MLS, Mobile Location Service

Roland Stühmer

Ausarbeitung zum Seminar Algorithmen für Ad-hoc- und Sensor-Netze im SS 07

Institut für theoretische Informatik
Prof. Dr. Dorothea Wagner
Betreuer: Bastian Katz, Steffen Mecke, Dorothea Wagner

28.9.2007

Universität Karlsruhe (TH)

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Ausarbeitung beschäftigt sich mit dem Paper *MLS: An Efficient Location Service for Mobile Ad Hoc Networks* von Roland Flury und Roger Wattenhofer, [FW06]. Das Paper stellt einen Mechanismus vor, um in einem Ad-hoc-Netzwerk Nachrichten an Knoten unbekannter Position zu schicken. Darüber hinaus dürfen die Knoten beweglich sein, auch während der gleichzeitig laufenden Nachrichtenzustellung. Dazu liefert der Location Service MLS eine Datenstruktur, in welcher über das Netzwerk verteilt Informationen zur momentanen Position der Knoten gespeichert werden, sowie einen Algorithmus, um diese Datenstruktur bei Positionsänderungen aktuell zu halten (sog. Publishing), und einen Routingalgorithmus (sog. Lookup), der diese Informationen nutzt.

MLS beschäftigt sich *nicht* mit dem grundlegenden Georouting von Knoten zu Knoten und löst nicht Probleme wie zum Beispiel das Umgehen von Seen, sprich Löchern in der Landschaft. Vielmehr setzt MLS diese Funktionalität voraus und nutzt sie, um zum Beispiel bei der Zustellung einer Nachricht einen bekannten Pointer zu besuchen, der den Weg zum (unbekannten) Zielknoten anzeigt.

Das Routing von MLS erfüllt die oben genannten Voraussetzungen in konstanter Routingqualität, also nur abhängig vom Abstand der beiden Knoten.

In Kapitel 2 werden zunächst einige Voraussetzungen vorgestellt, die MLS aus verwandten Arbeiten zu Ad-hoc-Netzwerken nutzt oder erweitert. Kapitel 3 betrachtet die Neuerungen von MLS, wie beispielsweise die Gleichzeitigkeit von Nachrichtenzustellung und Bewegung des Zielknotens. In Kapitel 4 wird eine mathematische Analyse von MLS durchgeführt, die unter anderem obere Schranken für den Aufwand einiger Operationen aufstellt, und zuletzt werden in Kapitel 5 Ergebnisse aus Simulationen von MLS vorgestellt.

2 Grundlagen und Voraussetzungen

In diesem Kapitel werden zunächst einige wichtige Voraussetzungen aufgezählt, auf die MLS baut, und danach Ergebnisse anderer Arbeiten zusammengetragen, die in MLS verwendet oder adaptierte werden.

2.1 Voraussetzungen

MLS bietet sicheres Routing von Nachrichten zu Zielknoten unbekannter Position bei konstanter Routingqualität. Dazu werden jedoch einige Annahmen gemacht, die das Ad-hoc-Netzwerk erfüllen muss.

Die Welt in der die Netzwerkknoten ausgestreut werden, kann **Land und Seen** enthalten. Letzteres sind Gebiete in denen keine Knoten existieren können. Um ein zusammenhängendes Netzwerk zu erhalten, wird außerdem angenommen, dass es keine unerreichbaren Inseln gibt.

In solch einem Netzwerk, das MLS einsetzt, müssen alle Knoten ihre eigene Position kennen. Da die Knoten mobil sind, müssen sie die Position selbst messen können, also jeder ein eingebautes System zur **Positionsbestimmung** besitzen. Welcher Art dieses System ist, spielt für MLS keine Rolle; Möglichkeiten sind zum Beispiel GPS, Cricket [SBGP04], Mobilfunk- oder WLAN-Triangulierung.

Darüber hinaus braucht jeder Knoten ein Mittel zur **Funkkommunikation**, wie zum Beispiel WLAN, das eine Mindestreichweite von r_{min} aufweist. Diese Zahl geht später mit ein in die Aufteilung des Ausstreuungsgebiets des Ad-hoc-Netzwerks.

An die **Knotendichte** werden ebenfalls Anforderungen gestellt. Sie muss recht hoch und gleichmäßig sein, auch nachdem sich Knoten bewegt haben. So muss es zu jeder Position auf dem Land einen Netzwerkknoten in maximal $\lambda = r_{min}/3$ Entfernung geben.

Desweiteren setzt MLS ein funktionierendes **Georouting** voraus, als unterliegendes Kommunikationsprotokoll. Dieses Protokoll bietet sichere Kommunikation zu einem Zielpunkt, dessen Position bereits bekannt ist und wird von MLS als sicheres Kommunikationsprimitiv vorausgesetzt.

Da MLS eine Art Geohashing einsetzt, die zu einer Knoten-ID eine beliebige aber feststehende Position auf dem Land zuordnet, müssen die Knoten die **Positionen aller Seen** kennen, um diese Hash-Funktion berechnen zu können.

2.2 Verwandte Arbeiten

Ein wichtiger Baustein von MLS ist, wie oben erwähnt, das Georouting. Facetten-Routing ([BMSU99]) ist ein prominenter Vertreter. Alle Georouting-Algorithmen kombinieren

Greedy Forwarding mit vielen Verbesserungen zur Umgehung von Löchern im Netzwerk und lokalen Minima an Löchern, etc. Allen gemeinsam ist, dass der Sender die Position des Empfängers kennen muss.

Dies erfordert linearen Speicheroverhead in jedem Knoten, wenn ein Knoten die Koordinaten von jedem anderen speichern muss. Im Falle eines *mobilen Ad-hoc-Netzwerkes* (MANET) kommt dazu noch der hohe Kommunikationsaufwand, wenn ein Knoten seine Positionsänderung jedem einzelnen anderen mitteilen muss.

Um den genannten Speicher- und Kommunikationsaufwand zu verringern, wurden so genannte *home-based* Ansätze vorgestellt ([EFK04], [RPSS03], [VAF+05], [WS01]). Dabei bekommt jeder Knoten ein weltweit bekanntes „Home“ zugeordnet, um seine gegenwärtigen Positionsdaten zu hinterlegen. Solch ein Home ist eine Koordinate an die die Daten geschickt werden. Ein vorhandenener Knoten in der Nähe dieser Koordinate nimmt die Daten an, und stellt den Speicher bereit. Somit fällt bei Bewegung nur noch linearer Kommunikationsaufwand mit dem Home an und jede Position muss nur noch in einem anderen Knoten gespeichert werden, nicht mehr in allen.

Das Home wird über eine Geohash-Funktion ermittelt, die es ebenfalls allen anderen Knoten ermöglicht, es zu finden. Dort ist dann die Position des Zielknotens hinterlegt und wird durch das Publishing bei Änderungen aktualisiert. Jedoch wird bei einem weit entfernten Home das Auffinden (Lookup) eines in der Nachbarschaft befindlichen Knotens unnötig erschwert, also die Routingqualität im Sinne von Dilation sehr verschlechtert. Im schlimmsten Fall führt die gefundene Route quer durch die gesamte Austreuungslandschaft und zurück, was nicht proportional zur Entfernung von Start zu Ziel ist. Dies will MLS jedoch leisten.

2.3 Modell

MLS unterteilt die Ausstreuungslandschaft in eine Hierarchie logischer Quadrate. Das größte Quadrat, die Hierarchiestufe L_M, umgibt die komplette Landschaft, in der sich die Knoten bewegen sollen. Jedes Quadrat unterteilt sich in seine vier Teilquadrate der nächsten Hierarchiestufe, bis zur kleinsten Hierarchiestufe L_0 mit Quadraten der Seitenlängen ρ. Diese Quadrate werden nicht weiter unterteilt; die enthaltenen Knoten können alle direkt kommunizieren. Die Seitenlänge ρ berechnet sich wie folgt: $\rho = \lambda/\sqrt{2} = \frac{r_{min}}{3}/\sqrt{2}$. Das Quadrat der Ebene i, das den Knoten t enthält, wird als L_i^t bezeichnet.

Jedes Quadrat, egal welcher Ebene, erhält nun ein *Home* für jeden enthaltenen Knoten. Ein Home ist eine Position auf der Landkarte, und ein nahegelegener Knoten übernimmt die Speicherung der Daten und fungiert damit als das Home. Ein Home enthält bei MLS jedoch nicht die endgültige Zielposition des gesuchten Knotens, sondern lediglich einen Zeiger auf das Teilquadrat der nächstkleineren Hierarchiestufe, in das abgestiegen werden muss. Die Homes werden bei MLS als *Level Pointer, LP,* bezeichnet.

Sobald beim Routing ein solcher Level Pointer LP_i gefunden wurde, folgt man seinem Zeiger in das zutreffende Unterquadrat zum dortigen LP_{i-1}, und so fort. Auf diese Weise steigt man die Hierarchie hinab zum kleinsten Quadrat der Seitenlänge ρ, welches den Zielknoten mit der gesuchten ID enthält. Siehe Abbildung 2.1. Die kleinsten Quadrate

(Ebene L_0) enthalten keine eigenen Level Pointer mehr.

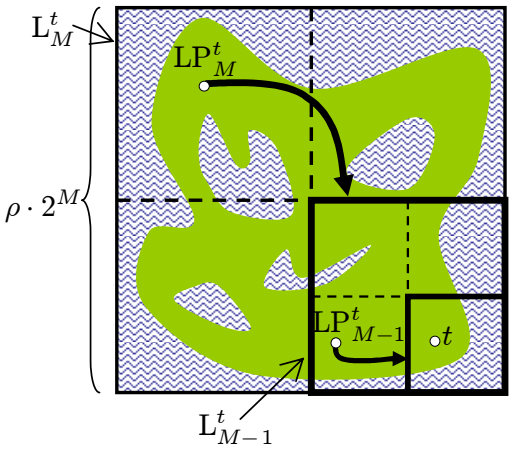

Abbildung 2.1: Hierarchie von Quadraten und Pointern

Die Position der Level Pointer einer Ebene werden aus der Kenntnis der Seen auf dieser Ebene und der Knoten-ID des Zielknotens berechnet. Diese Geohash-Funktion sollte garantieren, dass die Level Pointer verschiedener Knoten gleichmäßig über die Landschaft verteilt sind. So wird der Speicheroverhead von MLS gut verteilt und Flaschenhälse beim Routing vermieden. Die Anzahl der Hierarchiestufen wird nur bestimmt durch die Größe des Ausstreuungsgebiets und den Senderadius r_{min}. Somit speichert jeder Knoten im Schnitt nur eine Anzahl an Level Pointern abhängig von der Größe des Ausstreuungsgebiets, die zur Laufzeit konstant ist.

Für das Errechnen der Hashfunktion ist auf allen Knoten die Kenntnis der Seen erforderlich, so dass Level Pointer nur auf dem Land zu Liegen kommen; nicht etwa auf Wasser, wo keine Knoten die Information aufnehmen könnten, also als Level Pointer agieren könnten.

Jede Geohash-Funktion ist für MLS geeignet, die (mit Kenntnis der Lage der Seen) zu einer Knoten-ID ein Koordinatenpaar über Land ausgibt, und diese Koordinaten gleichmäßig über die Landschaft verteilt. Die Autoren schlagen folgende Funktion vor, bestehend aus $H_1(ID_t)$ und $H_2(ID_t)$: Das Argument ist jeweils die Knoten-ID des Knotens t, zu dem ein Level Pointer gefunden werden soll. H_1 bestimmt die y-Koordinate, und zwar den Anteil an Land, der oberhalb von y_t liegen soll. Die x-Koordinate wird daraufhin so bestimmt, dass die Streckensegmente mit $y = y_t$, die über Land liegen, also alle in Frage kommenden Teile des gefundenen „Breitengrades“ konkateniert werden. H_2 gibt dann die Position auf dieser Strecke an. Siehe dazu Abbildung 2.2. H_1 und H_2 geben ihre Funktionswerte im Intervall]0,1] aus, und werden dann mit den vorliegenden Streckenlängen multipliziert.

An die gefundene Position (x_t, y_t) wird ein Create Request gesendet, um den Level

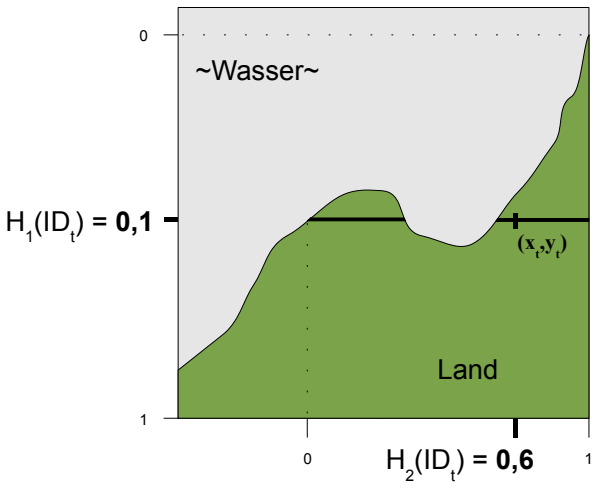

Abbildung 2.2: Geohash-Funktion für MLS

Pointer anzulegen. Der Knoten, der dieser Position am nächsten gelegen ist, verarbeitet die Anfrage. Sollte er sich später zu weit von der optimalen Position des Level Pointers wegbewegen, gibt er den Pointer an einen besser gelegenen Knoten weiter. Beim Routing kann der Level Pointer jeder Hierarchiestufe in gleicher Weise mit obiger Hashfunktion wiedergefunden und genutzt werden.

2.4 Lookup

Das Routing wird bei MLS als *Lookup* bezeichnet, um zu betonen, dass die Zielposition der Route unbekannt ist. Darüber hinaus ist zu beachten, dass die zu übertragende Nachricht beim Suchen der Route komplett mitgereicht wird, so dass beim Finden des Zielknotens die Nutzlast auch schon zugestellt ist.

Sobald ein in Abschnitt 2.3 beschriebener Pointer gefunden ist, kann ihm und nachfolgenden Pointern bis zum Zielknoten gefolgt werden. Siehe dazu auch Abbildung 2.3. Hier soll nun die erste Phase des Lookup beschrieben werden, das Auffinden eines ersten Pointers möglichst niedriger Ebene.

Wenn ein Knoten s einem Zielknoten t eine Nachricht zusenden will, geht s folgendermaßen vor: Zuerst befragt s seine Nachbarschaft, ob sie den Zielknoten kennt und sendet gegebenenfalls direkt. Wenn sich der gesuchte Knoten in der Nachbarschaft nicht meldet, sendet s den Lookup Request zur errechneten Position des Level Pointers im umgebenden Quadrat der Ebene L_1. Siehe Abbildung 2.3. Wenn sich an dieser Position kein Level Pointer zu t findet, dann ist der Zielpunkt t nicht im besuchten Quadrat L_1.

Daraufhin wird die Nachricht weitergeleitet, jedoch noch nicht in Richtung der vermeintlichen Position des nächsthöheren Level Pointers in L_2, sondern zunächst im Kreis

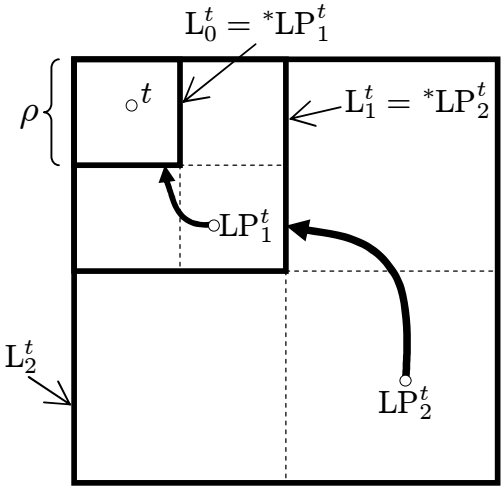

Abbildung 2.3: Unteres Ende der Pointerhierarchie

um die acht Nachbarquadrate der gleichen Ebene von L_1. Erst wenn dort auch an keiner der errechneten Positionen ein Level Pointer gefunden wurde, wird in der Hierarchie aufgestiegen zu L_2. Dort wird rekursiv fortgefahren mit dem Aufsuchen des potentiellen Level Pointers im Quadrat um s und danach deren in den acht Nachbarquadraten, und so weiter, bis aller spätestens im schlimmsten Fall der Level Pointer der obersten Ebene L_M besucht werden muss.

Das erwähnte Umkreisen eines Quadrats der aktuellen Ebene vor dem Aufstieg in die nächste Ebene findet statt, um die Dilation des Lookup klein zu halten. Falls zwei Knoten beispielsweise durch eine größere Achse der Hierarchie getrennt sind, muss der Lookup Request des einen Knotens sehr weit in der Hierarchie aufsteigen, um die Grenze zum anderen Knoten überschreiten zu können. Die beiden Positionen a und b in Abbildung 2.5 sind ein drastisches Beispiel hiervon. Wenn ein Lookup Request von a stets in der Hierarchie aufsteigt, anstatt auch die Nachbarschaft einer Hierarchie zu besuchen, muss er bis zum Level Pointer L_{i+2}^t und von dort wieder abwärts geroutet werden, anstatt den Zielknoten an Position b gleich beim ersten Umkreisen der Ebene L_{i-1} (im Bild ohne Bezeichnung) zu finden.

Durch das Umkreisen bleibt der zurückgelegte Weg des Lookup Requests von s nach t proportional zum tatsächlichen Abstand der Knoten. Der Weg eines solchen Lookup Requests, also eine gefundene Route in MLS, sieht beispielsweise aus wie in Abbildung 2.4. Die quadratischen roten Punkte bezeichnen Level Pointer des Zielknotens. Die Simulation in der Abbildung wurde mit einer Landschaft ohne Seen durchgeführt, weshalb in Phase 1 (linker Teil der Abbildung) das Besuchen der vermeintlichen Level Pointer sehr regelmäßige quadratische Bahnen zieht. Nachdem ein Level Pointer gefunden wurde, endet das regelmäßige Absuchen von Quadraten und der Lookup Request folgt in Phase 2 der Hierarchie abwärts, Pointer für Pointer.

Im selben Bild sieht man zwei Level Pointer von t, die vermeintlich verpasst wurden: Sie gehören jedoch zu höheren Ebenen und wurden deshalb nicht angesteuert. Aus genannten Gründen der Dilation darf ein Lookup Request nicht unnötig in der Pointerhierarchie aufsteigen.

In Kapitel 4 wird gezeigt, dass ein Lookup in $\mathcal{O}(d)$ abhängig von der Entfernung d zwischen Start und Ziel terminiert.

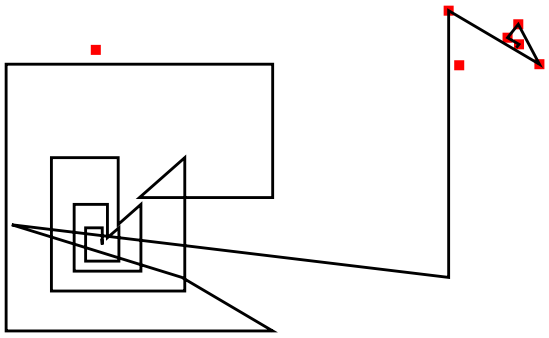

Abbildung 2.4: Lookup Request aus Simulation.

Ein *Lookup*, das Routing bei MLS, lässt sich also in zwei Phasen aufteilen. In der ersten Phase wird die Nachricht zu immer größeren umgebenden Quadraten weitergeleitet, aufwärts in der Hierarchie. In der zweiten Phase werden dann Pointer bis zum Ziel hin verfolgt, abwärts in der Hierarchie. Im Folgenden wird gezeigt, wie Knoten in Bewegung ihre Level Pointer aktuell halten.

2.5 Publishing

Da bei MLS alle Knoten mobil sein dürfen, reicht eine einmalige Initialisierung aller Level Pointer zu jedem Knoten nicht aus. Ein Knoten muss seine Positionsänderungen mitteilen können, diesen Vorgang nennt man *Publishing*. Sobald ein Knoten sich aus einem Quadrat L_i herausbewegt, muss er seinen Pointer auf L_i, der im umgebenden L_{i+1} liegt, ändern. Alle seine Pointer kleiner $i+1$ ändern sich damit auch, und benötigen ein Update auf ihr korrektes Unterquadrat, beziehungsweise müssen gelöscht und in ein neues Quadrat umgezogen werden.

Diese Änderungen müssen kommuniziert werden, was Netzwerklast verursacht. Gerade wenn viele Ebenen mit einer kleinen Bewegung überschritten werden, müssen viele *Delete-*, *Create-* und *Update-Requests* gesendet werden. Wenn ein Knoten auch noch in solch einer Bewegung oszilliert, wie beispielsweise zwischen den Punkten a und b in Abbildung 2.5, kann das Netzwerk verstopft werden. Um dies zu vermeiden, setzt MLS auf das so genannte Lazy Publishing.

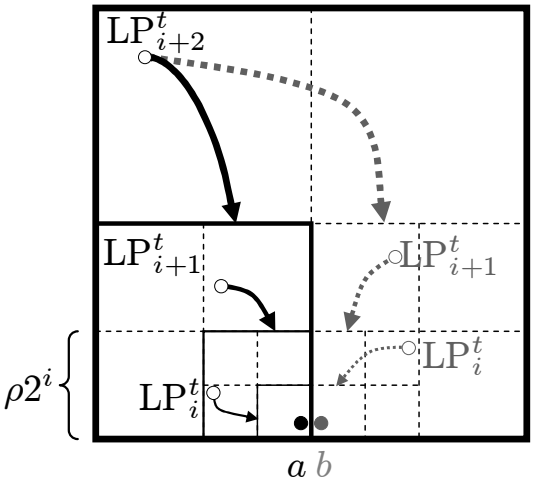

Abbildung 2.5: Problem oszillierender Knoten

2.6 Lazy Publishing

Lazy Publishing erlaubt es einem Knoten t, sein Quadrat in einem bestimmten Radius zu verlassen, ohne die bestehenden Level Pointer höherer Ebenen zu aktualisieren (siehe Abbildung 2.6). Um dennoch von einem Lookup Request gefunden zu werden, wird der Level Pointer des verlassenen Quadrats in einen Forwarding Pointer FP umgewandelt, der auf das neu betretene Quadrat zeigt, und dort wird ein Level Pointer angelegt.

Wenn t auch den Umgebungsradius verlässt, wird der Forwarding Pointer gänzlich gelöscht, und die höheren Ebenen müssen angepasst werden. Durch diese verzögerten, faulen, Änderungen verhindert Lazy Publishing das Überfluten des Netzwerks durch oszillierende Knoten.

Die Größe des Radius wird bestimmt durch $\alpha \times \rho \times 2^i$, also α mal der Seitenlänge des verlassenen Quadrats. Der Faktor $\alpha \in \mathbb{R}^+$, $0 \leq \alpha < 1$ ist ein beliebiger aber fester Parameter von MLS.

In Abschnitt 3.1 wird noch ein Mechanismus vorgestellt, der eine weitere Art von Pointern anlegt, damit Pointerinformation nicht zu schnell gelöscht wird, wenn sich ein Knoten weiterbewegt, aber ein Lookup noch unterwegs ist.

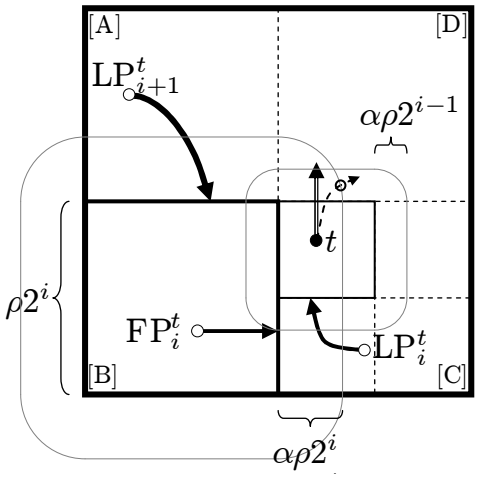

Abbildung 2.6: Lazy Publishing

3 Neuerungen

Die bisherigen Techniken waren größtenteils auch schon Bestandteil anderer Arbeiten und wurden für MLS übernommen und teilweise adaptiert. Nun folgen reine Neuerungen von MLS.

3.1 Concurrency

Es wurde bereits gesagt, dass Knoten mobil sein dürfen. Vorkehrungen dazu wurden von MLS getroffen. Jedoch unterstützt das bisher vorgestellte Publishing noch nicht die Mobilität eines Knotens, wenn gleichzeitig eine Nachricht zu ihm zugestellt werden soll. Folgt ein Lookup Request beispielsweise einem Pointer, dessen Update noch nicht bei ihm angekommen ist, und gelangt in ein Quadrat, wo der Pointer schon gelöscht wurde, ginge die Nachricht ins leere.

Also mussten bisher alle Publish Requests terminieren, bevor ein Lookup gestartet werden kann. Da man beweglichen Knoten somit fast nie etwas zuschicken könnte, unterstützt MLS die Gleichzeitigkeit von Publishing und Lookup.

Weil obige Wettlaufsituation durch das verfrühte Löschen eines Pointers verursacht wurde, löscht MLS seine Pointer nicht gänzlich, sondern hinterlässt beim Löschen so genannte Temporary Forwarding Pointers, TFP. Solch ein Pointer zeigt auf das benachbarte Quadrat, von dem aus der Löschauftrag gesendet wurde. Somit geraten Lookup Requests nicht mehr in Sackgassen.

Wie der Name *temporary* schon sagt, können die TFP_i nach einer Weile gelöscht werden, und zwar ist ihre Lebensdauer, TTL_i, nur abhängig von der Hierarchieebene i. In Kapitel 4 werden für die Lebensdauer Schranken angegeben. Damit kann die Lebensdauer jedes TFP statisch auf dem speichernden Knoten berechnet werden, und die Löschung bedarf keiner Kommunikation.

4 Analyse

Im Kapitel „Analyse“ des Papers beschäftigen sich die Autoren mit der Korrektheit von MLS. Es wird gezeigt, dass MLS funktioniert, insbesondere bei gleichzeitigem Auftreten von Lookup- und Publish Requests. Außerdem wird bewiesen, dass ein Lookup Request seinen Zielknoten in $\mathcal{O}(d)$ findet, wobei d die Entfernung von Start- zu Zielknoten ist. Auch wird gezeigt, dass die Gesamtkosten für das *Publishing* einer Positionsänderung in $\mathcal{O}(d \log d)$ liegen, wobei d hier die zurückgelegte Strecke der Positionsänderung ist.

Um diese Eigenschaften von MLS zu beweisen, wird die maximale Knotengeschwindigkeit v_{max}^{node} beschränkt, mit der sich die Knoten maximal fortbewegen dürfen. Dazu werden im Laufe der Beweise verschiedene Bedingungen an v_{max}^{node} gestellt, aus denen am Schluss die Geschwindigkeit ermittelt wird, die alle Bedingungen erfüllt.

Die Qualität des Lookup, $\mathcal{O}(d)$, basiert auf der Entfernung der Start- und Zielknoten $|st| = d$. Im Fall eines statischen Ad-hoc-Netzwerks ist diese Entfernung wohldefiniert, während der ganzen Zeit des Lookup. Bei einem mobilen Ad-hoc-Netzwerk mit gleichzeitigem Lookup und Publishing können sich jedoch beide Knoten bewegen, während der Lookup Request läuft. Die Distanz $|st|$ ändert sich. Für die Analyse wird deswegen die Distanz zum *Zeitpunkt des Absendens* des Lookup Request herangezogen.

4.1 Lookup Request

Für Phase 1 des Lookup wird gezeigt, dass jeder Lookup Request für einen beliebigen Zielknoten t so geroutet werden kann, dass er immer einen Level Pointer zu t findet. Für Phase 2 wird gezeigt, dass der Lookup Request stets entlang der Pointer zum Ziel geroutet werden kann.

4.1.1 Lookup – Phase 1

Zu Anfang wird eine untere Schranke ermittelt für TTL_i, die Lebensdauer der Temporary Forwading Pointer, so dass MLS auch bei Nebenläufigkeit keinen Pointer verpassen kann.

Lemma 4.1 *Wenn t sich in $L_i^s \cup (L_i^s)^8$ befindet, dort auch bleibt und einen LP_i^t auf diesem Gebiet unterhält, dann findet ein von s versandter Lookup Request einen Pointer auf t spätestens beim besuchen dieser ganzen Fläche $L_i^s \cup (L_i^s)^8$, wenn $TTL_i \geq \eta 2^i \rho(\sqrt{2} + 8\sqrt{5})$.*

Beweis. Wenn t seinen LP_i^t in sein aktuelles Quadrat L_i^t umziehen muss, sendet t einen Create Request m los, der höchstens $\Delta t_m \leq \eta 2^i \rho \sqrt{2}$ Zeit braucht, nämlich maximal quer durch L_i^t. (Der Faktor η ist der Routingoverhead, der durch das MLS zu Grunde

liegende Georouting verursacht wird.) Gleichzeitig sendet t einen Update- oder Delete Request m' in sein ehemaliges Quadrat, um den veralteten LP_i^t in einen korrekten FP_i^t oder TFP_i^t umzuwandeln. Für die Analyse des schlimmstenfalls, nehmen wir an, dass m' sofort zugestellt wird, was die erforderliche TTL nur vergrößert.

Ein Lookup Request findet weder den neuen noch den alten LP_i^t, falls er zwischen die Publish-Nachrichten gerät, so dass er vor m an der Position p des neuen ankommt, oder nach m' an der Position p' des inzwischen gelöschten alten. Schlimmstenfalls zieht t seinen Level Pointer genau nach L_i^s um, das Quadrat mit s, in der Mitte der kreisenden Suche. Wenn um Zeit T_0 der Umzug mit Absenden von m gestartet wird, aber der Lookup Request zum Zeitpunkt T_1 vor m bei Position p des Umzugsziels ankommt, dann macht der Lookup Request weiter mit den acht umliegenden Quadraten $(L_i^s)^8$. Je nachdem in welcher Reihenfolge die acht Quadrate besucht werden, könnte p' erst im achten gefunden werden, was nach spätestens $\Delta t_{lookup} \leq \eta 2^i \rho 8\sqrt{5}$ passiert (der Zeit um achtmal die längste Strecke durch zwei Quadrate zurückzulegen). Ein solcher Lookup Request erreicht p' spätestens zur Zeit $T_2 \leq T_1 + \Delta t_{lookup} < \eta 2^i \rho(\sqrt{2} + 8\sqrt{5})$. Wenn der alte LP_i^t wie angesprochen sofort zur Zeit T_0 in einen TFP_i^t umgewandelt wurde, existiert letzter bis $T_0 + TTL_i > T_2$ und der Lookup Request findet ihn. Für den Fall eines FP (Lazy Publishing) anstatt des TFP existiert ein Pointer noch viel länger, denn der FP wird erst noch durch einen TFP ersetzt, vor der endgültigen Löschung. □

Im nichtnebenläufigen, statischen Fall findet ein Lookup Request spätestens einen Pointer zu t beim Besuchen von $L_{k+1}^s \cup (L_{k+1}^s)^8$, wenn die Seitenlänge von L_k^s mindestens die Länge d hat, also $\rho 2^k \geq d$. (Auch mit Lazy Publishing muss t in der zusammengenommenen Fläche einen LP_{k+1}^t angelegt haben. Wenn nämlich s auf einer Außenkante von L_{k+1}^s liegt, und t orthogonal zur Kante in Entfernung d, dann bleibt ein Abstand von mindestens $\rho 2^k$ zwischen t und einem L_k ausserhalb von $L_{k+1}^s \cup (L_{k+1}^s)^8$, der größer ist, als der Radius des Lazy Publishing, denn die Zahl α, die diesen Radius festlegt, ist immer kleiner Eins, vergleiche Abschnitt 2.6.)

Für den nebenläufigen Fall schwächen wir das Ergebnis im folgenden Lemma 4.2 ab, zu $L_{k+\gamma}^s \cup (L_{k+\gamma}^s)^8$ (als maximal besuchte Fläche für das Auffinden eines Pointers zu t). Dabei gilt weiterhin $\rho 2^k \geq d > \rho 2^{k-1}$ und $\gamma \in \mathbb{N}^+$. Die Zahl $\gamma \geq 1$ wird ein Parameter von MLS, zum Perfomance-Tuning. Außerdem müssen die beiden folgenden Gleichungen (1) und (2) gelten, wie im Beweis zum nächsten Lemma gezeigt wird.

Die maximale Knotengeschwindigkeit muss folgendermaßen beschränkt sein:

$$v_{max}^{node} < \frac{1 - \frac{\alpha}{2} - 2^{-\gamma}}{\eta\sqrt{2}} \tag{1}$$

Die TFP müssen mindestens so lange existieren:

$$TTL_i \geq \eta 2^{i+1} \rho(1/\sqrt{2} + 8\sqrt{5}) \tag{2}$$

Man sieht, dass ein größeres γ eine höhere Knotengeschwindigkeit bringt, aber ein Lookup Request länger brauchen kann, um einen Pointer zu finden. Deswegen bleibt die Festlegung von γ wie erwähnt dem Benutzer zum Tuning überlassen.

Lemma 4.2 *Ein Lookup Request nach t wird von s ausgesandt, und d ist der Abstand der beiden Knoten zum Zeitpunkt des Absendens. Dann findet der Lookup Request spätestens einen Pointer zu t in einem der Quadrate von $L^s_{k+\gamma} \cup (L^s_{k+\gamma})^8$, wenn gleichzeitig folgendes gilt: $\gamma \geq 1, k \in \mathbb{N}$ so dass $\rho 2^{k-1} < d \leq \rho 2^k$ und die beiden Gleichungen (1) und (2) gelten.*

Beweis. Als erstes wird gezeigt, dass t einen $LP^t_{k+\gamma}$ angelegt hat, in $L^s_{k+\gamma} \cup (L^s_{k+\gamma})^8$, spätestens wenn der Lookup Request abgesandt wird. Das ist nötig, damit der Lookup Request nicht zu früh ankommt und den LP verpasst, weil dieser noch nicht eingerichtet wurde.

Per Definition ist s in $L^s_{k+\gamma}$, und damit ist t von diesem Quadrat höchstens $d \leq \rho 2^k$ entfernt. Gleichzeitig ist t von jedem $L_{k+\gamma-1}$ außerhalb der betrachteten Fläche $L^s_{k+\gamma} \cup (L^s_{k+\gamma})^8$ mindestens $\rho(2^{k+\gamma} - 2^k)$ entfernt. Man weiss, dass t trotz Lazy Publishing zu dieser Zeit schon einen Create Request für einen $LP^t_{k+\gamma}$ irgendwohin auf die betrachtete Fläche abgesandt haben muss. Zwischen dem Verlassen des Radius um das alte Quadrat und der aktuellen Position (in d-Entfernung von s) hat t eine Distanz zurückgelegt von mindestens $\Delta d = \rho 2^k (2^\gamma (1 - \frac{\alpha}{2}) - 1)$.

Der Update Request muss maximal *quer* durch $L_{k+\gamma}$ laufen, was eine Zeit von $\Delta t_{update} \leq \eta 2^{k+\gamma} \rho \sqrt{2}$ braucht. Aber der Lookup Request ist langsamer und kommt erst an, wenn der LP eingerichtet ist, denn der Lookup Request wird erst losgeschickt, nachdem sich t um Δd bewegt hat, was mindestens eine Zeit von $\Delta t_{move} \geq \Delta d / v^{node}_{max}$ braucht. Nach Gleichung (1) gilt $\Delta t_{move} > \eta 2^{k+\gamma} \rho \sqrt{2} \geq \Delta t_{update}$, also wird der Lookup Request erst losgeschickt, nachdem der $LP^t_{k+\gamma}$ in $L^s_{k+\gamma} \cup (L^s_{k+\gamma})^8$ angelegt ist.

Um den Beweis abzuschließen, muss noch gezeigt werden, dass ein $TFP^t_{k+\gamma}$ lang genug lebt, für den Fall, dass kein $LP^t_{k+\gamma}$ gefunden wird. Aus dem vorangegangenen Lemma 4.1 weiss man, dass ein Pointer zu t gefunden wird, wenn t innerhalb von $L^s_{k+\gamma} \cup (L^s_{k+\gamma})^8$ bleibt. Bevor t seinen Level Pointer $LP^t_{k+\gamma}$ aus der betrachteten Fläche wegziehen kann, muss t sich nach Start des Lookup mindestens um $\Delta d' = \rho 2^k (2^\gamma (1 + \frac{\alpha}{2}) - 1)$ bewegt haben. Das dauert nach (1) mindestens $\Delta t \geq \Delta d' / v^{node}_{max} > \eta 2^{k+\gamma} \rho \sqrt{2}$. Darum könnte t frühestens einen $TFP^t_{k+\gamma}$ im betrachteten Gebiet anlegen, wenn nach Absenden des Lookup Request die Zeit Δt vergangen ist. Wie schon im Beweis zu Lemma 4.1 erwähnt wurde, ist ein Lookup Request mit $L_{k+\gamma}$ fertig, nach höchstens $\Delta t_{lookup} \leq \eta 2^{k+\gamma+1} \rho (\sqrt{2} + 8\sqrt{5})$. Also muss der $TFP^t_{k+\gamma}$ mindestens $\Delta t_{lookup} - \Delta t$ bestanden haben. Wegen (2) ist dies gegeben. □

Das vorangegangene Lemma zeigt, dass ein Lookup Request einen ersten Pointer zu t finden kann. Das nächste Lemma zeigt darüber hinaus, dass der Lookup Request auch einen weiteren Pointer finden kann, wenn der Request einem Forwarding Pointer oder Temporary Forwarding Pointer folgen musste. Jedoch muss dazu die Knotengeschwindigkeit diesmal auf

$$v^{node}_{max} < \frac{\alpha}{\eta \cdot 2(3\sqrt{2} + \alpha)} \tag{3}$$

eingeschränkt werden, damit dies in allen Fällen funktioniert.

Lemma 4.3 *Ein Lookup Request, der einen FP_i^t oder einen TFP_i^t gefunden hat, findet auch einen weiteren Pointer auf t im ersten Pointer referenzierten Quadrat $^*FP_i^t$ beziehungsweise $^*TFP_i^t$, sofern die maximale Knotengeschwindigkeit die Gleichung (3) erfüllt.*

Es wurde bis hierhin dafür gesorgt, dass ein Lookup Request Pointer zu t finden kann, und diesen auch folgen kann, um weitere zu finden. Im folgenden werden obere Schranken aufgestellt, für die Zeit, einen LP_i^t aufzusuchen nach dem Erreichen eines FP_i^t oder TFP_i^t. Dazu wird eine Schranke aufgezeigt, für die maximale Anzahl an Weiterleitungen um den LP_i^t zu erreichen. Es wird gezeigt, dass bei einem FP_i^t diese Anzahl der Weiterleitungen $\beta > 2$ beträgt, sofern die Knotengeschwindigkeit folgender Gleichung genügt:

$$v_{max}^{node} \leq \frac{\alpha(\beta - 2)}{2\eta(\sqrt{2} + \alpha + \beta\sqrt{8})} \tag{4}$$

Bei einem TFP_i^t wird gezeigt, dass $\beta_T > 2$ ist, wenn gilt:

$$v_{max}^{node} \leq \frac{\alpha 2^{i-1}\rho(\beta_T - 2)}{\eta 2^i \rho(\sqrt{2} + \alpha + \beta_T\sqrt{8}) + TTL_i} \tag{5}$$

Die Werte β und β_T werden zwei weitere Tuning-Parameter für MLS, die die Knotengeschwindigkeit und auch die Laufzeit des Lookup beeinflussen.

Für spätere Lemmata wird folgendes Hilfslemma benötigt. Es besagt:

Lemma 4.4 *Zwischen zwei aufeinanderfolgenden Updates eines LP_i^t muss sich ein Knoten t mindestens um $\Delta d_{update} \geq \alpha 2^{i-1}\rho$ fortbewegt haben.*

Lemma 4.5 *Wenn Gleichung (4) gilt, dann kann ein Lookup Request, der einen FP_i^t erreicht hat, zum zugehörigen LP_i^t weitergeleitet werden in höchstens $\Delta t \leq \beta\eta 2^i\rho\sqrt{8}$ für ein gegebenes $\beta > 2$.*

Lemma 4.6 *Wenn Gleichung (5) gilt, dann kann ein Lookup Request, der einen TFP_i^t erreicht hat, zum zugehörigen LP_i^t weitergeleitet werden in höchstens $\Delta t \leq \beta_T\eta 2^i\rho\sqrt{8}$ für ein gegebenes $\beta_T > 2$.*

Die Beweise dieser Lemmata fehlen hier, der Kürze halber. Die Autoren haben sie aber in ihrem Paper vollständig vorgeführt. Nun können die Lemmata zusammengefügt werden, um zu zeigen, dass ein Lookup Request einen ersten LP^t in beschränkter Zeit finden kann.

Lemma 4.7 *Ein Lookup Request von einem Knoten s zu einem Knoten t findet einen Level Pointer LP^t in $\mathcal{O}(d)$, wenn die Gleichungen (1), (2), (3), (4) und (5) erfüllt sind.*

Beweis. Aus Kombination der Lemmata 4.2, 4.5 und 4.6 erhalten wir folgendes: Ein erster Pointer zu t wird spätestens in einem der Quadrate von $L_u^s \cup (L_u^s)^8$ gefunden, mit $u = \lceil \log_2 \frac{d}{\rho} \rceil + \gamma$ (Lemma 4.2). Die nötige Zeit, um all diese Ebenen zu besuchen, ist begrenzt durch $T_1 \leq \eta 2^{u+1}\rho(\sqrt{2} + 8\sqrt{5})$. Wenn p ein FP_i^t ist, erreicht der Lookup

Request den zugehörigen LP_i^t in $T_2 \leq \beta\eta 2^i \rho\sqrt{8}$ (Lemma 4.5). Wenn p ein TFP_i^t ist, erreicht der Lookup Request den zugehörigen LP_i^t in $T_3 \leq \beta_T \eta 2^i \rho\sqrt{8}$ (Lemma 4.6). Für $i < u$ ist die Gesamtzeit T um einen ersten LP_i^t zu finden wie folgt beschränkt:

$$\begin{aligned} T &\leq T_1 + \max(T_2, T_3) \\ &\leq \eta 2^{\lceil log_2(d/\rho)\rceil+\gamma+1}\rho(\sqrt{2} + 8\sqrt{5} + \max(\beta, \beta_T)\sqrt{2}) \\ &\leq d \cdot \underbrace{\eta 2^{\gamma+2}(\sqrt{2} + 8\sqrt{5} + \max(\beta, \beta_T)\sqrt{2})}_{\text{konstant}} \\ &\in \mathcal{O}(d) \end{aligned}$$

4.1.2 Lookup – Phase 2

Für Phase 2 des Lookup muss gezeigt werden, dass ein Lookup Request, der einen ersten Pointer LP_i^t gefunden hat, den Pointern auch folgen kann bis zum Zielknoten t. Es wird wieder ein Hilfslemma angegeben, das später benutzt wird:

Lemma 4.8 *Solange $\delta_i^t < \alpha 2^i \rho$ ist, enthält das Quadrat ${}^*LP_{i+1}^t$ entweder einen LP_i^t oder einen FP_i^t.*

δ_i^t ist die Entfernung des Knotens t von seinem Quadrat. Durch Lazy Publishing kann δ_i^t größer Null sein.

Das nächste Lemma sagt aus, dass ein Lookup Request, der bei einem LP_i^t angekommen ist, an Hand dieses Zeigers weitergeleitet werden kann zu LP_{i-1}^t.

Lemma 4.9 *Unter der Bedingung, dass $i > 0$ ist und dass alle Einschränkungen von v_{max}^{node} und TTL_i eingehalten werden, gilt: Ein Lookup Request kann einem LP_{i+1}^t folgen zu einem LP_i^t nach $\Delta t \leq \eta 2^i \rho\sqrt{8}(1 + \max(\beta, \beta_T))$.*

Mit dem vorangehenden Lemma 4.9 kann gezeigt werden, dass man von LP_i^t zu LP_{i-1}^t weitergeleitet werden kann, bis man LP_1^t erreicht, von wo man in das kleinste Quadrat mit t gelangt, ${}^*LP_1^t$. Es bleibt zu zeigen, dass man in diesem Quadrat direkt zu t finden kann, unter der Bedingung, dass

$$v_{max}^{node} < \frac{\sqrt{2} - \alpha}{\eta(\alpha + 4\sqrt{2})} \tag{6}$$

Lemma 4.10 *Wenn die Gleichung (6) erfüllt ist, dann kann ein Lookup Request, der LP_1^t gefunden hat, direkt zu t in ${}^*LP_1^t$ gesandt werden.*

Satz 4.11 (Lookup) *Wenn die Gleichungen (1) bis (6) erfüllt sind, dann braucht ein Lookup Request von s nach t ein Zeit in $\mathcal{O}(d)$, um t zu erreichen.*

Beweis. Nach Lemma 4.7 weiß man, dass die Zeit, um einen ersten Level Pointer zu finden beschränkt ist von $T_1 \leq d \cdot \eta 2^{\gamma+2}(\sqrt{2} + 8\sqrt{5} + \max(\beta, \beta_T)\sqrt{2})$. Desweiteren weiß man, dass dieser erste Level Pointer auf einem Level nicht größer als $\lceil \log_2 \frac{d}{\rho} \rceil + \gamma$ gefunden wird. Aus Lemma 4.9 leitet man ab, dass das Folgen der Pointer von Ebene $\lceil \log_2 \frac{d}{\rho} \rceil + \gamma$ bis hinunter zu Ebene 1 so lange dauert:

$$T_2 \leq \sum_{j=1}^{\lceil \log_2 d/\rho \rceil + \gamma} \eta 2^j \rho \sqrt{8}(1 + \max(\beta, \beta_T))$$

und die Zeit um von LP_1^t zu t zu gelangen wird mit Lemma 4.10 (der Beweis ist wiederum der Kürze halber ausgelassen) beschränkt auf:

$$T_3 \leq \eta \rho \sqrt{8}(1 + \max(\beta, \beta_T)) \qquad \square$$

Damit ist die Gesamtzeit für einen Lookup Request beschränkt von:

$$\begin{aligned}
T_{lookup} &\leq T_1 + T_2 + T_3) \\
&\leq T_1 + \sum_{j=1}^{\lceil \log_2 d/\rho \rceil + \gamma} \eta 2^j \rho \sqrt{8}(1 + \max(\beta, \beta_T)) \\
&\leq T_1 + d\eta 2^{\gamma+2} \sqrt{8}(1 + \max(\beta, \beta_T)) \\
&\leq d \cdot \underbrace{\eta 2^{\gamma+2}(3\sqrt{2}(1 + \max(\beta, \beta_T)) + 8\sqrt{5})}_{\text{konstant}} \\
&\in \mathcal{O}(d)
\end{aligned}$$

4.2 Maximale Knotengeschwindigkeit

Für die vorangegangenen Lemmata wurden verschiedene Anforderungen an v_{max}^{node} und TTL_i gestellt, die wie angekündigt nun zusammengefasst werden sollen, zu erschöpfenden Endergebnissen, mit denen MLS dann bewiesener Weise funktioniert.

Der Wert von TTL_i muss nur die Gleichung (2) erfüllen. Weil jede Erhöhung von TTL_i eine Senkung der maximalen Knotengeschwindigkeit nach (5) nach sich zieht, wird der kleinstmögliche Wert übernommen:

$$TTL_i = \eta 2^{i+1} \rho (1/\sqrt{2} + 8\sqrt{5}) \tag{7}$$

Um die maximale v_{max}^{node} herauszufinden, müssen die MLS-Parameter $0 < \alpha < 1, \beta > 2, \gamma \geq 1$ und $\beta_T > 1$ festgemacht werden. Der Wert von η ergibt sich aus der Topologie der Landschaft und der Qualität des gewählten unterliegenden Georoutings. Während v_{max}^{node} maximiert werden soll, sollte möglichst T_{lookup} minimiert werden, die maximale Zeit für einen Lookup Request, abhängig von den selben Parametern.

Für $\gamma \to \infty; \beta \to \infty; \beta_T \to \infty$ erhält man $v_{max}^{node} \approx \frac{0.0845}{\eta}$ mit $\alpha \approx 0.863$. Diese obere Schranke ist unerreichbar, denn die Laufzeit eines Lookup Request würde gegen unendlich gehen. Offensichtlich muss man abwägen zwischen der Maximierung der Knotengeschwindigkeit und der Minimierung der Lookup-Zeit. Die Autoren schreiben, dass sie durch Spielen mit den Parametern folgende guten Werte gefunden haben: $\gamma = 1, \beta = 5, \beta_T = 19$. Die Geschwindigkeit v_{max}^{node} ist damit am höchsten für $\alpha = 0.8$. Damit kann nun auch für die Maximale Knotengeschwindigkeit eine untere Schranke angegeben werden:

Satz 4.12 *Für $\alpha = 0.8, \gamma = 1, \beta = 5$ und $\beta_T = 19$ können die Netzwerkknoten sich mit maximal $v_{max}^{node} \leq \frac{1}{15 \cdot \eta}$ bewegen, so dass MLS noch funktioniert.*

Beweis. Durch Einsetzen der konkreten Parameter in die Gleichungen (1), (3), (4), (5) und (6). □

4.3 Kommunikationskosten bei Positionsänderungen

Als letzter Teil des Analysekapitels müssen noch die Kosten des Publish-Algorithmus ermittelt werden. Zuerst werden die ungüstigsten Kosten die Kosten des Publishing zu einer einzigen Ebene berechnet, danach werden die gesammelten Kosten des Publishing ermittelt, für einen Knoten der sich bewegt.

Die bisherigen Routingkosten des unterliegenden Georouting waren bisher als die Zeit $\eta \cdot d$ betrachtet, für eine Strecke der Länge d. Für die folgenden Lemmata betrachten wir diskrete Routingkosten in Anzahl von Hops, die für die Überbrückung der Entfernung d nötig sind. Wir nehmen an, dass die Anzahl der Hops proportional zur Entfernung ist, und schreiben dann $\tilde{\eta} \cdot d$ für die Anzahl der Hops, die das Georouting auf der Strecke d machen muss.

Lemma 4.13 *Die Kosten für das Publishing der Ebene i sind beschränkt von $\tilde{\eta} 2^{i+2} \rho(\sqrt{8} + \alpha)$ Hops.*

Daraus folgt für die gesammelten Kosten des Nachrichtenaufkommens für das Publishing (wieder ohne Beweis):

Satz 4.14 (Publish) *Die gesammelten Kosten des Publishing eines Knotens sind in $\mathcal{O}(d \log d)$ Nachrichten-Hops.*

Damit wurde durch die Analyse gezeigt, dass bei richtiger Wahl der Knotengeschwindigkeit und Lebensdauer der Pointer alle Wettlaufsituation in MLS verhindert werden, und MLS funktoniert. Genauso wurde die lineare Routingdilation mit $\mathcal{O}(d)$ gezeigt, und der maximale Aufwand für das Publishing in $\mathcal{O}(d \log d)$ eingeschränkt.

5 Simulation

Die Autoren von MLS haben eine Reihe von Simulationen durchgeführt, die zeigen, dass die Größen für die *durchschnittliche* Zeiten für Lookup und Publishing weit unter den oberen Grenzen liegen. Dazu wurde ein Simulationsframework entwickelt, zu dem noch folgende Besonderheiten zu erwähnen sind.

Vom unterliegenden Knoten-zu-Knoten-Routing wird der Einfachheit halber abstrahiert, es geht lediglich durch seine Verzögerung mit in die Simulation ein.

Die minimale Knotendichte wird nicht simuliert, das Framework speichert die Pointer intern und garantiert nicht, dass sich ein Knoten immer in der Nähe dieser Positionen befindet. So können zur Leistungssteigerung der Simulation auch nur Untermengen aller Knoten simuliert werden.

Für bessere Ergebnisse wurde der Lookup in alternativen Läufen parallelisiert, so dass anstatt der umkreisenden Suche durch die $(L_i)^8$) *nacheinander* gleich alle 9 Quadrate gleichzeitig den Lookup Request zugesandt bekommen. Diese Messungen mit dem „Fast Lookup“ sind in Abbildung 5.1 und Abbildung 5.2 als Linien mit runden Punkten zu sehen.

Die Simulation wurde mit 5000 Knoten laufengelassen, die insgesamt 10.000 Lookup Requests versandt haben. Dies wurde je für zwei maximale Knotengeschwindigkeiten und zwei verschiedene Landschaften durchgeführt. Die Landschaft$_1$ ist in Abbildung 2.1 zu sehen. Die Seitenlänge der Karte waren 5300 Längeneinheiten, ρ wurde als 1 LE gewählt. Die maximale Routingdilation (wegen der Seen) war 10.

Landschaft$_2$ enthielt kein Wasser, war aber gleich groß. Die Routingdilation ist in Abbildung 5.1 zu sehen.

Bei höherer Knotengeschwindigkeit produzieren die beweglichen Knoten mehr TFP, was dem Lookup Request hilft. Man sieht das an Abbildung 5.2, nämlich dass bei höherer Knotengeschwindigkeit mehr Zeit für das Verfolgen von FP und TFP genutzt wird.

Der geänderte Fast Lookup Algorithmus reduziert die Lookup-Zeit um die Hälfte, produziert jedoch einen zusätzlichen Nachrichtenoverhead, und es können Nachrichten mehrmals am Ziel ankommen. Identische Nachrichten müssen also noch erkannt und verworfen werden.

Mit diesen Änderungen an MLS kommt man auf auf eine Lookup-Dilation von etwa 6.

Für Publish Requests wurde eine durchschnittliche Konstante von 4,3 für das O-Kalkül erkannt, also ein Publish Overhead von $4,3 \cdot d \log d$, mit d als der zurückgelegten Strecke. Die Autoren waren mit diesen Werten laut eigener Aussage zufrieden, so dass ihr Netzwerk nicht gesättigt sei, mit Publish Requests von bewegenden Knoten.

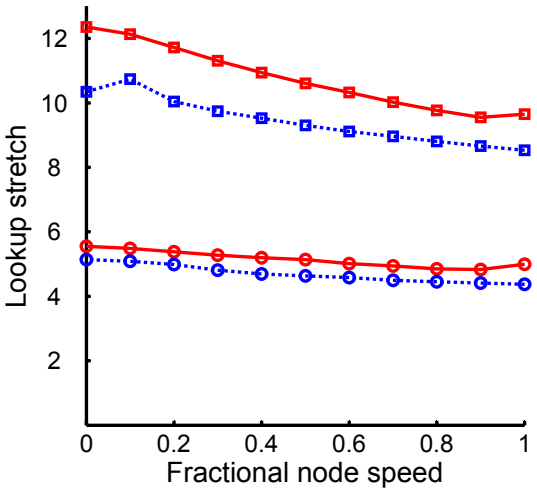

Abbildung 5.1: Lookup-Dilation. Durchgezogene Linien stammen von Landschaft$_1$, gepunktete Linien von Landschaft$_2$. Die Geschwindigkeit v_{max}^{node} liegt am rechten Ende der Skala, bei 1. Linien mit runden Punkten sind vom sog. Fast Lookup.

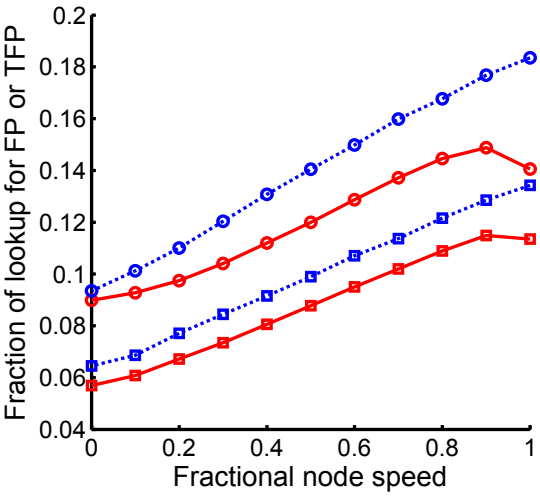

Abbildung 5.2: Anteil am Lookup, der für das Verfolgen von FP und TFP anfällt. Durchgezogene Linien stammen von Landschaft$_1$, gepunktete Linien von Landschaft$_2$. Geschwindigkeit v_{max}^{node} liegt bei 1. Linien mit runden Punkten sind vom sog. Fast Lookup.

6 Schluss

6.1 Schlussfolgerungen

MLS erlaubt also das Senden von Nachrichten an Knoten deren Position der Sender nicht kennt. Darüber hinaus dürfen alle Knoten mobil sein, und selbst der Zielknoten darf sich fortbewegen, während sich eine Nachricht auf dem Weg zu ihm befindet. Dazu musste die maximale Knotengeschwindigkeit beschränkt werden. Desweiteren wurde die Güte des Routings und der Kommunikationsaufwand nach einer Knotenbewegung berechnet.

Das alles leistet MLS unter einigen Voraussetzungen wie hoher Knotendichte, verlässlicher Kommunikation und der Kenntnis aller Seen durch jeden Knoten.

6.2 Offene Fragen

Offene Fragen bleiben zum Beispiel im Feld bidirektionaler Kommunikation bestehen, wo Knoten Positionsinformationen vom Sender oder dem Hinweg der Zustellung für die beschleunigte Zustellung einer Antwort nutzen könnten.

Es verbleiben außerdem die angesprochenen starken Modellannahmen (Knotendichte, verlässliche Kommunikation und Kenntnis der Seen), an denen die Autoren nach eigener Aussage in Zukunft noch arbeiten wollen.

Literaturverzeichnis

[FW06] Roland Flury, Roger Wattenhofer, *MLS: An Efficient Location Service for Mobile Ad Hoc Networks.* MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing, pages 226–237, 2006. 1

[SBGP04] A. Smith, H. Balakrishnan, M. Goraczko, and N. B. Priyantha. *Tracking Moving Devices with the Cricket Location System.* In MobiSys, 2004. 2.1

[BMSU99] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. *Routing With Guaranteed Delivery in Ad Hoc Wireless Networks.* In DIAL-M, pages 48–55, 1999. 2.2

[EFK04] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. *Scalable Ad Hoc Routing: The Case for Dynamic Addressing.* In INFOCOM, 2004. 2.2

[RPSS03] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. *Geographic Routing Without Location Information.* In MOBICOM, pages 96–108, 2003. 2.2

[VAF+05] A. C. Viana, M. D. de Amorim, S. Fdida, Y. Viniotis, and J. F. de Rezende. *Easily-Managed and Topology-Independent Location Service for Self-Organizing Networks.* In MobiHoc, 2005. 2.2

[WS01] S.-C. M. Woo and S. Singh. *Scalable Routing Protocol for Ad Hoc Networks.* Wireless Networks, 7(5):513–529, 2001. 2.2