Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für praktische Informatik und Medieninformatik

Fachgebiet Telematik

Prof. Dr. Dietrich Reschke

Hauptseminar Telematik

SS 2003

Namens- und Lokalisierungsmechanismen in dezentralen verteilten Systemen

vorgelegt von:
Unbehaun, Robert

Inhaltsverzeichnis

1 Einleitung

Dezentrale verteilte Systeme werden heute und in Zukunft immer wichtiger für die Kommunikation und den Datenaustausch in großen Rechnernetzen und über weite Strecken hinweg. Nicht allein aus diesem Grund sind sie auch immer mehr Gegenstand von gegenwärtigen Forschungsbemühungen. Im folgenden seien die derzeit wichtigsten Anwendungsgebiete dezentraler verteilter Systeme kurz erläutert (vgl. [1], [2]).

Die wohl bekanntesten und populärsten Anwendungsbereiche sind das Distributed File Sharing und das Person-to-Person Messaging. Distributed File Sharing Applikationen bzw. Protokolle, wie beispielsweise Freenet, Gnutella oder KaZaA, ermöglichen es den Nutzern, Dateien mit jedem anderen Nutzer entsprechender Applikationen innerhalb eines virtuellen Netzwerkes auszutauschen. Notwendige Voraussetzung für die Verwendbarkeit solcher Systeme ist das Auffinden relevanter Datenquellen in einem zeitlich zumutbaren Rahmen.

Instant Messaging Systeme wie zum Beispiel Jabber erlauben es Nutzern Textnachrichten auszutauschen und mit anderen Nutzern über große Strecken hinweg günstig und fast echtzeitnah zu kommunizieren. Daher benötigen Messaging Systeme Lokalisierungsmechnismen, die ein schnelles Auffinden anderer Personen und eine schnelle Aktualisierung der Liste von Nutzern im Onlinestatus ermöglichen.

Ein weiterer Forschungsschwerpunkt ist Distributed Computing in dezentralen Systemen, was sich derzeit noch in den Anfängen der Entwicklung befindet. Es ist allerdings wegen des enormen Potenzials an Rechenkapazität von großem Interesse für die Wissenschaft und Wirtschaft. Grundgedanke ist das Aufteilen rechenintensiver Prozesse auf mehrere verteilte Prozessoren zur schnelleren parallelen Berechnung. Eine der bekanntesten Anwendungen ist sicherlich SETI@home. Hiermit ist es möglich geworden, große Datenmengen, die sich bei der Beobachtung des Universums ergeben haben, mit Hilfe der ungenutzten Rechenleistung sehr vieler, weltweit verstreuter Rechner auszuwerten.

Es ist offensichtlich, dass für dezentrale verteilte Systeme effiziente Namens- und Lokalisierungsmechanismen benötigt werden, die ein Suchen von Ressourcen in dieser Art von Netzen ermöglichen. Der gegenwärtige Stand der Entwicklung soll in dieser Hauptseminararbeit behandelt werden. Dazu werden die verschiedenen Konzepte von Namens- und Lokalisierungsmechanismen gegenübergestellt. Der Schwerpunkt liegt dabei auf der Suche von Ressourcen, bei denen die Quelle erst ermittelt werden muss.

2 Grundlagen

2.1 Definition dezentraler verteilter Systeme

2.1.1 Verteile Systeme

In der Literatur existieren verschiedene Definitonen für den Begriff eines Verteilten Systems. Den weiteren Ausführungen sei die folgende Defintion zu Grunde gelegt:

Ein verteiltes System ist ein System, in dem Hard- und Softwarekomponenten, die sich auf miteinander vernetzten Computern befinden, miteinander kommunizieren und ihre Aktionen koordinieren, indem sie Nachrichten austauschen.

2.1.2 Topologien verteilter Systeme

Topologien können aus verschieden Blickwinkeln betrachtet werden. Generell wird untersucht, wie die Knoten in einem System miteinander verbunden sind. Dies kann auf physischer, logischer oder organisatorischer Ebene sowie auf Verbindungsebene geschehen. Verteilte Netze lassen sich anhand ihrer Topologien grundsätzlich in zentrale, hierarchische, dezentrale und Ringtopologien unterscheiden. Darüber hinaus existieren Mischformen. Die möglichen Topologien verteilter Systeme werden im Folgenden kurz voneinander abgegrenzt (vgl. [3]).

2.1.2.1 Reine Topologien

Zentrale Systeme

Zentralisierte Systeme sind heutzutage die am weitesten verbreiteten Systeme. Die Client/Server Architektur wird beispielsweise für Datenbanken, Webserver und andere verteilte Systeme verwendet. Alle Funktionen und Daten werden von einem Server zentral angeboten und von vielen Clients, die sich mit diesem Server verbinden müssen, in Anspruch genommen. Zentralisierte Komponenten sind auch in vielen Peer-to-Peer Applikationen enthalten. SETI@home zum Beispiel besitzt eine vollständig zentralisierte Architektur mit einem Job Dispatcher als Server. Auch Napsters Suchmechanimus ist zentralisiert, wenn auch der direkte Datenaustausch dezentral abläuft.

Ringsysteme

Da ein einzelner Server oftmals nicht ausreicht, um eine große Anzahl von Clientanfragen zu bearbeiten, ist es üblich, mehrere Rechner in Form eines Ringes zusammenzuschließen und sie dann als verteilten Server agieren zu lassen. Durch gemeinsame Kommunikation unter den Rechnern kann sichergestellt werden, dass die angebotenen Informationen identisch sind. Darüber hinaus wird Ausfallsicherheit und eine gezielte Verteilung der Arbeitslast gewährleistet.

Hierarchische Systeme

Hierarchische Topologien finden schon seit langer Zeit Anwendung im Internet, obwohl sie in der Praxis oftmals als verteiltes System übersehen werden. Ausgehend von einem Rootserver, bieten verschiedene weitere Server den selben Dienst an, wobei sie sich regelmässig mit dem Rootserver abgleichen, bzw. Anfragen an ihn weiterleiten, wenn im eigenen Bereich keine Lösung gefunden werden kann. Diese weiteren Server sind ausgehend vom Rootserver in verschiedenen Zweigen hierarchisch angeordnet. Hierarchische Netze lassen sich besonders

gut für Nachschlagesysteme einsetzen, wie beispielsweise der Domain Name Service (DNS) eines ist. Darüber hinaus gibt es das Network Time Protocol (NTP). Auch hier erfolgt der Zeitabgleich, der als Dienst vom Rootserver angeboten wird, in hierarchischer Anordnung. Der Baum besteht dabei aus den Computern die diesen Dienst in Anspruch nehmen und gleichzeitig Hosts sind.

Dezentralisierte Systeme

In dieser Form von Netzwerk existieren keine zentralen Server. Es wird hier von sog. Peers gesprochen, die auf gleichberechtigter Ebene miteinander kommunizieren. Dabei stellt jeder Peer einen Knoten in diesem Netzwerk dar. Den meisten Filesharing Applikationen liegt diese Topologie zugrunde, wenn auch vollständig dezentrale Systeme eher selten zu finden sind. Die wohl am meisten dezentralisierte Filesharing Applikation ist Gnutella. Hier ist lediglich die Funktion zum Einbinden eines neuen Hosts zentral gelöst.

2.1.2.2 Hybride Topologien:

Meist haben Computernetzwerke keine einfachen Topologien, sondern sind eine Mischung aus mehreren verschiedenen Formen von Topologien. Dies wird auch als Hybride Topologien bezeichnet. Dabei kann zum Beispiel ein Knoten einerseits in einem zentralen System interagieren und gleichzeitig Teil eines hierarchischen Netzwerks sein.

Zentral + Ring

Die Server sind hier als Ring angeordnet, um, wie bereits oben erwähnt, Ausfallsicherheit und Verteilung der Arbeitslast zu erreichen. Es ist damit möglich die Einfachheit eines zentralen Systems aus Sicht der Clients und die Robustheit eines Ringes zu vereinen.

Zentral + Zentral

Häufig ist es so, dass Server aus einem zentralen Netzwerk in einem anderen Netzwerk Clients sind, die ihrerseits wiederum Dienste von anderen Server aufrufen.

Zentral + Dezentral

Eine neuere Form von Peer-to-Peer Systemen vereint die zentrale Systemarchitektur mit der dezentralen. Hierbei ist das zentrale System in die dezentrale Architektur eingebettet. Die meisten Peers besitzen eine zentrale Verbindung zu einem Superknoten, zu dem Sie alle Suchanfragen weiterleiten. Die Superknoten sind dabei keine einzelnen unabhängigen Server, sondern stehen miteinander über ein dezentrales Netzwerk hinweg in Verbindung, um gemeinsam die einzelnen Suchanfragen besser zu bearbeiten.

2.1.3 Dezentrale Systeme

Dezentrale Systeme können in strukturierte und unstrukturierte Systeme unterteilt werden (vgl. [4]). Diese Unterscheidung wird hier hinsichtlich des Filesharings getroffen, da dieses eines der wesentlichsten Anwendungen dezentraler verteilter Systeme darstellt.

Dezentrale strukturierte Systeme

Bei dieser Art von Systemen wird die Menge der Verbindungen zwischen P2P Nutzern kontrolliert. Weiterhin werden Dateien nicht nur auf beliebigen Knoten abgespeichert, sondern darüber hinaus an weiteren spezifizierten Punkten abgelegt, was eine erneute Suche der selben Datei später erleichtert und beschleunigt. In locker strukturierten Systemen werden an den spezifizierten Punkten nicht die Dateien selbst, sondern lediglich Verweise (Hints) auf

die Datei vermerkt.

Dezentrale unstrukturierte Systeme

Hier existiert kein gezieltes Ablegen von häufig nachgefragten Dateien oder Verweisen an spezifizierten Punkten. Ebensowenig ist eine Kontrolle der Menge von Verbindungen zwischen den P2P Nutzern gegeben. Das Netzwerk bildet sich aus Knoten, die jederzeit zum System hinzukommen oder es verlassen können. Die Suche kann bei dieser Art von Topologie wesentlich aufwendiger und ressourcenintensiver sein. In der Regel fragt ein Knoten zunächst seine Nachbarn, um eine Datei zu finden. Die typischste Suchmethode nennt sich Flooding. Hierbei erreicht die Suchanfrage alle Knoten innerhalb eines bestimmten Umkreises.

2.2 Definition Lokalisierungsmechanismus

Da in der Literatur kaum die Begriffsklärung eines Lokalisierungsmechanismus vorgenommen wird, sei im Rahmen dieser Arbeit folgende Definition unterstellt.

Ein Lokalisierungsmechanismus ist ein Algorithmus zum Auffinden von Ressourcen innerhalb eines Netzwerks.

Ein effektiver Lokalisierungsmechanismus in Bezug auf dezentrale verteilte Systeme sollte fünf Anforderungen erfüllen (vgl. [5]).

- Effizientes Arbeiten in kleinen wie auch in großen Netzwerken
- Effizientes Arbeiten für eine geringe und eine große Menge von Ressourcen
- Unterstützung möglichst vieler verschiedener Arten von Metadaten und Suchanfragen
- Liefern von möglichst genauen und relevanten Informationen auf jede Suchanfrage
- Sicherheit gegenüber Angriffen und Ausbeutung von Bandbreite

Die ersten beiden Punkte beziehen sich im wesentlichen darauf, dass das System skalierbar bleibt, d.h. dass auch bei großen Netzwerken mit einer großen Menge von Ressourcen die Funktionsweise des Lokalisierungsmechanismus die Systemnutzung nicht unmöglich macht. Dieses Problem wird in dieser Abhandlung noch eine Rolle spielen. Darüber hinaus werden in den Punkten drei und vier Faktoren genannt, die mehr für die individuelle Nutzbarkeit des Systems von Bedeutung sind. Der letzte Punkt geht einher mit den ersten beiden Punkten, da ohne eine gewisse Störungssicherheit kein effizientes Arbeiten des Lokalisierungsmechanismusses möglich ist und damit auch die Skalierbarkeit beeinträchtigt wird.

2.3 Definition Namensmechanismus

Namen dienen in verteilten Systemen dazu, Objekte, wie Dateien, Computer, Benutzer, Netzadressen, Prozesse, Dienste und Ports zu benennen und eindeutig zu identifizieren (vgl.[6] S. 183 ff). Sie müssen aufgelöst werden können, d.h. es muss möglich sein, das Objekt, das ein Name bezeichnet, anhand dieses Namens zu finden. Dies ist nur möglich, wenn ein einheitliches Namensgebungssystem verwendet wird. In verteilten Systemen kann dieses Namensgebungssystem häufig über mehrere Rechner verteilt sein. Die Art und Weise dieser Verteilung hat einen starken Einfluss auf die Effizienz und Skalierbarkeit des Namensgebungssystems.

Um auf ein Objekt zugreifen zu können, benötigt man einen sog. Access Point. Der Name eines Access Points ist eine Adresse. Damit sind Adressen ebenfalls eine speziele Art von

Namen. Werden Adressen verwendet, um auf Objekte zu verweisen, so besteht die Gefahr, dass sich das Objekt nicht mehr an diesem Punkt befindet. Daher ist es günstig, das Objekt nicht allein durch eine Adresse zu referenzieren, sondern darüber hinaus mit einem eindeutigen Bezeichner (auch Identifier). Dabei sollte jeder Bezeichner nur auf ein Objekt verweisen und jedes Objekt auch nur einen Bezeichner besitzen. Außerdem verweist ein Bezeichner immer auf dasselbe Objekt und wird nicht wiederverwendet. Mit Hilfe eines Bezeichners lässt sich eine neue Adresse ermitteln, falls ein Objekt an einer bestimmten Adresse nicht gefunden werden konnte.

3 Namens- und Lokalisierungsmechanismen

Es existieren verschiedene Mechanismen, die sich hinsichtlich ihres Designs und ihrer Architektur zum Teil stark unterscheiden. Es wird im Folgenden auf die Mechanismen Flooding, Random Walk, Distributed Hashtables, Futella und Alpine näher eingegangen. Danach sollen noch einige Bemerkungen zum heutigen Stand der Technik gemacht werden.

3.1 Flooding

3.1.1 Funktionsweise

Die grundlegende Idee des Floodingmechanismusses lässt sich folgendermaßen erklären. Ein Nutzer innerhalb eines Peer-to-Peer Netzwerkes stellt eine Suchanfrage. Diese wird an bekannte Knoten weitergeleitet. Sofern diese Knoten das gewünschte Suchobjekt nicht haben, wird die Anfrage wiederum an alle bekannte Knoten weitergeleitet. Bei einem Treffer erhält der Ausgangsknoten eine Nachricht und kann sich dann gegebenenfalls zum Download entschließen.

Da dieses Verfahren sehr schlecht skaliert und daher für große Netzwerke in dieser Form nicht angewendet werden kann, existieren verschiedene Ansätze, um das unendliche Weiterversenden einer Nachricht im Netzwerk zu begrenzen. Allerdings lässt sich auch damit die Skalierbarkeit nur begrenzt verbessern. Alle diese Ansätze beziehen sich auf die Fragen, (a) wem ein Knoten die Suchanfrage weiterleitet, (b) wann er sie weiterleitet und (c) wie die Anfrage gestoppt wird. Die verschiedenen Varianten sind dabei folgende (vgl. [7]):

Zu (a):

- *Alle*: Ein Knoten aus der Menge des Frontier Sets leitet die Suchanfrage an alle bekannten Knoten weiter, außer an den Knoten von dem er die Anfrage erhalten hat. Das Frontier Set ist dabei die Menge von Knoten, die die Suchanfrage zwar erhalten, aber noch nicht weitergeleitet hat. Diese Methode verwendet Flooding. Skalierbarkeit ist dabei nicht gegeben.
- *Teilmenge*: Ein Knoten sendet die Suchanfrage nur zu einem Teil der ihm bekannten Knoten weiter. Wenn der Knoten weiß, über welche Ressourcen die benachbarten Knoten verfügen, so ist eine gezielte Weiterleitung möglich.

Zu (b):

Jede Suchanfrage enthält ein Feld, dass die Zahl der Sprünge, die sie gemacht hat zählt. Das Feld wird immer dann inkrementiert, wenn ein Knoten die Suchanfrage weiterleitet. Die Zahl der gesendeten Nachrichten, die während einer Suchanfrage entstehen, erhöht sich enorm mit der Zahl der Sprünge, womit sich auch die Inanspruchnahme von Bandbreite erhöht. Es besteht daher die Möglichkeit, die Weiterleitung der Suchanfrage zu verzögern. Es seien hier drei mögliche Formen der Weiterleitung erwähnt:

- *Unmittelbar*: Ein Knoten des Frontier Sets leitet eine Suchanfrage sofort weiter ohne jegliche Verzögerungen.
- *Konstant*: Ein Knoten verzögert die Weiterleitung einer Nachricht für eine fest vorgegebene Zeit.
- *Linear*: Das Zeitintervall, um das die Weiterleitung einer Nachricht verzögert wird,

erhöht sich linear mit der Zahl der bereits unternommenen Sprünge.

Zu (c):

Eine Suchanfrage sollte gestoppt werden, wenn das damit beabsichtigte Ziel erreicht wurde. Jedes weitere Versenden von Nachrichten wäre eine Verschwendung von Ressourcen. Folgende Optionen zum Stoppen einer Suchanfrage sind denkbar:

- *Nie*: Ein Knoten aus der Menge des Fontier Set stoppt niemals eine Anfrage. In diesem Fall endet das Broadcast erst, wenn alle Rechner des Netzwerkes die Suchanfrage einmal erhalten haben.
- *TTL*: Jede Suchanfrage besitzt eine bestimmte Lebensdauer abhängig von den Sprüngen, die sie machen kann. Alle Knoten des Frontier Set unterlassen die Weiterleitung der Anfrage, wenn die Zahl der Sprünge gleich TTL ist (Time to live).
- *Periodische Genehmigung*: Der Suchanfrage wird vom Ursprungsknoten ein sog. HCP Parameter (Hop Count Period) angehängt. Beträgt die Zahl der unternommenen Sprünge ein Vielfaches dieses Parameters, so meldet sich der entsprechende Knoten beim Ursprungsknoten und bittet um die Genehmigung zur Weiterleitung der Anfrage. Wurde der Ursprungsknoten bis dahin mit den bereits erzielten Suchergebnissen zufriedengestellt, so verweigert er die Genehmigung und die Anfrage wird nicht weitergeleitet.
- *Hit Genehmigung*: Sollte ein Knoten eine Suchanfrage beantworten können, so meldet er sich beim Ursprungsknoten und fragt, ob die Anfrage weitergeleitet werden soll. Akzeptiert der Ursprungsknoten den Treffer, so wird die Suchanfrage gestoppt.
- *NEGs*: Dies sind Nachrichten, die einen Knoten veranlassen, eine Suchanfrage zu stoppen, wenn er sie rechtzeitig vorher erhält. Sollte er diese Nachricht nach dem Versenden der Suchanfrage erhalten, so wird er sie an die selben Knoten weiterleiten, an die er die Suchanfrage geschickt hat. Es wird deutlich, dass die Geschwindigkeit der NEG Nachricht größer sein muss als die der eigentlichen Suchanfrage. Um dies zu erreichen, kann beispielsweise die Weiterleitung der Suchanfrage mit Verzögerung geschehen, während alle NEG Nachrichten unmittelbar weitergeleitet werden.

3.1.2 Expanding Ring

Das Verfahren des Expanding Ring basiert auf dem Floodingmechanismus und unternimmt den Versuch die Netzlast zu begrenzen, indem eine Suchanfrage zunächst mit einer TTL von eins verschickt wird. Ergeben sich keine Treffer, so wird die TTL inkrementiert und die Suchanfrage erneut verschickt. Das System ist besser skalierbar als das einfache Flooding. Die Objekte, die häufig nachgefragt werden und somit auch an vielen Stellen im Netz liegen, können schon mit einer geringen TTL gefunden werden. Damit lässt sich die Bandbreitenauslastung verringern. Auf der anderen Seite werden bei seltenen Objekten viele Knoten durch wiederholte Suchanfragen mit erhöhter TTL mehrfach abgefragt, was immer noch zur Verschwendung von Bandbreite führt.

3.1.3 Beispiel Gnutella Version 0.4

Gnutella ist eines der bekanntesten Peer-to-Peer Protokolle, das auf dem Flooding Prinzip aufbaut. Für die Begrenzung der Suchanfrage verwendet Gnutella TTL (*T*), d.h. die Sprünge, die eine Nachricht machen kann und eine Festlegung über die Anzahl der Verbindungen, die zu anderen Knoten bestehen (*N*). Damit ist gemeint, dass eine Suchanfrage von einem Knoten nur an eine bestimmte Anzahl anderer Knoten weitergeleitet wird.

Unter den folgenden Annahmen kann nun veranschaulicht werden, in welchem Umfang bei diesem Mechanismus Bandbreite in Anspruch genommen wird. Es wird davon ausgegangen, dass jeder Knoten die gleichen Werte für *N* und *T* verwendet. Darüber hinaus sei hier ein 18 Byte langer Suchstring betrachtet, wobei sich das zu sendende Paket durch IP-, TCP- und Gnutellaheader noch auf 83 Byte vergrößert.

Der Datenumfang für die Antwort auf eine Suchanfrage sei 788 Byte. Hierin enthalten sind neben dem IP-, TCP- und Gnutellaheader die Länge des Antwortstrings, die Anzahl der gesendeten Antworten eines Knotens und der Servant Identifier, der die Identifizierung einzelner Knoten ermöglicht. Geht man nun davon aus, dass in dem betrachteten Netzwerk 30% der Knoten Inhalte anbieten und nur 50% von diesen auf Suchanfragen antworten, so ergibt sich bei 10 queries per second die folgende Tabelle (vgl. [8]):

Bandbreitenauslastung bei 10 qps								
N	***T=1***	***T=2***	***T=3***	***T=4***	***T=5***	***T=6***	***T=7***	***T=8***
3	5.4KBps	21.6KBps	65.6KBps	176.2KBps	443.2KBps	**1.1MBps**	**2.4MBps**	**5.8MBps**
4	7.2KBps	39.8KBps	171.8KBps	670KBps	**2.4MBps**	**8.8MBps**	**30.6MBps**	**104MBps**
5	8.8KBps	63.4KBps	356.6KBps	**1.8MBps**	**9MBps**	**42.2MBps**	**194.8MBps**	**882.6MBps**
6	10.6KBps	92.2KBps	642.4KBps	**4.2MBps**	**25MBps**	**146.8MBps**	**845.2MBps**	**4.8GBps**
7	12.4KBps	126.8KBps	**1.1MBps**	**8MBps**	**58.4MBps**	**412.4MBps**	**2.8GBps**	**19.2GBps**
8	14.2KBps	166.6KBps	**1.6MBps**	**14.2MBps**	**121MBps**	**995.4MBps**	**8GBps**	**63.4GBps**

Tabelle 3.1.3/1 Bandbreitenauslastung Gnutella Version 0.4

Aus Tabelle 3.1.3/1 ist ersichtlich, dass beim Floodingmechanismus in großem Umfang Bandbreite in Anspruch genommen wird und diese oftmals schon allein deshalb verschwendet wird, weil viele Knoten Anfragen doppelt gesendet bekommen oder die Anfragen selbst übermäßig lange im Netzwerk zirkulieren.

Gnutella wurde seit den ersten Anfängen im Jahr 2002 stark verbessert (vgl.[5]). Während die ursprüngliche Version für kleine Netzwerke gedacht war und wie oben gezeigt in großen Netzen nicht skaliert, konnte mit einigen Änderungen die Skalierbarkeit erhöht und damit die Verwendbarkeit in größeren Systemen verbessert werden. Eine wichtige Änderung war der Ausschluss von webbasierten Clients. Diese ermöglichten es Nutzern, Suchanfragen an das Netzwerk zu stellen ohne selbst Inhalte anzubieten. In der neuen Konzeption (ab Version 0.6)teilen Knoten nur noch Inhalte mit Knoten die auch ihrerseits Daten zum Download anbieten. Weiterhin wurde das Broadcasting durch gezielte Weiterleitung der Suchanfragen zu bestimmten Knoten ersetzt. Mit Einführung des Clip2 Reflectors wurde es darüber hinaus möglich, dass Knoten mit großer Bandbreite als Proxies für Knoten mit geringer Bandbreite fungieren konnten.

3.2 Random Walk

Die Idee von Random Walk ist es, eine Suchanfrage an einen beliebigen anderen bekannten Knoten weiterzuleiten, bis das gewünschte Objekt gefunden wurde. Die Nachricht, die dieser Suchanfrage zugrunde liegt und im Netzwerk zirkuliert, wird „Walker“ genannt. Ausgehend von einem Walker ist hier zwar eine Verringerung des Trafficaufkommens zu verzeichnen, jedoch dauert auch das Auffinden von Objekten entsprechend länger. Um dieser Verzögerung entgegenzuwirken, können statt einem Walker mehrere verwendet werden. Das heißt anstatt

eine einzige Nachricht zu versenden, verschickt der Ursprungsknoten k Nachrichten, wobei jede dieser Nachrichten ihren eigenen zufälligen Weg durch das Netzwerk nimmt. Es kann bestätigt werden, dass k Walker nach T Schritten die gleiche Anzahl von Knoten erreichen wie ein einzelner Walker nach kT Schritten (vgl. [4]). Damit lässt sich die Verzögerung der Suchergebnisse um den Faktor k verringern. Solange die Menge der erreichten Knoten signifikant größer wird und eine Verschwendung von Bandbreite begrenzt ist, kann die Zahl der Walker erhöht werden.

Um einen Walker zu terminieren, können beispielsweise TTL oder periodische Genehmigungen verwendet werden (siehe Kapitel 3.1.1), wobei auch die periodische Genehmigung eine TTL enthält, die hier allerdings sehr groß gewählt ist und dem Zweck dient, Schleifen zu verhindern. Während periodische Genehmigungen sich beim Floodingmechanismus weniger anbieten, da sie zu einem hohen Nachrichtenaufkommen beim Ursprungsknoten führen können, lässt sich beim Random Walk ein gutes Gleichgewicht zwischen Nachrichtenflut beim Ursprungsknoten und den Vorteilen von periodischen Genehmigungen erreichen, wenn eine Rückfrage nach jedem vierten Sprung erfolgt. Weiterhin ist es günstig, der Suchanfrage eine ID zu geben, die jeder Walker mit sich trägt, so dass ein Knoten 2 Walker, die zur selben Suchanfrage gehören, nicht zum selben Knoten weiterleitet. Dabei müssen sich die Knoten merken, welche Walker sie bereits wohin weitergeleitet haben.

Der wesentlichste Unterschied zum Floodingmechanismus liegt beim Random Walk darin, dass mit einem zusätzlichen Sprung die Menge der erreichten Knoten nicht in dem selben Maß ansteigt. Während beim Floodingmechanismus die Zahl der besuchten Knoten exponentiell zunehmen kann, erhöht sich beim Random Walk die Zahl lediglich um die Anzahl der zirkulierenden Walker (k). Damit skaliert dieses Verfahren in großen Netzwerken besser als Flooding. Bezüglich des Expanding Ring liegen die Vorteile darin, dass die mehrfachen Abfragen desselben Knotens vermieden werden können und dennoch dieselbe Treffergenauigkeit erreicht werden kann.

3.3 Distributed Hashtables

Im Bereich der Distributed Hashtables (DHT) existieren verschiedene Varianten, die sich in der Form der Lokalisierung nur in begrenztem Maß unterscheiden. In diesem Abschnitt sollen die Verfahren Chord und Freenet vorgestellt werden. Desweiteren existieren zum Beispiel Content Addressable Networks (CAN) [9], Pastry [10] und Tapestry [12].

3.3.1 Allgemeine Funktionsweise

Den meisten der DHT Lokalisierungsmechanismen ist gemein, dass aus allen suchbaren Objekten mit Hilfe eines Hashalgorithmus wie beispielsweise SHA-1 ein Key-Value Paar erzeugt wird. Ebenso können alle Knoten mit Hilfe eines Hashalgorithmus einen Key bzw. Identifier erhalten, dem dann zum Beispiel die IP-Adresse zugeordnet ist.

In großen Netzwerken ist es nicht möglich, dass jeder Knoten einen vollständige Hashtabelle besitzt, die alle restlichen Knoten des Netzwerkes enthält. Dies begründet sich schon allein dadurch, dass es schwierig ist allen Knoten mitzuteilen, wenn neue Knoten zum Netzwerk hinzukommen, bzw. dieses verlassen. Es werden daher Distributed Hashtables verwendet. Hier kennt jeder Knoten nur eine bestimmte Menge weiterer Knoten. Sie werden auch als Nachbarn bezeichnet. In dieser Art von Netzen kennt meist der Knoten die Position eines Objektes, dessen Identifier dem Objekt-Key am nächsten ist. Damit werden Suchanfragen, die grundsätzlich anhand des Objekt-Keys erfolgen, durch Befragung der Nachbarn immer näher zu einem Knoten mit ähnlichem oder gleichem Identifier geleitet. Wurde der Knoten, der das

Objekt besitzt, gefunden, so kann der Wert bzw. Value des entsprechenden Keys an den Ursprungsknoten zurückgegeben werden. Dieser Wert kann entweder das Objekt selbst oder eine Referenz auf das Objekt sein.

Probleme mit den Distributed Hashtables ergeben sich daraus, dass nur Objekte, deren Schlüssel bekannt, ist gesucht werden können. Es besteht keine Möglichkeit, ähnliche Objekte zu finden. Es ist zwar beispielsweise möglich, eine Keywordsuche zu implemtieren, da aber die Suchstrings gehasht werden, können nur Objekte mit genau diesem Suchstring aufgespürt werden. Objekte mit einem ähnlichen Suchstring sind nicht zu finden. Weiterhin besteht bei Distributed Hashtables das Problem, dass Angriffe auf das Netz nicht allein zur Verschwendung von Bandbreite führen können, sondern durch Einführung falscher Knoten- oder Objekt-Keys das Suchen im Netz erheblich beeinträchtigt werden kann.

3.3.2 Chord

Die Vorgehensweise von Chord entspricht weitestgehend der oben im Allgemeinen beschriebenen Form (vgl. [11]). Allerdings wird hier eine Consistent Hashing Funktion verwendet, die jedem Knoten und Objekt-Key einen *m*-bit Identifier zuweist. Der Knotenidentifier ergibt sich dabei aus dem hashen der IP-Adresse, der Keyidentifier aus dem hashen des Keys selbst. *m* muss groß genug sein, um zu verhindern, dass 2 Knoten bzw. 2 Keys den selben Identifier erhalten. Die Identifier sind kreisförmig modulo 2^m angeordnet, d.h ihr Wertebereich reicht von 0 bis 2^m-1. Der Key *k* wird dem im Uhrzeigersinn ersten Knoten zugeordnet, dessen Identifier größer oder gleich dem von *k* ist. Die Suchanfragen an sich werden ebenfalls nur im Uhrzeigersinn weitergeleitet, d.h. die Weiterleitung erfolgt unidirektional.

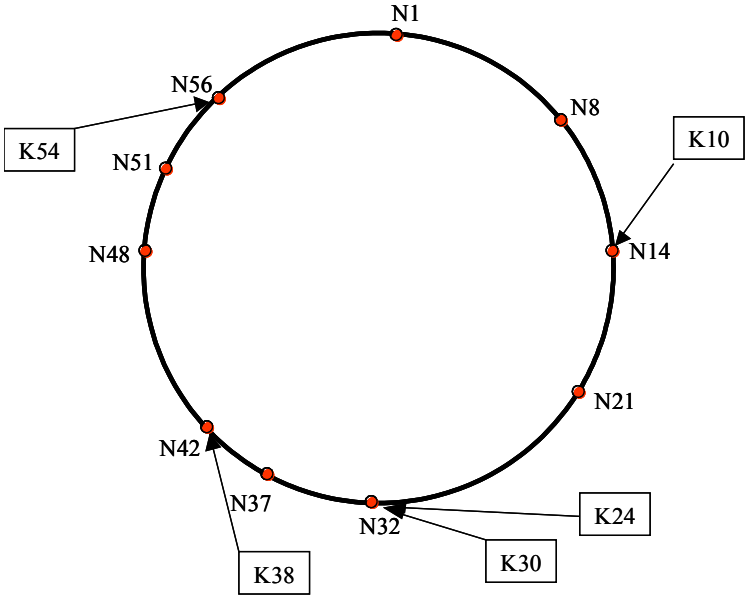

Abbildung 3.3.2/1 Einfacher Chord Ring

Abbildung 3.3.2/1 zeigt einen Kreis von Identifiern, wobei $m = 6$ ist. In diesem Kreis befinden sich 10 Knoten und 5 Keys. Der Folgeknoten des Identifiers 10 ist Knoten 14. Damit ist der Key mit dem Identifier 10 dem Knoten 14 zugeordnet. Weiterhin sind Key 24 und Key 30 bei Knoten 32 zu finden, Key 38 bei Knoten 42 und Key 54 bei Knoten 56.

Kommt ein Knoten hinzu, so werden alle Keys, die zwischen ihm und dem Identifier seines Vorgängers liegen, ihm zugeordnet. Verlässt ein Knoten das Netzwerk, so werden die ihm

zugeordneten Keys seinem Folgeknoten übergeben. Jeder Knoten besitzt eine Stabilierungsmethode, die ein fehlerfreies Ein- und Austreten aus dem Kreis sicherstellt, sowie den Knoten die Möglichkeit gibt, die Gegenwärtigkeit anderer Knoten zu prüfen. Fällt ein Knoten aus, so sind die ihm zugeordneten Schlüssel verloren. Anfragen für die der ausgefallene Knoten Folgeknoten war, werden an den von ihm aus nächsten Folgeknoten geleitet.

Soll nun ein Objekt innerhalb des Netzwerkes lokalisiert werden, so braucht jeder Knoten, ausgehend vom Ursprungsknoten, nur im Uhrzeigersinn seinen Folgeknoten zu fragen, bis der Knoten, der für den Key verantwortlich ist, erreicht wurde. Dieser kann dann den entsprechenden Wert, welcher dem Key zugeordnet ist, an den Ursprungsknoten senden.

Da diese Methode besonders in großen Netzen sehr lange dauern kann, bietet es sich an, den Knoten Kenntnis über eine bestimmte Menge weiterer Knoten zu geben. Damit kann eine Suchanfrage gezielter in das nähere Umfeld des Zielknotens geleitet werden.

Startet beispielsweise Knoten 14 eine Suchanfrage nach Key 54 (Siehe Abbildung 3.3.2/2) und kennt er dabei bereits Knoten 42, so kann er die Anfrage direkt an ihn leiten und die Knoten 32 und 38 übergehen. Kennt Knoten 42 dann beispielsweise Knoten 51, so leitet er die Anfrage an diesen weiter. Knoten 51 wird die Anfrage an Knoten 56 weiterleiten. Dieser kennt den Wert des Keys und wird diesen schliesslich an Knoten 14 zurücksenden.

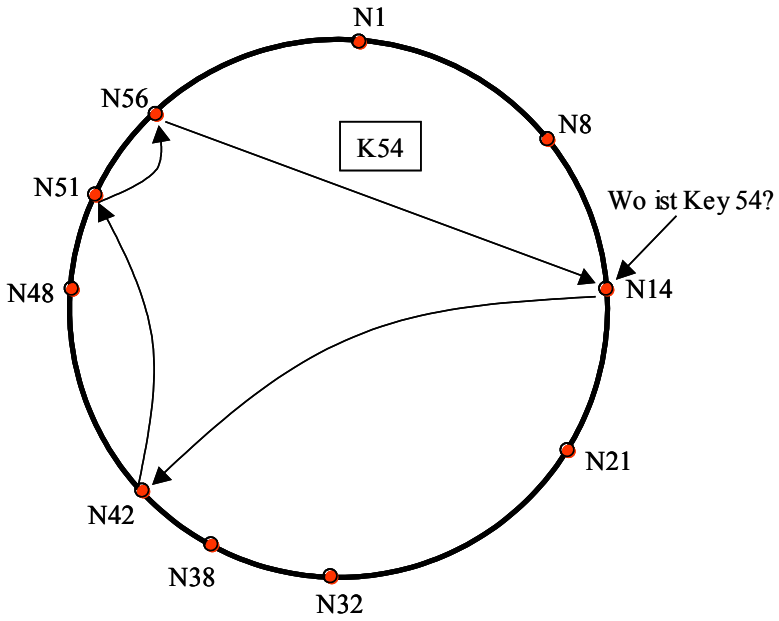

Abbildung 3.3.2/2 Chordring mit Kenntnis weiterer Knoten

3.3.3 Freenet

Auch Freenet verwendet ein Verfahren, dass Daten anhand von Keys innerhalb des Netzwerks findet, wobei jeder Knoten nur eine begrenzte Anzahl von Nachbarn kennt, an die Suchanfragen gesendet werden können. Dies geht einher mit den hohen Anforderungen an Sicherheit und Anonymität. Jeder Knoten besitzt damit nur lokales Wissen über seine nähere Umgebung. Mit der Architektur von Freenet sind folgende Ziele verbunden (vgl. [13]):

- Anonymität für Anbieter und Nachfrager von Informationen

- Keine strafrechtliche Belastbarkeit von Informationsanbietern
- Vermeiden von Informationszensur
- Effiziente dynamische Datenhaltung und Informationsweiterleitung
- Dezentralisation sämtlicher Netzwerkfunktionen

Sämtlich Daten, die innerhalb des Netzwerk ausgetauscht werden, lassen sich anhand von Keys identifizieren. Hierbei werden 3 Formen von Keys unterschieden. Das erste ist der Keyword-signed Key. Hier wird lediglich der Beschreibungsstring, der ein Datenobjekt beschreibt, gehasht. Problematisch ist dabei, dass 2 Nutzer den selben String für 2 verschiedene Objekte vergeben können, womit Datenobjekte nicht mehr eindeutig zu identifizieren wären. Daher existiert zum zweiten der Signed-subspace Key. Der Nutzer hat damit die Möglichkeit, einen eigenen Namesspace für seine Datenobjekte zu generieren. Der Beschreibungsstring des Datenobjektes und die Bezeichnung des Namespaces werden dann getrennt gehasht und XOR verknüpft. Der dritte Key ist der Content-hash Key, Hierbei wird der gesamte Inhalt eines Datenobjektes gehasht. Dies kann sinnvoll sein, um Update und Splitting Funktionen zu realisieren. Stellt ein Knoten eine Suchanfrage, so muss er zunächst den Key ermitteln, anhand welchem er suchen will. Erhält ein Knoten eine Suchanfrage, so überprüft er als erstes seinen eigenen Datenbestand und liefert das entsprechende Datenobjekt zurück, sofern er es besitzt, mit dem Vermerk, dass er die Quelle ist. Hat er das gesuchte Objekt nicht, so sucht er in seiner Routingtabelle nach dem Key der dem Suchkey am ähnlichsten ist und leitet die Anfrage zu dem entsprechenden Knoten weiter. Ist die Anfrage beim Folgeknoten erfolgreich, so wird er das gesuchte Datenobjekt in seinen Datenbestand aufnehmen und die Antwort an den Ursprungsknoten weiterleiten. Ist die Anfrage beim Folgeknoten nicht erfolgreich, so schickt er die Anfrage an den Knoten mit dem zweitbesten Key.

3.3.4 Weitere DHT-Verfahren

3.3.4.1 Kademlia

Kademlia verwendet eine auf XOR basierende Metrik. Im Gegensatz zu den anderen Verfahren ist es möglich, gleichzeitig Konsistenz und Performance, wartezeitminimiertes Routing und eine symmetrische unidirektionale Topologie zu realisieren (vgl. [16]).

Jeder Knoten erhält eine 160 bit ID und stellt ein Blatt in einem binären Suchbaum dar, wobei die Position eines Knotens durch seinen kürzesten eindeutigen Präfix ermittelt wird. Objekte bzw. Key-Value-Paare werden auf den Knoten gespeichert, deren ID dem Key am ähnlichsten ist. Bei einer Suche werden entsprechend der Such-ID die Teilbäume ausgeschlossen, die den entsprechenden Knoten nicht enthalten können. Der größte Teilbaum stellt dabei die Hälfte des gesamten Suchbaums dar. Das Kademliaprotokoll stellt sicher, dass jeder Knoten aus jedem anderen Teilbaum mindestens einen Knoten kennt. Bei einer Suche lässt sich damit die Suchanfrage immer weiter in die Nähe eines Knotens leiten, der den gewünschten Key kennt und dann den Wert des Keys zurückliefern kann.

3.3.4.2 Content Addressable Network (CAN)

Ein CAN stellt ein Netz aus Knoten in einem virtuellen, mehrdimensionalen Koordinaten-system dar (vgl. [2]). Dieses Koordinatensystem wird dynamisch partitioniert, so dass jeder Knoten eine festgelegte Zone innerhalb des Raumes besitzt. Die Objekte bzw. Key-Value-Paare stellen Punkte in diesem Raum dar. Alle Punkte liegen in einer Zone. Damit lassen sich alle Objekte entsprechend ihres Punktes im Raum einem Knoten zuweisen. Um den Wert

eines Objekt-Keys zu erhalten, braucht nur anhand des Keys zu dem entsprechenden Punkt geroutet zu werden. CAN ist fehlertolerant, robust und selbstorganisierend (vgl. [17]).

3.3.4.3 Plaxton

Plaxton verwendet Routing Maps, um Knoten und Objekte zu lokalisieren (vgl. [12]). Dabei werden Nachrichten Stelle für Stelle in die Nähe der entsprechenden ID weitergeleitet (z.B. ***1 → **31 → *731 → 5731, * sind dabei Wildcards). Die Stellen werden von rechts nach links angepasst. Damit dieser Algorithmus funktionieren kann, muss jeder Knoten einen weitere Knoten kennen, für jeden Wert, den eine Stelle einer ID annehmen kann, d.h. geht der Wertebereich der Stellen von 0 bis 9, so muss ein Knoten für jede Stelle der ID mindestens 10 weitere Knoten kennen, die alle möglichen Werte für die jeweilige Stelle einmal annehmen. Kritikpunkt an diesem Verfahren ist erstens die Notwendigkeit eines globalen Wissens zum Zeitpunkt der Netzwerkinitialisierung, was auch das Hinzukommen bzw. Verlassen von Knoten erschwert. Zweitens stellt der Rootknoten, der für ein Objekt verantwortlich ist, eine Schwachstelle dar, da, wenn er ausfällt, das Objekt für andere Knoten unauffindbar werden kann.

3.3.4.4 Tapestry

Die Funktionsweise von Tapestry baut auf der von Plaxton auf (vgl. [12]), beseitigt allerdings die Kritikpunkte, die sich bei Plaxton ergeben haben und bietet darüber hinaus Selbstverwaltung, Robustheit, dynamische Adaption sowie Verminderung von Fehlerquellen und Bandbreitenauslastung.

3.3.4.5 Pastry

Pastry besitzt einige Gemeinsamkeiten mit Tapestry (vgl. [12]). Es wird in beiden Fällen ein Präfix/Suffix Routing verwendet. Sie besitzen ähnliche Algorithmen zum Einfügen und Entfernen von Knoten bzw. Objekten. Unterschiede liegen im wesentlichen bei der Replikation von Objekten auf Knoten entlang des Weges einer Suchanfrage.

3.4 Interessengruppenbasierte Suchansätze

Das Konzept interessengruppenbasierter Suchansätze soll hier beispielhaft mit Hilfe von FUtella, einer contentbasierten Peer-to-Peer Applikation, dargestellt werden (vgl. [14]). Diese Lösung macht sich die positven Eigenschaften von Gnutella, Freenet, JXTA sowie JXTA Search zu nutze. JXTA ist eine Middleware, die Entwicklern die Möglichkeit bietet, auf dieser Grundlage eigene Peer-to-Peer Applikationen zu erstellen. Hierbei ist JXTA Search ein eigener Dienst zur effizienten Bearbeitung von Suchanfragen. Die Suchanfragen gehören zu sogenannten Queryspaces und werden nur an Knoten weitergeleitet, die im Netzwerk für den entsprechenden Queryspace registriert sind.

Der FUtella Architektur liegen folgende Konzepte zugrunde:

- Ein Peer kann entweder ein vollwertiger oder einfacher Peer sein, wobei einfache Peers keine Suchanfragen weiterleiten.
- Peers, die einen gemeinsamen Kompentenzbereich (Queryspace) besitzen, können sich dynamisch zu Interessengruppen (Knowledge Groups) zusammenschließen.
- Interessengruppen registrieren sich im Netzwerk und stehen damit für Suchanfragen, die den entsprechenden Kompetenzbereich betreffen, zur Verfügung.
- Um Broadcastnachrichten wie bei Gnutella zu vermeiden, werden Suchanfragen aus-

schliesslich an Interessengruppen gestellt, die für den entsprechenden Kompetenzbereich registriert sind.

- Das Routing zum Auffinden von Interessengruppen innerhalb des Netzwerks basiert auf dem effizienten Retrieval Routing Mechanismus von Freenet.
- Der Freenet Routingmechanismus wird um die Möglichkeit erweitert, registrierte Interessengruppen wieder zu löschen.
- Für Übertragungen wird das kompakte Nachrichtenformat von Gnutella verwendet.

Um nach Informationen im FUtella Netzwerk zu suchen, muss der Ursprungsknoten eine Interessengruppe kennen, die für den Kompetenzbereich der Suchanfrage zuständig ist. Jeder Knoten besitzt einen Cache, in dem die ihm bekannten Interessengruppen inclusive ihres Kompetenzbereichs enthalten sind. Die Suchanfrage wird an die Interessengruppe gesendet, deren Queryspace dem der Suchanfrage am ähnlichsten ist. Dabei geht die Anfrage immer an den Kopf der Gruppe. Von dort aus wird sie an die anderen Mitglieder weitergeleitet. Die Ergebnisse werden beim Kopf der Gruppe gesammelt und von dort an den Ursprungsknoten zurückgesendet.

Kennt der Ursprungsknoten keine Interessengruppe, an die er die Suchanfrage senden kann, so muss er zunächst eine Suche nach einer geeigneten Interessengruppe starten. Hierbei fragt er zuerst bei Knoten nach, die laut seiner Routingtabelle eine Interessengruppe entsprechend dem Kompetenzbereich der Suchanfrage kennen könnten. Der Knoten, der die Anfrage nach der Interessengruppe erhält, verfährt genauso, sofern er kein entsprechendes Ergebnis liefern kann. Sobald ein Knoten eine positive Antwort liefert, sendet er sie entlang des Pfades, den die Nachricht genommen hat, zurück, wobei die auf dem Pfad liegenden Knoten die Interessengruppe mit ihrem Kompetenzbereich ebenfalls speichern. Nach dem Auffinden der Interessengruppe kann die Anfrage an den Kopf der Gruppe gesendet werden.

Interessengruppen können sich jederzeit im Netzwerk bilden oder wieder auflösen. Sie können zu mehreren Kompentenzbereichen gehören, ebenso, wie es zu einem Kompetenzbereich mehrere Interessengruppen geben kann.

3.5 ALPINE

Ein weiterer Ansatz ist Alpine (vgl. [5]). Es implementiert eine Methode, die von den Entwicklern als „Adaptive Social Discovery Mechanism for Large Peer Based Networks“ bezeichnet wird. Folgende Ideen liegen diesem Ansatz zugrunde.

Alpine verlangt eine direkte Verbindung zwischen jedem Knoten, d.h. Suchanfragen werden nur direkt an andere Knoten gestellt und nicht von Knoten zu Knoten weitergegeben. Dabei kontrolliert jeder Knoten mit welchem anderen Knoten er kommuniziert, wieviel Bandbreite verbraucht wird und wie das Netzwerk genutzt wird. Darüber hinaus wird eine Verbindung länger aufrecht erhalten als bei den anderen Verfahren, damit jeder Knoten eine History über seine Interaktionen mit anderen Knoten aufbauen kann. Diese wird dann verwendet, um ein Profil jedes einzelnen Knotens aufzubauen und zu ermitteln, wie wertvoll er für das Netzwerk ist. Zu boshaften Knoten, die keinen Wert bringen, dafür aber einen hohen Verbrauch von Bandbreite oder anderer Ressourcen aufweisen, kann die Verbindung terminiert werden. Knoten, die keine Inhalte anbieten, werden ebenfalls als wertlos angesehen. Die Terminierung dieser Verbindungen beugt einem Missbrauch des Netzwerks vor.

Das Nachrichtenprotokoll ist so kompakt gestaltet, dass der Nachrichtenoverhead beim Versenden einer Nachricht so gering wie möglich gehalten wird. Dies ist notwendig, da

Suchanfragen eine gewisse Kommunikation zwischen Knoten erfordern und damit den größten Teil der Bandbreite in Anspruch nehmen, besonders, wenn gleichzeitig mehrere Verbindungen offengehalten werden.

Ausgehend vom Ursprungsknoten beginnt die Suche nach Daten bei den Knoten, die in der Vergangenheit die qualitativ hochwertigsten und relevantesten Daten geliefert haben. Diese können anhand des Profils identifiziert werden. Um sich mit einem hochwertigen Knoten verbinden zu können muss der Ursprungsknoten selbst entsprechend hochwertig sein. Dies soll boshaften Knoten und Knoten, die keine Inhalte anbieten, vorbeugen.

Darüber hinaus unterstützt Alpine Peer Groups, d.h. Knoten, die für einen bestimmten Themenbereich hochwertig sind, bilden eine Gruppe. Dies ist sinnvoll, da jeder Knoten nur auf bestimmten Gebieten gute Suchergebnisse liefern kann und das Profil eines Knotens unter Suchanfragen außerhalb seines Kompetenzbereichs leiden würde. Somit kann jeder Knoten seine Suchanfragen direkt an Knoten des entsprechenden Kompentenzbereichs stellen. Die einzelnen Peer Groups entwickeln sich dabei dynamisch im Netzwerk.

Weiterhin haben Nutzer die Möglichkeit eine eigene Menge von Metadaten und Protokollerweiterungen zu definieren. Diesen muss eine eindeutige Erweiterungs ID zugewiesen werden, die mit jeder Suchanfrage verschickt wird. Knoten, die diese Metadaten unterstützen, können dann wesentlich effizienter und genauer Suchanfragen beantworten.

3.6 Heutiger Stand der Technik im Bereich Distributed Filesharing

Die zur Zeit bekanntesten Systeme wie beispielsweise KaZaA oder Morpheus wurden mit Hilfe des Clip2 Reflectors, der es ermöglicht, dass Knoten mit großer Bandbreite als Proxies für Knoten mit geringer Bandbreite fungieren, sowie mit JXTA Search implementiert (vgl. [5]). Anstelle von zentralen Servern werden hier Superpeers verwendet. Dies sind Knoten mit erhöhter Bandbreite und größerer Rechenleistung. Für Suchanfragen sendet ein Knoten die entsprechende Suchanfrage an einen oder mehrere Superpeers und erhält von dort entsprechende Quellenangaben für das Suchobjekt zurück. Superpeers bilden sich abhängig von der Bandbreite und der Rechenleistungen eines Knotens dynamisch innerhalb des Netzwerks.

Eine andere Technik verwendet Overnet, der derzeitige Nachfolger von eDonkey. Die Funktionsweise entspricht im wesentlichen der allgemeinen Vorgehensweise für Distributed Hashtables aus Kapitel 3.3.1. Jeder Knoten kennt eine bestimmte Menge anderer Knoten im Netzwerk. Jeder Knoten und jedes Datenelement werden durch einen 128 bit Identifier gekennzeichnet. Es ist der Knoten für ein Datenobjekt verantwortlich, dessen Key dem des Datenobjektes am ähnlichsten ist. Möchte ein Knoten ein Datenobjekt veröffentlichen, so wird der verantwortliche Knoten ermittelt und das Datenelement ihm zugesandt. Die Suche von Daten erfolgt anhand des Identifiers. Nach Ermittlung des zuständigen Knotens kann von diesem die Liste der Datenelemente abgerufen werden, die dem entsprechenden Identifier zugeordnet sind.

4 Zusammenfassung und Fazit

In diesem Kapitel sollen die einzelnen Namens- und Lokalisierungsmechanismen gegenübergestellt werden. Dies kann anhand von 3 Kriterien geschehen:

Skalierbarkeit: Wie groß kann das System werden, ohne dass die Systemeigenschaften die Systemfunktion übermäßig stören?

Sicherheit: Wie leicht lässt sich das System beispielsweise durch boshafte Knoten stören oder in seinem Ablauf behindern?

Treffer: Wie schnell können in dem System vorhandene Objekte lokalisiert werden?

	Skalierbarkeit	**Sicherheit**	**Treffer**
Flooding (Gnutella)	Skaliert in großen Netzen nicht und kann zum Systemstillstand führen	Anfällig für Denial of Service Angriffe sowie Spam als Antwort auf Suchanfragen	variabel
Random Walk	Skaliert besser als Flooding	Walker können umgeleitet oder gelöscht werden	Abhängig von der Zahl der Walker
Chord	Gut skalierbar	Anfällig für Fehlleitung von Anfragen und Denial of Service Angriffe	In O(log N) Schritten, wenn N die Anzahl der Knoten ist
Freenet	Gut skalierbar	Anfällig für Denial of Service Angriffe	In O(log N) Schritten
FUtella	Gut skalierbar	Keine Angaben	Liefert Treffer in 98% aller Fälle
KaZaA, Morpheus	Gut skalierbar	Anfällig für Denial of Service Angriffe und fehlerhafte Indexierung von Datenbeständen	Keine Angaben
Overnet	Gut skalierbar	Anfällig für Denial of Service Angriffe	Keine Angaben
Alpine	Gut skalierbar	Sicher gegenüber Denial of Service Angriffen	Keine Angaben

Tabelle 4/1 Gegenüberstellung von Lokalisierungsmechnanismen

Bezüglich der Skalierbarkeit lässt sich sagen, dass alle Systeme bis auf Gnutella in der Version 0.4 gut skalierbar und auch in sehr großen Netzwerken mit vielen Ressourcen anwendbar sind. Dies beruht auf den Erfahrungen, die zunächst mit dem Flooding Lokalisierungsmechanismus gemacht wurden. Da dieser einer der ersten Mechanismen in dezentralen verteilten Netzen war, konnten spätere Lösungsansätze die entstandenen Probleme berücksichtigen und beseitigen.

Die Sicherheit gegenüber äußeren Angriffen und Störungen des Systems kann für die meisten Mechanismen noch nicht gewährleistet werden. Generell lässt sich sagen, dass jeder Mechanismus, der auf Vertrauen und Gutgläubigkeit in offenen dezentralen Netzen aufbaut,

Gefahr läuft, Opfer von Missbrauch und boshaften Aktivitäten zu werden. In den meisten Fällen handelt es sich dabei um Erzeugung vieler sinnloser Anfragen um die Auslastung des Netzes bzw. einzelner Knoten, bis zum Zusammenbruch dieser, zu erhöhen. Besonders die Entwickler von Alpine betonen, dass ihr System dieses Problem umgeht. Alpine steht unter der GNU Lesser General Public License und befindet sich noch in der Entwicklung. Ob diese Art von Sicherheit hier wirklich gegeben ist kann zu diesem Zeitpunkt noch nicht mit Bestimmtheit gesagt werden.

Hinsichtlich der Treffer konnten nur teilweise Angaben ermittelt werden. Bei DHT-Mechanismen kann in der Regel die Zahl der Schritte bis zum Auffinden eines Suchobjektes als logarithmische Funktion in Abhängigkeit von der Zahl der Knoten dargestellt werden. Darüber hinaus finden diese Mechanismen das Objekt immer, sofern es im Netzwerk vorhanden ist und keine Störungen vorliegen. In stark dynamischen Netzen, in denen Knoten häufig eintreten und verschwinden, besteht beim interessengruppenbasierten Ansatz von FUtella immer noch eine Wahrscheinlichkeit von 98%, dass ein vorhandenes Suchobjekt gefunden wird. Im Vergleich dazu besteht bei Gnutella 0.4 nur eine Wahrscheinlichkeit von ca. 14% (vgl. [15]).

Derzeit basieren die bekanntesten und am häufigsten verwendeten Applikationen, wie beispielsweise KaZaA oder Morpheus, auf dem Konzept der Superpeers, da hier ein guter Mittelweg zwischen zu zentralen Lokalisierungsmechanismen, wie beispielsweise bei Napster und zu dezentralen wie Gnutella, gefunden wurde. Es existieren weiterer Ansätze wie Alpine, Overnet und FUtella, für die man erst in Zukunft sehen wird, ob sie sich durchsetzen und wo sich eventuell neue Problembereiche ergeben.

5 Verzeichnisse

5.1 Tabellenverzeichnis

5.2 Abbildungsverzeichnis

6 Literatur

1. Kelaskar, M., Matossian, V., Mehra, P., Paul, D., Parashar, M..“A Study of Discovery Mechanisms for Peer-to-Peer Applications“
2. Parashar, M., Kelaskar, M., Matossian, V., Mehra, P., Paul, D.,Vaidhyanathan, A.. “Requirements of Discovery mechanisms in peer-to-peer applications“
3. Minar, N. 2001.“Distributed Systems Topologies: Part 1“. http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html , Abruf am 27.05.2003
4. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S..“Search and Replication in Unstructured Peer-to-Peer Networks“
5. Alpine Network. “Decentralized Resource Discovery in Large Peer Based Networks“. http://cubicmetercrystal.com/alpine/discovery.html . Abruf am 27.05.2003.
6. Tanenbaum, A. S., Steen, M. 2002.“Distributed Systems“, Prentice Hall, London.
7. Bawa, M., Garcia-Molina, H. 2001.“Dampened Broadcast in Peer-to-Peer Networks“. http://dbpubs.stanford.edu:8090/pub/2001-42 . Abruf am 27.05.2003
8. Ritter, J. 2001.“Why Gnutella Can't Scale. No, Really.“ http://www.darkridge.com/~jpr5/doc/gnutella.html. Abruf am 27.05.2003
9. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S. 2001.“A Scalable Content-Addressable Network“.http://www.acm.org/sigs/sigcomm/sigcomm2001/p13-ratnasamy.pdf.
10. Rowstron, A., Druschel, P. 2001.“Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.“.http://research.microsoft.com/~antr/PAST/pastry.pdf.
11. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M. F., Dabek, F., Balakrishnan, H. 2002.“ Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications.“ citeseer.nj.nec.com/stoica01chord.html .
12. Zhao, B., Kubiatowicz, J., Joseph, A. 2001.“Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing.“http://www.cs.berkeley.edu/~ravenben/publications/CSD-01-1141.pdf.
13. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W. 2001.“Freenet: A Distributed Anonymous Information Storage and Retrieval System“. http://citeseer.nj.nec.com/clarke00freenet.html
14. Zahn, T., Ritter, H., Schiller, J., Schweppe, H. 2002.“FUtella Analysis and Implementation of a Content-Based Peer-to-Peer Network“. http://page.inf.fu-berlin.de/~zahn/FUtella_netties.pdf .
15. Zahn, T., Ritter, H., Schiller, J., Schweppe, H..“FUtella Performance Analysis of a Content-Based Peer-to-Peer Infrastructure“. http://page.inf.fu-berlin.de/~zahn/FUtella_Berkeley.pdf .
16. Maymounkov, P., Mazieres, J. 2002.“Kademlia: A Peer-to-peer based information System Based on the XOR Metric“. http://citeseer.nj.nec.com/529075.html . Abruf am 07.06.2003

17. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S. 2001.“A Scalable Content-Addressable Network“. http://citeseer.nj.nec.com/460728.html . Abruf am 07.06.2003