

Peer-to-Peer-Systeme
Content Addressable Networks (CAN)

Björn Schießle

13. Juli 2007

This work is licensed under the
Creative Commons Attribution-Share Alike 3.0 License.
To view a copy of this license, visit
<http://creativecommons.org/licenses/by-sa/3.0/deed>
or send a letter to
Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305,
USA.

Zusammenfassung

Die Algorithmen die zum auffinden von Daten in einem Peer-to-Peer Systems verwendet werden, sind für das effiziente Arbeiten eines solchen Systems sehr entscheidend. Mit Hashtabellen kennen wir bereits eine effiziente Möglichkeit Daten zu finden, indem jedem Datum ein Schlüssel zugeordnet wird. Mit verteilten Hashtabellen lässt sich dieses Konzept auf verteilte Strukturen und Peer-to-Peer-Systeme übertragen. Content Addressable Networks (CAN) sind eine mögliche Umsetzung dieser Idee. Im folgenden soll ein konkretes CAN Design vorgestellt und Erweiterungen zur Optimierung des Routing diskutiert werden.

1 Einleitung

Im Gegensatz zum klassischen Client-Server Modell werden die Daten bei Peer-to-Peer-Systemen nicht auf einem zentralen Server sondern auf den einzelnen Clients (Peers) gespeichert und direkt zwischen den Peers übertragen.

Bei den ersten Peer-to-Peer Systemen (z.B. Napster [1]), wurden die Information über die vorhandenen Daten im Netzwerk auf einem zentralen Server verwaltet. Wollte ein Peer bestimmte Daten abrufen, wurde eine Suchanfrage an den Server geschickt, welcher dem Peer daraufhin die IP Adresse des Peers übermittelte, auf dem die Daten zu finden waren. Danach bauten die beiden Peers eine direkte Verbindung auf um die Daten zu übertragen. Obwohl Napster also ein Peer-to-Peer Modell für die Datenübertragung verwendet, wird die Liste der vorhandenen Daten zentral gespeichert. Dieses Verfahren macht das Netzwerk sowohl teuer (Skalierung des zentralen Verzeichnis) als auch fehleranfällig (Stichwort: single point of failure).

Spätere Peer-to-Peer Systeme (z.B. Gnutella[2]) gingen einen Schritt weiter und dezentralisierten auch die Information über die vorhandenen Daten, diese wurden auf den einzelnen Peers verwaltet. Bei der Suche wurde dann Flooding verwendet. Das bedeutet, dass jeder Peer die Suchanfrage an alle ihm angeschlossenen Peers weiterleitet. Da das Flooding irgendwann abgebrochen werden muss, besteht allerdings die Möglichkeit, dass Daten nicht gefunden werden, obwohl sie sich im System befinden.

Die Frage, wie man die Information über die vorhanden Daten im Netzwerk verwaltet und zugänglich macht ist eine entscheidende Frage für die Leistungsfähigkeit von Peer-to-Peer Systemen. Mit Hashtabellen kennen wir bereits eine Struktur, die diesen Zugriff sehr effizient ermöglicht. Content Addressable Networks bieten eine Möglichkeit dieses Verfahren auf Peer-to-Peer Systeme abzubilden.

CANs werden nicht nur in Peer-to-Peers Systemen eingesetzt, sondern auch in anderen Systemen, z.B. großen Speichermanagementsystemen, für die ein effizientes einfügen und abrufen von Daten ebenfalls erforderlich ist. Allerdings soll der Schwerpunkt an dieser Stelle auf Peer-to-Peer Systemen liegen.

Das CAN Modell bietet aber noch Spielraum für Optimierungen im Bereich des Routing. In Abschnitt 3 werden wir uns hierzu ein paar Möglichkeiten ansehen, u.a. die Optimierung durch “Expressways” [3].

Zuvor werde ich aber in Abschnitt 2 den Aufbau und die Arbeitsweise eines Content Addressable Networks beschreiben. Dabei halte ich mich an das Design aus “A scalable content-addressable network” [4].

2 Ein Content Addressable Network

An dieser Stelle soll der Aufbau und die Funktionsweise eines Content Addressable Networks beschrieben werden. Ich beziehe mich dabei auf “A Scalable Content-Addressable Network” [4].

Wie bereits erwähnt, haben Content Addressable Networks Ähnlichkeiten mit Hashtabellen. Zu den grundlegenden Operationen gehört also das einfügen, nachschlagen und löschen von (Schlüssel,Wert)-Paaren. Dieses CAN besteht aus vielen einzelnen Knoten. Jeder dieser Knoten speichert einen Teil der Hashtabelle (diese Teile werden im weiteren Verlauf “Zonen” genannt). Außerdem enthält jeder Knoten auch Informationen über seine Nachbarknoten, welche für das Routing benötigt werden. Das vorgestellte CAN ist vollständig verteilt (es benötigt keinerlei zentrale Steuerung oder Konfiguration), skalierbar (die Knoten verwalten nur eine kleine Menge von Informationen und das System kann jederzeit erweitert oder verkleinert werden) und Fehlertolerant (Knoten können Fehler umgehen).

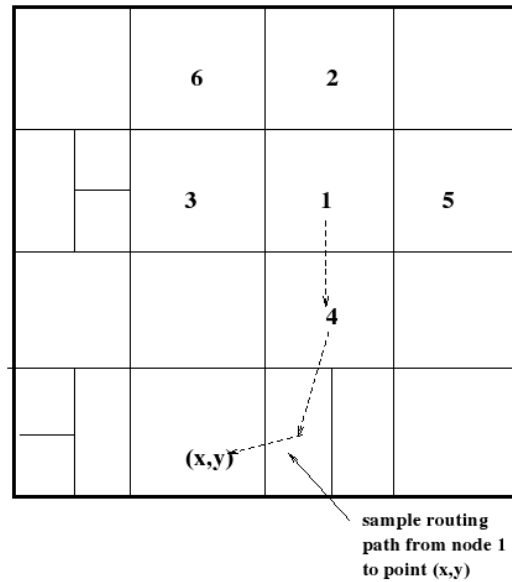
2.1 Aufbau

Man kann sich das CAN als einen d-dimensionalen kartesischen Koordinatenraum vorstellen.

Dieser Koordinatenraum wird unter den Knoten des Systems vollständig aufgeteilt, so dass jeder Knoten einen bestimmten Zone dieses Raums belegt. Abbildung 1 zeigt ein 2-dimensionalen $[0, 1] \times [0, 1]$ Koordinatenraum den sich 5 Knoten im CAN teilen.

In diesem Koordinatenraum werden dann (Schlüssel, Wert)-Paare angeordnet. Dazu wird mittels einer Hash-Funktion dem Schlüssel S deterministisch ein Punkt P im Koordinatensystem zugeordnet. Das entsprechenden

ten 2 gleich sind. Dagegen sind Knoten 1 und Knoten 6 keine Nachbarn, da deren Koordinaten sowohl entlang der X-Achse als auch entlang der Y-Achse aneinander angrenzen.



1's coordinate neighbor set = {2,3,4,5}
7's coordinate neighbor set = {}

Abbildung 2: 2-dimensionaler Koordinatenraum bevor Knoten 7 eingefügt wurde (aus [4])

Mit diesen Informationen über die Nachbarn wird dann eine Nachricht immer von einem Knoten zu dem Nachbarknoten weiter geleitet, der den Zielkoordinaten am nächsten ist. Abbildung 2 zeigt ein Beiepfad. Da es mehrere mögliche Pfade zwischen zwei Punkten gibt, kann der Knoten auch beim Ausfall eines oder mehrerer Nachbarknoten noch einen Pfad finden. Wenn ein Knoten allerdings alle Nachbarknoten in eine Richtung verliert, kann es passieren, dass die Routing Strategie zeitweise nicht möglich ist. In diesem Fall versucht der Knoten durch Flooding einen Knoten zu finden, der näher am Zielknoten dran ist als er selber, von diesem Knoten aus kann dann die normale Routing Strategie wieder fortgesetzt werden.

In einem d-dimensionalen Koordinatenraum, welcher in n gleiche Zonen aufgeteilt wurde, beträgt die durchschnittliche Pfadlänge $(d/4)(n^{1/d})$ Sprünge und jeder Knoten verwaltet $2d$ Nachbarn².

²In Abschnitt 3 werden wir Verfahren kennenlernen, die die durchschnittliche Pfadlänge

2.3 Ein Knoten betritt das CAN

Der gesamte Vektorraum muss zu jeder Zeit unter den Knoten im System aufgeteilt sein. Damit das System auch wachsen kann, muss es möglich sein, einen neuen Knoten im CAN aufzunehmen. Dazu muss ein Knoten der sich bereits im System befindet seinen Zone teilen um Platz für den neuen Knoten zu schaffen.

Dafür sind drei Schritten nötig:

1. Der neue Knoten muss ein Knoten finden, der bereits im CAN ist.
2. Im CAN muss einen Knoten gefunden werden, dessen Zone geteilt werden soll.
3. Die Nachbarn des geteilten Knoten müssen benachrichtigt werden, damit deren Routingtabelle angepasst werden kann.

2.3.1 Einen Knoten im CAN finden

Als erstes muss ein neuer Knoten die IP Adresse eines beliebigen Knoten im Netzwerk herausfinden um über diesen Zugang zum CAN zu bekommen. Dafür gibt es mehrere Möglichkeiten, die konkrete Umsetzung ist für das CAN aber nicht von Bedeutung, daher möchte ich an dieser Stelle mit YOID[5] nur auf eine Möglichkeit hinweisen und diese kurz beschreiben.

Bei YOID gibt es sogenannte "Rendezvous Hosts", die man über einen DNS Domainnamen ansprechen kann. Diese Knoten verwalten eine Liste von Knoten die bereits im CAN eingebunden sind. Ein neuer Knoten verbindet sich also über den Domainnamen mit einem "Rendezvous Host", welcher dem Knoten dann ein zufällig ausgewählten Knoten aus dem CAN mitteilt, über den der neue Knoten das CAN betreten kann.

2.3.2 Einen Zone im CAN finden

Als nächstes wählt der neue Knoten einen beliebigen Punkt im Netz aus und senden an diesen Punkt eine JOIN-Anfrage, dies geschieht von dem Knoten aus, den er von "Rendezvous Host" erhalten hat. Diese Anfrage wird nach dem in Abschnitt 2.2 beschriebenen Routing-Verfahren weitergeleitet, bis sie am Punkt ankommt.

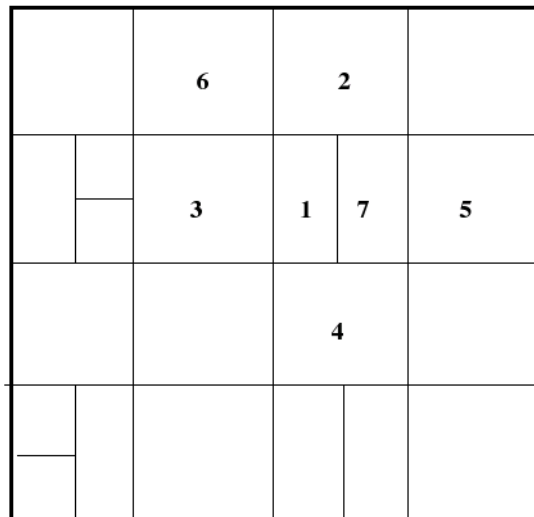
Der Knoten der die Zone des Punktes verwaltet, halbiert dann seinen Zone, teilt die zweite Hälfte dem neuen Knoten zu und übergibt ihm die Schlüssel, Wert)-Paare die in der Hälfte des neuen Knoten liegen.

verbessern.

2.3.3 Knoten ins Routing einbinden

Nachdem der neue Knoten einen Platz im CAN gefunden hat, müssen noch die Routingtabellen der Knoten selber und die der Nachbarknoten angepasst werden. Zuerst aktualisieren sich der neue Knoten und der Knoten der seine Zone geteilt hat. Der alte Knoten gibt die Nachbarn, die jetzt an den neuen Knoten grenzen, an den neuen Knoten ab und beide Knoten tragen ihr Nachbarschaftsverhältnis in ihre Routingtabelle ein.

Als nächstes müssen die Nachbarn darüber informiert werden, das ihr Nachbarschaftsbereich neu aufgeteilt wurde. Dafür sendet in regelmäßigen Abständen jeder Knoten eine Update-Nachricht an seine Nachbarn, dass führt dazu, dass alle Knoten im Netz die neuen und aktuellen Nachbarschaftsverhältnisse lernen. Abbildung 2 und Abbildung 3 veranschaulichen nochmal, wie sich die Zonen aufteilen, wenn ein neuer Knoten das CAN betritt (hier Knoten 7).



1's coordinate neighbor set = {2,3,4,7}
7's coordinate neighbor set = {1,2,4,5}

Abbildung 3: 2-dimensionaler Koordinatenraum nachdem Knoten 7 eingefügt wurde (aus [4])

Das Einfügen neuer Knoten beeinflusst nur einen kleinen Bereich des Koordinatenraums und damit auch eine kleine Anzahl von Knoten. Die Anzahl

der Nachbarn, die ein Knoten verwaltet, hängt nur von der Dimension d des Vektorraums ab und ist unabhängig von der Anzahl der Knoten. Dadurch werden durch das Einfügen eines neuen Knoten nur $O(d)$ bereits existierende Knoten beeinflusst.

2.4 Ein Knoten verlässt das CAN

Wenn ein Knoten das CAN verlässt, muss die Zone des Knoten unter den verbliebenen Knoten wieder aufgeteilt werden. Dafür übergibt der Knoten seine Zone und die zugehörigen (Schlüssel, Wert)-Paare an einen seiner Nachbarknoten.

2.5 Fehlerbehandlung

Das CAN soll auch auf Netzwerk- und Knotenfehler reagieren können, bei denen ein oder mehr Knoten nicht mehr erreichbar sind.

Jeder Knoten sendet in regelmäßigen Abständen Update-Nachrichten an seine Nachbarn mit den Koordinaten seiner Zone, einer Liste seiner Nachbarn und die Koordinaten deren Zonen. Werden solche Update-Nachrichten von einem Knoten nicht mehr empfangen, gehen die Nachbarknoten davon aus, dass ein Fehler vorliegt.

Wenn ein Knoten entscheidet, dass einer seiner Nachbarn nicht mehr aktiv ist, startet dieser einen Übernahme-Timer, der Timer wird dabei in Abhängigkeit der Größe der eigenen Zone initialisiert. Die anderen Nachbarn werden unabhängig voneinander die gleichen Maßnahmen einleiten. Wenn der Timer abgelaufen ist sendet der Knoten eine Übernahme-Nachricht mit der Größe seiner Zone an alle Nachbarn des fehlerhaften Knoten. Empfängt ein Knoten eine solche Übernahme-Nachricht, vergleicht er die Größe seiner Zone mit der Größe der Zone, von dem die Nachricht stammt. Wenn die Zonengröße in der Nachricht kleiner ist, als die eigene Größe, beendet der Knoten seinen Timer, ansonsten antwortet er mit einer eigenen Übernahme-Nachricht. Auf diese Weise wird ein Nachbarknoten ausgewählt, welcher noch am Leben ist und dessen Zone der kleinste ist.

Da in dieser Situation aber keine Verbindung mehr zum fehlerhaften Knoten besteht, sind die (Schlüssel, Wert)-Paare, welche auf dem Knoten gespeichert waren, solange verloren, bis der Knoten, auf dem die Daten liegen, eine Aktualisierungsnachricht schickt. Solche Nachrichten werden von jedem Knoten im CAN in regelmäßigen Abständen generiert.

Unter bestimmten Umständen, z.B. wenn mehrere benachbarte Knoten gleichzeitig ausfallen, kann es passieren, dass ein Knoten ein Fehler erkennt, aber nur noch weniger als die Hälfte aller Nachbarn eines fehlerhaften Knoten

aktiv sind. Wenn der Knoten unter diesen Umständen einen fehlerhaften Zone übernimmt, kann es passieren, dass ein inkonsistenter Zustand entsteht. In einem solchen Fall führt der Knoten eine Ringsuche durch, um aktive Knoten um die fehlerhaften Zonen herum zu finden um damit die Nachbarschaftsinformationen um die fehlerhaften Zonen aufzubauen. Mit diesen Informationen kann dann der Übernahme-Algorithmus, wie weiter oben beschrieben, gestartet werden.

3 Optimierungsmöglichkeiten des Routing

In dem vorherigen Abschnitt haben wir anhand eines konkreten Beispiels [4] den Aufbau und die Funktionsweise eines Content Addressable Networks kennen gelernt. Beim Routing hat das vorgestellte CAN eine durchschnittliche Pfadlänge von $O(dn^{1/d})$, für d Dimensionen und n Knoten. Im folgenden wollen wir uns Möglichkeiten ansehen um das Routing (die durchschnittliche Pfadlänge) zu optimieren.

3.1 Mehrdimensionaler Koordinatenraum

Als eine der naheliegenden Optimierungsmöglichkeiten wird in [4] die Erhöhung der Dimension des Koordinatenraums beschrieben.

Es fällt einem sofort auf, dass das in Abschnitt 2 beschriebene CAN aus einem 2-dimensionalen Koordinatenraum besteht und es eigentlich keinen Grund gibt, den Koordinatenraum nicht auf mehrere Dimensionen auszuweiten. Wenn man die Dimension erhöht, reduziert man die Pfadlänge beim Routing, dafür nimmt man allerdings eine leichte Vergrößerung der Routingtabellen auf den einzelnen Knoten in Kauf. Abbildung 4 zeigt wie sich die Pfadlänge in Abhängigkeit von der Dimension des Koordinatenraums verändert.

Mit der Erhöhung der Dimensionen verbessert man nicht nur das Routing, sondern auch die Fehlertoleranz, da bei einer höheren Dimension die Knoten mehr Nachbarn haben und man daher auch mehr Möglichkeiten hat um Fehler herum zu routen.

3.2 Mehrere Koordinatenräume

Die nächste Möglichkeit die in [4] genannt wird ist, dass man nicht nur einen Koordinatenraum hat, sondern mehrere.

Dabei bekommt jeder Knoten im System einen unterschiedlichen Zone von jedem Koordinatenraum (wir nennen einen solchen Koordinaten-

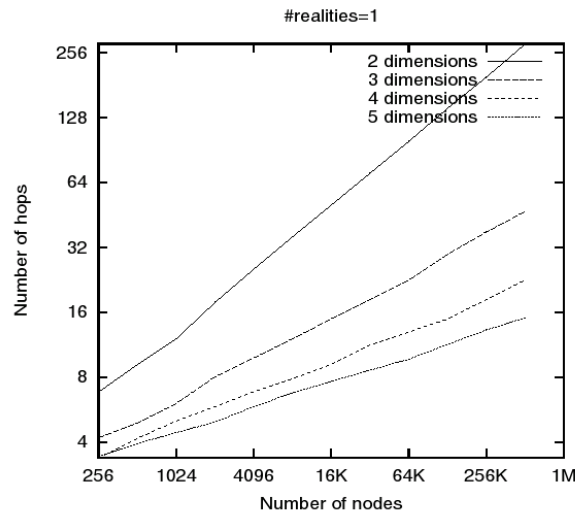


Abbildung 4: Auswirkung der Dimension auf die Pfadlänge (aus [4])

raum“Reality”) zugeteilt. Das bedeutet, dass in einem CAN mit r Koordinatenräumen jeder Knoten r Zonen und r voneinander unterschiedliche Nachbarschaftsverhältnisse verwaltet.

Der Inhalt der Hashtabelle wird auf jeder Reality gespeichert, was die Verfügbarkeit der Daten erhöht. Nehmen wir an, ein Verweis zu einer bestimmten Datei wird an dem Punkt (x, y, z) gespeichert. Bei 4 unabhängigen Realities, würde dieser Verweis auf 4 unterschiedlichen Knoten gespeichert und erst dann nichtmehr erreichbar sein, wenn alle 4 Knoten ausfallen. Das erhöht auch die Fehlertoleranz beim Routing, wenn das Routing in einer Reality nichtmehr möglich ist, kann es in einer anderen Reality fortgesetzt werden.

Da der Inhalt der Hashtabelle auf jeder Reality vorhanden ist, reicht es aus, den Zielpunkt über eine Reality zu erreichen. Da hier ein Knoten eine Zone in jeder Reality verwaltet und diese Zonen beliebig weit auseinander liegen können, besteht die Möglichkeit, auch weit entfernte Koordinaten mit einem Sprung zu erreichen, was die durchschnittliche Pfadlänge deutlich reduziert, wie man in Abbildung 5 sehen kann.

3.3 Optimierung durch Expressways

In “A scalable Content-Addressable Network” [4] haben wir ein CAN kennen gelernt, dessen Routing einen Aufwand von $O(n^{1/d})$ hat. Expressways[3]

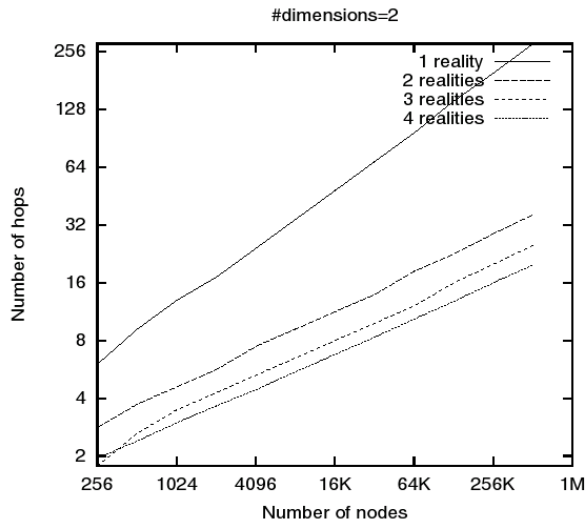


Abbildung 5: Auswirkung mehrere Realities auf die Pfadlänge (aus [4])

bieten eine Möglichkeit das Routing auf $O(\log_K(n))$ zu optimieren¹. Diese Möglichkeit soll im folgenden genauer betrachtet werden.

Abbildung 6 veranschaulicht nochmal die Performanceunterschiede zwischen dem Expressway-Routing und dem normalen CAN-Routing.

3.3.1 Übersicht

Expressways für CANs optimieren das Routing durch Routingtabellen mit einer größeren Ausdehnung.

Dazu wird der ganze Koordinatenraum in Zonen mit unterschiedlicher Ausdehnung aufgeteilt. Die kleinsten Bereiche sind die CAN-Zonen, wie wir sie aus Abschnitt 2 kennen, und die größeren Bereiche bilden sogenannte Expressway-Zonen. Jeder Knoten gehört also zu einer CAN-Zone und liegt gleichzeitig auch in Expressway-Zonen.

Abbildung 7 veranschaulicht dies nochmal. Die CAN-Zonen liegen auf Ebene 3 und machen je $1/64$ des gesamten Koordinatenraums aus. 4 nebeneinander liegende CAN-Zonen ergeben dann eine Ebene-2 Expressway-Zone und 4 benachbarte Eben-2 Expressway-Zonen bilden dann wiederum eine Ebene-1 Expressway-Zone.

¹Ich werde an dieser Stelle nur den Aufbau und die Funktionsweise von Expressways beschreiben. Eine genaue Analyse des Aufwands findet man in "Building Low-maintenance Expressways" [3]

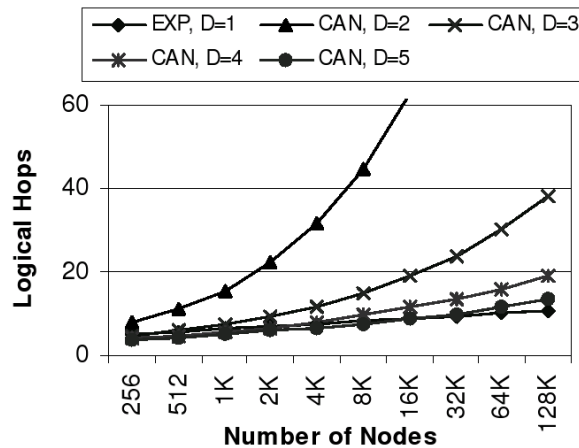


Abbildung 6: Performance Vergleich zwischen Expressways und dem normalen CAN-Routing mit unterschiedlichen Dimensionen (aus [3])

Diese Expressway-Zonen und CAN-Zonen werden in jedem Knoten in einer Datenstruktur gespeichert, die wir “vollständigen Routingtabelle” nennen. $R_T = \langle R_0, R_1, \dots, R_L \rangle$, R_L ist die Standard-Routingtabelle des entsprechenden Knoten, die das CAN bereits aufgebaut hat. R_i ($i = 0$ bis $i = L - 1$) sind die Expressway-Routingtabellen. Je kleiner das i , desto größer die Expressway-Zone. Das Routing mit einem kleineren i geschieht also auf einer höheren Ebene des Expressways. Jede R_i enthält die i . größte Zone die den Knoten einschließt und die Nachbarzonen der gleichen Größe. Die Expressway-Routingtabelle enthält jeweils einen oder mehrere Knoten der angrenzenden Expressway-Zonen.

Abbildung 7 veranschaulicht dies nochmal. Knoten 1 liegt in einer CAN-Zone (der dunkel-graue Bereich links-oben) und gehört gleichzeitig zu den Ebene-2 und Ebene-1 Expressway-Zonen, welche die CAN-Zone umschließen. Die vollständige Routingtabelle von Knoten 1 besteht nun aus der Standard-Routingtabelle des CAN, welche nur auf die direkten Nachbarn des Knotens verweist, und den Expressway-Routingtabellen die jeweils zu einem Knoten der Nachbar-Expressway-Zonen auf Ebene-2 und Ebene-1 verweisen. In Abbildung 7 kann man auch sehen, wie Knoten 1 Knoten 9 mit Hilfe der Expressways erreichen kann.

3.3.2 Aufbau der Expressways

In Abschnitt 3.3.1 wurde der prinzipielle Aufbau von Expressways beschrieben. Die Frage ist, wie man diese Expressways nun in einem Netz aufbaut, in-

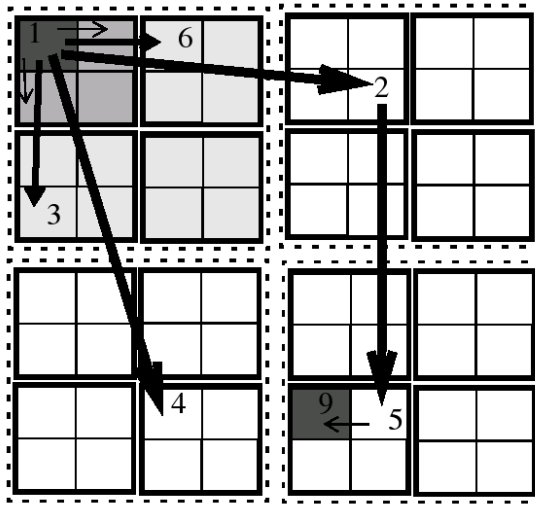


Abbildung 7: Expressways in einem CAN (aus [3])

dem jederzeit neue Knoten hinzu kommen können und Knoten das Netz auch wieder verlassen können. Eine Mögliche Lösung ist der “envolving snapshot”-Algorithmus, wie er auch hier[3] beschrieben wird.

Die Idee hinter diesem Algorithmus ist relativ einfach. Es wird in regelmäßigen Abständen ein Snapshot der aktuellen Routingtabelle gemacht. Dieser Snapshot wird dann auf die vollständige Routingtabelle R_T von x gepusht.

Die Routingtabelle $x.R$ des Knoten x sieht folgendermaßen aus. Die Routingtabelle beinhaltet die aktuelle Zone des Knoten, diese wird mit $x.R.Z$ bezeichnet, die Nachbarzonen $x.R.N_d$ für jede der d Dimensionen, und die Adresse einer oder mehrere Knoten in den Nachbar-Expressway-Zonen. Dieser Snapshot wird dann auf die vollständige Routingtabelle von x gepusht, R_T .

Die Knoten beobachten die Größe ihrer Zone und erstellt ihren Snapshots unabhängig voneinander. Wenn die aktuelle Zone $x.R_L.Z$ von Knoten x auf die Größe $x.R_{L-1}.Z/K$ schrumpft, erzeugt er einen neuen Snapshot indem er L erhöht und R_L von $x.R_{L-1}$ kopiert.

Das CAN beginnt mit einem Knoten x im System. Die vollständige Routingtabelle sieht dann so aus $R_T = \langle R_0, R_1 \rangle$ und $R_0.Z$ beinhaltet den gesamten Koordinatenraum. Wenn jetzt ein zweiter Knoten y dazu kommt, der sich die Zone mit x teilt, übernimmt er alle Einträge der vollständigen Routingtabelle von x , außer x aktueller Routingtabelle ($x.R_L$). $x.R_L$ und $y.R_L$ werden entsprechend des normalen CAN Algorithmus gebildet. Mit der

weiteren Ausbreitung des Systems, erstellen die Knoten in regelmäßigen Abständen Snapshots und sammelt damit die bisherigen Routingtabellen an.

<p>Procedure for a node y joins node x $y.R_T = \langle x.R_0, \dots, x.R_{L-1}, y.R_L \rangle$ Repeat procedure for testing for new snapshot</p>
<p>Procedure for testing for new snapshot // executed by both x and y If $(R_L.Z \leq R_{L-1}.Z/K)$ { $R_{L+1} = R_L$ $R_T = \langle R_0, R_1, \dots, R_L, R_{L+1} \rangle$ $L = L+1$ }</p>

Abbildung 8: Algorithmus, wenn ein neuer Knoten das System betritt (aus [3])

Abbildung 8 zeigt was zusätzlich zum normalen CAN Algorithmus noch gemacht werden muss, wenn ein neuer Knoten das System betritt. Der neue Knoten übernimmt die vollständige Routingtabelle von dem Knoten der sich aufteilt, danach testen beide Knoten ob sie auf das $1/k$ -te im Vergleich zum letzten Snapshot geschrumpft sind. Ist das der Fall, wird ein neuer Snapshot erzeugt.

3.3.3 Routing

<p>Route with Expressway If $(pt \in R_L.Z)$ Return; For $(i=0; i \leq L; i++)$ If $(pt \notin (R_i.Z))$ Route using R_i;</p>
<p>Route with R_i For $(j=0; j < d; j++)$ If $(pt < R_i.Z.L_j \parallel pt > R_i.Z.U_j)$ { Route to $x \in R_i.N_j$ that is closest to pt Break; }</p>

Abbildung 9: Beschreibung des Routing in Expressways in Pseudo-Code (aus [3])

Das Routing-Protokoll ist relativ einfach. Wenn sich der Zielpunkt bereits in der aktuellen CAN-Zone ($R_L.Z$) des Knoten befindet, haben wir bereits das Ziel erreicht. Ansonsten iterieren wir durch die vollständige Routingtabelle, angefangen mit der größten Expressway-Zone, bis wir eine Routingtabelle R_i finden, so dass $R_i.Z$ den Zielpunkt nicht einschließt. Dann wird die Nachricht zu einem der Nachbarn von R_i geroutet, welcher am nächsten am Zielpunkt liegt. Dadurch erreichen wir, dass sich die Nachricht mit den größt möglichen Sprüngen in Richtung Ziel bewegt.

In Abbildung 9 sehen sie die Beschreibung des Routing-Algorithmus in Pseudo-Code, d beschreibt die Dimension des Koordinatenraums, pt den Zielpunkt und $R_i.Z.L_j$ und $R_i.Z.U_j$ beschreiben die untere beziehungsweise obere Grenze von $R_i.Z$ entlang der j -ten Dimension.

4 Zusammenfassung

Mit Content Addressable Networks haben wir eine Möglichkeit kennen gelernt, um die Idee von Hashtabellen auf verteilte Strukturen und Peer-to-Peers Systeme zu übertragen. Dies ermöglicht eine wichtige Funktion in solchen Systemen, das auffinden von Daten, sehr effizient zu gestalten. Mit der Beschreibung eines relativ einfachen CAN in Abschnitt 2 haben wir uns den Aufbau eines solchen Systems genauer angesehen und dabei einen mittleren Pfadlänge von $O(dn^{1/d})$ erreicht. In Abschnitt 3 haben wir dann Optimierungsmöglichkeiten kennen gelernt und es geschafft, die durchschnittliche Pfadlänge mittels Expressways 3.3 auf $O(\log_K(n))$ zu reduzieren.

CANs ermöglichen also sehr effektive Routing-Algorithmen, darüber hinaus haben CANs eine sehr hohe Fehlertoleranz, da es immer mehrere mögliche Wege zu einem gesuchten Punkt im Koordinatenraum gibt.

Ein weitere Vorteil den uns CANs bieten ist, dass Daten, die sich im CAN befinden, unter normalen Umständen auch garantiert gefunden werden, da jedem Datum ein eindeutiger Punkt im Koordinatenraum des CAN zugeordnet wird. Dagegen kann es bei Systemen wie Gnutella[2] auch passieren, dass Daten nicht gefunden werden, obwohl sie sich im System befinden und jeder Knoten im Netz korrekt arbeitet.

Gerade auch Expressways sind Gegenstand weiterer Forschung, deren Ziel es ist, das Routing weiter zu optimieren, indem man die Expressways besser auf die Ansprüche der Anwendungsbereiche und den zugrundeliegenden physikalischen Netzwerken abstimmt. Es sind also durchaus noch weitere Optimierungen im Bereich des Routing zu erwarten.

Literatur

- [1] Napster. <http://www.napster.com/>.
- [2] Gnutella. <http://www.limewire.com/>.
- [3] Zhichen Xu and Zheng Zhang. Building low-maintenance expressways. Hewlett-Packard, 2002.
- [4] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.
- [5] Paul Francis. Yoid: Extending the internet multicast architecture. In <http://www.aciri.org/yoid/docs/>, 2000.