Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Lehrstuhl für Rechnernetze und Telematik

SS 2009

Seminararbeit

P2P-Ansatz mit konstantem Lookup-Overhead

Grundlage:
LAND: Stretch $(1+\epsilon)$ Locality-Aware Networks for DHTs
(Ittai Abraham, Dahlia Malkhi, Oren Dobzinski)

Thomas Stocker

23. Juli 2009

Betreut durch Prof. Dr. Christian Schindelhauer

Inhaltsverzeichnis

1 Einleitung

Seit den ersten Entwicklungen von Peer-to-Peer (P2P) Netzwerken hat sich der Ansatz verteilter Speicherung von Daten im Hinblick auf das beständig wachsende Datenvolumen im Internet als vorteilhaft erwiesen. Die Netzlast (Datenaufkommen, Lookup-Anfragen) wird abhängig vom verwendeten Ansatz auf mehrere oder gar alle beteiligten Peers verteilt und damit der für Client-Server-Architekturen typische Flaschenhals am Server umgangen. Da P2P-Netzwerke kein Wissen über die tatsächlich zugrundeliegende Netzwerkstruktur haben und deshalb ein logisches Overlay-Netzwerk bilden [2], ergeben sich zwangsweise suboptimale Routen bei Lookup-Vorgängen. In vielen P2P-Netzwerken wird die Performance eines Lookup-Vorgangs in Form der benötigten Zwischenschritte (Hops) gemessen, wobei dieses Maß meistens logarithmisch in der Anzahl der Peers ist [7] [6] [8]. Allerdings ist eine Hopcount-Metrik wenig aussagekräftig, was die tatsächliche Distanz zwischen Knoten in einem Netzwerk betrifft. Knoten, die im Overlay-Netzwerk nahe beieinander liegen, können im Underlay-Netzwerk sehr weit voneinander entfernt sein. Es ist also möglich, dass ein Lookup zwar lediglich 5 Hops benötigt, die beteiligten Peers sich jedoch alle auf verschiedenen Kontinenten befinden. Die Ursache für dieses Problem liegt im Aufbau des Overlay-Netzwerks, was beim Erstellen von Verbindungen zwischen Knoten typischerweise deren tatsächliche Distanz nicht mit berücksichtigt. Ein weitaus aussagekräftigeres Maß ist der sog. *stretch*, der anhand einer Kostenfunktion die tatsächliche Entfernung zwischen zwei Knoten zu den Kosten der Verbindungsroute ins Verhältnis setzt. Dieser Ansatz wurde schon 1997 von Plaxton, Rajamaran und Richa in [4] verfolgt und in erfolgreichen P2P-Netzwerken wie Pastry [6] und Tapestry [8] verwendet und weiterentwickelt. Der *stretch* ergab sich dabei entwerder als verhältnismäßig große Konstante oder verhielt sich proportional zum Durchmesser des Netzwerkes [1]. Das Ziel von Abraham, Malkhi und Dobzinski ist es, aufbauend auf o.g. Arbeiten in einem P2P-Netzwerk einen konstanten *stretch* von $1 + \epsilon$ zu garantieren. Dabei wird eine wachstumsorientierte Metrik verwendet, die gewünschte Eigenschaften des Underlay-Netzwerkes in Bezug auf dessen Struktur und Ausdehnungsdynamik beschreibt. Diese Metrik ist zugleich die zetrale Annahme, die getroffen wird, um das Ziel eines konstanten *stretch* zu erreichen.

2 Architektur

Die **LAND**-Architektur baut auf dem *Distributed Hash Table* (DHT) Ansatz auf, bei dem Netzwerk-Knoten und Objekten gleichermaßen eindeutige IDs zugeordnet werden und Lookup-Anfragen durch das Netzwerk zu dem Knoten geroutet werden, der entweder das Objekt selbst speichert, oder den Speicherort kennt. Dabei haben alle Knoten partielles Wissen über die Speicherorte von Objekten. Dieses Wissen wird innerhalb des Routing-Algorithmus kombiniert, um Lookup-Anfragen zu beantworten. Eine Möglichkeit Lookup-Anfragen auszuführen ist das sog. *prefix routing* [4], welches auch in **LAND** verwendet wird. Ergänzt wird dieses Schema in **LAND** um einen Mechanismus, der Netzwerk-Verbindungen (Links) ausschließlich innerhalb einer adäquaten Distanz wählt und dadurch den *stretch* verringert.

Prefix routing. Das von Plaxton u.A. in [4] vorgestellte *prefix routing* bestimmt von einem zu suchenden Objekt *obj* zunächst einen Hash-Wert $\mathcal{H}(obj)$. Ausgehend von einem Startknoten v_1 wird dann in jedem Schritt eine Stelle $\mathcal{H}(obj)$ adjustiert, indem für das i-te Sprungziel ein Nachbarknoten von v_i gewählt wird, der genau diese Stelle adäquat belegt. Am Endknoten v_n angekommen, kann dort die Information über den Speicherort des Objektes gewonnen werden. Diese sukzessive Annäherung an die Zielknoten-ID erfüllt eine Lookup-Anfrage in $O(log(n))$ Schritten. Voraussetzung dafür ist, dass jeder Knoten eine Liste verwaltet, die IDs von Nachbarknoten enthält, um die Adjustierung vornehmen zu können. Im Einzelfall kann es dabei zu nicht existenten Linkzielen kommen, was eine Strategie zur Fehlerbehandlung voraussetzt. Im Fall von **Tapestry** wird *surrogate routing* [8] dafür eingesetzt, die Autoren von **LAND** bauen auf das *shadow router* Konstrukt, bei dem fehlende Links durch die eigene Instanz emuliert werden, um Lookup-Anfragen nicht ins Leere laufen zu lassen. Das hier verwendete Prinzip ist stark angelehnt an Zhao, Kubiatwicz und Josephs **Tapestry** [8].

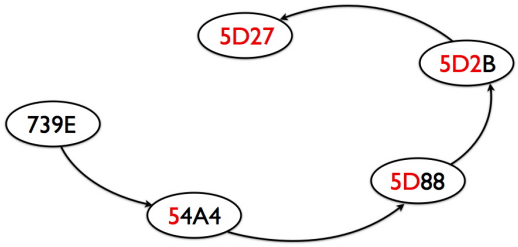

Abbildung 1: Lookup nach Knoten-ID 5D27.

Prinzipiell geht der Ansatz von **LAND** davon aus, dass für ein beliebiges gewünschtes ϵ ein Netzwerk so konstruiert werden kann, dass ein *stretch* von $(1+\epsilon)$ gewährleistet wird. D.h. die Kosten einer Route zwischen zwei Knoten ergeben sich zu $(1+\epsilon)\cdot c(v_1, v_2), v_1, v_2 \in V$. Die Rahmenbedingungen verändern sich mit dem gewählten ϵ, sodass für jedes ϵ ein Szenario entsteht, das alle Anforderungen erfüllt, um den $(1+\epsilon)$-*stretch* zu garantieren. Wie realistisch diese Szenarien zu beurteilen sind, wird in Kapitel 4 diskutiert.

2.1 Grundlagen

Das Netzwerk besteht aus einer Menge von Knoten V ($|V| = n$). Die dem *stretch* zugrundeliegende Kostenfunktion wird als $c : V^2 \mapsto \mathbb{R}_+$ definiert, wobei vorausgesetzt wird, dass sie stets positiv und reflexiv ist und die Dreiecksungleichung erfüllt. Wie in [3] wird angenommen, dass die Struktur des Underlay-Netzwerkes einer gewissen Regelmäßigkeit unterworfen ist. Betrachtet wird dabei die Anzahl der Knoten, die sich innerhalb eines Kreises mit Radius r um einen bestimmten Knoten v befinden: $N(v, r) = \{w \in V | c(v, w) \leq r\}$. Bei einer Verdopplung des Radius soll die Knotenanzahl höchstens um einen konstanten Ausdehnungsfaktor $\triangle$ steigen. Es ergibt sich also: $|N(v, 2 \cdot r)| \leq \triangle |N(v, r)|$. Im Unterschied zu [4] wird auf die dort verwendete untere Schranke zugunsten eines allgemeingültigeren Ansatzes verzichtet.

Vergebene IDs haben eine Basis $B = 2^b$, wobei $B \geq \triangle^2$ gelten muss und die Länge M mit $B^M = n$. Weiterhin wird eine Konstante α gewählt, sodass $\alpha > ln(B)$. $A_i(x)$ bezeichnet den kleinsten Kreis um x mit Radius $a_i(x)$ mit $\min\{\alpha \cdot B^i, n\}$ Knoten. Für $r = a_i(x)$ gilt $A_i(x) = N(x, r)$. Mithilfe von α und A_i werden die Bereiche definiert, innerhalb derer naheliegende Knoten ausgewählt werden, um die Lokalitäts-Eigenschaft von **LAND** zu realisieren.

2.2 Netzwerkstruktur

Jeder Knoten v verwaltet $M + 1$ Router $R_i, i \in \{1, .., M + 1\}$ die jeweils eine eigene ID und ein Level i besitzen. Router verfügen über eine Anzahl von Links zu Routern anderer (benachbarter) Knoten und unterscheiden dabei zwischen *neighbor*-Links, die für das *prefix routing* verwendet werden und *publish*-Links, die eine Menge von Knoten referenzieren, denen während einer Publish-Operation (siehe Abschnitt 2.3) eingegangene Informationen über den Speicherort eines bestimmten Objekts weitergereicht werden (Shortcut-Funktion). Ein Router mit Level $\ell \leq M$ hat genau B *neighbor*-Links $L(0), ..., L(B - 1)$ zu Knoten, die im Bereich $A_\ell(v)$ liegen. Für jeden Link $L(i)$ teilt der Zielknoten die ersten $\ell - 1$ Stellen der Router ID und adjustiert die ℓ-te Stelle auf i.

Falls in diesem Bereich kein Router gefunden werden kann, der diese Eigenschaft besitzt, wird stattdessen ein *shadow router* generiert, der von v verwaltet wird, inclusive dessen *neighbor*- und *publish*-Links. Dieses Verfahren wird bei Bedarf rekursiv fortgesetzt. Der Level eines Routers bestimmt also die Länge des gemeinsamen Präfixes mit dem verbundenen Router. Die *publish*-Links eines Routers auf Level ℓ formen sich aus *allen* Knoten innerhalb des Bereichs $A_{\ell+d+5}$, die einen Router auf Level $\ell + 1$ beinhalten, der die ersten $\ell - 1$ Stellen seiner ID teilt. Die Konstante d orientiert sich an ϵ und liegt in der Größenordnung $O(log(\frac{1}{\epsilon}))$.

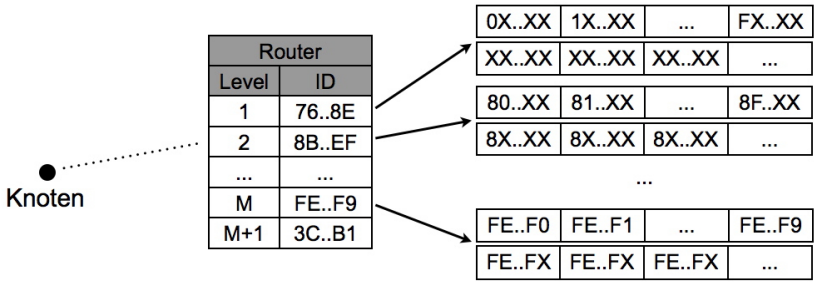

Abbildung 2: Knoten mit zugehörigen Routern und Links.

Diese Vorgehensweise erinnert stark an **Tapestry** [8]. Dort werden ebenfalls Links in Level eingeteilt und benachbarte Knoten für das *prefix routing* in das *neighborhood set* gewählt. Die *links* in **Tapestry** erhöhen wie die *publish*-Links in **LAND** die Konnektivität des Netzwerks und damit auch dessen Robustheit. Allerdings wird in Tapestry nicht zwischen Knoten und Routern unterschieden. Während bei **LAND** ein Knoten $log_B(n)$ Router mit jeweils einem Level hostet, existiert bei Tapestry pro Knoten nur eine ID, jedoch mit Linklisten für alle Level. Insgesamt dürfte der Grad der Knoten in **LAND** deutlich höher sein, denn im Gegensatz zu Tapestry wird die Anzahl der *publish*-Links nicht nach oben beschränkt und zudem für jedes Level ein separater *publish*-Link Pool gehalten.

2.3 Publish und Lookup

Publish. Beim Hinzufügen von neuen Ressourcen zu einem bestehenden Netzwerk, muss ausgehend vom publizierenden Knoten v eine Publish-Operation ausgeführt werden, um anderen Knoten diese Information mitzuteilen. Für das einzufügende Objekt *obj* wird zunächst ein Hashwert $\mathcal{H}(obj)$ berechnet. Anhand dieses Hashwertes werden enstprechende

Links auf der Publish-Route gesetzt, die es später ermöglichen, innerhalb eines Lookups den Publish-Pfad rückwärts nachzulaufen und so zum Speicherort zu gelangen. Dabei wird in Schritt i die i-te Stelle des Hashwertes fixiert, indem der passende *neighbor*-Link eines Level-i Routers gewählt wird, der zu einem Router mit Level $i+1$ führt, der dieses Präfix teilt. Diese Operation wird solange ausgeführt, bis alle Stellen des Hashwertes fixiert sind und keine weiteren Links mehr nachgelaufen werden können. Entlang dieser *publish*-Route speichert jeder beteiligte Knoten eine Referenz zu *obj* und zum Vorgängerknoten. Außerdem gibt er diese Referenzen an alle Router weiter, die er über seine *publish*-Links erreicht.

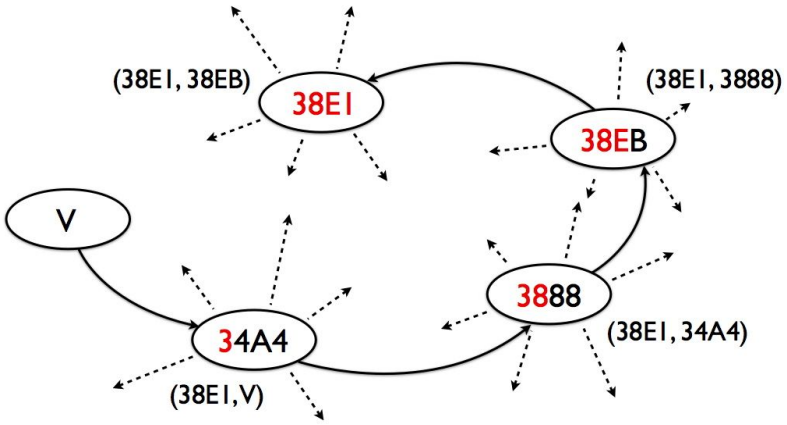

Abbildung 3: Knoten V gibt die Freigabe von Objekt 38E1 bekannt.

Lookup. Eine Lookup-Operation versucht einen möglichst naheliegenden Knoten zu finden, der das zu suchendende Objekt speichert und kann von jedem Knoten im Netzwerk aus gestartet werden. Generell wird das Standard-*prefix routing* durchgeführt, indem Stelle für Stelle von $\mathcal{H}(obj)$ durch wiederholtes Springen zu passenden Routern gefixt wird. Enthält unterwegs ein Router eine Referenz auf das Objekt, wird stattdessen diese Referenz in Richtung Ziel nachgelaufen. Spätestens der Router R mit $R.ID = \mathcal{H}(obj)$ hat eine Referenz, die genutzt werden kann, um den Speicherort zu finden.

2.4 Änderungsdynamik

P2P-Netzwerke unterliegen typischerweise einer starken Änderungsdynamik. Peers treten dem Netzwerk spontan bei oder verlassen es ohne Vorwarnung. In jedem Fall müssen

Links gesetzt oder geändert werden, um die Funktionalität des Systems nicht zu gefährden.

2.4.1 Einfügen

Soll ein neuer Knoten v zum Netzwerk hinzugefügt werden, werden zunächst die IDs der initialen Router auf zufällige Werte gesetzt, wie in Abschnitt 2.1 beschrieben. Im zweiten Schritt müssen deren *neighbor* und *publish*-Links angelegt werden. Die Schwierigkeit dabei ist, für einen Router R_ℓ mit Level ℓ geeignete $R_{\ell+1}$ im näheren Umfeld von v zu finden, genauer in $A_\ell(v)$ und $A_{\ell+d+5}(v)$. Zusätzlich müssen alle $R_{\ell-1}$ über die Präsenz von v informiert werden, um ihre Links zu aktualisieren.

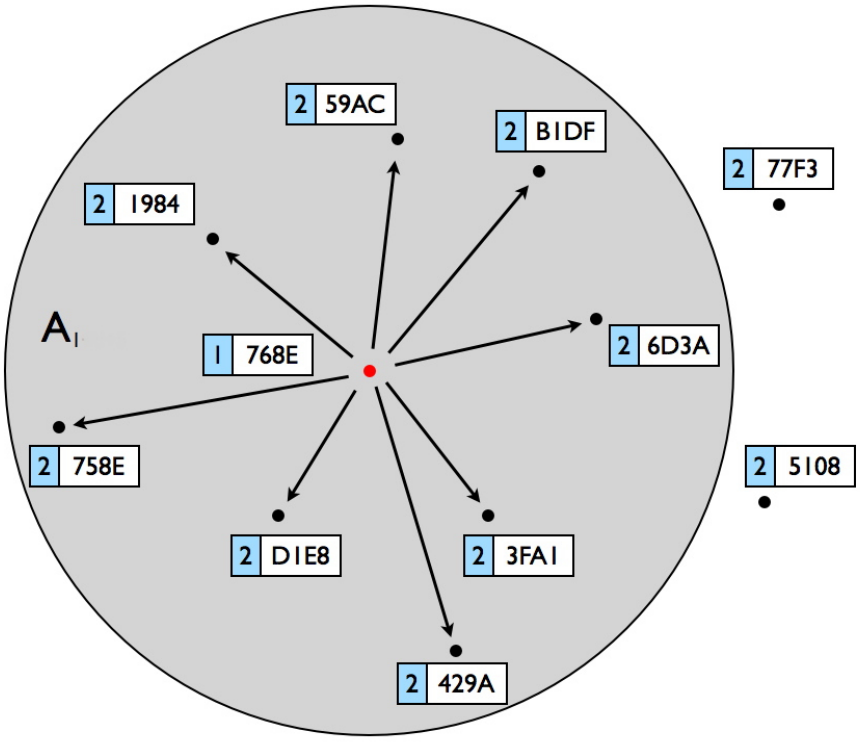

Abbildung 4: Beispiel für die *publish*-Link-Generierung von Router R_1

Algorithm 1 Find-Closest(v, ℓ) at router $u.R_\ell$

1: **if** $\ell = 1$ **then**
2: send u to v and return
3: **end if**
4: $S_i \leftarrow$ all incoming links to $u.R_\ell$
5: $w \leftarrow$ closest node in S_i
6: FIND-CLOSEST(v,$\ell - 1$)

Die Idee ist, den v am nächsten liegenden Knoten s zu benutzen, um eine ausreichend große Menge von Routern zu finden, die potentiell als Linkziele geeignet sind. Dafür muss s jedoch zuerst bestimmt werden. Beginnend bei einem beliebigen bekannten Knoten im Netzwerk u wird dazu der Algorithmus 1 angestoßen[1] .

Gestartet wird der Algorithmus mit Level $\ell = M + 1$ und einem beliebigen bekannten Knoten u. In jedem Schritt wird genau der Router w aus allen eingehenden Links von $u.R_\ell$ gewählt, der dem neuen Knoten v am nächsten ist. Dabei wird angenommen, dass die Knoten ihre tatsächliche Distanz $c(v_1, v_2)$ berechnen können, während sie kommunizieren. Bei $\ell = 1$ angekommen endet der Algorithmus und gibt den aktuell nächsten Knoten zurück. Der Algorithmus findet in jedem Fall den naheliegendste Knoten (Beweis siehe [1]). Für diese Aufgabenstellung wäre es eventuell auch möglich Lösungsansätze für das *Closest-Pair*-Problem in Betracht zu ziehen.

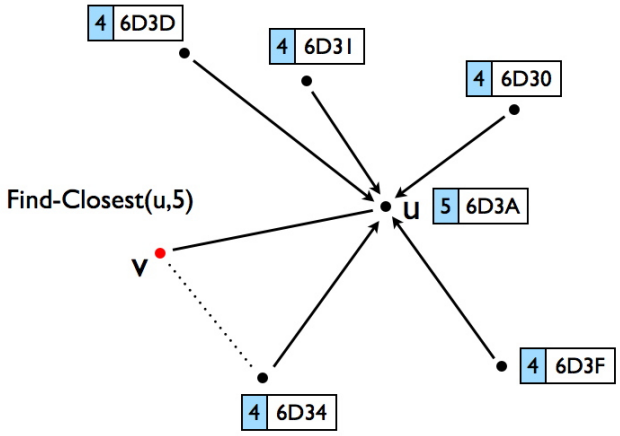

Abbildung 5: Initialer Aufruf von Find-Closest mit dem v bekannten Knoten u

Wenn der v naheliegendste Knoten s gefunden wurde, werden für jeden Router $v.R_\ell$ mittels Lookup-Operationen durch s Knoten gesucht, die auf Level $\ell + 2$ liegen und mit R_ℓ ein Präfix der Länge $\ell - 1$ teilen. Um die so erhaltene Knotenmenge Y zu vergrößern, werden in einem nächsten Schritt alle Router gewählt, die *publish*-Link-Ziele von Routern $y \in Y$ sind. In der Hoffnung mithilfe dieser Menge $S(\ell + 3)$ Router zu finden, die näher an v liegen und auf Level $\ell+1$ liegen, wird wiederholt aus der gegebenen Menge $S(i)$ eine Menge $S(i-1)$ generiert, indem die Quell-Router eingehender *publish*-Links auf Router in $S(i)$ in die neue Menge $S(i-1)$ übernommen werden. Aus $S(\ell+1)$ werden dann Router als Ziele für *neighbor*-Links gewählt, sofern sie im Bereich A_ℓ liegen oder für *publish*-Links, wenn v aus Sicht des betreffenden Routers in $A_{\ell+d+5}$ liegt. Kommen mehrere Kandidaten für einen *neighbor*-Link $L(i)$ in Frage, so wird stets der Knoten w_i gewählt der $c(v, w_i)$ minimiert. Es wird angenommen, dass $c(v, w_i)$ berechnet werden kann, während die beiden Router kommunizieren. Danach werden alle Router in $S(\ell-1)$ über die Präsenz von v informiert, damit sie ggf. ihre Links aktualisieren können. Interessant wäre an dieser Stelle gewesen, ab welcher Knotenzahl sich die Anzahl der *shadow router* auf ein sinnvolles Maß reduziert und wie der Netzaufbau von Null an von statten geht, bzw. welche Probleme auftreten, wenn ein neuer Knoten keine Knoten im Netzwerk kennt und wie damit umgegangen wird.

2.4.2 Entfernen

Verlässt ein Knoten das Netzwerk, müssen die Links aller verlinkender Router erneuert werden. Dabei wird anstatt des zuvor verlinkten ein Alternativrouter aus dem Pool von *publish*-Links gewählt, oder bei Fehlschlagen dieses Unterfangens ein *shadow router* emuliert. In diesem Fall müssen dessen Links wie in Abschnitt 2.4.1 beschrieben gesetzt werden. Offen bleibt die Frage, wie ein Router das Fehlen eines Linkzieles bemerkt. Wird ein Verlassen des Netzwerks von verlinkenden Knoten/Routern nicht bemerkt, kann kein Update vorgenommen werden, was zwangsläufig zu Routing-Probemen führt. An dieser Stelle sollte also ein Erkennungs-Mechanismus eingeführt werden, der beispielsweise aufgrund fehlender ACKs auf die Abwesenheit eines Knotens schließt und dann entsprechende Maßnahmen einleitet.

3 Analyse

Um die Ausführung möglichst kurz zu halten, wird an dieser Stelle der Beweis für den konstanten $1+\epsilon$-*stretch* nur grob umrissen, Einzelheiten finden sich im Originalpaper [1].

Betrachtet wird eine Route von Startknoten s zum Zielknoten t wie folgt:
$s = x_1, x_2, x_3, ..., x_k, w_{k-1}, w_{k-2}, ..., w_1 = t$ wobei für die relevanten Router gilt $x_i.r.level = i$. Im ersten Teil der Route wird ein *prefix routing* nach dem Hashwert des Objektes $\mathcal{H}(obj)$ ausgeführt, sodass $x_i.r.id[i-1] = \mathcal{H}(obj)[i-1]$. Ab dem Router w_{k-1} sind Referenzen vorhanden, die nachgelaufen werden können, bis mit $w_1 = t$ der Speicherort erreicht wird. Dabei kann es sein, dass aufeinanderfolgende Knoten identisch sind,d.h. *shadow router* hosten, die am Routing beteiligt sind.

Zuerst wird bewiesen, dass $\frac{\gamma}{\gamma-1} \cdot a_{i+1}(s)$ eine obere Schranke für die Kosten der Route $s = x_1, ..., x_i$ ist, wobei diese Schranke erneut mit $\frac{2 \cdot \gamma^{1-d}}{\gamma-1} \cdot c(s,t)$ abgeschätzt wird. Dabei gilt $\gamma = B^{log_{\triangle} 2}$. Die Kosten des Hops von x_k zu w_{k-1} werden mithilfe der Dreiecksungleichung abgeschätzt: $c(x_k, w_{k-1}) \leq a_{k+1}(s) + c(s,t) + a_k(t) \leq (2 \cdot \gamma^{-d} + 1 + \gamma^{-d-1}) \cdot c(s,t)$. Essentiell dafür ist der Sachverhalt $x_i \in A_i(x_{i-1}) \subseteq A_{i+1}(s)$, der in [1] bewiesen wird. Die Entfernung von x_k zu s kann also mit dem Radius von $A_{k+1}(s)$ abgeschätzt werden, analog für $A_k(t)$. In einem dritten Schritt wird bewiesen, dass die Kosten der Route $w_{k-1}, ..., w_1 = t$ mit $\frac{\gamma^{-d}}{\gamma-1} \cdot c(s,t)$ abgeschätzt werden können. Die kumulierten Kosten der vollständigen Route betragen also
$(1 + (\frac{3}{\gamma-1} + \frac{1}{\gamma} + 4) \cdot \gamma^{-d}) \cdot c(s,t)$.

Mit einer adäquaten Wahl von $d = O(log(\frac{1}{\epsilon}))$ kann die Behauptung bewiesen werden. Genauer muss gelten: $\epsilon \geq \frac{1}{\gamma^d} \cdot (\frac{2 \cdot \gamma}{\gamma-1} + 2 + \frac{1}{\gamma} + \frac{1}{\gamma-1}) = (\frac{3}{\gamma-1} + \frac{1}{\gamma} + 4) \cdot \gamma^{-d}$.

Ausgehend von den Konstanten $\triangle$, $B \geq \triangle^2$, $\alpha > ln(B)$ und d kann man die erwartete Anzahl von Links eines Knotens berechnen. Für M gilt $M = log_B(n)$. Die Anzahl der *neighbor*-Links ergibt sich zu $(M+1) \cdot B = B + B \cdot log_B(n)$. Die Anzahl der *publish*-Links ist schwieriger zu bestimmen. Für jeden Router auf Level ℓ werden alle Router mit Level $\ell + 1$ mit dem gleichen $(\ell - 1)$-Präfix von Knoten aus dem Bereich $A_{\ell+d+5}$ gewählt. Die Wahrscheinlichkeit, dass ein Knoten einen solchen Router hostet, ist $\frac{1}{B^{\ell-1} \cdot (1 - B \cdot e^{-\alpha})}$ [1]. Die erwartete Anzahl von *publish*-Links P ergibt sich also zu:
$|P| = \sum_{\ell=1}^{M} \frac{|A_{\ell+d+5}|}{B^{\ell-1} \cdot (1 - B \cdot e^{-\alpha})} = M \cdot \frac{|A_{\ell+d+5}|}{B^{\ell-1} \cdot (1 - B \cdot e^{-\alpha})} = M \cdot \frac{\alpha \cdot B^{d+6}}{1 - B \cdot e^{-\alpha}} = \frac{\alpha \cdot B^{d+6}}{1 - B \cdot e^{-\alpha}} \cdot log_B(n)$. Die Anzahl der Links liegt also in $O(log(n))$, doch abhängig vom gewünschten ϵ und α kann der Grad der Knoten sehr hoch werden, was den Aufwand diese Links aufzubauen und zu verwalten signifikant steigen lässt.

4 Fazit

Grundsätzlich ist zu sagen, dass der Ansatz von **LAND** zwar auf theoretischer Basis beweisbare Performance nachweisen kann, sich dabei jedoch auf ein idealisiertes Netzwerkmodell bezieht, was abhängig vom gewünschten Stretch $1 + \epsilon$ so definiert wird, dass entsprechende Ergebnisse erzielt werden (was vor allem den Faktor d betrifft). Insbesondere die Annahme, dass ein Netzwerk eine konstanten Ausdehnungsfaktor $\triangle$ aufweist ist, fernab realer Netzwerkstrukturen, die typischerweise Areale mit vergleichsweise hoher Knotendichte aufweisen. Selbst die grundlegende Annahme der Dreiecksungleichung für Pfadkosten kann nicht als allgemeingültig angesehen werden, wie neueste Untersuchungen nachweisen [5]. Ob ein praktischer Einsatz von **LAND** sinnvoll ist und inwieweit sich eine Performanceverbesserung im Vergleich zu anderen P2P-Netzwerken ergeben würde, ist schwer zu sagen. Vor allem die erhebliche Einschränkung auf Netzwerke mit konstantem Ausdehnungsfaktor stellt dies in Frage.

Ein weiterer Kritikpunkt ist der fehlende Erkennungsmechanismus für Knoten, die das Netzwerk spontan verlassen haben. In diesem Fall müssen alle ins Leere laufende Links geändert werden, um die Stabilität des Systems nicht zu gefährden (siehe Abschnitt 2.4.2).

Auch das *shadow router*-Konstrukt wirft einige Fragen auf. So wird zwar bewiesen, dass die Anzahl dieser Router konstant bleibt, doch es ist offensichtlich dass kleinere Netzwerke bis zu einer bestimmten Grenze deutlich mehr *shadow router* aufweisen, weil schlicht keine geeigneten Knoten als Linkziele gefunden werden können und damit ein solcher Knoten emuliert werden muss. Es wäre durchaus interessant gewesen, ab welcher Knotenanzahl **LAND** sinnvoll operieren kann. Ob ein emulierter Router in jedem Fall entfernt wird, wenn ein neuer Knoten dem Netzwerk beitritt, der ihn ersetzen kann, bleibt ebenfalls zu klären.

Die Lektüre des Papers war stellenweise mit hohem Aufwand verbunden, weil Sachverhalte teilweise inkonsistent und unvollständig dargestellt wurden (Vergleich Kapitel 7 im Originalpaper [1]). Insbesondere ließen Algorithmenbeschreibungen die Erläuterung von Sonderfällen bezüglich Indexgrenzen missen.

Literatur

[1] Ittai Abraham, Dahlia Malkhi, and Oren Dobzinski. Land: Stretch $(1 + \epsilon)$ locality-aware networks for dhts. In *Proc. 15th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA*, pages 550–559, 2004.

[2] Vinay Aggarwal, Stefan Bender, Anja Feldmann, and Arne Wichmann. Methodology for estimating network distances of gnutella neighbors. In *INFORMATIK 2004 - Informatik verbindet, Band 2, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, volume 51 of *LNI*, pages 219–223. GI, September 2004.

[3] David R. Karger and Matthias Ruhl. Finding nearest neighbors in growth-restricted metrics. In *In 34th Annual ACM Symposium on the Theory of Computing*, pages 741–750, 2002.

[4] Xiaozhou Li and C. Greg Plaxton. On name resolution in peer-to-peer networks. In *In Proceedings of the 2nd ACM Worskhop on Principles of Mobile Commerce (POMC*, pages 82–89, 2002.

[5] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle inequality and routing policy violations in the internet. In *PAM '09: Proceedings of the 10th International Conference on Passive and Active Network Measurement*, pages 45–54, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, 2001.

[7] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. pages 149–160, 2001.

[8] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, 2001.