgewertet.

5.2 Nachrichteneinsparung bei MO - LS Kommunikation

5.2.1 Auswertung von Fall 1 (MO - LS)

Im Folgenden wird die Kommunikation zwischen dem mobilen Objekt und den Location-Servern betrachtet.
In Fall 1 werden Update-Nachrichten eingespart, wenn sich der Benutzer nur sehr wenig oder gar nicht bewegt. Befindet sich der Benutzer nach einer Bewegung noch innerhalb des k-ten Teilkreises (beginnend vom Teilkreis des innersten Refinement-Shares S_n), so müssen lediglich n-k Shares (beginnend vom n-ten Share) neu berechnet und verschickt werden. Eine prozentuale Einsparung p_e von Nachrichten lässt sich daher allgemein über folgende Formel berechnen:

$$p_e = \frac{n-k}{n} \tag{5.1}$$

Im best-case, wenn also $k = 1$ ist, und sich der Benutzer somit innerhalb des Teilkreises des feinsten Refinement-Shares befindet, muss lediglich ein Refinement-Share neu berechnet und geschickt werden. Bei $n = 5$ entspricht das einer Einsparung von 80%, bei $n = 20$ einer Einsparung von 95%.
Im worst-case befindet sich der Benutzer gerade noch innerhalb des Teilkreises des Master-Shares, also $k = n$-1. Dies entspricht für $n = 5$ einer Einsparung von 20% und für $n = 20$ einer Einsparung von 5%. Betrachtet man die gesamte Anzahl an Nachrichten N_1, die in Fall 1 geschickt werden müssen, so ist diese durch folgende Formel gegeben:

$$N_1 = k \tag{5.2}$$

Abhängig vom k-ten Teilkreis, in dem sich der Benutzer befindet, müssen dementsprechend viele Refinement-Shares neu berechnet und an die Location-Server geschickt werden.

5.2.2 Auswertung von Fall 2 (MO - LS)

Im Folgenden wird die Kommunikation zwischen dem mobilen Objekt und und den Location-Servern betrachtet.
In Fall 2 werden Update-Nachrichten eingespart, wenn sich der Benutzer zwischen den Updates mindestens so weit bewegt hat, dass sich die Teilkreise der Master-Shares nicht überschneiden. Ist dies der Fall, muss lediglich der Master-Share S_0 neu gesendet werden, da die Refinement-Shares eine relative Positionsangabe darstellen und einfach an den Mittelpunkt des Master-Share Kreises angehängt werden können.
Kann also Fall 2 angewandt werden, sieht die Formel für die prozentuale Einsparung p_e von

Update-Nachrichten gegenüber dem naiven Fall (alle Shares updaten) folgendermaßen aus:

$$(5.3)\quad p_e = \frac{n-1}{n}$$

Bei $n = 5$ entspricht das einer Einsparung von 80%, bei $n = 20$ einer Einsparung von 95%. Betrachtet man die Anzahl an Nachrichten N_2, die in Fall 2 geschickt werden müssen, so ist diese durch folgende Formel gegeben:

$$(5.4)\quad N_2 = 1$$

Da immer nur der Master-Share upgedatet wird, muss immer nur 1 Nachricht geschickt werden (nämlich der Master-Share S_0 an den jeweiligen Server, der den Master-Share abspeichert).

5.2.3 Analytischer Vergleich von Fall 1 und Fall 2

Vergleicht man Fall 1 mit Fall 2, könnte man schnell zum Schluss kommen, dass Fall 2 zu bevorzugen ist, da dort (außer für $k = 1$) immer weniger Update-Nachrichten gesendet werden müssen. Wenn man lediglich die Kommunikation zwischen dem mobilen Gerät und dem Location-Server betrachtet, ist diese Annahme richtig. Betrachtet man allerdings zusätzlich die Kommunikation zwischen LBS und Location-Server, ist Fall 2 oftmals ungünstiger:
Im Folgenden wird davon ausgegangen, dass der Location-Server bei der Aktualisierung eines Shares durch das mobile Gerät diese Information an jeden LBS weiterleitet, der auf diesen Share Zugriffsrechte besitzt (ihn also für seine Berechnungen kennen muss). Wird also der Master-Share neu berechnet (Fall 2), muss dieser vom entsprechenden Location-Server an alle LBS weitergeleitet werden (da der Master-Share immer benötigt wird). Muss man allerdings z. B. nur die letzten beiden Shares updaten (Fall 1), hängt es von den Zugriffs-Wahrscheinlichkeiten der LBS auf diese Shares ab, ob sie an diese weitergeleitet werden müssen oder nicht. Da manche LBS die exakte Benutzerposition kennen müssen, manche wiederum nicht, ist nun ein möglich realitätsnahes, theoretisches Modell zu erarbeiten, mit dessen Hilfe man die Zugriffe der LBS auf die einzelnen Share-Sets berechnen kann. Zunächst werden die Anfrage-Wahrscheinlichkeiten für die Share-Sets beispielhaft mathematisch hergeleitet.

5.2.4 Beispiel: Mathematische Herleitung der Anfrage-Wahrscheinlichkeiten

Das Anfragen von k aus n Refinement-Shares kann als mathematisches Zufallsexperiment gesehen werden. Folgendes Beispiels soll als Motivation dienen: Angenommen, es gibt $n = 5$ Shares, also den Master-Share S_0, sowie das Set von Refinement-Shares $[S_1, S_2, S_3, S_4]$. Dabei ist S_0 auf dem Location-Server LS_0 abgelegt, S_1 auf dem Location-Server LS_1, usw. Ein LBS fragt nun entweder keinen, genau einen, zwei, drei oder alle vier Refinement-Shares an. Die Ergebnismenge Ω lautet also wie folgt: $\Omega =$

$\{0, S_1, S_2, S_3, S_4, S_1S_2, S_1S_3, S_1S_4, S_2S_3, S_2S_4, S_3S_4, S_1S_2S_3, S_1S_2S_4, S_1S_3S_4, S_2S_3S_4, S_1S_2S_3S_4\}$ (Dabei heißt 0, dass kein Share angefragt wird).
Mittels Kombinatorik können zu den einzelnen Ergebnissen jeweils die Wahrscheinlichkeiten für deren Eintritt berechnet werden. Die Grundannahme hierbei ist, dass jeder LBS den Master-Share S_0 bereits kennt (da er ja sonst mit einem Refinement-Share nichts anfangen kann) und die Anfrage-Wahrscheinlichkeit jedes einzelnen Refinement-Shares die selbe ist, nämlich $\frac{1}{n-1}$, also $\frac{1}{4}$. Es ergeben sich folgende Formeln:

(5.5) $P(0\,aus\,4) = 0,75^4 = 0,32$

(5.6) $P(1\,aus\,4) = 0,75^3 * 0,25^1 * 4 = 0,42$

(5.7) $P(2\,aus\,4) = 0,75^2 * 0,25^2 * 6 = 0,21$

(5.8) $P(3\,aus\,4) = 0,75^1 * 0,25^3 * 4 = 0,047$

(5.9) $P(4\,aus\,4) = 0,25^4 = 0,004$

Da die Vektoren der Refinement-Shares in beliebiger Reihenfolge an den Master-Share gehängt werden können, ist es egal, ob zum Beispiel das Ereignis S_1 oder das Ereignis S_2 eintritt. Die beiden Ereignisse beschreiben beide den Fall, dass genau einer von vier Shares angefragt wird, daher werden in Formel 5.6 die Wahrscheinlichkeiten der vier Einzelereignisse addiert, analog zu den Formeln 5.7 und 5.8. Abbildung 5.2 stellt die Verteilung der Wahrscheinlichkeiten für $n = 4$ Refinement-Shares grafisch dar.

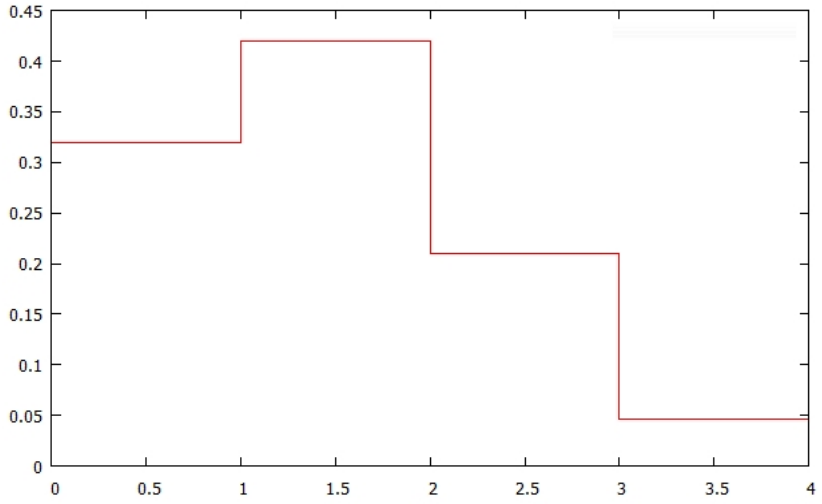


Abbildung 5.2: Mögliche Zugriffs-Wahrscheinlichkeitsverteilung der Refinement-Shares für den Fall $n = 4$

Mathematisch betrachtet ist die Wahrscheinlichkeit, dass ein LBS alle vier Refinement-Shares benötigt also nur 0,4%. Das würde bedeuten, dass von 1000 LBS nur 4 Stück die exakte Position des Benutzers kennen müssen, was kaum der Realität entsprechen wird. Darum wird im Anschluss ein möglichst realitätsnahes Modell vorgestellt, mit dessen Hilfe man die Zugriffswahrscheinlichkeiten auf die einzelnen Share-Sets modellieren kann.

5.2.5 Realitätsnahes Modell

Um die Anfrage-Wahrscheinlichkeiten realistischer zu gestalten, sollen folgende Überlegungen in das Modell mit aufgenommen werden:
Wird das System nun so implementiert, dass ein LBS bei Anfrage eines einzelnen Refinement-Shares immer auf LS_1 zugreift, bei Anfrage von zwei Shares immer auf LS_1 und LS_2, bei Anfrage von drei Shares immer auf LS_1,LS_2 und LS_3 und bei Anfrage von vier Shares immer auf LS_1,LS_2,LS_3 und LS_4, so lässt sich die Ergebnismenge des obigen Zufallsexperiments deutlich reduzieren, und zwar auf $\Omega = \{0, S_1, S_1S_2, S_1S_2S_3, S_1S_2S_3S_4\}$. Der Fall, dass kein Refinement-Share angefragt wird, ist äquivalent zu dem Fall, dass nur der Master-Share S_0 angefragt wird. Daher kann dieser Fall als S_0 bezeichnet werden, und es ergibt sich als finale Ergebnismenge $\Omega = \{S_0, S_1, S_1S_2, S_1S_2S_3, S_1S_2S_3S_4\}$. Unter der Annahme, dass S_0 immer eine Anfrage-Wahrscheinlichkeit von 100% besitzt, da ihn jeder LBS kennen muss, lässt sich die Anfrage-Wahrscheinlichkeit P_a eines Share-Sets $[S_0...S_i]$ über folgende Formel berechnen:

$$P_a([S_0...S_i]) = 1 - \frac{i}{n} \tag{5.10}$$

Hierbei ist n die Gesamtanzahl an Shares (inklusive Master-Share), S_i ist der i-te Share des Sets (da das Set nicht bei S_1 beginnt, sondern bei S_0) und $\frac{1}{n}$ ist die Anfrage-Wahrscheinlichkeit eines einzelnen Refinement-Shares. So ist für $n = 5$ Shares die Anfrage-Wahrscheinlichkeit P_a für das Share-Set $[S_0...S_0] = [S_0]$ = 100%, für das Share-Set $[S_0...S_1]$ = 80%,für das Share-Set $[S_0...S_2]$ = 60%, für das Share-Set $[S_0...S_3]$ = 40%, und für das Share-Set $[S_0...S_4]$ = 20%. Die Formel berücksichtigt also die Tatsache, dass die Wahrscheinlichkeit einer Anfrage von Refinement-Shares linear mit steigendem Grad des Refinement-Shares abnimmt.
Abbildung 5.3 stellt die Verteilung der Wahrscheinlichkeiten für $n = 5$ Shares grafisch dar.

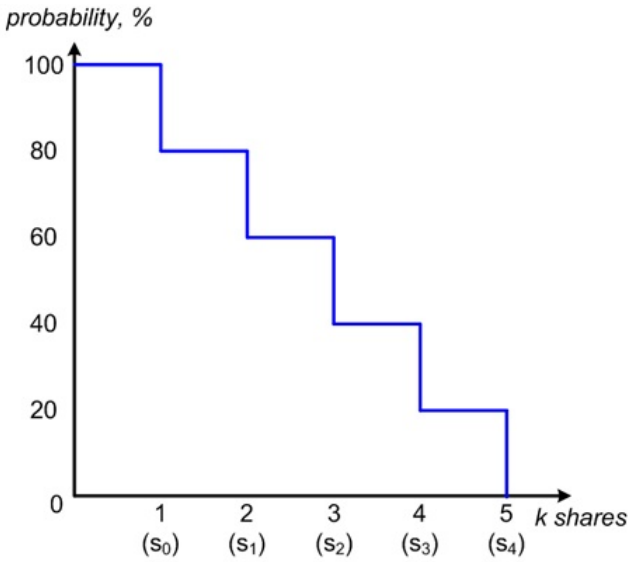


Abbildung 5.3: Lineare Wahrscheinlichkeitsverteilung von Share-Sets (Quelle: [Skv12])

Daher ist es für die LS - LBS Kommunikation sicherlich oftmals besser, wenn nicht durch jede Update-Message der Master-Share upgedatet wird, da dann der LS diese Änderung wiederum an alle LBS weiterleiten muss. Wird hingegen zum Beispiel nur der letzte Refinement-Share S_n upgedatet, der laut obiger Formel im Share-Set $[S_0...S_n]$ eine weitaus geringere Zugriffswahrscheinlichkeit besitzt (im Vergleich zum Master-Share S_0, der immer eine Zugriffswahrscheinlichkeit von 1 besitzt), muss der LS demnach weniger LBS's informieren.
Im Anschluss werden jeweils die Kommunikationskosten zwischen den Location-Servern und den location-based Services für Fall 1 und Fall 2 ausgewertet.

5.3 Nachrichteneinsparung bei LS - LBS Kommunikation

5.3.1 Auswertung von Fall 1 (LS - LBS)

Im Folgenden wird die Kommunikation zwischen den Location-Servern und den location-based Services betrachtet.
Die prozentuale Nachrichteneinsparung P_e in Fall 1 gegenüber dem naiven Fall (alle Shares werden upgedatet) lässt sich durch folgende Formel berechnen:

$$P_e = \frac{\sum_{i=0}^{n-1} P_a([S_0..S_i]) - \sum_{i=n-k}^{n-1} P_a([S_0..S_i])}{\sum_{i=0}^{n-1} P_a([S_0..S_i])} \tag{5.11}$$

Formel 5.11 lässt sich vereinfachen, da die beiden Summen im Zähler direkt subtrahiert werden können:

$$P_e = \frac{\sum_{i=0}^{n-1-k} P_a([S_0..S_i])}{\sum_{i=0}^{n-1} P_a([S_0..S_i])} \quad (5.12)$$

Betrachtet man wieder das Standardbeispiel im best-case mit $n = 5$ und $k = 1$, so spart man laut Formel 5.12 im Fall 1 ganze 93,3% an Update-Nachrichten. Ist $k = 2$, kann man noch 80% einsparen. Ab $k = 3$ können im Fall 1 nur noch 60% eingespart werden.
Betrachtet man nicht die prozentuale Einsparung, sondern die Gesamtanzahl an Nachrichten N_1, die in Fall 1 geschickt werden müssen, lässt sich diese durch folgende Formel berechnen:

$$N_1 = n_{LBS} * \sum_{i=n-k}^{n-1} P_a([S_0..S_i]) \quad (5.13)$$

Die Nachrichtenanzahl ergibt sich aus der Anzahl an LBS n_{LBS} multipliziert mit der Summe der Anfrage-Wahrscheinlichkeiten der Share-Sets, die als letzte Shares jeweils die upzudatenden Shares beinhalten.

5.3.2 Auswertung von Fall 2 (LS - LBS)

Im Folgenden wird die Kommunikation zwischen den Location-Servern und den location-based Services betrachtet.
Die prozentuale Nachrichteneinsparung P_e in Fall 2 gegenüber dem naiven Fall (alle Shares werden upgedatet) lässt sich durch folgende Formel berechnen:

$$P_e = \frac{(\sum_{i=0}^{n-1} P_a([S_0..S_i]) - P_a([S_0])) * n_{LBS}}{\sum_{i=0}^{n-1} P_a([S_0..S_i]) * n_{LBS}} \quad (5.14)$$

Der Zähler gibt die Anzahl an Nachrichten an, die im Falle eines Updates des Master-Shares S_0 weniger gesendet werden, der Nenner gibt die Anzahl an Nachrichten an, die im naiven Fall geschickt werden müssen. Sind zum Beispiel sowohl n als auch $n_{LBS} = 5$, können im Fall 2 laut Formel 5.14 66,6% an Nachrichten gegenüber dem naiven Fall eingespart werden.
Formel 5.14 lässt sich noch weiter vereinfachen, da $P_a([S_0])$ direkt von der Summe subtrahiert werden kann, und sich n_{LBS} kürzen lässt. Es ergibt sich folgende Formel:

$$P_e = \frac{\sum_{i=1}^{n-1} P_a([S_0..S_i])}{\sum_{i=0}^{n-1} P_a([S_0..S_i])} \quad (5.15)$$

Betrachtet man nicht die prozentuale Einsparung, sondern die Gesamtanzahl an Nachrichten N_2, die in Fall 2 geschickt werden müssen, lässt sich diese durch folgende Formel berechnen:

(5.16) $N_2 = n_{LBS}$

Da die Anfrage-Wahrscheinlichkeit für den Master-Share S_0 immer 1 ist, und dieser somit an alle LBS übermittelt werden muss, ist die Gesamtanzahl an übermittelten Nachrichten gleich der Anzahl an LBS n_{LBS}.

5.3.3 Gewichtung der LS - LBS Kommunikation

Da es wichtiger ist, die Kommunikationskosten zwischen MO und LS zu reduzieren, sollte die LS - LBS Kommunikation nicht so stark ins Gewicht fallen. Ist es besser, wenn 5 Nachrichten zwischen MO und LS eingespart werden können, oder stattdessen 10 Nachrichten zwischen LS und LBS eingespart werden können? Daher wird ein Gewichtungsfaktor G_0 eingeführt, der beliebig gewählt werden kann. So kann die Nachrichteneinsparung zwischen MO und LS nach belieben gewichtet werden. Soll z. B. eine gesparte Nachricht zwischen MO und LS genauso viel Wert sein wie eine Nachricht zwischen LS und LBS, kann G_0 auf 1 festgelegt werden. Soll sie doppelt so viel Wert sein, wird G_0 auf 2 festgelegt, usw.
Im Anschluss wird eine Funktion präsentiert, die berechnet, wie viele Teilkreise (k) sich der Benutzer zwischen den Updates bewegt haben darf, sodass durch die Anwendung von Fall 1 mehr Nachrichten eingespart werden können als durch die Anwendung von Fall 2. Dadurch kann dann bestimmt werden, wann es besser ist, Fall 1 oder Fall 2 anzuwenden.

5.4 Minimierung der Nachrichtenanzahl

Anhand der oben präsentierten Formeln kann nun eine Funktion vorgestellt werden, die anhand der Parameter n, G_0 und n_{LBS} bestimmt, ob es in der jeweiligen Situation vorteilhafter ist, Fall 1 oder Fall 2 auszuwählen. Dazu werden die Formeln, die die Nachrichtenanzahl in Fall 1 und Fall 2 berechnen, miteinander gleichgesetzt (Gleichung 5.17). Der linke Teil der Gleichung berechnet die benötigte Anzahl an Update-Nachrichten im Fall 1 und der rechte Teil die benötigte Anzahl an Update-Nachrichten im Fall 2:

(5.17) $(k * G_0) + (n_{LBS} * \sum_{i=n-k}^{n-1} P_a([S_0..S_i])) = G_0 + n_{LBS}$

Um herauszufinden, wann welcher Fall zu bevorzugen ist, kann anhand von Gleichung 5.17 durch geschicktes Einsetzen von k der Wert für k herausgefunden werden, bis zu dem Fall 1 noch besser ist als Fall 2. In der Annahme, die Anzahl an Location-Servern n sei bekannt, ebenso wie die Anzahl an LBS n_{LBS} und der Gewichtungsfaktor G_0, kann man im Voraus berechnen, im wievielten Teilkreis (k) sich der Benutzer mindestens befinden muss, sodass durch Anwendung von Fall 1 mehr Nachrichten eingespart werden können als durch

Anwendung von Fall 2. Ist dieser Wert überschritten, wird abgewartet bis die Bedingung von Fall 2 erfüllt ist, und Fall 2 wird angewendet. Sollte auch eine Anwendung von Fall 2 nicht möglich sein, muss der worst-case, also Fall 3 angewandt werden.
Um den Wert k zu finden, der die Grenze markiert, wird eine Art brute-force Methode angewandt, das heißt es werden von 1 bis n alle möglichen Werte für k durchprobiert. Da k niemals größer sein kann als n, ist die Laufzeit der brute-force Methode auf *O(n)* beschränkt. k muss zwingend ein Integer sein, da es im System-Modell keine halben oder viertel Teilkreise gibt (und da k in der Summe vorkommt), daher wird k pro Schleifendurchlauf inkrementiert. Die Funktion, die berechnet, bis zu welchem k Fall 1 noch besser ist, wird durch Algorithmus 5.1 beschrieben.
In (2) und (3) werden zunächst die Variablen *kGuessed* und *kGuessedNext* mit 1 und 2 initiali-

Algorithmus 5.1 Minimierungs-Funktion

1: **function** MINIMIZE_MESSAGES(n, G_0, n_{LBS})
2: $kGuessed \leftarrow 1$;
3: $kGuessedNext \leftarrow 2$;
4: $numberOfMessagesCase2 \leftarrow G_0 + n_{LBS}$;
5: $numberOfMessagesCase1Next \leftarrow (kGuessedNext * G_0) + (n_{LBS} * \sum_{i=n-kGuessedNext}^{n-1} P_a([S_0..S_i]))$;
6: **while** $((numberOfMessagesCase2 >= numberOfMessagesCase1Next))$ **do**
7: $kGuessed++$;
8: $kGuessedNext++$;
9: $numberOfMessagesCase1Next \leftarrow (kGuessedNext * G_0) + (n_{LBS} * \sum_{i=n-kGuessedNext}^{n-1} P_a([S_0..S_i]))$;
10: **end while**
11: **return** *kGuessed*
12: **end function**

siert. Dann werden jeweils die Anzahl der Nachrichten für Fall 1 und die Nachrichtenanzahl in Fall 1, wenn statt k das um eins größere k benutzt wird, berechnet (4+5). Solange nun die Nachrichtenanzahl von Fall 2 größer ist, als die von Fall 1 mit dem nächstgrößeren k (6), werden kGuessed und kGuessedNext inkrementiert (7 + 8), und der Wert für die Anzahl von Fall 1 mit dem nächstgrößeren k aktualisiert (9). Ist die Bedingung erfüllt, wird die while-Schleife verlassen, und das aktuelle k wird ausgegeben (11).
Das gefundene k gibt nun den Teilkreis an (ausgegangen vom innersten Teilkreis), in dem sich der Benutzer nach einem Update maximal befinden darf, sodass Fall 1 gerade noch weniger Nachrichten produziert als Fall 2.
Zur Veranschaulichung soll folgendes Beispiel dienen: Sind sowohl n und auch n_{LBS} = 5, so ergeben sich laut Gleichung 5.17 Nachrichtenzahlen, die in Abbildung 5.4 veranschaulicht sind.
Befindet sich der Benutzer also nach einem Update noch innerhalb des 2. Teilkreises, spart man durch die Anwendung von Fall 1 gegenüber Fall 2 eine Nachricht. Befindet sich der

k	Nachrichten Fall 1	Nachrichten Fall 2
1	2	6
2	5	6
3	8	6

Abbildung 5.4: Anzahl an Nachrichten in den einzelnen Fällen für n und $n_{LBS} = 5$

Benutzer allerdings innerhalb des 3. Teilkreises, ist Fall 2 um zwei Nachrichten sparsamer als Fall 1. Da k lediglich die Nachrichten-Anzahl in Fall 1 beeinflusst, wird mit steigendem k die Nachrichten-Anzahl in Fall 1 immer höher. Das heißt, dass hier ab $k = 3$ immer Fall 2 angewendet werden sollte um Update-Nachrichten einzusparen. Es ist noch zu zeigen, dass Algorithmus 5.1 immer terminiert:
Sofern folgende Aussage gilt, terminiert Algorithmus 5.1.

$$(5.18)\quad \forall k, k \in [1,n] \exists k : |(k * G_0) + (n_{LBS} * \sum_{i=n-k}^{n-1} P_a([S_0..S_i]))| > |G_0 + n_{LBS}|$$

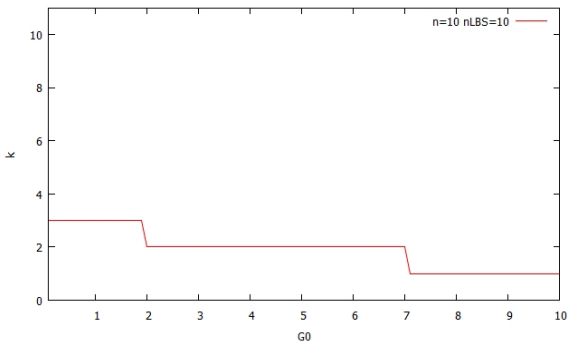
Wenn sich während der Laufzeit von Algorithmus 5.1 n und n_{LBS} nicht mehr ändern, ist der rechte Teil aus Gleichung 5.18, der die Anzahl an Nachrichten in Fall 2 angibt, ein konstanter Wert. Setzt man nun für k den Maximalwert n ein, ergibt sich Gleichung 5.19:

$$(5.19)\quad |(n * G_0) + (n_{LBS} * (\frac{n}{2} + 0,5))| > |G_0 + n_{LBS}|$$

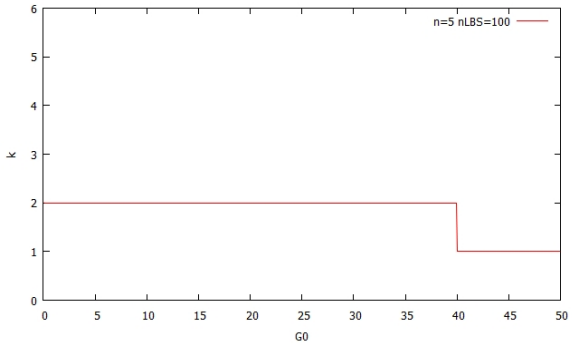
Da $n > 1$, ist Gleichung 5.19 auf jeden Fall erfüllt. Es wurde also ein Wert für k gefunden, der die Bedingung aus Gleichung 5.18 erfüllt. Somit terminiert Algorithmus 5.1 spätestens beim Wert $k = n$.
In den folgenden beiden Unterkapiteln wird nun untersucht, inwiefern sich verschiedene Werte von G_0 und $nLBS$ auf k auswirken. Die Ergebnisse werden grafisch dargestellt.

5.4.1 Grafische Auswertung von *k* in Abhängigkeit von G_0

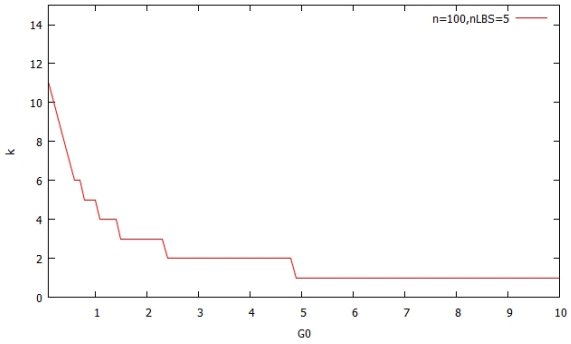
Eine Auswertung mit verschiedenen Werten der Parameter und dem daraus resultierenden Wert für k ist in Abbildung 5.5 zu sehen.
Abbildung 5.5a zeigt, dass sich k bei gleicher Anzahl von LBS und n durch das Erhöhen von G_0 relativ schnell dem minimalen Wert von $k = 1$ nähert, ab $G_0 = 7$ verändert sich k nicht mehr. Gibt es deutlich mehr LBS als Location-Server (wie auf Abbildung 5.5b zu sehen), so wird ein deutlich höherer Wert von G_0 benötigt, um k zu minimieren. Abbildung 5.5c zeigt den Fall, in dem es deutlich mehr Location-Server gibt als LBS. k fällt wieder relativ schnell auf das Minimum von k=1, allerdings ist der Startwert von k im Vergleich zu dem Fall aus Abbildung 5.5a deutlich höher.
Die Abbildungen 5.5d, 5.5e und 5.5f visualisieren die Verteilung, die angibt welcher der beiden Fälle ab welchem k besser ist. Dabei stellt die grüne Fläche Fall 1 und die rote Fläche Fall 2 dar. Es ist deutlich zu erkennen, dass Fall 2 weitaus öfter angewendet wird als Fall 1.



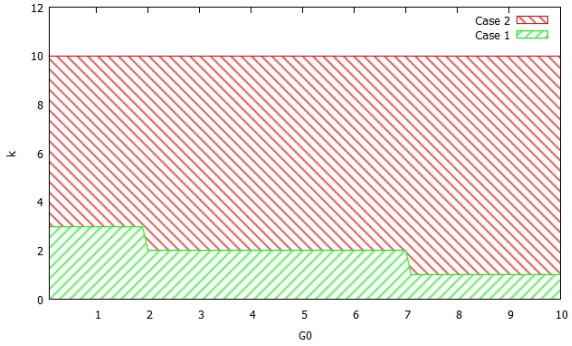
(a) k in Abhängigkeit von G_0 ($n = 10$, $n_{LBS} = 10$)



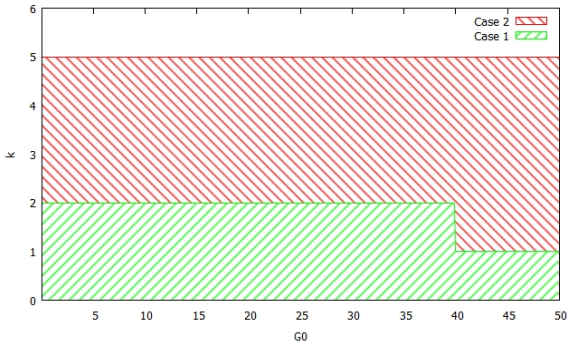
(b) k in Abhängigkeit von G_0 ($n = 5$, $n_{LBS} = 100$)



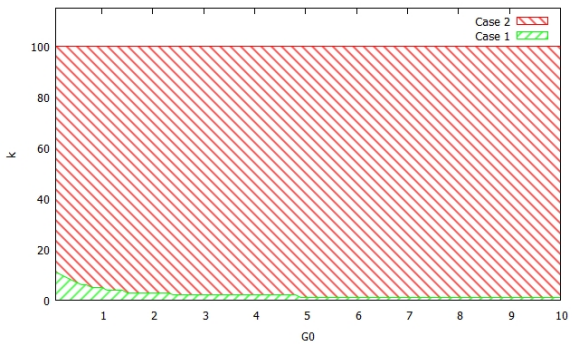
(c) k in Abhängigkeit von G_0 ($n = 100$, $n_{LBS} = 5$)



(d) Verteilung der beiden Fälle bei variierendem G_0 ($n = 10$, $n_{LBS} = 10$)



(e) Verteilung der beiden Fälle bei variierendem G_0 ($n = 5$, $n_{LBS} = 100$)



(f) Verteilung der beiden Fälle bei variierendem G_0 ($n = 100$, $n_{LBS} = 5$)

Abbildung 5.5: k in Abhängigkeit von G_0

Dies lässt sich dadurch erklären, dass für G_0 ausschließlich positive Werte eingesetzt wurden, denn je größer G_0 ist, desto mehr ist eine Nachricht zwischen MO und LS Wert, und desto öfter wird Fall 2 angewendet.

5.4.2 Grafische Auswertung von *k* in Abhängigkeit von *nLBS*

Ebenfalls interessant ist das Verhalten im Falle eines fixen Wertes für G_0 und n und einem variierenden n_{LBS}. Abbildung 5.6 stellt diesen Fall grafisch dar.

Abbildung 5.6a zeigt die Abhängigkeit von k im Falle eines klein gewählten festen G_0: k startet bei einem Wert von 7 und erreicht relativ schnell das lokale Maximum, das bei 13 liegt. Wird G_0 auf 1 gesetzt, startet k wie in Abbildung 5.6b zu sehen bei 1 und erreicht das lokale Maximum von 13 erst ab ca. 50 LBS. Abbildung 5.6c visualisiert den Fall, dass G_0 auf einen relativ hohen Wert gesetzt wird, nämlich 10. k startet wieder bei 1 und hat bei 100 LBS erst das lokale Maximum von 7 erreicht.
Mit steigender Anzahl an LBS bei festem G_0 konvergiert k also gegen einen Maximalwert, G_0 beeinflusst, wie schnell dieser Wert erreicht wird.
Die Abbildungen 5.6d, 5.6e und 5.6f visualisieren die Verteilung, die angibt welcher der beiden Fälle ab welchem k besser ist. Dabei stellt die grüne Fläche Fall 1 und die rote Fläche Fall 2 dar. Es ist wieder zu erkennen, dass die Kurve, die angibt, ab wann noch Fall 1 benutzt wird, mit steigendem G_0 abflacht. Auch ist wieder eine Dominanz von Fall 2 vorhanden, da ausschließlich positive Werte für G_0 eingesetzt wurden.
Anschließend wird überprüft, ob die Kapitel 4 gestellte Bedingung an die Nachrichtenoptimierung durch die vorgestellten Ansätze erfüllt werden kann.

5.5 Verifizierung der Optimierungs-Bedingung

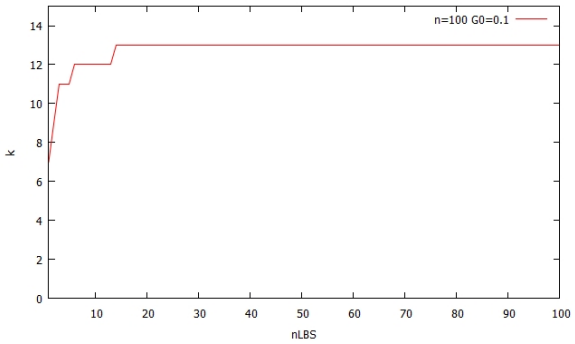
Die in Kapitel 4 an das System gestellte Bedingung soll nun abschließend noch überprüft werden.
Zunächst werden $|S_{opt}|und|S_{basic}|$ folgendermaßen umformuliert:

$$|S_{opt}| = |S_{Fall3}| + \sum_{i=2}^{n_{Traj}} (|S_{Fall1}[i]|or|S_{Fall2}[i]|or|S_{Fall3}[i]|) \tag{5.20}$$

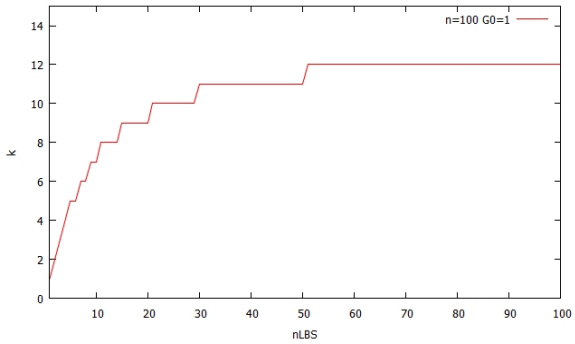
und

$$|S_{basic}| = \sum_{i=2}^{n_{Traj}} |S_{Fall3}[i]| \tag{5.21}$$

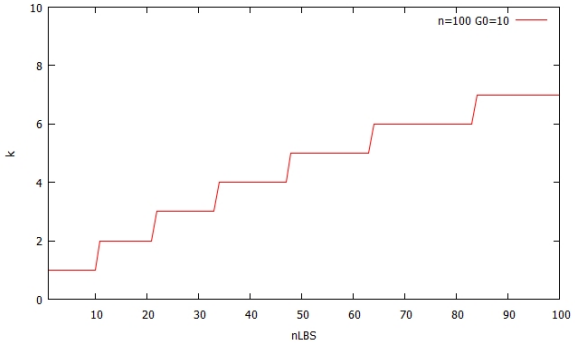
Formel 5.20 sagt aus, dass sich die Gesamtnachrichtenanzahl bei Eingabe einer Benutzer-Trajektorie mit n_{Traj} Benutzerpositionen im optimierten System aus der einmaligen Anwendung von Fall 3 (erste Benutzerposition) und der Summe der Anwendungen von Fall 1 oder Fall 2 oder Fall 3 auf die jeweilige Benutzerposition berechnen lässt.



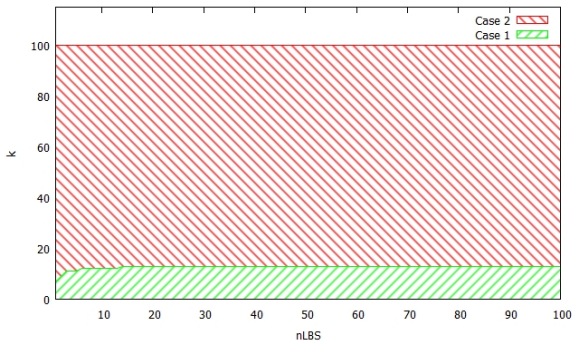
(a) k in Abhängigkeit von n_{LBS} (n=100, G_0=0.1)



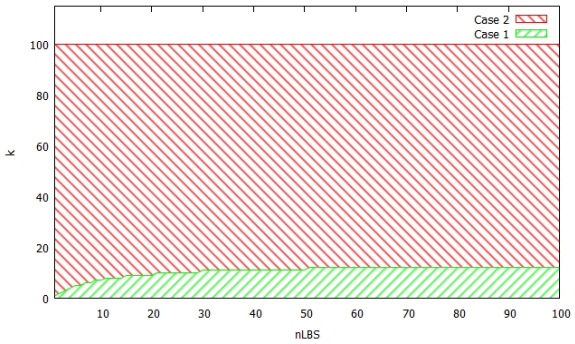
(b) k in Abhängigkeit von n_{LBS} (n=100, G_0=1)



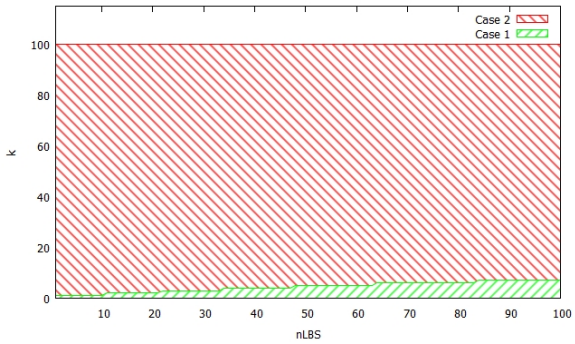
(c) k in Abhängigkeit von n_{LBS} (n=100, G_0=10)



(d) Verteilung der beiden Fälle bei variierendem n_{LBS} (n = 100, G_0 = 0.1)



(e) Verteilung der beiden Fälle bei variierendem n_{LBS} (n = 100, G_0 = 1)



(f) Verteilung der beiden Fälle bei variierendem n_{LBS} (n = 100, G_0 = 10)

Abbildung 5.6: k in Abhängigkeit von n_{LBS}

Formel 5.21 sagt aus, dass sich die Gesamtnachrichtenanzahl bei Eingabe einer Benutzer-Trajektorie mit n_{Traj} Benutzerpositionen im Basis-System aus der Summe der Anwendungen von Fall 3 auf die jeweilige Benutzerposition berechnen lässt.
Im Anschluss ist zu zeigen, dass Fall 1 und Fall 2 jeweils weniger Nachrichten produzieren als Fall 3. Hierzu kann man die folgenden Formeln betrachten, die jeweils die Nachrichtenanzahl in Fall 1, Fall 2 und Fall 3 berechnen (G_0 wurde der Einfachheit halber auf 1 gesetzt).

$$|S_{Fall1}| = k + (n_{LBS} * \sum_{i=n-k}^{n-1} P_a([S_0..S_i])) \tag{5.22}$$

$$|S_{Fall2}| = 1 + n_{LBS} \tag{5.23}$$

$$|S_{Fall3}| = n + \sum_{i=0}^{n-1} P_a([S_0..S_i]) * n_{LBS} \tag{5.24}$$

Da per Definition gelten muss, dass $k < n$, ist es offensichtlich, dass $|S_{Fall1}| < |S_{Fall3}|$, da sowohl die in Formel 5.22 vorkommende Summe als auch der Summand betragsmäßig kleiner sind als die entsprechende Summe und der entsprechende Summand in Formel 5.24. Ähnlich lässt sich zeigen, dass $|S_{Fall2}| < |S_{Fall3}|$, da immer von mindestens zwei Location-Servern ausgegangen werden kann, da sonst das gesamte System keinen Sinn machen würde. Also gilt $n >= 2$, womit der Summand in Formel 5.23 kleiner wäre als der entsprechende Summand in Formel 5.24. Das Ergebnis der Summe in Formel 5.24 ist ebenfalls größer als 1, da die Obergrenze der Summe $\frac{n}{2} + 0,5$ ist, was also bei $n = 2$ dem Wert 1,5 entspricht. Da n_{LBS} in Formel 5.23 keinen Faktor besitzt und der entsprechende Faktor in Formel 5.24 immer größer als 1 ist, gilt $|S_{Fall2}| < |S_{Fall3}|$.
Nun ist lediglich noch zu zeigen, dass pro Trajektorie mindestens immer einmal Fall1 oder Fall 2 angewendet werden kann, damit auch tatsächlich Nachrichten gegenüber dem Basis-Ansatz eingespart werden. Da die Anwendung der Fälle von Bedingungen abhängig sind, die wiederum vom Radius des Master-Shares abhängig sind, kann der Radius immer so gewählt werden, dass mindestens einmal einer der beiden Fälle angewendet werden kann: Ist r_0 zu klein um Fall 1 anzuwenden, kann man ihn entsprechend vergrößern. Ist r_0 andererseits zu groß um Fall 2 anzuwenden, kann man ihn entsprechend verkleinern.
Ebenfalls wurde die Bedingung gestellt, dass sich die Genauigkeit der Positionen nicht verschlechtern darf. Die Genauigkeit wurde so definiert, dass sie durch jeden hinzugefügten Vektor erhöht wird. Da bei n Location-Servern sowohl beim Basis-Ansatz als auch beim optimierten Ansatz jeweils n Vektoren erzeugt werden, und man bei beiden Ansätzen anhand von allen Vektoren die genaue Benutzerposition erhält, ist die Genauigkeit beider Ansätze gleich. Insbesondere ist die Genauigkeit des optimierten Ansatzes nicht schlechter, und somit ist auch diese Bedingung erfüllt.

5.6 Fazit und Schlussfolgerung

Der analytische Vergleich der beiden Fälle hat ergeben, dass es Situationsabhängig ist, ob einer der Fälle besser ist als der andere (Ein Fall ist besser, wenn insgesamt weniger Nachrichten geschickt werden müssen). Außerdem ist es wichtig, nicht nur die Kommunikation zwischen MO und LS zu betrachten, sondern auch die Kommunikation zwischen LS und LBS. Zu diesem Zweck wurde ein Gewichtungsfaktor G_0 eingeführt, der es ermöglicht, jeweils eine der beiden Kommunikationsarten zu gewichten. Zum Schluss wurde eine Funktion vorgestellt, die berechnen kann, wann welcher Fall besser ist, d.h. mit welchem Fall sich wann gegenüber dem naiven Fall mehr Nachrichten einsparen lassen.
Ein bisher ungelöstes Problem ist die Festlegung einer sinnvollen Gewichtung G_0. Natürlich könnte man den Benutzer entscheiden lassen, aber da dieser den mathematischen Hintergrund nicht kennt, ist dies nicht sehr sinnvoll. Betrachtet man Formel 5.17, dann erkennt man, dass für größer werdende G_0 der Wert von k immer kleiner wird. Umso stärker die MO-LS Kommunikation also gewichtet wird, desto weniger darf sich der Benutzer seit dem letzten Update bewegt haben, damit Fall 1 besser ist als Fall 2. Dies lässt sich insofern erklären, dass Fall 1 in den meisten Fällen mehr Nachrichten zwischen LS und LBS einspart als Fall 2, und Fall 2 meistens mehr Nachrichten zwischen MO und LS einspart als Fall 1. Ein größeres G_0 führt also zwangsläufig dazu, dass Fall 2 öfters zur Anwendung kommt als Fall 1. Betrachtet man zum Beispiel die Parameter $n = 10$, $n_{LBS} = 5$ und $G_0 = 1$, dann ergibt sich nach Algorithmus 5.1 für k der Wert 3. Befindet sich der Benutzer zwischen zwei Updates also noch innerhalb des dritten Teilkreises, können durch Anwendung von Fall 1 mehr Nachrichten eingespart werden als durch Anwendung von Fall 2. Lässt man alle Parameter gleich, ändert aber G_0 auf 2 (zwei eingesparte Nachrichten zwischen MO und LS sind also so viel Wert wie eine eingesparte Nachricht zwischen LS und LBS), ergibt sich für k nur noch der Wert 2.
Im nächsten Kapitel wird das System, das die vorgestellten Ansätze zur Nachrichtenoptimierung implementiert, simuliert und evaluiert. Die Ergebnisse der Evaluierung werden vorgestellt und diskutiert.

6 Evaluierung

6.1 Einleitung

Das System mit den in Kapitel 4 vorgestellten Verbesserungsvorschlägen (Fall 1: nur k Shares updaten und Fall 2: nur Master-Share updaten) soll nun unter möglichst realen Bedingungen simuliert werden, um die Effektivität der Verbesserungsvorschläge evaluieren und beurteilen zu können. Als Eingabedaten sollen echte Benutzer-Trajektorien dienen, die anschließend vorgestellt werden.

6.2 Datensatz

Zheng, Xie und Wei-Ying stellen in [ZXWY10] ein System namens „GeoLife“ vor, das Benutzern durch GPS-Daten die Möglichkeit gibt, ihre Reiseerfahrungen mit Freunden zu teilen. So können vergangene Reisen anhand von aufgezeichneten GPS-Trajektorien direkt im System nach erlebt werden. Durch das Auswerten von vielen Benutzerdaten können außerdem die interessantesten Reiserouten ermittelt werden, und das System kann als Reiseführer benutzt werden. Ebenfalls können durch GeoLife Benutzer mit ähnlichen Reiserouten gefunden werden, das System schlägt also potenzielle Freunde und auch beliebte Orte vor. GeoLife wurde von Microsoft Research Asia entwickelt, und über 3 Jahre hinweg wurden von 178 Benutzern GPS-Trajektorien aufgezeichnet. Diese gesammelten Daten umfassen Längengrad, Breitengrad, Höhe und einen Zeitstempel. Abbildung 6.1 zeigt einen Ausschnitt aus dem GeoLife-Datensatz.

```
Geolife trajectory
WGS 84
Altitude is in Feet
Reserved 3
0,2,255,My Track,0,0,2,8421376
0
39.984702,116.318417,0,492,39744.1201851852,2008-10-23,02:53:04
39.984683,116.31845,0,492,39744.1202546296,2008-10-23,02:53:10
39.984686,116.318417,0,492,39744.1203125,2008-10-23,02:53:15
39.984688,116.318385,0,492,39744.1203703704,2008-10-23,02:53:20
39.984655,116.318263,0,492,39744.1204282407,2008-10-23,02:53:25
39.984611,116.318026,0,493,39744.1204861111,2008-10-23,02:53:30
39.984608,116.317761,0,493,39744.1205439815,2008-10-23,02:53:35
39.984563,116.317517,0,496,39744.1206018519,2008-10-23,02:53:40
39.984539,116.317294,0,500,39744.1206597222,2008-10-23,02:53:45
39.984606,116.317065,0,505,39744.1207175926,2008-10-23,02:53:50
```

Abbildung 6.1: Ausschnitt aus dem GeoLife-Datensatz(Quelle: [ZXWY10])

Der Datensatz umfasst neben täglichen Routine-Trajektorien wie z. B. dem Weg zur Arbeit und zurück auch Trajektorien durch Wanderausflüge oder Fahrradtouren. Die meisten Trajektorien wurden in Abständen von 1-5 Sekunden oder 5-10 Metern aufgezeichnet (und fallen somit in die Kategorie on-the-fly tracking). Ebenfalls ist bei manchen Datensätzen auch das verwendete Transportmittel angegeben.
Ebenfalls wurden zur Simulation auch von mir selbst gesammelte GPS-Trajektorien benutzt, die mit der Software „GPS Logger for Android“ ([Men12]) auf einem HTC Wildfire Mobiltelefon aufgenommen wurden. Diese wurden aber in erster Linie zum Verifizieren des Programmes genutzt, das für die Simulation entwickelt wurde.
Im Anschluss wird die Simulationsumgebung präsentiert, die das verbesserte System simulieren soll.

6.3 Simulationsumgebung

Im Rahmen dieser Arbeit wurde ein Java-Programm entwickelt, das es ermöglicht, Datensätze von Trajektorien einzulesen, zu visualisieren und eine Simulation des Systems durchzuführen. Das Einlesen von Geolife-Trajektorien wird durch einen selbst implementierten Reader realisiert, zum Einlesen von Daten im gpx-Format wird die Bibliothek GPS Analysis ([Tay11]) verwendet. Zur Visualisierung wird der Webservice von OpenStreetMap ([Ope12]) verwendet, um den entsprechenden Kartenausschnitt anzuzeigen. Des Weiteren wird zum einen die Bibliothek „Image2Html“ ([Sab09]) verwendet, die die Antwort des Webservices, der das Bild als img-Tag zur Einbindung in Webseiten zurückliefert, in ein Java BufferedImage-Objekt umwandelt. Ebenfalls wird die Bibliothek „Geodesy'"([Gav08]) verwendet, die es ermöglicht, die Distanz zwischen zwei Punkten auf der Erde (oder anderen Ellipsoiden) zu berechnen. Da die Erde keine perfekte Kugel ist, wäre dies mit herkömmlichen Methoden sehr kompliziert. Abbildung 6.2 zeigt einen Screenshot der Benutzeroberfläche des Simulationsprogramms.

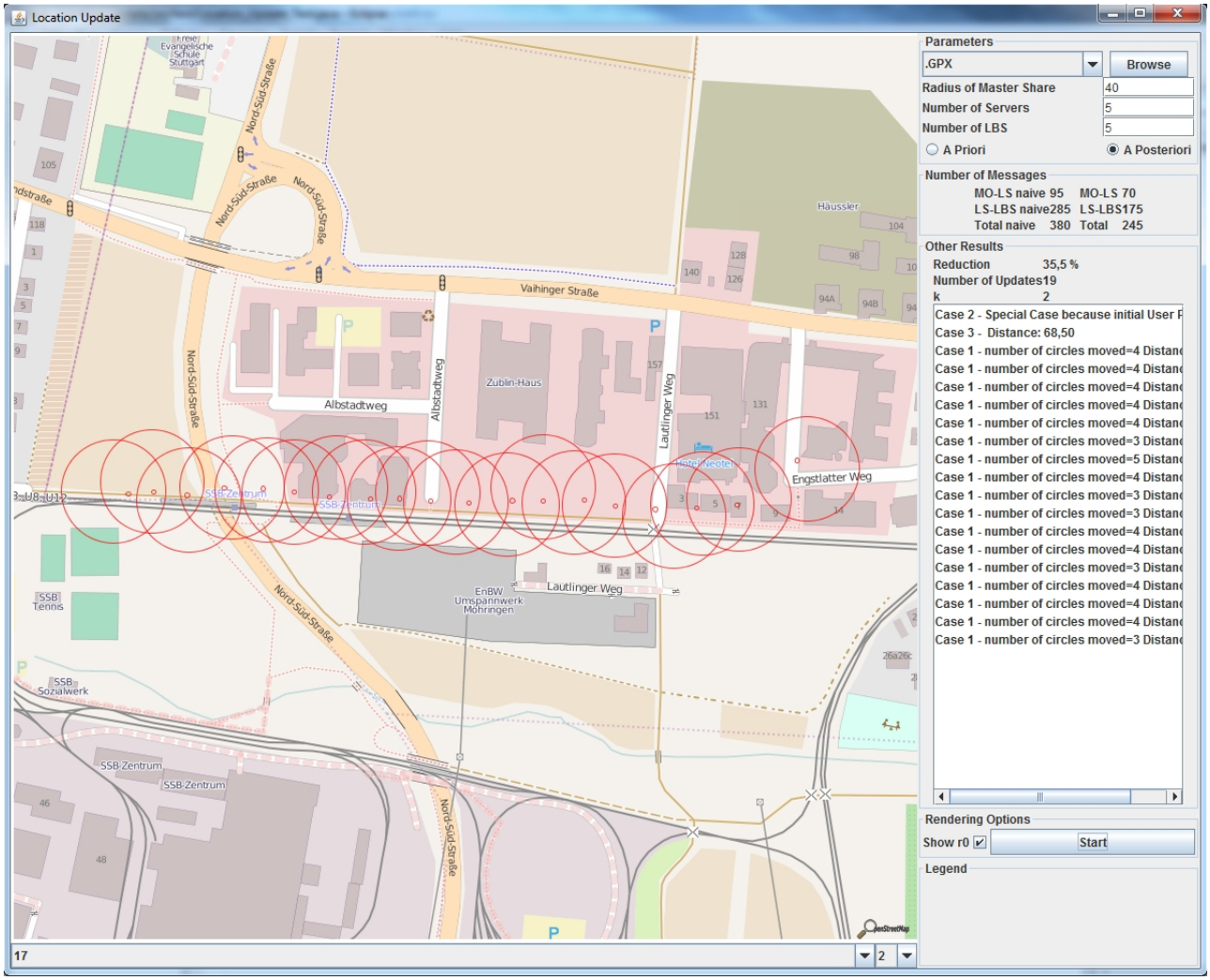


Abbildung 6.2: Benutzeroberfläche des Visualizer-Programms

Links ist der Visualisierungs-Bereich zu sehen, der die Antwort des OpenStreetmap-Webservices darstellt. Dabei werden die Benutzerpositionen als rote Punkte dargestellt, deren Größe mit der Combo-Box unten rechts verändert werden kann. Ebenfalls ist einstellbar, ob der Kreis des Master-Shares angezeigt werden soll oder nicht. Unten links ist eine Combo-Box, die die Zoom-Stufe des Webservice-Bildes regelt.
Oben rechts ist die Combo-Box zu sehen, die die Auswahl eines Datensatzes ermöglicht. Momentan werden die Datensätze von Geolife und Datensätze im gpx-Format unterstützt. Eine Erweiterung um weitere Datensätze ist allerdings kein Problem. Der Browse-Button öffnet einen Dateiauswahldialog, der die Auswahl einer einzulesenden Datei ermöglicht. Darunter sind Eingabefelder für den Radius des Master-Shares, die Anzahl an Servern und die Anzahl an LBS, sowie eine Auswahl des Share-Generation Algorithmus (A Priori oder A Posteriori).
Darunter sind die Ausgaben der Simulation zu sehen: Die Anzahl an Nachrichten im naiven

Fall (immer alle Shares updaten), sowie die Anzahl an Nachrichten mit dem verbesserten System. Darunter zu sehen ist die Nachrichtenreduktion, die Gesamtanzahl an Updates, das ausgerechnete k und eine Liste, die folgende Angaben erhält: Den verwendeten Fall, in Fall 1 die Anzahl an Kreisen, die der Benutzer seit dem letzten Update überquert hat und die Distanz zum vorhergehenden Update.
Die darunter angezeigten Rendering-Optionen ermöglichen das Anzeigen/Ausblenden der Master-Share Kreise, sowie der Start-Button, der die Simulation startet.

Ebenfalls gibt es noch ein kleines Zusatzprogramm, mit der man die beiden Share-Generation Algorithmen visualisieren kann. Die Oberfläche davon ist auf Abbildung 6.3 zu sehen. Rechts

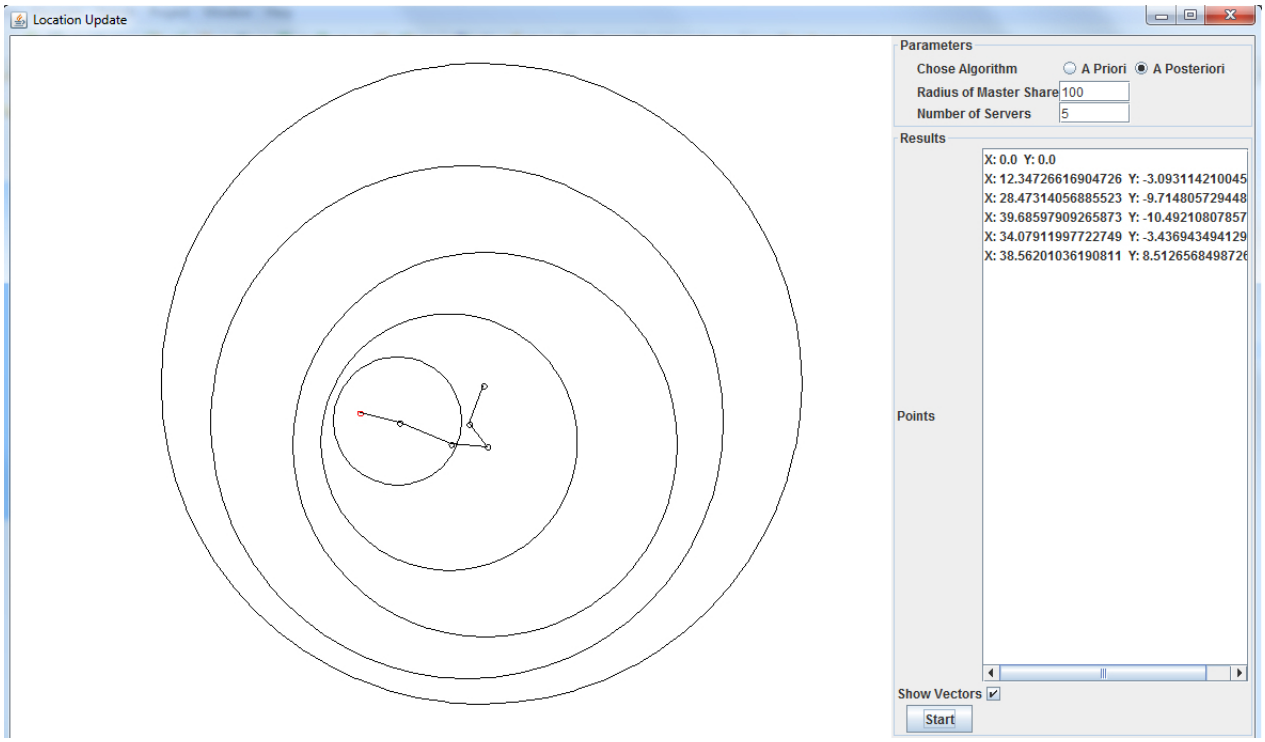


Abbildung 6.3: Benutzeroberfläche des Visualizer-Programms, Share-Generation

oben kann man den zu verwendenden Algorithmus einstellen, sowie die Anzahl an Location-Servern und der Radius des Master-Shares. Das Programm erzeugt nun entsprechend Shares und visualisiert die Ergebnisse im Panel links, sowie in der Liste rechts. Ebenfalls ist noch auswählbar, ob die einzelnen Kreismittelpunkte durch Vektoren verbunden sein sollen.
Im Anschluss wird gezeigt, wie der Systemablauf des verbesserten Systems abläuft. Dazu wird das System anhand eines Flussdiagrammes visualisiert.

6.4 Systemablauf

Das reale System soll dem Benutzer die Möglichkeiten bieten, sowohl sporadische Updates als auch kontinuierliche Updates durchzuführen. Die Update-Möglichkeiten sind also folgendermaßen gegeben:

- Durchführung eines Updates zu jeder Zeit (Sporadic Update)
- Update der Benutzerposition nach X Metern (Distance-based Update)
- Update der Benutzerposition nach X Sekunden (Time-based Update)

In allen drei Fällen gestaltet sich der Systemablauf gleich. Anhand von Schaubild 6.4 soll veranschaulicht werden, wie sich das System verhält.
Zunächst wird anhand von Funktion 5.1 berechnet, in welchem Teilkreis sich der Benutzer seit dem letzten Update befinden müsste, sodass Fall 1 besser wäre als Fall 2. Ist Fall 2 besser, muss überprüft werden, ob Fall 2 überhaupt angewendet werden kann (aufgrund möglicher Überschneidungen der Teilkreise). Kann Fall 2 angewendet werden, wird dies getan. Kann Fall 2 nicht angewendet werden (oder fiel die Entscheidung zuvor schon für Fall 1), muss überprüft werden, ob Fall 1 angewendet werden kann (Befindet sich der Benutzer mindestens innerhalb des Kreises des Master-Shares?). Kann Fall 1 angewendet werden, wird dies getan, ansonsten wird Fall 3 angewendet (Alle Shares werden neu berechnet und upgedatet).
Es ist eventuell sinnvoll, im Falle eines sporadischen Updates G_0 auf einen Wert größer gleich 1 festzulegen, sodass Fall 2 öfters angewendet wird als Fall 1. Dass der Benutzer von selbst so oft ein Update macht, und sich dabei immer noch innerhalb des Teilkreises des Master-Shares befindet, ist eher unwahrscheinlich. Andererseits ist es eventuell sinnvoll, im Time-based oder im Distance-based Szenario G_0 auf einen Wert kleiner 1 festzulegen, sodass Fall 1 öfters angewendet wird als Fall 2. Natürlich kommt es in diesem Szenario darauf an, wie schnell sich der Benutzer bewegt, und wie die Update-Intervalle festgelegt sind, aber es ist wahrscheinlicher, dass sich der Benutzer zwischen zwei Updates noch innerhalb des Teilkreises des Master-Shares befindet und eher unwahrscheinlich, dass sich der Benutzer so weit bewegt hat, dass Fall 2 angewendet werden kann.
Im folgenden Unterkapitel wird der Algorithmus vorgestellt, der das verbesserte System simuliert.

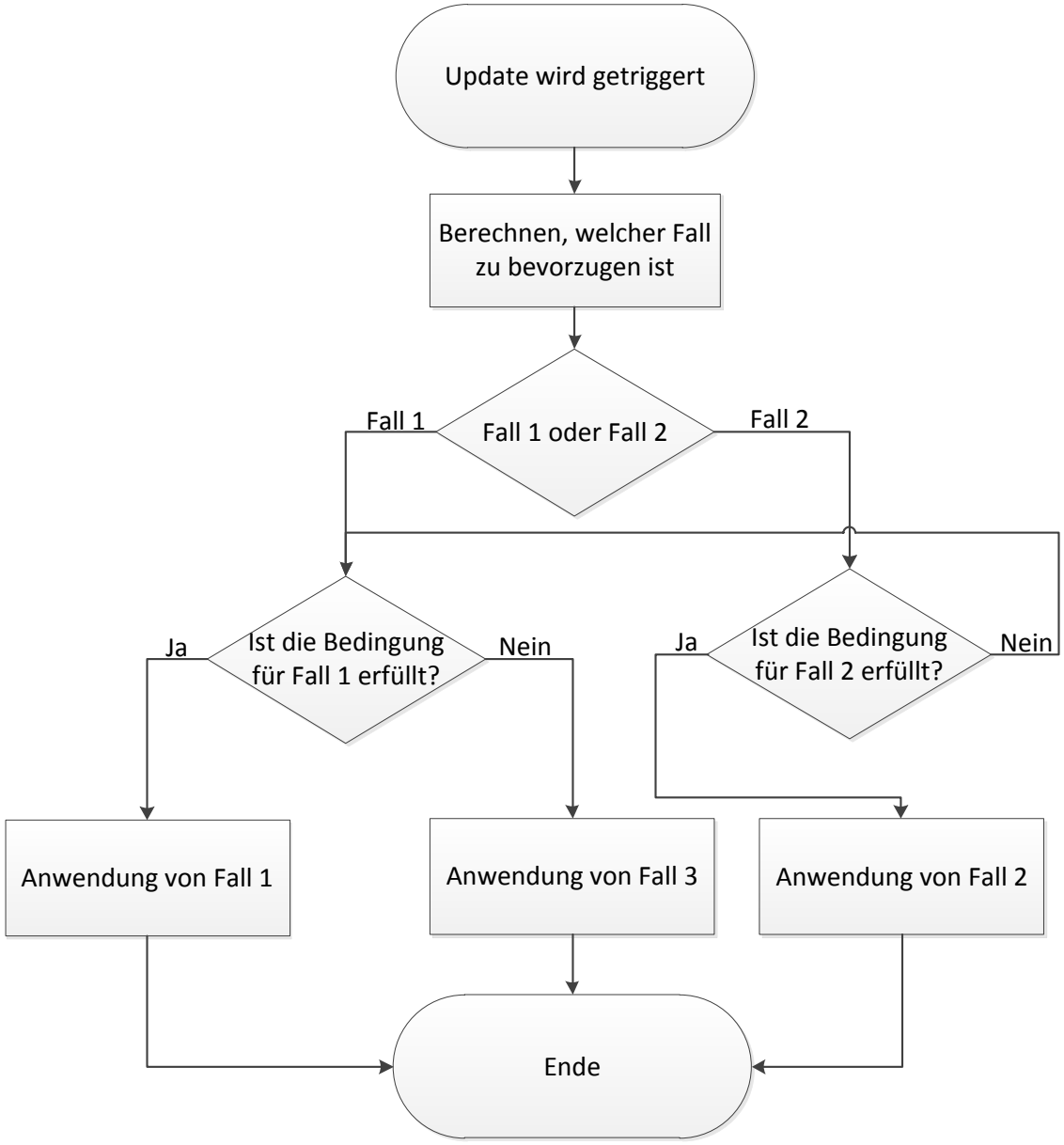


Abbildung 6.4: Systemablauf im Falle eines durch den Benutzer ausgelösten Updates

6.5 Simulations-Algorithmus

Algorithmus 6.1 simuliert das verbesserte System.
Der Algorithmus erhält als Parameter die Anzahl an Location-Servern n, die Anzahl an LBS $nLBS$, den Radius des Master-Shares r_0, den zu verwendenden Share-Generations Algorithmus und die Trajektorien-Daten.

Algorithmus 6.1 Simulations-Algorithmus

```
1: function SIMULATE_MINIMIZATION(n, nLBS, r_0, algorithm, TrajectoryData)
2:     result ← newResult()
3:     Δ_Φ ← r_0 / n
4:     k ← minimizeMessages(n, 1, nLBS)
5:     maxDistance ← k * Δ_Φ
6:     CenterOfMasterShare[] ← newArray()
7:     UseCase3ForFirstUserPosition()
8:     for (i = 0 → TrajectoryData.size()) do
9:         if (algorithm == aposteriori) then
10:            CenterOfMasterShare[i] ← GenerateNSharesAPosteriori(TrajectoryData, r_0, n)[n].center
11:        else
12:            CenterOfMasterShare[i] ← GenerateNSharesAPriori(TrajectoryData, r_0, n)[n].center
13:        end if
14:    end for
15:    for (i = 1 → TrajectoryData.size()) do
16:        data ← TrajectoryData[i]
17:        data2 ← TrajectoryData[i − 1]
18:        travelledDistance = calculateDistance(data, data2)
19:        if (travelledDistance <= maxDistance) then
20:            numberOfCircles ← Math.floor(travelledDistance / deltaPhi) + 1
21:            UseCase1()
22:        else
23:            if (travelledDistance > 4 * r_0 || calculateDistance(CenterOfMasterShare[i], CenterOfMasterShare[i −
    1]) > 2 * r_0) then
24:                UseCase2()
25:            else
26:                if (travelledDistance < n * Δ_Φ) then
27:                    UseCase1()
28:                else
29:                    UseCase3()
30:                end if
31:            end if
32:        end if
33:    end for
34:    return result
35: end function
```

In (2-6) werden dann das Result-Objekt, das unter anderem die Nachrichtenanzahl speichert, Δ_Φ, das die Verringerung des Radius pro Teilkreis angibt, k, das angibt, wie viele Kreise der Benutzer sich bewegt haben darf, damit Fall 1 besser ist als Fall 2, *maxDistance*, das k in eine Strecke umrechnet, sowie *CenterOfMasterShare*, welcher ein Array ist, der die Mittelpunkte der erzeugten Master-Share Kreise speichert, initialisiert.

In (7) wird dann für die erste Benutzerposition Fall 3 angewendet, da zunächst jeder Server alle Shares kennen muss. Intern werden im Result-Objekt die entsprechenden Nachrichten-Anzahlen gesetzt.
In (8 - 14) werden entsprechend des ausgewählten Algorithmus für jede Benutzerposition n Shares erzeugt und in *CenterOfMasterShare* gespeichert.
Anschließend werden für jede Trajektorie im Datensatz (mit Ausnahme der ersten) immer die zwei aufeinanderfolgenden Trajektorien, beginnend ab Trajektorie 1 (was in diesem Fall die zweite Trajektorie ist, da der Array bei 0 beginnt) miteinander verglichen (16+17). In (18) wird die Distanz ausgerechnet, die der Benutzer zwischen den beiden Updates zurückgelegt hat.
In (19) wird nun überprüft, ob Fall 1 besser ist als Fall 2 (*maxDistance* ist k, umgerechnet in eine Strecke). Wenn dem so ist, wird in (20) und (21) Fall 1 angewendet, und wieder intern im Result-Objekt die Nachrichtenanzahl gesetzt. Falls dies nicht der Fall ist, muss in (23) überprüft werden, ob Fall 2 überhaupt angewendet werden kann. Hierzu werden die gespeicherten Mittelpunkte der Master-Share Kreise genutzt. Falls Fall 2 angewendet werden kann, wird dieser in (24) angewandt, ansonsten wird überprüft, ob Fall 1 angewendet werden kann (26). Wenn ja, wird dieser in (27) angewandt, ansonsten muss Fall 3 angewendet werden (29). Am Ende wird das Result-Objekt zurückgegeben (34).
In den nachfolgenden Unterkapiteln werden die Datensätze vorgestellt, mit denen die Simulation durchgeführt wurde. Die Ergebnisse der Simulation werden ebenfalls visualisiert und diskutiert.

6.6 Simulations-Daten

Die Simulation des Systems wird mit den folgenden Daten durchgeführt:

- 2 Datensätze sporadische Updates
- 2 Datensätze kontinuierliche Updates
- 2 Datensätze mixed Updates

Die Datensätze stammen allesamt aus dem GeoLife Datensatz. Die mixed Updates und ein sporadic Update wurden von mir noch ein wenig modifiziert (Updates wurden manuell gelöscht). Durch die verschiedenen Arten der Daten soll eine optimale Abdeckung bezüglich möglicher realer Fälle erreicht werden. Zunächst soll der Nachrichtenverlauf evaluiert werden: Hierzu wird für jeden Datensatz ausgewertet, wie viel Nachrichten pro Update geschickt werden, und welcher Fall angewandt wird. Das Ganze geschieht jeweils mit 3 verschiedenen Werten für r_0: 5 Meter, 50 Meter und 100 Meter. Für n und *nLBS* wurde jeweils als Wert 5 festgelegt (Was einigermaßen realistisch erscheint).
Im Anschluss wird die Reduktion von Update-Nachrichten evaluiert. Hierzu werden jeweils verschiedene Werte für den r_0-Parameter benutzt (1 Meter - 1000 Meter). Die Shares werden dabei immer mit dem a Posteriori Algorithmus erzeugt, da die Auswahl des Algorithmus nicht allzu viel am Ergebnis ändert und der a Priori Algorithmus zu viel Rechenzeit in Anspruch nimmt.

6.7 Durchführung der Simulation

6.7.1 Sporadische Updates

Die Visualisierung der Datensätze sind in den Abbildungen 6.5 und 6.6 zu sehen.

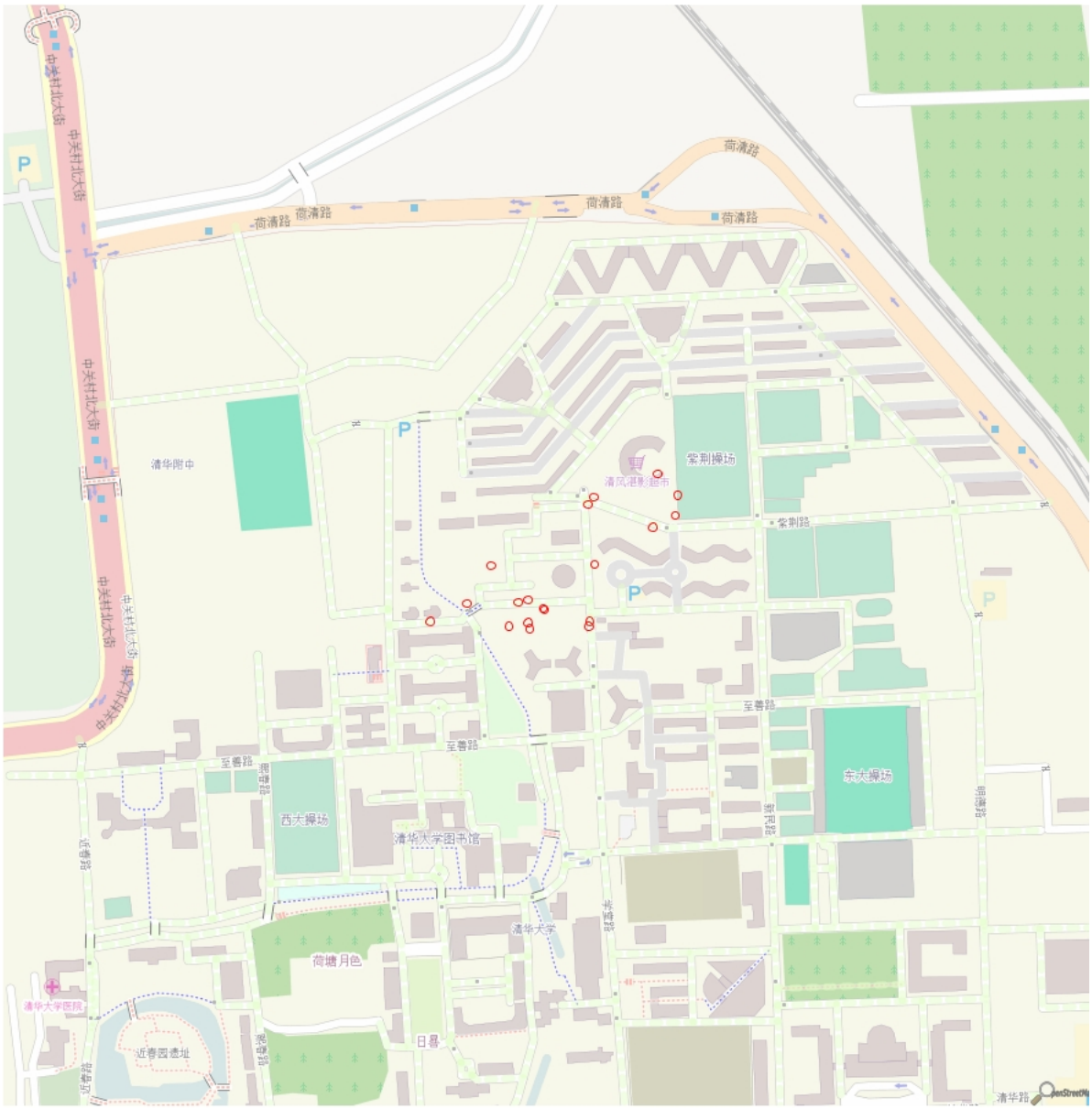


Abbildung 6.5: Visualisierung von Datensatz 1 der sporadischen Updates des GeoLife-Datensatzes, Updates sind durch rote Kreise visualisiert

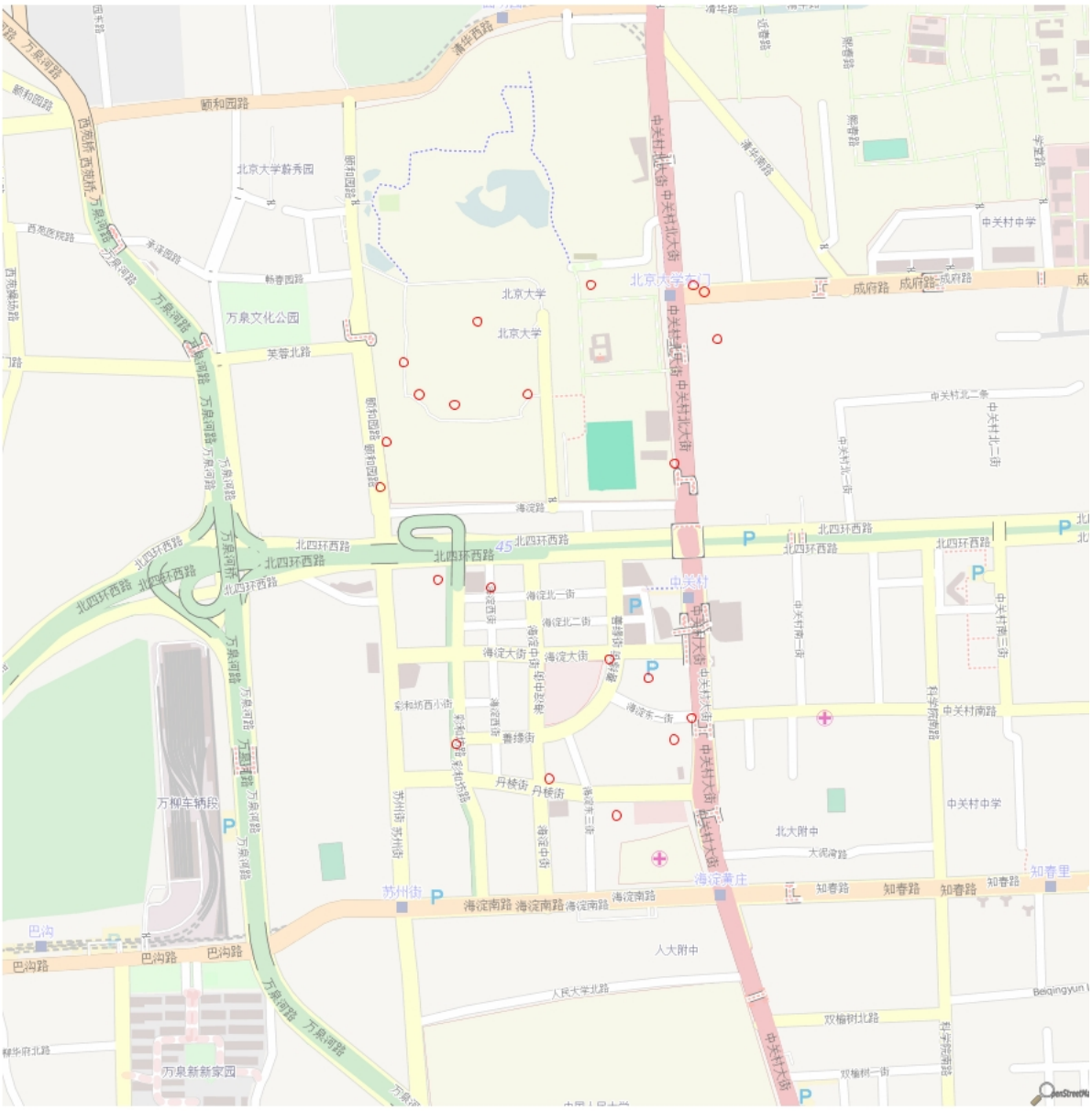
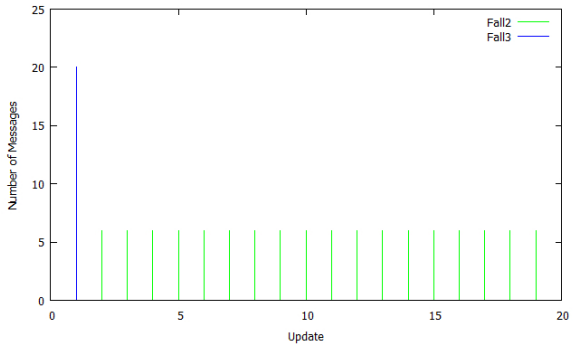


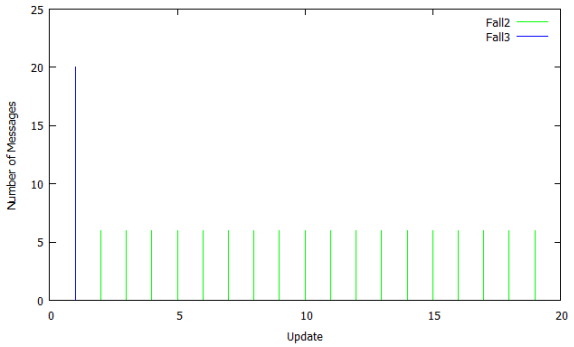
Abbildung 6.6: Visualisierung von Datensatz 2 der sporadischen Updates des GeoLife-Datensatzes, Updates sind durch rote Kreise visualisiert

Die Abbildung 6.7 zeigt jeweils den Update-Verlauf der einzelnen Datensätze. Auf der X-Achse sind die einzelnen Updates zu sehen, die Y-Achse gibt an, wieviele Nachrichten jeweils geschickt werden, und die farbige Markierung zeigt, welcher Fall angewendet wurde.

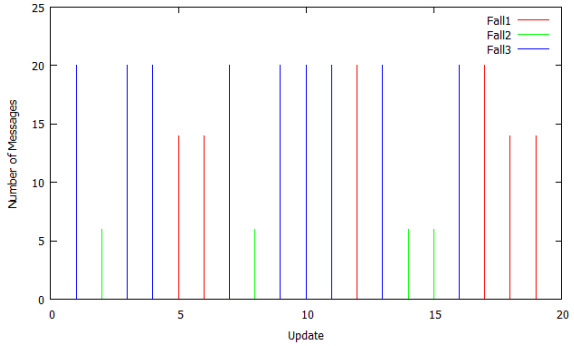
An den Ergebnissen sieht man deutlich, dass die Updates in Datensatz 2 räumlich weiter auseinander liegen. Verhalten sich die Datensätze bei einem Radius von 5 Metern zwar noch



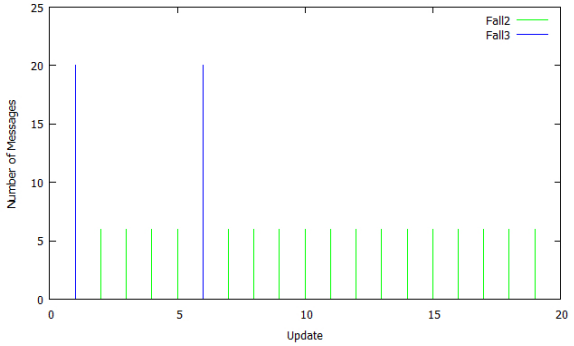
(a) Sporadische Updates Datensatz 1, r_0 = 5 m, Gesamtanzahl Naiv: 380, Gesamtanzahl Optimiert: 128, Nachrichteneinsparung: 66,3%



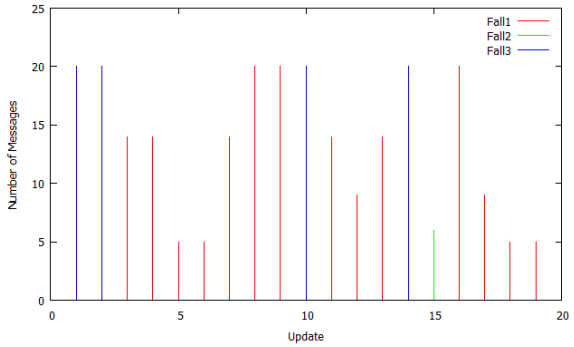
(b) Sporadische Updates Datensatz 2, r_0 = 5 m, Gesamtanzahl Naiv: 420, Gesamtanzahl Optimiert: 140, Nachrichteneinsparung: 66,7%



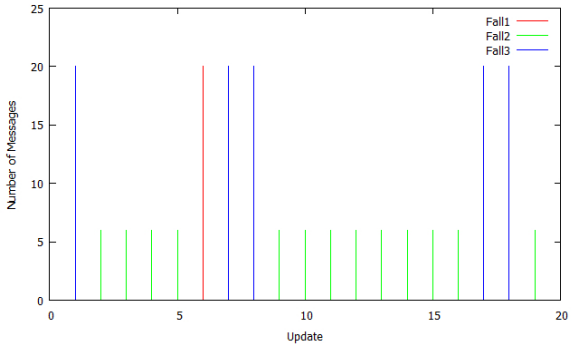
(c) Sporadische Updates Datensatz 1, r_0 = 50 m, Gesamtanzahl Naiv: 380, Gesamtanzahl Optimiert: 272, Nachrichteneinsparung: 28,4%



(d) Sporadische Updates Datensatz 2, r_0 = 50 m, Gesamtanzahl Naiv: 420, Gesamtanzahl Optimiert: 154, Nachrichteneinsparung: 63,3%



(e) Sporadische Updates Datensatz 1, r_0 = 100 m, Gesamtanzahl Naiv: 380, Gesamtanzahl Optimiert: 254, Nachrichteneinsparung: 33,2%



(f) Sporadische Updates Datensatz 2, r_0 = 100 m, Gesamtanzahl Naiv: 420, Gesamtanzahl Optimiert: 224, Nachrichteneinsparung: 46,7%

Abbildung 6.7: Nachrichtenverlauf von sporadischen Updates

gleich, können ab einem Radius von 50 Metern bei Datensatz 2 deutlich mehr Nachrichten durch die Anwendung von Fall 2 eingespart werden. Dadurch lässt sich zwar ab 100 m in Datensatz öfters Fall 1 anwenden, allerdings spart man in Datensatz 2 dennoch mehr Nachrichten durch die Anwendung von Fall 2. Es lässt sich bereits erahnen, dass für sporadische Updates kleinere Radien besser geeignet sind, da Fall 2 dann öfters angewendet werden kann. Bestätigt wird dies durch Abbildung 6.8.

Abbildung 6.8 zeigt die prozentuale Nachrichteneinsparung in Abhängigkeit von r_0.

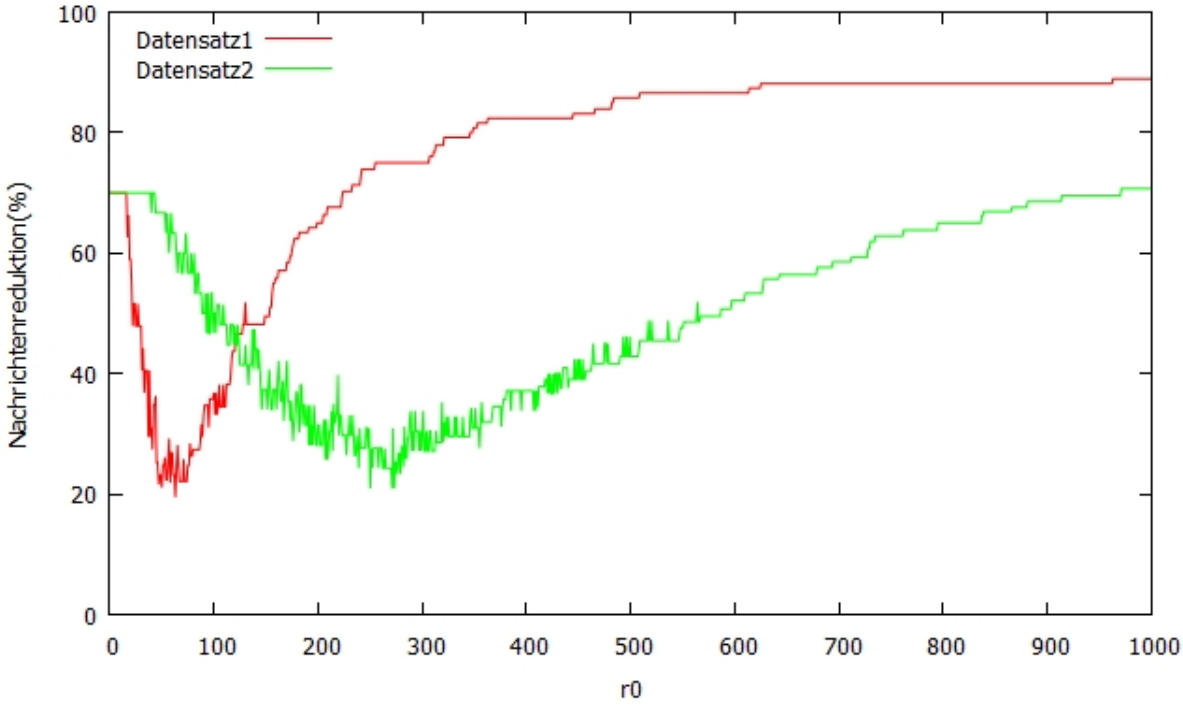


Abbildung 6.8: Abhängigkeit der Nachrichteneinsparung von r_0

Im Fall von sporadischen Updates, wo der zeitliche (und damit meist auch der räumliche) Abstand zwischen zwei Nachrichten generell höher ist als im kontinuierlichen Fall, ist zu erkennen, dass durch sehr kleine Radien von r_0 bereits viele Update-Nachrichten eingespart werden können (ca. 70%). Dies lässt sich so erklären, dass durch den kleinen Radius Fall 2 so gut wie jedes Mal angewendet werden kann.
Für einen mittelgroßen Radius des Master-Shares wird das schlechteste Ergebnis erzielt, da sehr oft weder Fall 1 noch Fall 2 angewendet werden kann und somit alle Shares gesendet werden müssen. Dies liegt daran, dass ein mittelgroßer Radius zu groß ist, um Fall 2 durchgehend anzuwenden, aber wiederum zu klein ist, um Fall 1 durchgehend anzuwenden. Datensatz 1, bei dem die Updates näher zusammen liegen als bei Datensatz 2, erreicht das lokale Minimum von ca. 20% schon ab einem Radius von ca. 60 Metern, wohingegen Datensatz 2 dieses erst bei ca. 250 Metern erreicht. Wird der Radius dann erhöht, steigt die Nachrichtenreduktion wieder an, bei Datensatz 1 schneller als Datensatz 2. Dies ist ebenfalls

durch den unterschiedlichen räumlichen Abstand der Updates voneinander zu erklären: dadurch, dass die Updates bei Datensatz 1 näher zusammen liegen, reicht ein kleinerer Radius bereits aus, um früher Fall 1 anwenden zu können, und dadurch mehr Nachrichten einzusparen.

6.7.2 Kontinuierliche Updates

Die Visualisierung der Datensätze sind in den Abbildungen 6.9 und 6.10 zu sehen.

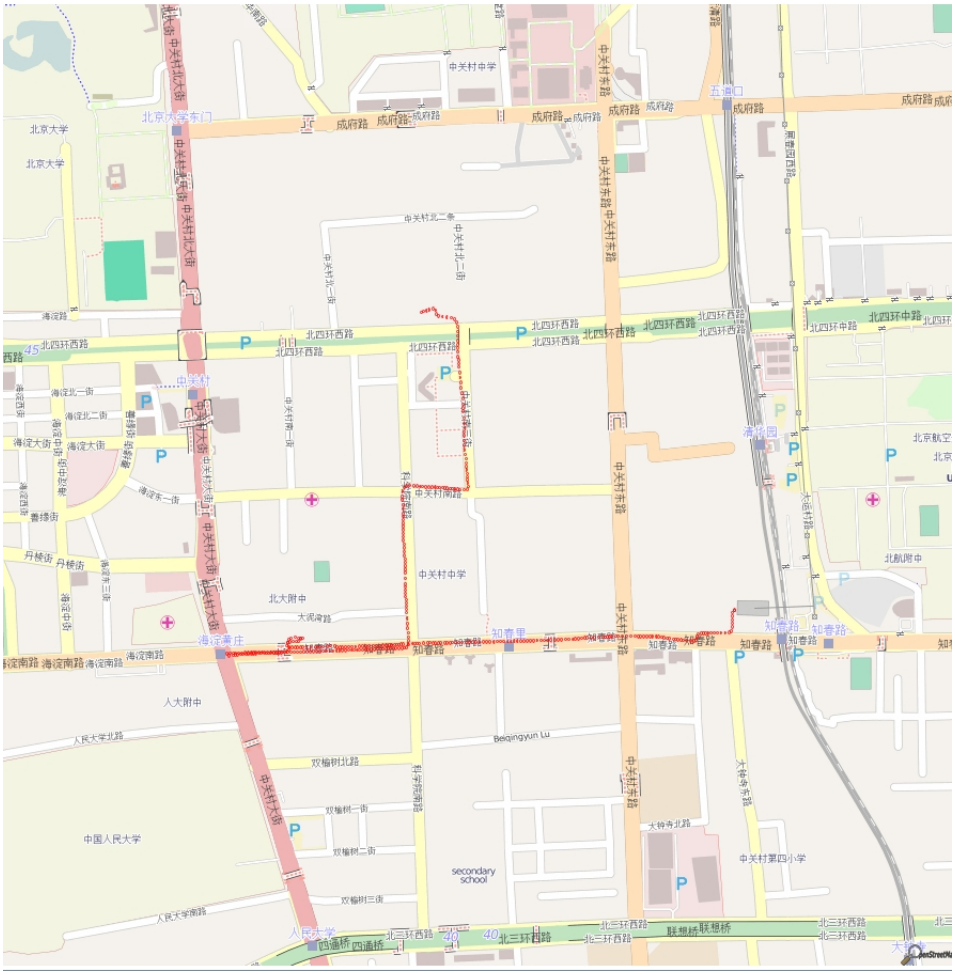


Abbildung 6.9: Visualisierung von Datensatz 1 der kontinuierlichen Updates des GeoLife-Datensatzes, Updates sind durch rote Kreise visualisiert

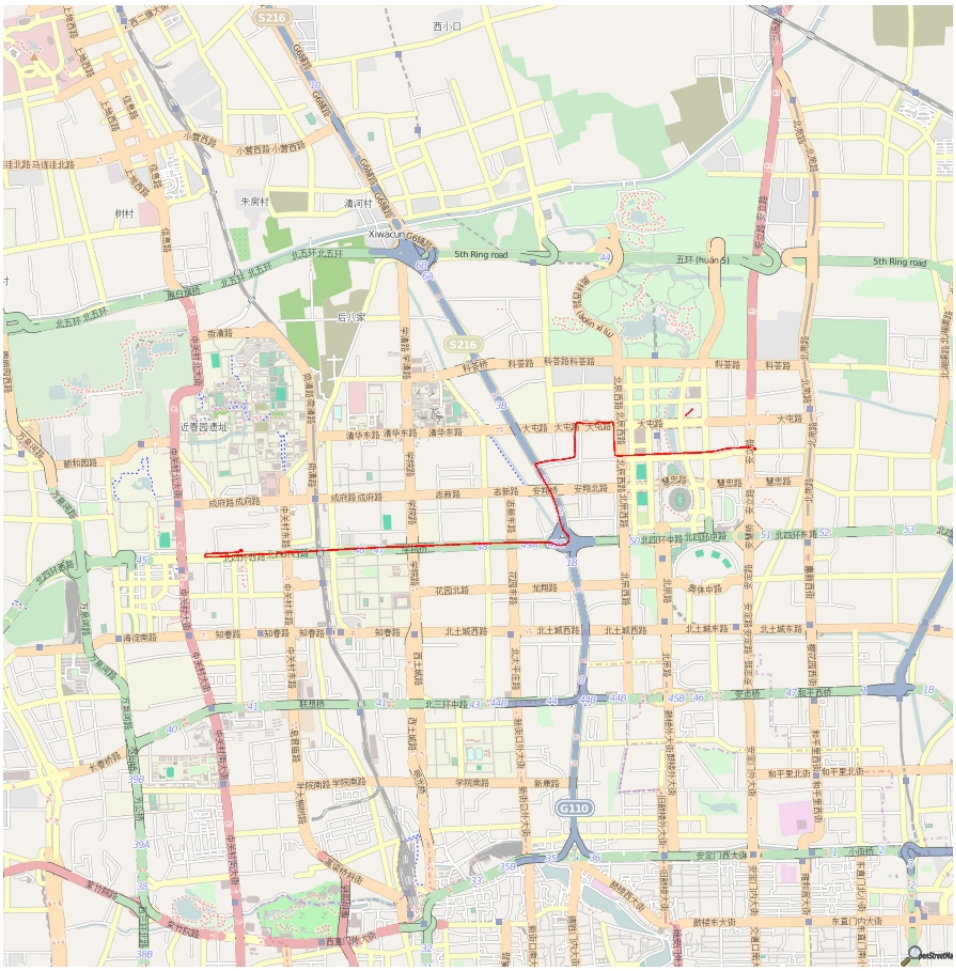
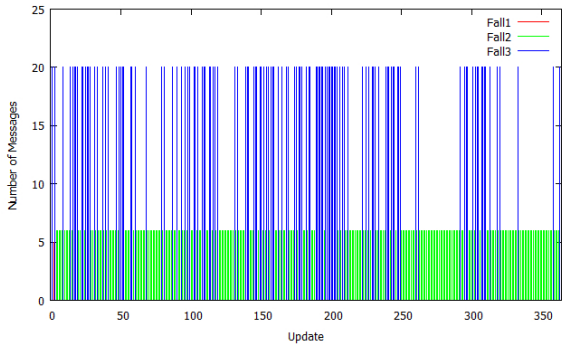


Abbildung 6.10: Visualisierung von Datensatz 2 der kontinuierlichen Updates des GeoLife-Datensatzes, Updates sind durch rote Kreise visualisiert

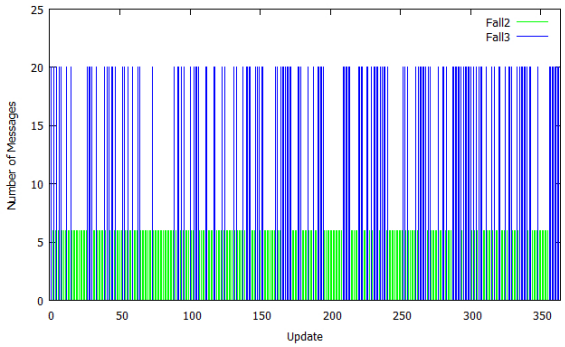
Die Abbildung 6.11 zeigt jeweils den Update-Verlauf der einzelnen Datensätze. Auf der X-Achse sind die einzelnen Updates zu sehen, die Y-Achse gibt an, wieviele Nachrichten jeweils geschickt werden, und die farbige Markierung zeigt, welcher Fall angewendet wurde.

Die kontinuierlichen Updates verhalten sich im Vergleich zu den sporadischen Updates gegensätzlich. Für kleine r_0, wie z. B. 5m können nur wenig Nachrichten eingespart werden, da Fall 1 nicht angewendet werden kann, und Fall 2 nur selten (da die Updates zu nahe beieinander liegen). Ab 50 Metern wird bereits eine sehr hohe Einsparung von ca. 80% erreicht. Ab einem Radius von 100 Metern nähert sich die Einsparung in beiden Datensätzen dem Maximalwert (für die ausgewählten Parameter) von ca. 89% an. Da die Updates in den beiden Datensätzen im selben Abstand durchgeführt wurden, verhalten sich beide Datensätze sehr ähnlich, was ebenfalls durch Abbildung 6.12 bestätigt wird.

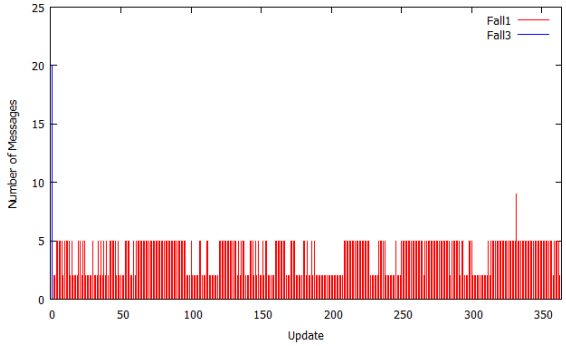
Abbildung 6.12 zeigt die prozentuale Nachrichteneinsparung in Abhängigkeit von r_0.



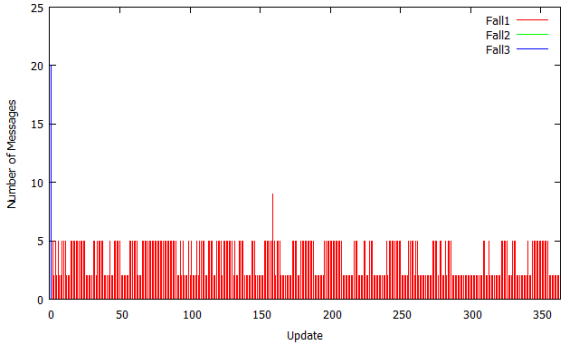
(a) Kontinuierliche Updates Datensatz 1, r_0 = 5 m, Gesamtanzahl Naiv: 7280, Gesamtanzahl Optimiert: 4143, Nachrichteneinsparung: 43,1%



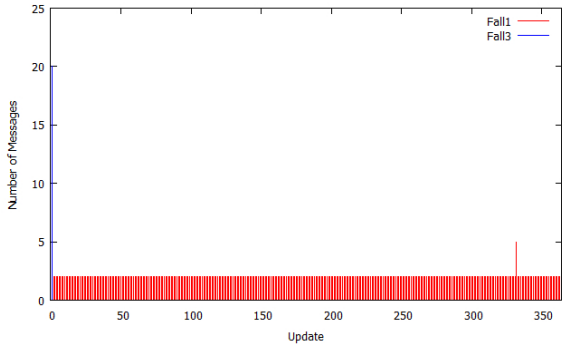
(b) Kontinuierliche Updates Datensatz 2, r_0 = 5 m, Gesamtanzahl Naiv: 20060, Gesamtanzahl Optimiert: 11520, Nachrichteneinsparung: 42,6%



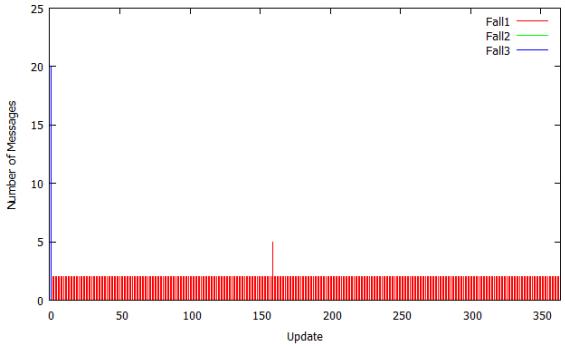
(c) Kontinuierliche Updates Datensatz 1, r_0 = 50 m, Gesamtanzahl Naiv: 7280, Gesamtanzahl Optimiert: 1446, Nachrichteneinsparung: 80,1%



(d) Kontinuierliche Updates Datensatz 2, r_0 = 50 m, Gesamtanzahl Naiv: 20060, Gesamtanzahl Optimiert: 3856, Nachrichteneinsparung: 80,8%



(e) Kontinuierliche Updates Datensatz 1, r_0 = 100 m, Gesamtanzahl Naiv: 7280, Gesamtanzahl Optimiert: 749, Nachrichteneinsparung: 89,7%



(f) Kontinuierliche Updates Datensatz 2, r_0 = 100 m, Gesamtanzahl Naiv: 20060, Gesamtanzahl Optimiert: 2041, Nachrichteneinsparung: 89,8%

Abbildung 6.11: Nachrichtenverlauf von kontinuierlichen Updates

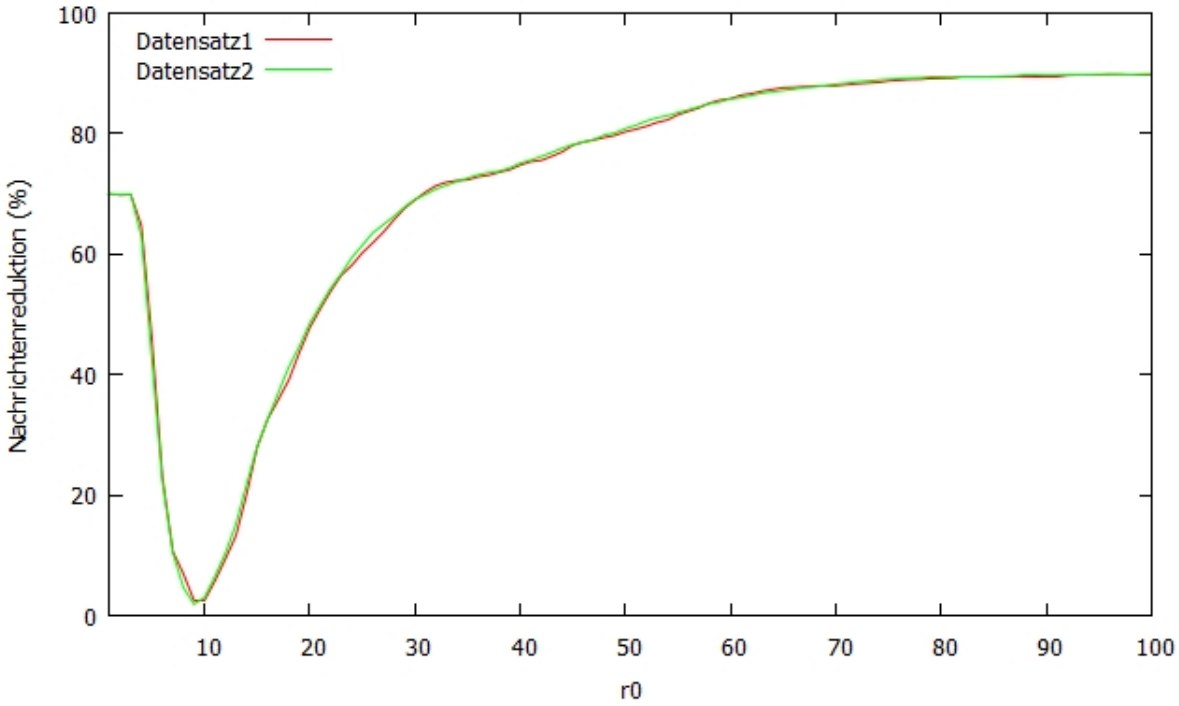


Abbildung 6.12: Abhängigkeit der Nachrichteneinsparung von r_0

Ähnlich wie im sporadischen Fall lassen sich mit einem kleinen Master-Share Radius schon einige Nachrichten einsparen, allerdings muss hier r_0 viel kleiner sein als im sporadischen Fall, da bereits ab einem Radius von ca. 10 Metern das globale Minimum erreicht ist. Dies ist logisch, da bei kontinuierlichen Updates der Abstand zwischen den Nachrichten in der Regel viel kleiner ist, und somit Fall 2 nur bei sehr kleinen r_0 angewandt werden kann.
Im Gegensatz zum sporadischen Fall steigt die Nachrichteneinsparung viel schneller an, und es reicht bereits ein r_0 von ca. 70 Metern um die maximale Einsparung von ca. 90% zu erreichen. Dies ist ebenfalls dadurch erklärbar, dass die Updates viel näher beieinander liegen. Auch fällt auf, dass die Ergebnisse der Datensätze viel näher beieinanderliegen als im sporadischen Fall, was dadurch erklärbar ist, dass der Abstand zwischen den Updates konstant ist und nicht wie im sporadischen Fall willkürlich vom Benutzer gewählt wird.

6.7.3 Mixed Updates

Die Visualisierung der Datensätze sind in den Abbildungen 6.13 und 6.14 zu sehen.

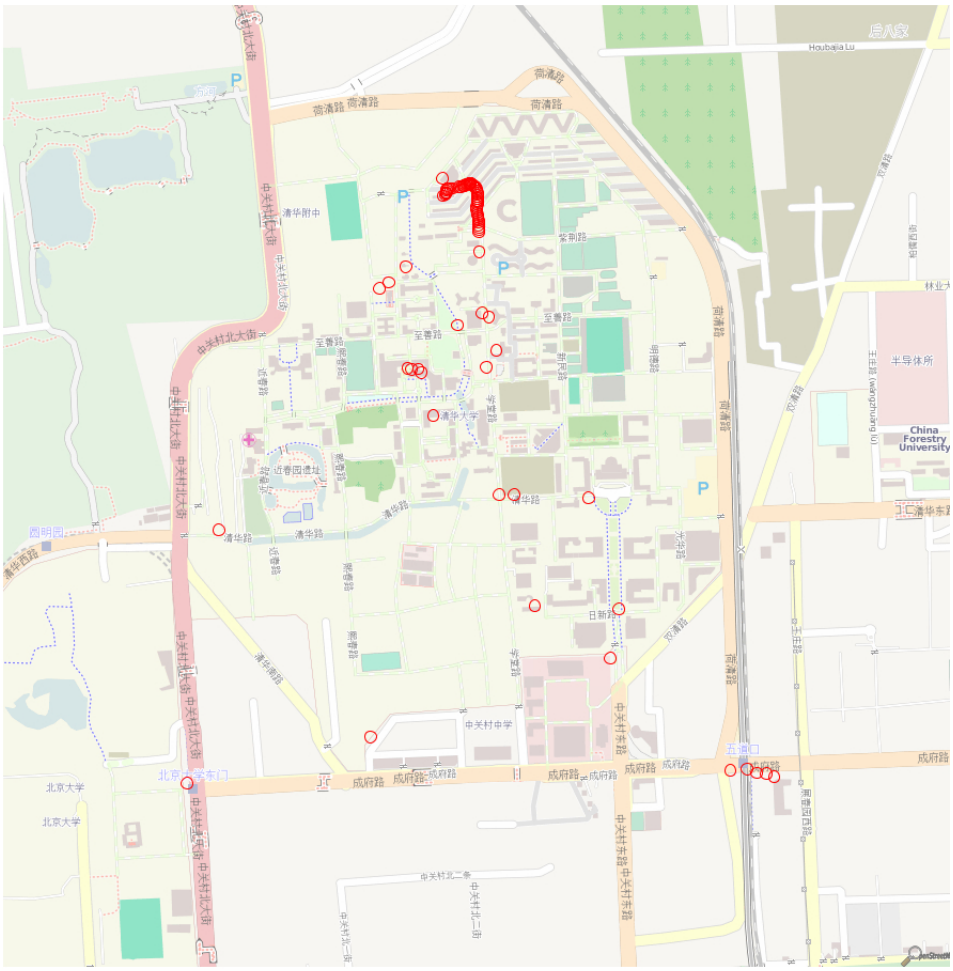


Abbildung 6.13: Visualisierung von Datensatz 1 der mixed Updates des GeoLife-Datensatzes, Updates sind durch rote Kreise visualisiert

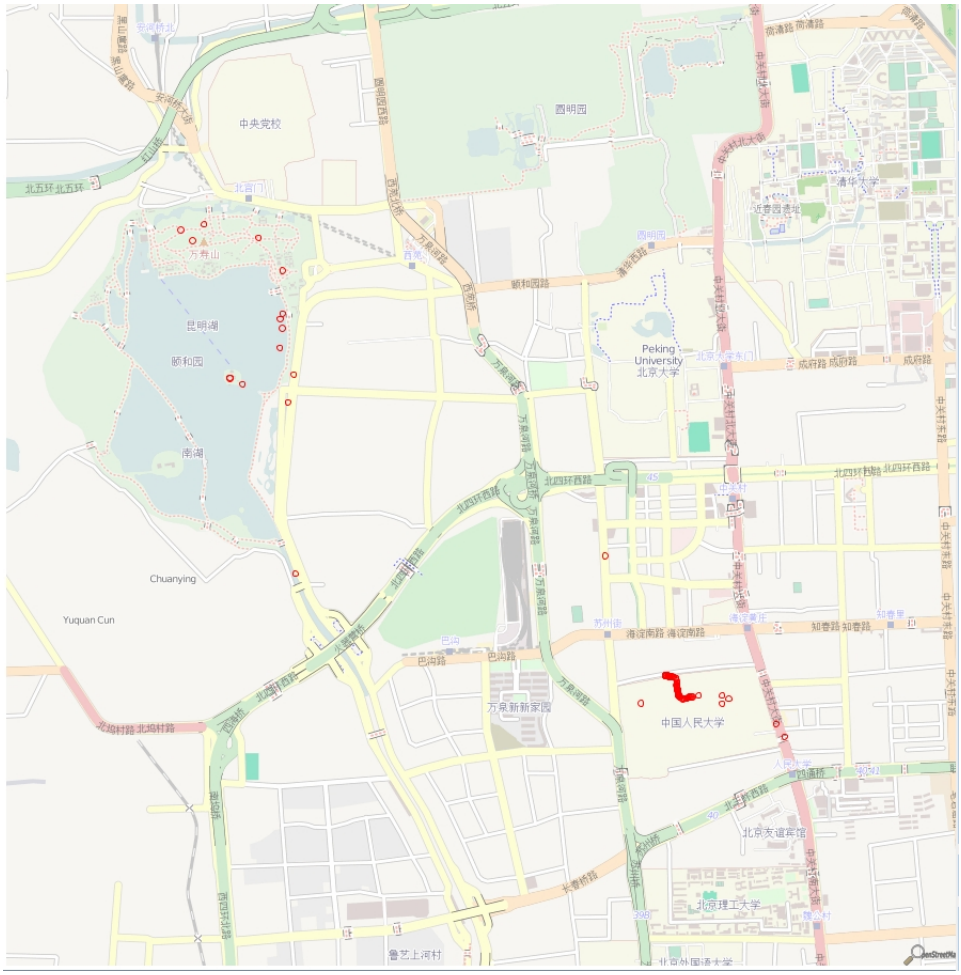
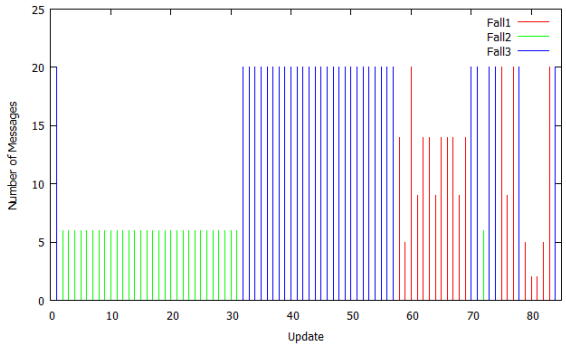


Abbildung 6.14: Visualisierung von Datensatz 2 der mixed Updates des GeoLife-Datensatzes, Updates sind durch rote Kreise visualisiert

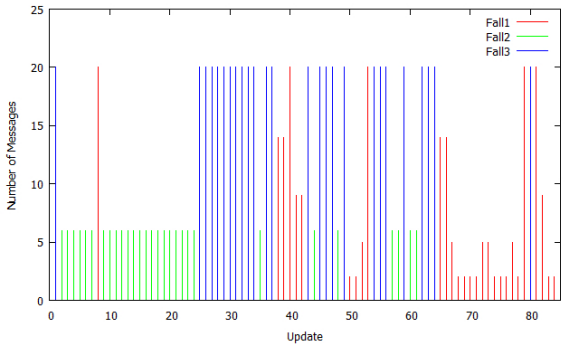
Die Abbildung 6.15 zeigt jeweils den Update-Verlauf der einzelnen Datensätze. Auf der X-Achse sind die einzelnen Updates zu sehen, die Y-Achse gibt an, wieviele Nachrichten jeweils geschickt werden, und die farbige Markierung zeigt, welcher Fall angewendet wurde.

Im Falle von mixed Updates lassen sich keine klaren Aussagen treffen. Je nachdem, ob die Trajektorie mehr kontinuierliche oder mehr sporadische Updates enthält, eignet sich ein größerer oder ein kleinerer Radius. Im Fall der beiden ausgewählten Datensätze eignen sich größere Radien, da dann oft Fall 1 angewendet werden kann.

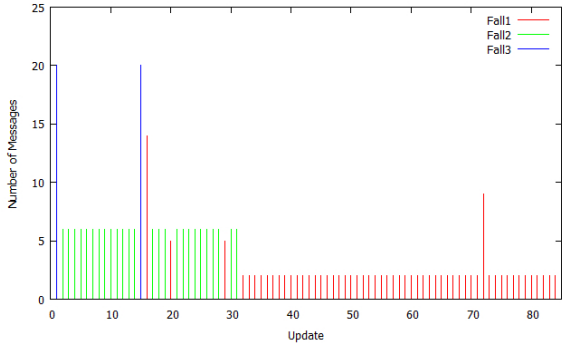
Abbildung 6.16 zeigt die prozentuale Nachrichteneinsparung in Abhängigkeit von r_0.



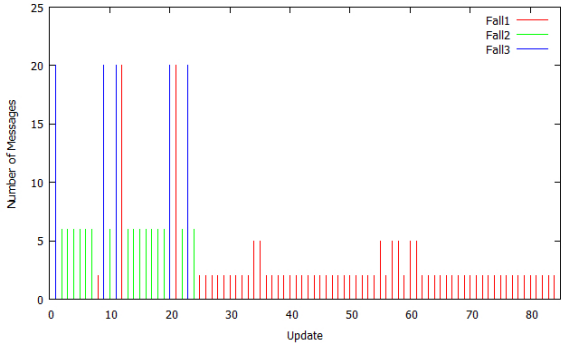
(a) Mixed Updates Datensatz 1, r_0 = 5 m, Gesamtanzahl Naiv: 1680, Gesamtanzahl Optimiert: 1051, Nachrichteneinsparung: 37,4%



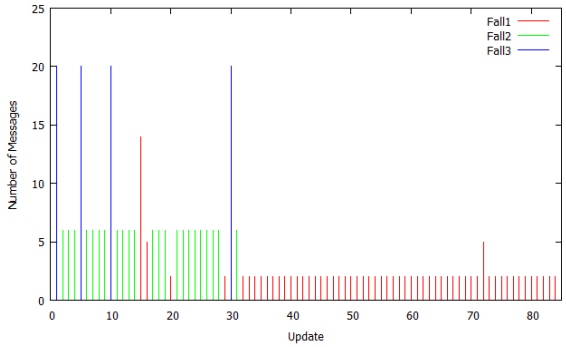
(b) Mixed Updates Datensatz 2, r_0 = 5 m, Gesamtanzahl Naiv: 1800, Gesamtanzahl Optimiert: 941, Nachrichteneinsparung: 47,7%



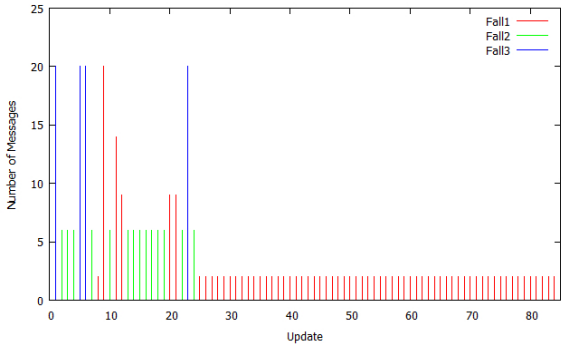
(c) Mixed Updates Datensatz 1, r_0 = 50 m, Gesamtanzahl Naiv: 1680, Gesamtanzahl Optimiert: 333, Nachrichteneinsparung: 80,4%



(d) Mixed Updates Datensatz 2, r_0 = 50 m, Gesamtanzahl Naiv: 1800, Gesamtanzahl Optimiert: 377, Nachrichteneinsparung: 79,1%



(e) Mixed Updates Datensatz 1, r_0 = 100 m, Gesamtanzahl Naiv: 1680, Gesamtanzahl Optimiert: 350, Nachrichteneinsparung: 79,2%



(f) Mixed Updates Datensatz 2, r_0 = 100 m, Gesamtanzahl Naiv: 1800, Gesamtanzahl Optimiert: 387, Nachrichteneinsparung: 78,5%

Abbildung 6.15: Nachrichtenverlauf von mixed Updates

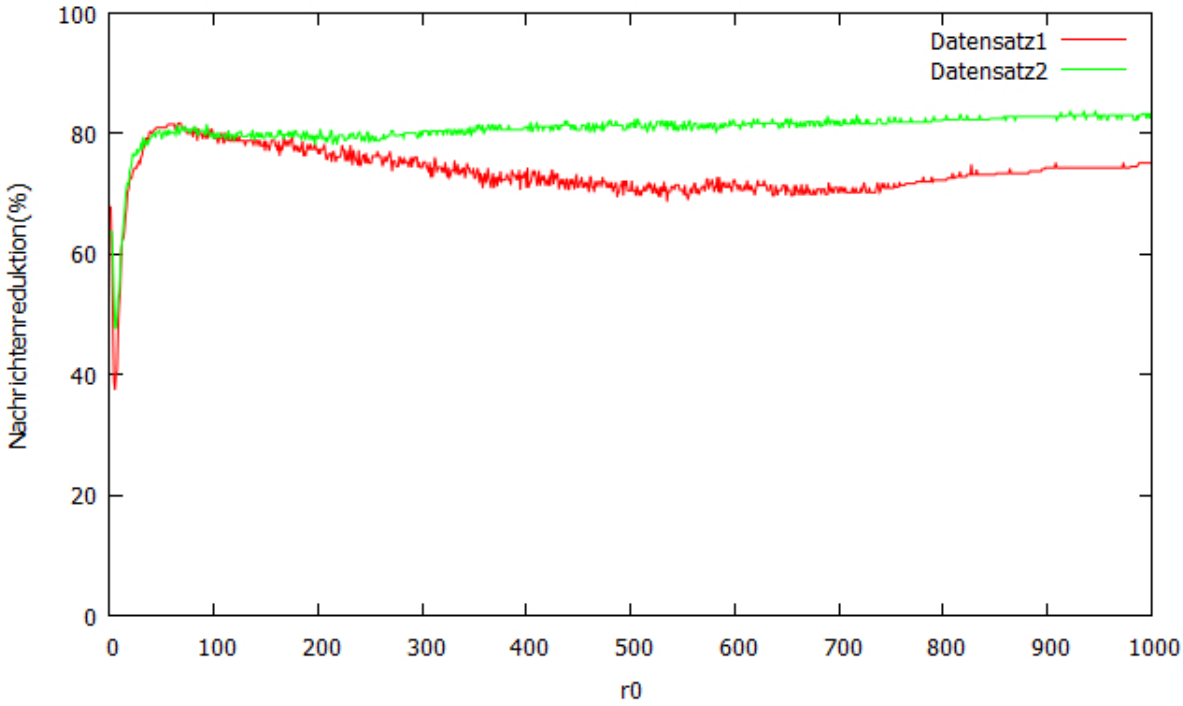


Abbildung 6.16: Abhängigkeit der Nachrichteneinsparung von r_0

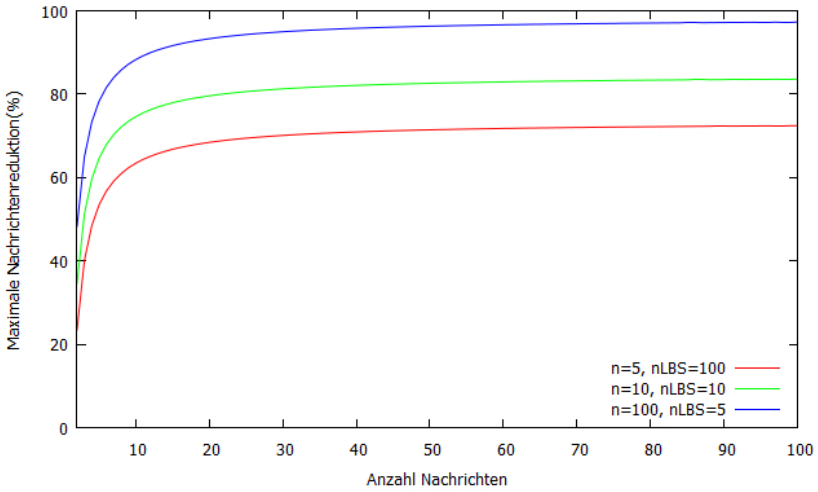
Ähnlich wie bei den kontinuierlichen und den sporadischen Updates weisen beide Datensätze den typischen Kurvenverlauf auf. Die lokalen Minima sind nah beieinander, allerdings werden für größere Werte von r_0 bei Datensatz 1 nicht so hohe Einsparungen erreicht wie bei Datensatz 2, was daran liegt, dass die durchschnittliche Distanz zwischen den Updates bei Datensatz 1 größer ist.
Im nachfolgenden Unterkapitel wird die maximal mögliche Nachrichteneinsparung der beiden Fälle untersucht, die Ergebnisse werden visualisiert und diskutiert.

6.8 Maximale Nachrichteneinsparung

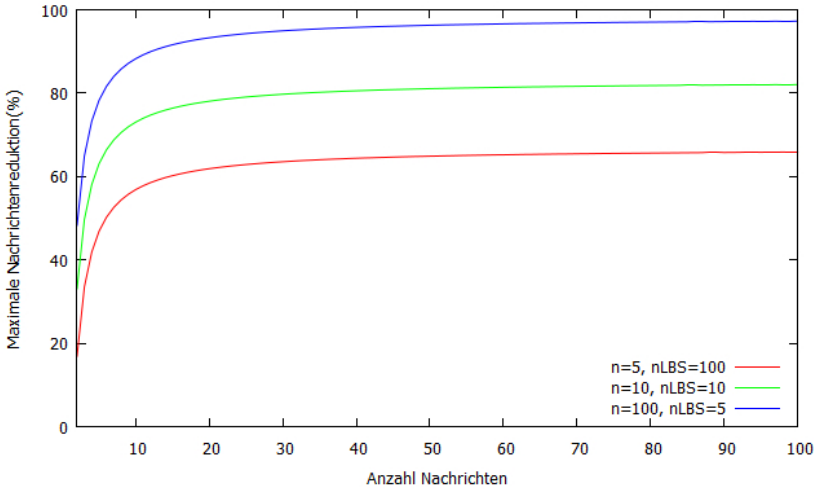
Interessant wäre es noch, für bestimmte Konstellationen von n und $nLBS$ die maximale Nachrichteneinsparung auszurechnen, die jeweils durch Fall 1 oder Fall 2 erzielt werden kann. Daher wurde dieser Wert mit den folgenden beiden Formeln für verschiedene Parameter bestimmt, es wurde jeweils immer von best-case ausgegangen. G_0 wurde der Einfachheit halber auf 1 gesetzt.

$$(6.1)\quad MinMessages_{case1} = (((\frac{n}{2} + 0,5) * n_{LBS}) + n) + n_{updates} + (n_{updates} * (n_{LBS} * (1 - \frac{1}{n})))$$

Formel 6.1 setzt sich aus den folgenden drei Summanden zusammen: Anzahl an Updates für die erste Nachricht (hier wurde für die Summe bereits der Maximalwert $\frac{n}{2} + 0,5$ eingesetzt),



(a) Maximale Nachrichteneinsparung in Fall 1 in Abhängigkeit von n, $nLBS$ und der Anzahl der Updates



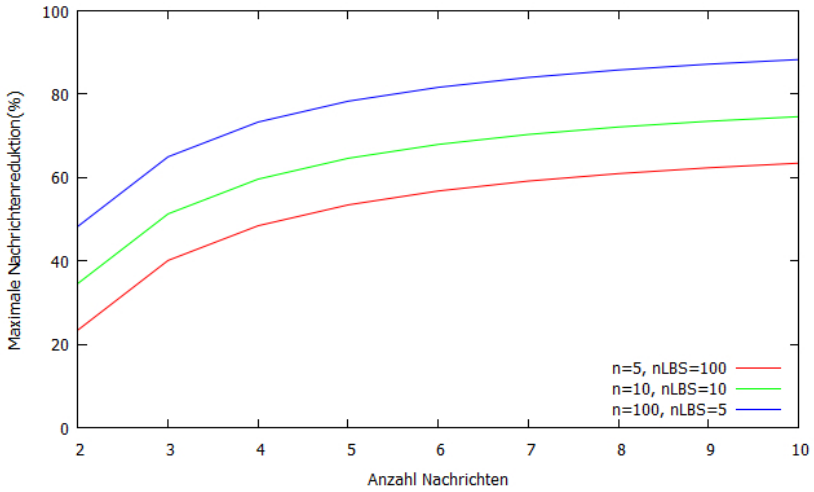
(b) Maximale Nachrichteneinsparung in Fall 2 in Abhängigkeit von n, $nLBS$ und der Anzahl der Updates

Abbildung 6.17: Vergleich der maximal möglichen Nachrichteneinsparung von Fall 1 und Fall 2

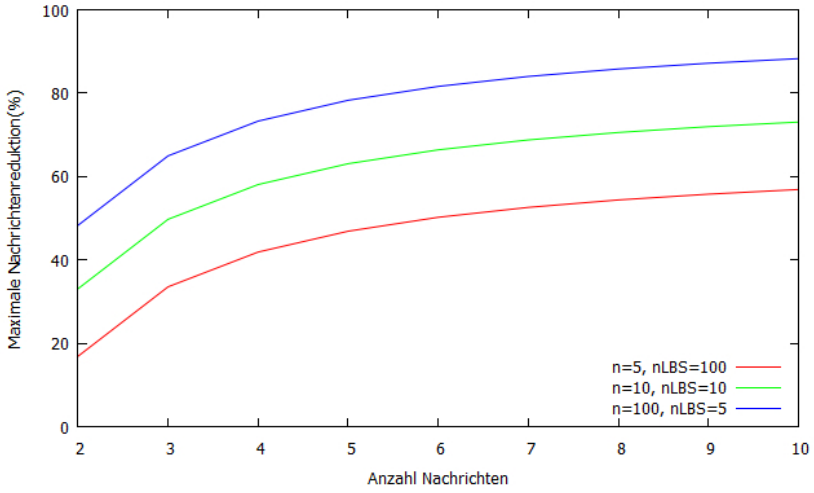
die Anzahl an Nachrichten zwischen MO und LS und die Anzahl an Nachrichten zwischen LS und LBS (hier wurde für die Summe bereits der kleinste mögliche Wert eingesetzt, da immer nur der letzte Share updgedatet wird).

$$MinMessages_{case2} = (((\frac{n}{2} + 0,5) * n_{LBS}) + n) + n_{updates} + (n_{updates} * n_{LBS}) \quad (6.2)$$

Formel 6.2 setzt sich aus den selben Summanden zusammen wie Formel 6.1. Die Resultate werden in Abbildung 6.17 dargestellt. Man erkennt, dass die maximal mögliche Nachrichtenreduktion stark von der Anzahl der LBS und der Location-Server abhängt. Umso mehr



(a) Maximale Nachrichteneinsparung in Fall 1 in Abhängigkeit von n, $nLBS$ und der Anzahl der Updates, andere Skalierung



(b) Maximale Nachrichteneinsparung in Fall 2 in Abhängigkeit von n, $nLBS$ und der Anzahl der Updates, andere Skalierung

Abbildung 6.18: Vergleich der maximal möglichen Nachrichteneinsparung von Fall 1 und Fall 2, andere Skalierung

Location-Server und umso weniger LBS vorhanden sind, desto mehr Nachrichten können maximal eingespart werden. Klar ist auch, dass die maximale Nachrichtenreduktion mit der Anzahl der Nachrichten immer mehr ansteigt. Die Schaubilder für Fall 1 und Fall 2 scheinen auf den ersten Blick identisch zu sein. Ändert man aber die Skalierung der X-Achse, kann man doch marginale Unterschiede erkennen. Diese sind auf Abbildung 6.18 zu sehen. Es ist zu erkennen, dass vor allem die rote Linie als auch die grüne Linie bei einer niedrigeren Nachrichtenreduktion beginnt und auch wieder endet. Bei der blauen Linie gibt es ebenfalls Unterschiede, die aber anhand des Schaubildes alleine nicht erkennbar sind. Die Messdaten

zeigen allerdings, dass die maximale Reduktion in Fall 2 immer ca. 1% unterhalb der von Fall 1 liegt. Insgesamt ist also ein ganz leichter Vorteil von Fall 1 gegenüber Fall 2 vorhanden, wenn es um die maximale Einsparung geht. Dieser ist aber so marginal, dass allgemein die Aussage getroffen werden kann, dass sich beide Fälle gleich gut zum Einsparen von Nachrichten eignen.
Abschließend wird noch eine Performance-Analyse durchgeführt, in der die Ausführungszeit der share-generation Algorithmen, sowie dem Algorithmus der das optimale k berechnet, auf einem Smartphone gemessen wird.

6.9 Performance-Analyse

Die hier vorgestellten Algorithmen wurden ebenfalls wie im Basis-Ansatz auf einem HTC Desire Smartphone getestet. Die Ergebnisse des share-generation Algorithmus für Fall 1 (hier wurde der a-posteriori Algorithmus benutzt, da der a-priori Algorithmus nicht so effizient ist), des share-generation Algorithmus für Fall 2, sowie dem Algorithmus, der für eine gegebene Anzahl an Location-Servern, LBS und G_0 das optimale k bestimmt, sind in den Abbildungen 6.19, 6.20 und 6.21 zu sehen.

n	2	4	8	16	32	64	128
Fall 1 (a Posteriori)	0.34	0.81	1.12	1.85	2,44	5.63	13.49

Abbildung 6.19: Performance des share-generation Algorithmus für Fall 1 auf einem HTC Desire, Ausführungszeit in ms

n	2	4	8	16	32	64	128
Fall 2 (nur Master-Share)	0.11	0.11	0.11	0.11	0.11	0.11	0.11

Abbildung 6.20: Performance des share-generation Algorithmus für Fall 2 auf einem HTC Desire, Ausführungszeit in ms

n	1	100	10	5	10	100
nLBS	100	1	10	10	5	100
Finde optimales k	0.01	0.03	0.02	0.01	0.01	0.08

Abbildung 6.21: Performance des Algorithmus zum Finden eines optimalen k auf einem HTC Desire, Ausführungszeit in ms

Zunächst ist zu sagen, dass alle Algorithmen in Echtzeit arbeiten (sofern nicht der a-priori Algorithmus benutzt wird), und somit das entwickelte System zur Echtzeit-Verschleierung von Trajektorien geeignet ist. Der Algorithmus für Fall 1 benötigt mehr Zeit, desto mehr Shares er erzeugen muss. Allerdings steigt die benötigte Zeit nur geringfügig an. Der Algorithmus für Fall 2 benötigt logischerweise immer gleich viel Zeit, da immer nur ein Share (und zwar der Master-Share) gefunden werden muss, der alle anderen Shares (bzw. den letzten Share) vollständig enthält. Der Algorithmus, der ein optimales k findet, benötigt auch für unterschiedliche Konstellationen von n und $nLBS$ kaum merkbar mehr Zeit als für andere Konstellationen. Die Parameter haben also keinen wirklichen Einfluss auf die Ausführungszeit (sofern realistische Werte dafür verwendet werden).
Zudem ist anzumerken, dass das HTC Desire nun auch schon ein Gerät der älteren Generation ist und die Algorithmen auf einem neueren Gerät wie z. B. einem Samsung Galaxy SIII oder einem IPhone 5 vermutlich noch schneller laufen würden.
Im nächsten Kapitel wird die Arbeit zusammengefasst, und es werden eventuelle Anknüpfungspunkte und weitere Verbesserungsvorschläge für die Zukunft vorgestellt.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde der in Kapitel 3 vorgestellte Ansatz so verbessert, dass in manchen Fällen bis zu über 90% an Update-Nachrichten eingespart werden können. Diese Einsparung an Nachrichten wird dabei ohne jeglichen Verlust der Sicherheit oder Genauigkeit erzielt. Die Verbesserung basiert hauptsächlich auf zwei Ansätzen:
Fall 1 kann angewendet werden, wenn sich der Benutzer zwischen den Updates nur wenig oder gar nicht bewegt, und wenn der Radius des Master-Shares ausreichend groß gewählt ist. Der Trick ist, dass man in Fall 1 nicht alle Verschiebungs-Vektoren neu erzeugen (und per Update-Nachricht schicken) muss. Es wird berechnet, wie viele Kreise der Benutzer überschritten hat, und lediglich die Vektoren, die auf die Mittelpunkte der überschrittenen Kreise gezeigt haben, werden neu berechnet und geschickt. Da die Vektoren eine relative Positionsangabe sind, können die neu berechneten Vektoren einfach an die alten Vektoren angehängt werden.
Fall 2 hingegen kann angewendet werden, wenn sich der Benutzer zwischen den Updates viel bewegt, und der Radius des Master-Shares kleiner ist. In Fall 2 wird nur der Master-Share neu berechnet, und die restlichen Verschiebungs-Vektoren werden an den neu berechneten Master-Share angehängt.
Die beiden Fälle können jedoch nur angewendet werden, wenn bestimmte Voraussetzungen erfüllt sind. In Fall 1 muss sich der Benutzer mindestens noch im Kreis des Master-Shares befinden, und in Fall 2 dürfen sich die Kreise der beiden Master-Shares aus Sicherheitsgründen nicht überschneiden.
Die beiden Verbesserungsvorschläge wurden dann analytisch ausgewertet, und es wurde eine Formel aufgestellt, die angibt, bis zu welchem überschrittenen (k-ten) Kreis Fall 1 noch mehr Nachrichten einspart als Fall 2, sowie ein Algorithmus implementiert, der dieses k berechnet. Hier wurden nicht nur die Nachrichten zwischen MO und LS berücksichtigt, sondern auch die Nachrichten zwischen LS und LBS. Ebenfalls wurde ein Java-Programm entwickelt, das es ermöglicht, aufgezeichnete Benutzer-Trajektorien einzulesen, zu visualisieren und das System zu simulieren. Die Simulation wurde mit verschiedenen echten Trajektorien aus dem GeoLife-Datensatz durchgeführt und anschließend ausgewertet.
In der durchgeführten Simulation mit echten Benutzer-Trajektorien aus dem GeoLife-Datensatz hat sich gezeigt, dass in den meisten Fällen viele Update-Nachrichten gegenüber dem naiven Ansatz (alle Shares neu berechnen und schicken) eingespart werden können. Ein weiterer positiver Effekt ist die Tatsache, dass der Ansatz zumindest für kontinuierliche Updates mit steigendem r_0 besser wird. Ist also ein Benutzer um seine Sicherheit besorgt, kann er problemlos einen großen Master-Share Radius auswählen. Ebenfalls positiv ist, dass dies die Servicequalität des LBS nicht im Geringsten einschränkt, da dieser, wenn notwendig, alle Shares abfragen kann und somit trotzdem die exakte Benutzerposition erhält.
Abschließend wurde noch ausgewertet, welche maximale Nachrichtenreduktion mit den

beiden Ansätzen erreicht werden kann. Die Ergebnisse zeigten, dass Fall 1 hier eine geringfügig höhere Einsparung erzielen kann, der Unterschied aber sehr marginal ist. Ebenfalls wurde eine Performance-Analyse auf einem HTC Desire Gerät durchgeführt, um die Effizienz der implementierten Algorithmen beurteilen zu können. Es hat sich gezeigt, dass die Algorithmen die entsprechenden Ergebnisse in Echtzeit liefern können, und somit auch für den Einsatz in on-the-fly tracking Anwendungen bestens geeignet sind.

Ausblick

Der in dieser Arbeit vorgestellte Verbesserungsvorschlag könnte in der Zukunft noch weiter optimiert werden:
Da an der Nachrichtenanzahl eines einzelnen Benutzers nicht mehr viel zu optimieren ist, wäre es zum Beispiel möglich, dass nicht nur die Nachrichtenanzahl eines einzelnen Benutzers optimiert und verschleiert wird, sondern von einer ganzen Gruppe aus Benutzern, die zusammen unterwegs sind. Ebenfalls könnte man das Ganze so erweitern, dass nicht nur einzelne Benutzerpositionen verschleiert werden können, sondern ganze Trajektorien.

Auch könnte man das System adaptiver gestalten:
Da der Mensch ein in allem nach einer gewissen Zeit eine Routine entwickelt, kann man das System an die Gewohnheiten eines Benutzers anpassen. So geht man z. B. jeden Tag zur selben Zeit arbeiten und fährt meistens auch zur selben Zeit wieder nach Hause. Man geht immer an einem Tag in der Woche einkaufen, usw. So fährt/läuft man auch täglich dieselben Strecken ab. Wenn man das System also so abändern würde, dass es zunächst eine Art Lernphase durchläuft, um sich an die täglichen Abläufe des Benutzers zu gewöhnen, könnte es ähnlich wie beim dead-Reckoning irgendwann die Positionen des Benutzers vorausberechnen. Auf diese Weise könnte man nochmal einiges an Update-Nachrichten einsparen.

Ein weiterer Verbesserungsvorschlag wäre es, die Akkulebensdauer des mobilen Objekts zu schonen. Ähnlich wie in [KrLGTr09] könnte man versuchen, energieverbrauchende Komponenten des mobilen Objekts wie z. B. den GPS-Sensor auszuschalten, wenn dieser nicht benötigt wird. Auch könnte man ab einem gewissen Akkustand G_0 automatisch so verändern, dass bevorzugt mehr Nachrichten zwischen MO und LS eingespart werden, da das schicken von Positionsupdates ebenfalls viel Akku verbraucht. Umso niedriger der Akkustand ist, desto höher wird G_0 gewichtet, was dazu führt, dass Fall 2 öfters angewendet wird (wir erinnern uns, dass Fall 2 besser zur Nachrichteneinsparung zwischen MO und LS geeignet ist). Auch wäre es denkbar, dass gleichzeitig mit einer Inkrementierung von G_0 eine Dekrementierung von r_0 stattfindet, was ebenfalls dazu führt, dass Fall 2 öfters angewendet wird. Hier könnte man den Benutzer einen festen Minimalwert angeben lassen, der nicht unterschritten werden darf. Umso niedriger der Akkustand ist, desto mehr nähert sich das System an diesen Minimalwert an.
Zu guter Letzt kann noch das zeitliche Intervall zwischen den Nachrichten erhöht werden (sofern ein time-based Protokoll gewählt wurde), bzw. im Falle eines distance-based Protokolls

kann die Distanz zwischen den Updates vergrößert werden. Beides würde ebenfalls dazu führen, dass Fall 2 bevorzugt angewandt wird, und somit weniger Nachrichten zwischen MO und LS geschickt werden müssen.

Literaturverzeichnis

[DSR11] F. Dürr, P. Skvortsov, K. Rothermel. Position sharing for location privacy in non-trusted systems. In *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications*, PERCOM '11, S. 189–196. IEEE Computer Society, Washington, DC, USA, 2011. URL http://dx.doi.org/10.1109/PERCOM.2011.5767584. (Zitiert auf den Seiten 7, 12, 24, 33, 34, 37, 39, 41, 42 und 44)

[Gav08] M. Gavaghan. Java Geodesy Library for GPS – Vincenty's Formulae, 2008. URL http://www.gavaghan.org/blog/free-source-code/geodesy-library-vincentys-formula-java/. (Zitiert auf Seite 64)

[KrLGTr09] M. B. Kjæ rgaard, J. Langdal, T. Godsk, T. Toftkjæ r. EnTracked: energy-efficient robust position tracking for mobile devices. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, S. 221–234. ACM, New York, NY, USA, 2009. URL http://doi.acm.org/10.1145/1555816.1555839. (Zitiert auf den Seiten 41 und 88)

[LDR08] R. Lange, F. Dürr, K. Rothermel. Online trajectory data reduction using connection-preserving dead reckoning. In *Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Mobiquitous '08, S. 52:1–52:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2008. URL http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3460. (Zitiert auf Seite 26)

[LDR11] R. Lange, F. Dürr, K. Rothermel. Efficient real-time trajectory tracking. *The VLDB Journal*, 20(5):671–694, 2011. URL http://dx.doi.org/10.1007/s00778-011-0237-7. (Zitiert auf Seite 20)

[LNR02] A. Leonhardi, C. Nicu, K. Rothermel. A Map-Based Dead-Reckoning Protocol for Updating Location Information. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, S. 15–. IEEE Computer Society, Washington, DC, USA, 2002. URL http://dl.acm.org/citation.cfm?id=645610.662039. (Zitiert auf den Seiten 7, 18, 24 und 25)

[LR01] A. Leonhardi, K. Rothermel. A Comparison of Protocols for Updating Location Information. *Cluster Computing*, 4(4):355–367, 2001. URL http://dx.doi.org/10.1023/A:1011872831932. (Zitiert auf den Seiten 7 und 15)

[Men12] Mendhak. Gps Logger for Android, 2012. URL http://www.androidpit.de/de/android/market/apps/app/com.mendhak.gpslogger/GPS-Logger-for-Android/. (Zitiert auf Seite 64)

[Ope12] staticMapLite - simple map for your website, 2012. URL http://staticmap.openstreetmap.de/. (Zitiert auf Seite 64)

[RPA+12] K. Ramya, K. Priyanka, K. Anusha, K. Brahmini, G. Lakshmi, K. Abraham. Advanced Query Evaluation Techniques for Preserving Privacy and Efficiency of Mobile Objects. In *International Journal of Modern Engineering Research (IJMER)*, S. 417–423. 2012. (Zitiert auf Seite 29)

[Sab09] V. Sabino. Image2Html, 2009. URL http://sourceforge.net/projects/image2html/. (Zitiert auf Seite 64)

[Skv12] P. Skvortsov. Optimizing Location Update Costfor Position Sharing. Internal Talk, 2012. (Zitiert auf den Seiten 7, 8, 23, 40, 43, 45 und 52)

[SYZ05] J. Shen, K. Yang, S. Zhong. *A Prediction-Based Location Update Algorithm in Wireless Mobile Ad-Hoc Networks*, S. 692–701. Springer-Verlag Berlin Heidelberg, 2005. (Zitiert auf den Seiten 7 und 20)

[Tay11] F. Taylor. GPS Analysis, 2011. URL http://sourceforge.net/projects/gpsanalysis/. (Zitiert auf Seite 64)

[WDR12] M. Wernke, F. Dürr, K. Rothermel. PShare: Position sharing for location privacy based on multi-secret sharing. In S. Giordano, M. Langheinrich, A. Schmidt, Herausgeber, *PerCom*, S. 153–161. IEEE, 2012. URL http://dblp.uni-trier.de/db/conf/percom/percom2012.html. (Zitiert auf Seite 46)

[WW10] S. Wang, X. S. Wang. In-Device Spatial Cloaking for Mobile User Privacy Assisted by the Cloud. In *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*, MDM '10, S. 381–386. IEEE Computer Society, Washington, DC, USA, 2010. URL http://dx.doi.org/10.1109/MDM.2010.82. (Zitiert auf den Seiten 7, 28, 29, 30 und 31)

[XC09] T. Xu, Y. Cai. Feeling-based location privacy protection for location-based services. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, S. 348–357. ACM, New York, NY, USA, 2009. URL http://doi.acm.org/10.1145/1653662.1653704. (Zitiert auf den Seiten 7, 27 und 28)

[ZXWY10] Y. Zheng, X. Xie, M. Wei-Ying. GeoLife: A Collaborative Social Networking Service among User, location and trajectory. Technischer Bericht, 2010. (Zitiert auf den Seiten 8 und 63)

Alle URLs wurden zuletzt am 14.10.2012 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Simon Haenle)