Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Praktische Informatik und
Medieninformatik
Fachgebiet Telematik

# Studienjahresarbeit

# Agentenbasierte Suche in Peer-To-Peer Netzen

Elke Gerdes

Juni 2004

Betreut durch Dipl.-Inf Thorsten Strufe

# Inhaltsverzeichnis

# 1. Ziel und Motivation der Arbeit

In den letzten Jahren haben immer mehr Nutzer ihre Personal Computer (PC) an das Internet angeschlossen. Das wurde möglich durch die höhere Verfügbarkeit von leistungsstarken Rechnern im privaten Bereich, aber auch durch die Bereitstellung von mehr Bandbreite durch die Netzbetreiber. Hierdurch waren dann die Bedingungen bereitgestellt, die eine Zusammenarbeit der zum Teil räumlich weit entfernten Nutzer möglich machen.

## 1.1. Was ist ein Peer-to-Peer Netz ?

Ein Peer-to-Peer (P2P) Netz ist ein Netzwerk, welches aus einer großen Anzahl von Teilnehmern besteht. Jeder Teilnehmer kann sich jederzeit dem Netz anschließen, aber auch das Netz verlassen. Jeder Teilnehmer ist mit einer relativ geringen Anzahl von weiteren Teilnehmern verbunden. Diese Teilnehmer werden als Nachbarn bezeichnet.

Es wird klar, daß P2P keine komplexe Technologie darstellt. Auch ist der Gedanke nicht neu. Ein offenkundiger ökonomischer Nutzen ist nicht sichtbar. Allerdings hat der Ruf der Peer-to-Peer Netze in den letzten Jahren stark gelitten. Das lag vor allem an der weiten Verbreitung sogenannter Tauschbörsen. Hier wurden, bzw. werden, Daten zwischen den Nutzern der Tauschbörsen ausgetauscht. Oft wurden hier Urheberrechte verletzt oder illegale pornographische Darstellungen verbreitet, was dann auch zu rechtlichen Auseinandersetzungen führte.

Ohne die Akteure würden wohl keine P2P Netze existieren. Die Akteure handeln aktiv, indem sie P2P Produkte entwickeln und vermarkten. Kommerzielle Akteure versuchen vor allem sich ein Marktsegment zu erschließen und Kunden zu akquirieren, während Nichtkommerzielle Akteure vor allem Öffentlichkeitsarbeit leisten und auf Tagungen in Erscheinung treten. Einzelpersonen forschen meistens individuell oder aus akademischen Gründen. Zahlreiche P2P-Projekte wurden von Universitäten gestartet und unterstützt. Viele dieser Projekte unterstützen sogar den Betrieb der P2P Infrastruktur. Aber auch Unternehmen beteiligen sich im P2P Bereich und veröffentlichen Studien über eigene Marktsegmente und technologische Vergleiche.

Hinter dem Begriff P2P verstecken sich viele verschiedene Anwendungen. Unter Groupware versteht man das Zusammenarbeiten, Kooperieren oder Kommunizieren von Teilnehmern. Das Instant Messaging beschäftigt sich unter anderem mit der IP-Telefonie oder IP-

Videokonferenzen. Distributed Computing ermöglicht das verteilte Rechnen. Hinter dem Begriff Filesharing stehen Tauschbörsen wie KaZaa oder eDonkey. Auch im Bereich Gaming und E-Commerce ist P2P nicht unbedeutend.

Um alle diese verschiedenen Anwendungen zu unterstützen ist auch eine spezielle P2P Infrastruktur nötig. Diese Infrastruktur besteht aus Netzwerken, die sowohl virtuell organisiert sein können, wie beispielsweise bei Gnutella, als auch physikalisch organisiert sein können, wie beispielsweise Bluetooth, Firewire (IEEE 1394) oder Wireless LAN (WLAN). Um den Netzbetrieb zu ermöglichen werden des Weiteren Protokolle benötigt. Beispiele für diese Protokolle sind das Gnutella Protokoll oder die JXTA Protokollfamilie.
Außerdem existieren bereits relevante Standards und Normen. Zusätzlich gibt es weitere Standardisierungsbestrebungen.
Weiterer Bestandteil der Infrastruktur sind sogenannte Schlüsseltechnologien, wie die Extensible Markup Language (XML) oder dem XML Remote Procedure Call (XML RPC).
Grundlegende Zugangstechniken für P2P sind beispielsweise das Kabelnetz oder xDSL-basierte Zugänge zum Internet.

## 1.2. Gegenstand der Arbeit

P2P umfasst alle möglichen Anwendungsfelder, die unter Zuhilfenahme eines vernetzten Computers bearbeitet werden können. Damit ist P2P keine homogene Technologie oder festgelegte Anwendung, sondern ein Paradigma.

Aufgrund des erheblichen Wachstums im Bereich des P2P steigt auch die Komplexität der Aufgaben. Die Skalierbarkeit der verschiedenen P2P Systeme wird mit zunehmender Anzahl von Teilnehmern immer wichtiger. Schließlich sollen die Netze weiter wachsen können und weitere Dienste, die erst bei vielen Teilnehmern möglich oder sinnvoll sind, anbieten können. So sind doch die Teilnehmer das eigentliche Kapital und der eigentliche Wert eines P2P Systems. Allerdings steigt mit der Anzahl der Teilnehmer nicht nur der Nutzen; auch die Probleme werden größer.

Eines dieser Probleme ist das Suchen und Finden von Ressourcen in einem P2P Netzwerk. In kleinen Netzwerken ist die Suche relativ leicht durchzuführen und führt auch schnell zu einem Ergebnis. Das Problem der Ressourcenverschwendung zum Auffinden eines Objektes spielt in diesem Fall nur eine untergeordnete Rolle, da das System bis zu einer bestimmten Anzahl von Teilnehmern weiterhin funktionsfähig bleiben kann.
Leider ist das Aufspüren von Daten, oder das Information Retrieval, in größeren Netzwerken weniger einfach. Das Netz muß so weit skalierbar sein, daß auch bei vielen Teilnehmern und vielen verteilten Ressourcen eine Suchoperation jederzeit möglich ist ohne das ganze Netz lahmzulegen.

Diese Arbeit will sich dementsprechend mit den verschiedenen Suchstrategien in P2P Netzwerken beschäftigen. Diese Strategien sollen vorgestellt und evaluiert werden. Das Ziel ist das Auffinden einer oder mehrerer praktikabler Suchstrategien, die auch in großen Netzwerken anwendbar sind.
Um dies zu erreichen werden verschiedene Netzwerktopologien betrachtet. Es werden sowohl bereits bestehende P2P Systeme betrachtet, wie auch Alternativen zu P2P aufgezeigt.

# 2. Alternativen zu P2P

Wer glaubt daß man nur mit P2P Anwendungen Filesharing betreiben kann, der irrt. Schon bevor P2P populär wurde, gab es Möglichkeiten Daten im Internet zu tauschen. Da das Ziel dieser Arbeit das Suchen und Auffinden von Informationen im Netz ist, soll hier nur auf die Möglichkeiten des Filesharing eingegangen werden. Möglichkeiten zur Realisierung von Instant Messaging, IP-Telefonie oder Distributed Computing werden hier nicht berücksichtigt.

## 2.1. Das Usenet

Das Usenet ist traditionell eine Sammlung von Diskussionsforen zu vielen verschiedenen Themen. Es verfügt über ein eigenes Protokoll und einer verteilten Serverstruktur.

Das Usenet wurde 1979 von Tom Truscott und Jim Ellis entwickelt. Das UNIX User Network wurde gestaltet als ein Verbund von Rechnern. Charakteristisch war, daß diese Rechner nicht über eine Standleitung verfügten, sondern über Wählverbindungen mit dem Internet verbunden waren. Zum Austausch von Nachrichten in den verschiedenen Gruppen diente das Protokoll UNIX to UNIX Copy, kurz: UUCP. Um Nachrichten auszutauschen muß sich der betreffende Rechner mit den Servern verbinden. Von diesem werden dann neue Nachrichten heruntergeladen. Neue Nachrichten können dann auch hochgeladen werden. Da kein zentraler Server existiert kann der Nutzer dabei sowohl die Rolle als Server als auch als Client einnehmen.

Ab dem Jahre 1986 wurde ein neues Protokoll mit Namen NNTP eingeführt. Der Grund hierfür war daß die Wählverbindungen immer mehr von Standleitungen abgelöst wurden. In diesem Zuge wurde dann auch das Anlegen von Gruppen völlig dezentralisiert. Die verfügbare Bandbreite der Teilnehmer nahm kontinuierlich zu. Daher konnte eine neue Hierarchie eingeführt werden, die alt.binaries-Hierarchie, die gedacht war um Binärdateien anzubieten. Ursprünglich wurden hauptsächlich kleine Sounddateien oder pornographische Dateien getauscht. Heutzutage ist das Angebot merklich angewachsen. So ist es mittlerweile auch möglich MP3, Programme, Videos und sonstige Dateien anzubieten.

Der Client muß keine kompletten Dateien herunterladen. Er lädt lediglich eine Liste bestehend aus Kopfzeilen, Inhaltsinformationen und Absenderinformationen herunter. Der

Benutzer durchsucht diese Liste, die durchaus bis zu mehreren Tausend Einträge enthalten kann, und markiert dann die Dateien, die er herunterladen möchte.

Als Nachteil ist festzuhalten, daß die auf dieser Weise zur Verfügung stehende Datenmenge geringer ist als bei einer Tauschbörse wie beispielsweise Napster. Aufgrund der interessenbedingten Gruppenbildung ist das Angebot besser sortiert.

Es hat immer wieder Versuche gegeben die Usenet Gruppen zu zensieren. Diese Versuche waren allerdings wenig erfolgreich. Der Grund hierfür ist, daß nicht jeder Server jede eingehende alt-Gruppe auf ihre Legitimität prüfen kann.

Um Usenet verwenden zu können ist der Einsatz einer speziellen Software nötig. Sowohl Netscape als auch Microsoft legen ihren Browsern kombinierte Newsreader und Mailer bei. (vgl. auch [HEISE1]) Diese Software ist für die Teilnahme an Diskussionen hinreichend. Wenn aber Dateien getauscht werden sollen, sollte andere Software verwendet werden, die für diesen Zweck besser geeignet ist, und für viele Betriebssysteme erhältlich ist.
Auch in Zukunft wird Usenet wahrscheinlich nicht verschwinden. Es ist als Tauschbörse gut geeignet. Die wachsenden Bandbreiten und Massenspeicher kommen dem Usenet weiter entgegen. Eine allgemeine Zensur des Usenet ist schwierig und würde das Recht auf freie Meinungsäußerung verletzen. Und freie Meinungsäußerung möglich zu machen ist schließlich das erklärte Ziel von Usenet.

## 2.2. Der Internet Relay Chat

Ein weiterer „Klassiker“ im Sinne des Filesharing ist der Internet Relay Chat (IRC), der allerdings ursprünglich nicht zum Tauschen von Dateien konzipiert worden war. Der Internet Relay Chat wurde 1988 von Jarkko Oikarinen als Textzeilen Kommunikationsdienst entwickelt. Hier kann jeder Teilnehmer themenspezifische Gruppen, die auch als Channels bezeichnet werden, erzeugen. Danach können dann unter Benutzung der Computertastatur Gespräche geführt werden. Der Teilnehmer tippt eine Zeile ein und schickt sie dann in den Channel. Das durch die Zusammenschaltung von IRC-Servern entstehende Netz bildet eine Kette. Wenn einer der Server ausfällt kann diese Kette in zwei Teile auseinandergerissen werden. Hier wird dann auch der Unterschied zu Usenet deutlich. Anders als hier gibt es im IRC kein einzelnes IRC-Netz, sondern viele unterschiedliche Netze.

Der direkte Austausch von Dateien ist über den Internet Relay Chat so nicht möglich. Allerdings stellt er für den Nutzer eine Hilfe dar um „Gleichgesinnte“ über ein textbasiertes Medium ausfindig zu machen. Der eigentliche Austausch von Dateien erfolgt dann über eine Verbindung zwischen den beteiligten Nutzern. Diese Verbindung wird als Client to Client Transfer (DCC-Transfer) bezeichnet.

Moderne Clienten besitzen Skript Fähigkeiten. Dieses kann dann verwendet werden um beispielsweise die lokale Datenhierarchie mittels einer Textoberfläche zugänglich zu machen. Der Dateiserver eines Nutzers (FServe) kann aufgerufen werden, indem im Channel ein Trigger angegeben wird.

Zum Beispiel:

> Enter “!Search Pink Floyd“ for Pink Floyd MP3s!
>
> Der Nutzer gibt dann allgemein sichtbar “!Serarch Pink Floyd“ ein. Daraufhin wird

eine DCC-Verbindung zum Rechner von MP3 Pink Floyd hergestellt. Ein Chat, bzw. ein Gespräch, wird dann aber nicht begonnen. Der Client fängt statt dessen die Nachrichten ab um sie zu interpretieren. Der Anfrager gelangt in das Hauptverzeichnis des Servers, kann in andere Verzeichnisse wechseln und Dateien anfordern.

Auch im Internet Relay Chat ist die Auswahl an Dateien nicht so umfangreich wie bei den allgemein bekannten Tauschbörsen. Anderseits ist die Suche thematisch präziser, da jeweils in unterschiedlichen Channels gesucht wird.

Auch beim Internet Relay Chat ist wie beim Usenet eine spezielle Software nötig. Der Internet Relay Chat wurde nicht für den Austausch von Dateien entwickelt, aber er bietet eine gute Möglichkeit Nutzer mit ähnlichen Interessen zu finden und mit diesen zu kommunizieren. Auf diese Weise ist das Herstellen von Tauschkontakten nicht sehr schwierig. Auch hier ist von rechtlicher Seite ein Unterbinden der Tauschkontakte erstrebenswert. Allerdings ist auch hier eine Zensur nicht einfach. Werden Channels aufgrund von Tauschaktionen geschlossen, kann jeder Nutzer jederzeit neue Channels aufbauen.

In Zukunft wird der Internet Relay Chat möglicherweise durch spezialisiertere und komfortablere Systeme zum Tauschen von Dateien abgelöst (vgl. [HEISE1]). Dennoch wird der Internet Relay Chat auch weiterhin zur Verfügung stehen. Der Internet Relay Chat wird möglicherweise in Zukunft breiten Zulauf erhalten, falls die entsprechenden Spezialsysteme aufgrund rechtlicher Auseinandersetzungen verboten werden.
Ein weiterer Vorteil von IRC ist der geringe Bedarf an Bandbreite. Schließlich werden nur Textzeilen ausgetauscht.

## 2.3. Das File Transfer Protocol

Das File Transfer Protocol (FTP) ist leider nicht sonderlich komfortabel. Es handelt sich hierbei um ein Client-Server Protokoll. Dieses wurde 1985 für das Tauschen von Textdateien und Binärdateien entwickelt.

Die Daten werden in Archiven gespeichert. Die Nutzer besitzen jeweils unterschiedliche Rechte zum Upload und zum Download. Jeder Nutzer kann sich selber einen FTP-Server installieren. Mithilfe von speziellen Suchmaschinen oder auch im Internet Relay Chat können FTP-Server aufgefunden werden.

Der Gebrauch von spezieller Software ist für FTP nicht erforderlich. Die Browser stellen in der Regel bereits einen FTP-Clienten zur Verfügung.

## 2.4. Tauschen im World Wide Web

Es ist auch möglich im World Wide Web (WWW) Dateien auszutauschen. Mehrere Möglichkeiten stehen hierzu zur Verfügung.

Zunächst gibt es die Möglichkeit zur Nutzung von Webspace. Webspace Anbieter bieten den Nutzern Platz um Daten zu speichern. Diese Daten können dann von jedermann über einen Browser abgerufen werden.

Speicherplatzanbieter bieten eine weitere Möglichkeit zum Datentausch. Sie bieten auch Platz zur Speicherung von Daten an. Es gibt aber einen wichtigen Unterschied zu den

Webspaceanbietern. Bei den Speicherplatzanbietern ist ein bestimmter Speicherbereich nur dem Besitzer zugänglich. Der kann dann sein Passwort an andere Personen weitergeben und so einer Anzahl von Personen den Zugriff erlauben.

In sogenannten Clubs ist es den Mitgliedern möglich eigene Dateien auf einem Server zu speichern. Allerdings ist in diesen Clubs oft eine Registrierung notwendig. Manchmal werden neue Mitglieder auch überprüft.

Diskussionsforen bieten den Nutzern die Gelegenheit sich zu treffen und Links zu veröffentlichen. Der Austausch von Dateien kann dann direkt oder auf jedem beliebigen anderen Weg erfolgen.

## 2.5. Fazit

Alle vorgestellten Methoden haben eins gemeinsam. Sie wurden allesamt nicht zum Tauschen von Dateien entwickelt, sondern sollten einem ganz anderen Zweck dienen. Es ist allerdings anzumerken, daß keine dieser Methoden unterscheidet, ob die Nutzung dem eigentlich bestimmten Gebrauch entspricht, oder ob sie zweckentfremdet oder sogar mißbraucht werden.

Der Tauschbörse Napster wurde unterstellt, daß sie von Anfang an dazu entwickelt wurde um illegale Musikdateien auszutauschen. Das war dann auch der Grund, daß es zu einem rechtmäßigen Verbot dieser Tauschbörse kam. Dies kann natürlich auch in Zukunft auf andere Tauschbörsen zutreffen.

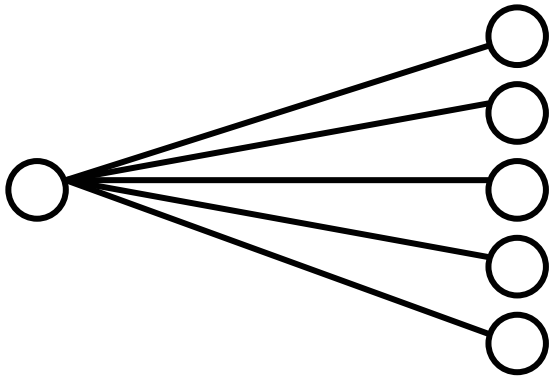
Den „klassischen“ Methoden kann man diese Gründe nicht unterstellen oder nachsagen. Weil sie aus ganz anderen Gründen und zu einem ganz anderen Zweck entwickelt wurden, ist es auch nicht zu erwarten, daß sie auf dieser Grundlage per Gerichtsbeschluß verboten werden können.

# 3. Topologien für Peer to Peer Netze

Bei Computer Netzwerken, dementsprechend auch bei P2P Netzen, gibt es verschiedene Netzwerktopologien. Eine Topologie, oder Anordnung, hat Auswirkungen auf die Funktion des Netzwerkes. Das ist auch der Grund weshalb im Rahmen dieser Arbeit auf Netzwerktopologien eingegangen werden soll.

## 3.1. Zentralisierte Netzwerktopologie

Die bekannteste Netzwerktopologie ist wahrscheinlich die zentralisierte Topologie. Diese Topologie wird oft als Client-Server Architektur realisiert. Der Server stellt in diesem Fall Informationen und/oder Dienstleistungen zur Verfügung. Beispiele hierfür sind ein Fileserver, eine Datenbank oder die Übernahme von Berechnungen. Die Clienten stellen Anforderungen an den Server. Dafür verbinden sich die Clienten direkt mit dem Server. (vgl. Abb. 3.1.(1))



| Abb. 3.1.(1) | Darstellung einer zentralisierten Netzwerktopologie |
|---|---|

Viele P2P Anwendungen setzen auf eine zentralisierte Architektur auf. Beispiele sind SETI@home oder die ursprüngliche Version der Tauschbörse Napster.
Bei SETI@home verteilt ein Server die anstehenden Jobs auf die Clienten. Die Architektur ist vollständig zentralisiert.

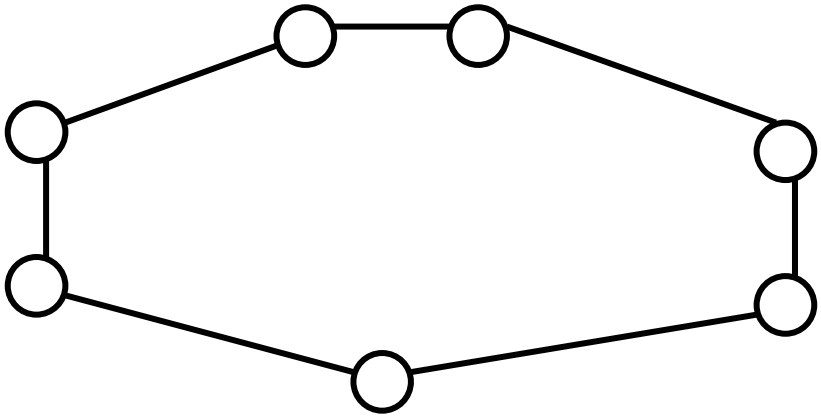
Bei der Tauschbörse Napster in seiner ursprünglichen Version war die Suchmaschine zentral organisiert. Der Server enthielt eine Datenbank, in der alle verfügbaren Musiktitel verzeichnet waren. Hier suchten die Clienten nach dem gewünschten Titel. Die Verzeichnisse und die Suche darin waren hier also zentral organisiert. Der eigentliche Austausch der Dateien hingegen fand dezentral statt. Hier war der Server nicht mehr beteiligt, da der Austausch direkt zwischen den beteiligten Clienten stattfand.

Diese zentralisierte Topologie hat Vorteile. Der wichtigste Vorteil liegt in der Einfachheit der Architektur (vgl. auch [MINAR1]) Die Dateien können damit an einem Ort zusammengefaßt werden. Aufgrund dieser Eigenschaft wird das System leichter verwaltbar und es gibt keine Probleme mit der Datenkonsistenz.
Weiterhin ist es auch einfacher das System gegen Angriffe von außen zu sichern. Das spielt insbesondere auch wegen der immer häufiger werdenden Schadensfälle eine immer größere Rolle. Bei der zentralisierten Architektur muß nur ein Teilnehmer, nämlich der Server, gegen Manipulation geschützt werden.

Allerdings ergeben sich aus der zentralisierten Topologie auch Nachteile (vgl. auch [MINAR1]). Der Server stellt den „single point of failure“ dar. Das bedeutet, daß das ganze System zusammenbricht, wenn der Server ausfällt. Schließlich ist der Server für die Clienten absolut notwendig.
Ein weiteres Problem stellt die rechtliche Seite dar. Aufgrund der zentralen Rolle des Servers ist es relativ einfach das ganze Netzwerk per Gesetz zu Fall zu bringen. Das ist so geschehen bei der Tauschbörse Napster. In einem Gerichtsverfahren ist es nicht nötig gegen alle Clienten vorzugehen. Es ist hinreichend wenn der Server Gegenstand eines Gerichtsverfahren ist.
Schwierig ist auch das Einfügen von neuen Dateien. Hier erweist sich die zentralisierte Architektur als sehr unflexibel. Der Server allein kann neue Daten aufnehmen und verwalten. Wer neue Daten einfügen möchte, braucht dazu das Einverständnis des Betreibers des Servers.
Auch das Hinzufügen von Clienten ist nur sehr begrenzt möglich. Ein einzelner Server kann nur eine begrenzte Anzahl von Clienten bedienen. Der Grund hierfür ist die Begrenztheit des Servers selber. Insbesondere dieses Problem ist möglicherweise nicht so gravierend wie es zunächst scheint. Abhängig von der technischen Ausstattung des Servers und der benötigten Anwendung wächst die Anzahl der Clienten möglicherweise nicht so stark an wie die Verbesserung des technischen Standards. In diesem Fall wird der Grenzfall nicht erreicht.

## 3.2. Die Ringtopologie

Der Ring versucht die Probleme der zentralisierten Netzwerktopologie so weit es geht auszugleichen. Ansatzpunkt ist es die Last vom Server zu nehmen wenn die Anzahl der Clienten zu groß wird. Zu diesem Zweck wird ein Ring aus Servern gebildet. Dieser Ring arbeitet dann als verteilter Server.
Die Teilnehmer des Ringes tauschen untereinander Zustandsdaten aus. Dann kann jeder Teilnehmer die gleiche Funktion erfüllen. Die Last wird also auf eine Gruppe von Servern verteilt. Damit ist ein größerer Schutz gegen Ausfälle gewährleistet (vgl. Abb. 3.2.(1))



| Abb. 3.2.(1) | Darstellung einer Ringtopologie |
|---|---|

Auch diese Topologie hat Vorteile (vgl. auch [MINAR1]). Der Ring hat insbesondere die Vorteile der zentralisierten Topologie. Der Grund hierfür ist daß ein Ring meistens nur einen Besitzer hat. Außerdem ist der Ring besser gegen Ausfälle gesichert. Wenn ein Teilnehmer versagt, kann die Funktionsweise des Ringes einen Totalausfall vermeiden. In diesem Fall können die anderen Teilnehmer die Aufgaben des ausgefallenen Servers übernehmen.
Neue Server können dem Ring leicht zugefügt werden. Dadurch kann die Kapazität des Ringes erhöht werden.

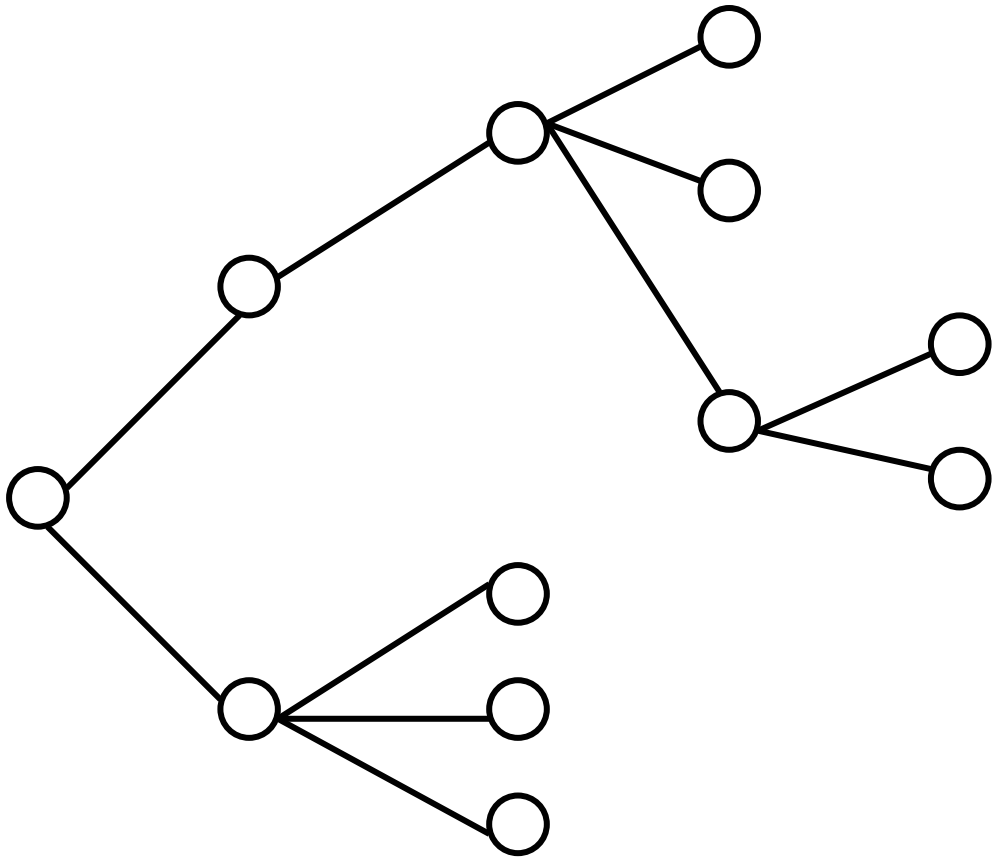
Ein Nachteil des Ringes ist, daß wie auch bei der zentralen Topologie, das Hinzufügen von Dateien schwierig, und nur mit dem Einverständnis des Betreibers möglich ist. Die Verwaltung und Speicherung bleiben weiterhin zentral organisiert.
Auch bei der Verwundbarkeit des Systems gegen gesetzliche Angriffe konnte durch den Ring nicht verkleinert werden.

# 3.3. Hierarchische Topologie

Eine hierarchische Struktur bildet eine klare Kette (vgl. Abb. 3.3.(1)). Das System ist dadurch leichter vorhersehbar, verwaltbar und durch Dateien erweiterbar. Jeder Teilnehmer kann Daten oder Informationen einfügen. Dies kann jedoch durch Regeln des Managements eingeschränkt werden.

Die hierarchische Architektur ist relativ robust gegen Ausfälle. Fällt ein Teilnehmer aus, kann ein anderer Teilnehmer der Hierarchie dessen Aufgaben übernehmen.
Angriffe seitens der Gerichtsbarkeit werden schwieriger, da nicht mehr nur ein Server angegriffen werden muß, sondern alle Teilnehmer, die beteiligt werden.

Als Nachteil ist anzumerken, daß der Wurzelknoten der „single point of failure“ ist. Fällt er aus, so fällt auch der Rest des Systems in sich zusammen. Es ist bei dieser Architektur auch schwieriger das System gegen Angriffe oder Manipulationen von außen zu schützen. Hier müssen alle beteiligten Knoten gesichert werden, nicht nur ein zentraler Server.



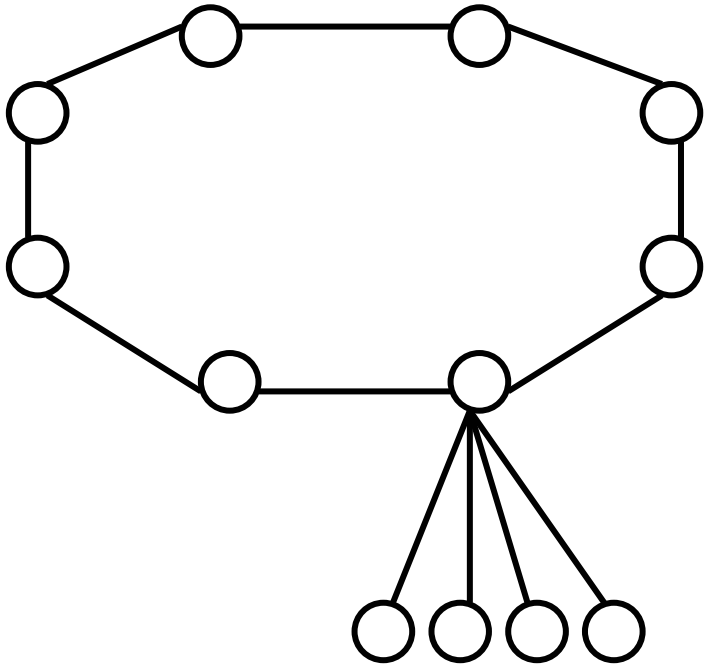
| Abb. 3.3.(1) | Darstellung einer hierarchischen Topologie |
|---|---|

## 3.4. Hybridtopologien

Die vorgestellten Topologien sind allesamt nicht in der Lage zufriedenstellen zu arbeiten. So entsteht das Bedürfnis eine Topologie zu entwickeln, die alle Vorteile der bekannten Topologien vereint, ohne die Nachteile zu enthalten. Die Vereinigung der Vorteile soll durch eine Vereinigung der Topologien zu einer Hybridtopologie ermöglicht werden. Für die Schaffung einer Hybridtopologie gibt es viele Möglichkeiten. Da die Anzahl möglicher Hybridtopologien groß ist, da praktisch alle Topologien beliebig miteinander kombiniert werden können, soll hier nur eine Möglichkeit, nämlich die Kombination aus zentralisierter Topologie und Ring, exemplarisch vorgestellt werden (vgl. Abb. 3.4.(1)).

Der Vorteil dieser Topologie liegt in der Redundanz des Ringes (vgl. auch [MINAR1]). Außerdem hat diese Topologie alle Vorteile der zentralisierten Topologie, zuzüglich der Erleichterung neue Daten einzufügen. Es wird nur wenig Komplexität hinzugefügt in Gegensatz zur Verwendung eines einzelnen zentralen Servers.

Diese Hybridarchitektur erfreut sich vor allem im Bereich des Web Commerce, aber auch bei hochverfügbaren Datenbanken großer Beliebtheit.



| Abb. 3.4.(1) | Darstellung einer Hybridtopologie |
|---|---|

## 3.5. Dezentrale Topologien

Dezentrale Topologien zeichnen sich damit aus, daß sie ohne einen zentralen Server auskommen. Hier wird die Last auf alle Teilnehmer oder Clienten verteilt. Diese Clienten können sowohl gleichberechtigt sein, als auch unterschiedliche Rechte und Pflichten haben. Diese Rechte und Pflichten können dann über vorher festgelegte Metriken, wie zum Beispiel die Proszessorleistung, Speicherplatz oder verfügbare Bandbreite, bestimmt werden.

Es gibt eine große Anzahl verschiedener dezentralen Topologien. Jede von ihnen erfüllt einen bestimmten Zweck und ist für bestimmte Anwendungen optimiert. Auch für die Suche von Informationen und Dateien sind bestimmte Topologien für bestimmte Algorithmen und Verfahren besser geeignet als andere. Ausgewählte Topologien werden, soweit benötigt, in dem Kontext, in dem sie bei der Suche nach Dateien und Informationen benötigt werden, an der betreffenden Stelle näher erläutert.

# 4. Dezentrale Beispiele aus der Praxis

Abgesehen von den Klassikern (vgl. Kapitel 2) gibt es auch schon „echte“ P2P Systeme, die sich im praktischen Betrieb schon bewährt haben. Da in dieser Arbeit die Suche von Dateien und Informationen untersucht werden soll, sollen in diesem Kapitel insbesondere Tauschbörsen vorgestellt werden (vgl. auch [HEISE1])

## 4.1. Gnutella

Justin Frankel war Chefentwickler des MP3 Players Winamp. Sein Ziel bei der Entwicklung von Gnutella war die Erstellung einer Open Source Software zum Tauschen von Dateien. Das „GNU“, welches für „General Public License“ steht, ist im Namen von Gnutella enthalten. Die Bearbeitung des Systems lief zunächst von der Öffentlichkeit weitgehend unbeachtet ab. Aber obwohl das Programm noch nicht fertig war, erhöhte sich die öffentliche Aufmerksamkeit nennenswert.
Seit 1999 gehörte Frankels Softwarefirma Nullsoft zu AOL. Die offizielle Entwicklung von Gnutella wurde eingestellt und der Quellcode nicht veröffentlicht. Das öffentliche Interesse an dem Programm nahm aber dennoch weiter zu. Vermutlich lag das auch an der Berichterstattung in den Medien. In einer IRC Sitzung gab ein anonymer Teilnehmer Details über das Gnutella Protokoll bekannt. Dieses Detailwissen führte zur Entstehung von Gnutella Clones. Heute ist das Protokoll von Gnutella vollständig dokumentiert.

Gnutella ist ein Netzwerk aus Rechnern. Es werden Suchanfragen gestellt und beantwortet. In der ersten Version des Programmes verband sich der interessierte Rechner beim Start mit dem Server „findshit.gnutella.org“. Dieser Server versorgte den Rechner mit gültigen IP Adressen. Damit war es nicht mehr erforderlich mühsam die IP Adressen anderer Gnutella Nutzer herauszufinden.

Der eigene Rechner pflegt dann eine gewisse Anzahl von Verbindungen mit anderen Rechnern. Außerdem läßt er es zu daß sich andere Rechner mit ihm verbinden. Soll eine Suchanfrage gestartet werden, wird jedem Nachbarn diese Anfrage zugeschickt. Diese Anfrage enthält auch einen Time to Live (TTL) Zähler. Jeder Rechner, der diese Anfrage erhält, subtrahiert 1 vom aktuellen TTL Wert und leitet die Anfrage erst dann weiter.

Ein Rechner, der eine Anfrage erhält sucht in seiner lokalen Datenliste nach Treffern. Diese werden dann auf dem gleichen Weg zum Absender der Anfrage geschickt, den schon die Anfrage genommen hat. In der Theorie sind aufgrund dessen, daß viele Zwischenstationen passiert werden, Anfrage und Antwort anonym. Sie können daher nicht einer speziellen IP Adresse zugeordnet werden. In der Praxis ist das aber anders. In der Antwort muß die IP Adresse enthalten sein. Andernfalls wäre ein Dateiaustausch nicht möglich. Der Dateitransfer wird direkt abgewickelt. Es wird das Übertragungsprotokoll Hypertext Transfer Protocol (HTTP) verwendet. Das hat den Vorteil daß Webbrowser auf Gnutella Dateien zugreifen können. In der Gnutella Gemeinde ist diese Eigenschaft jedoch sehr unbeliebt. Es ist nämlich möglich das Gnutella Netz zur Suche nutzen kann, ohne daß der betreffende Nutzer selber Dateien zum Tausch anbietet.

Auch problematisch sind die sogenannten PUSH Requests. Notwendig sind sie, da Teilnehmer, die sich hinter Firewalls befinden, die Dateien nicht über herkömmliche Wege übertragen können. Auch wenn die PUSH Requests den gleichen Weg nehmen sollen wie die Suchantworten, gibt es einige Implementierungen, die sie über das ganze Netz verteilen.

Weitere Probleme im Gnutella Netzwerk bereiten die sogenannten Pings und Pongs. Diese werden über das ganze Netzwerk geroutet und dienen dem Auffinden und Abzählen von anderen Gnutella Nutzern. Die Teilnehmer schicken regelmäßig Ping Pakete an ihre Nachbarn, die sie dann weiter im Netz verteilen. Auch die Pings haben einen TTL Zähler. Jeder Teilnehmer, der einen Ping empfängt, antwortet dann mit einem Pong. Dieser wird wie eine Suchantwort zurückgeroutet und enthält die IP Adresse, den Port und die Anzahl und Größe der verfügbaren Dateien.

Der größte Teil des Datenverkehrs im Gnutella Netz besteht aus PUSH, Ping und Pong Botschaften. Dieser Bereich der Entwicklung befindet sich zurzeit in einem Stillstand. Da es kein offizielles Gnutella Entwicklungsteam gibt, ist die Frage der Kompatibilität zu anderen Gnutella Versionen nicht leicht zu klären. Ist ein Client mit dem alten Protokoll nicht mehr kompatibel, ist es ihm unmöglich sich mit dem Gnutella Netz zu verbinden.

Der Erfinder von Gnutella erkannte dies und auch andere Probleme. Ihn störte vor allem das schlechte Ping/Pong Routing und daß die Teilnehmer mit schlechten Verbindungen zu viel Datenverkehr bewältigen müssen. Frankel schlug vor Routingtabellen mit Host IDs zu verwenden, und die nachrichtenspezifischen IDs abzulösen. Glaubt man Frankel, so kann nach Behebung der Mängel eine bis zu 6-stellige Anzahl von Teilnehmern partizipieren. Augenblicklich liegt die Obergrenze bei 2000 Nutzern (vgl. [HEISE1]).
Als die Tauschbörse Napster gerichtlich geschlossen wurde stürmten die Nutzer das Gnutella Netz geradezu. Das führte dazu, daß Gnutella für Teilnehmer mit langsameren Wählverbindungen kaum mehr nutzbar war.

Das sind allerdings nicht die einzigen Probleme. Beispielsweise benötigen grobe Anfragen wie nach „mp3“ oder „jpg“ sehr viel Bandbreite. Das liegt daran, daß fast jeder Empfänger dieser Suchanfragen lange Trefferlisten zurückschickt. Die Netzbelastung steigt damit stark an. Derartige Anfragen werden vor allem von Anfängern verschickt, oder von Nutzern, die eine große Auswahl haben wollen.
Dieses Problem ist allerdings verhältnismäßig einfach zu lösen. Die Implementierung des Gnutella Protokolls sollte eine derart allgemeine Anfrage einfach ablehnen.

Ein weiteres Problem von Gnutella ist die Sabotage. In den ersten Versionen wurde den Nutzern hinsichtlich ihrer Gutmütigkeit vertraut. Dennoch wurde das Netz häufig mit sinnlosen Suchanfragen überflutet. Die Suchmonitore wurden auch zum Chatten verwendet.
Eine Lösung gegen das Fluten, der Belastung des Netzes durch Suchanfragen, ist, daß jede Suchanfrage die IP Adresse des Anfragestellers enthalten sollte. Damit kann ein Teilnehmer, der eine positive Suchantwort liefern kann, direkt antworten. Die Netzlast läßt sich so verringern.
Der Anfrageweg kann außerdem durch Bestätigungsanfragen verifiziert werden. Damit kann die Anzahl der Anfragen pro IP Adresse begrenzt werden um ein Fluten des Netzes zu vermeiden.

## 4.2. Jungle Monkey

Jungle Monkey (JM) existiert schon seit 1998 und ist damit älter als Gnutella. Zwischen beiden Modellen gibt es gewisse Ähnlichkeiten. Jungle Monkey scheint allerdings besser skalierbar zu sein. Hier sind die Tauschbörsen wie beim IRC in Channels unterteilt. Wer herausfinden möchte welche Channels existieren, muß zuerst den sogenannten Master Channel kennen. Dieser wird in allen Protokollen als „Initial Connection Point“ geführt. Bei JM wird dabei standardmäßig eine Verbindung zum Server junglemonkey.net hergestellt. Dieser bietet den Channel „Monkey Central“ an. Hier werden Channels und Chats angekündigt. Dateien werden hier nicht zum Tausch angeboten. Die Bedeutung von Monkey Central ist nur gering, da jeder Teilnehmer auch einen solchen Root Channel anlegen kann.

Die Netztopologie ist bei Jungle Monkey durchdachter als bei Gnutella, wo das Netz einem Spinnennetz ähnelt. Bei Jungle Monkey sieht das Netz eher aus wie eine Baumstruktur. Aufspaltungen im Netz sind daher leichter zu erkennen. Die Weiterleitung der Nachrichten läuft entlang des Baumnetzes. Es ist beispielsweise möglich vom Channel „mp3“ aus den Channel „Pink Floyd“ zu erzeugen. Dieser ist dann für alle Teilnehmer des Channel „mp3“ sichtbar. Diese selbstorganisierende Struktur ermöglicht eine effizientere Suche als bei Gnutella. Es ist auch für den Nutzer nicht nötig beispielsweise Anfragen für Pornobilder weiterzuleiten, wenn er selber nur Textdateien anbietet. Aufspaltungen des Netzes lassen sich leider nicht ganz vermeiden. Das kann die Suche erschweren.

Die Aufspaltung des Netzes in thematische Unternetze kann auch aus juristischer Sicht problematisch werden. Die Industrie tendiert dazu schneller gegen kleinere Gruppen von Nutzern von relevantem Material vorzugehen als gegen große unübersichtliche Gruppen, wo das relevante Material nur einen kleinen Unterbereich darstellt.

Zum Anfordern von Dateien wird zuerst bei einem „Rendez-vous“ Server angefragt welcher Host die nachgefragte Datei gespeichert hat. Die heruntergeladenen Dateien werden dann im Gegenzug sofort wieder zum Download angeboten. Folglich werden populäre Dateien vielfach an vielen Orten im Netz angeboten. Die Netzlast wird auf dieser Weise besser verteilt. Es ist außerdem schwierig festzustellen welcher Nutzer diese Datei erstmalig in das Netz eingespeist hat. Das ist vor allem juristisch relevant.
Der „Rendez-vous“ Server enthält zudem Informationen über die Entfernung der Hosts zueinander. Diese Entfernung wird aus den Ping Zeiten berechnet. Der „Rendez-vous“ Server läßt sich nicht nach Dateien durchsuchen. Zu diesem Zweck gibt es den Search Server, in den sich Channels installieren lassen.

Jungle Monkey scheint besser skalierbar zu sein als Gnutella. Es lassen sich immer neue Channels anlegen. Diese Channels skalieren einzeln, nicht wie bei Gnutella als Ganzes. In der

Öffentlichkeit ist Jungle Monkey nur wenig relevant. Das liegt daran daß von Jungle Monkey nur wenige Implementationen verfügbar sind.

## 4.3. Freenet

Freenet hat sich zum Ziel gemacht ein Netz zum Publizieren und Abfragen von Informationen bereitzustellen. Außerdem soll das Netz noch zensurresistent, anonym und effizient sein. Zum Erreichen dieser Ziele baut Freenet auf vier Prinzipien (vgl. auch [HEISE1]):

- Redundanz: populäre Dateien sind häufiger im Netz zu finden. Dabei ist es nicht möglich festzustellen wer diese Datei als Erster eingespielt hat. Damit wird eine gewisse Anonymität, aber auch Rechtssicherheit gewährleistet. Die Effizient wird verbessert, da eine Wahlmöglichkeit zwischen verschiedenen Anbietern besteht. Ist beispielsweise ein Anbieter zu langsam, ist es möglich die benötigte Datei von einem anderen Anbieter zu beziehen.
- Dezentralisiertheit: Freenet ist dezentralisiert und verläßt sich nicht auf einzelne Server
- Verschlüsselung: alle Daten, die in das Netz eingespielt werden, werden vorher verschüsselt.
- Dynamisches Routing: die Route, die eine Anfrage durch das Netz nimmt, wird durch die lexikographische Nachbarschaft der Anfrage zu den Einträgen in der vom Host gespeicherten Tabelle aus Prüfsummen und IP Adressen bestimmt.

Daß die Suche im Freenet bislang noch nicht implementiert ist, soll sich demnächst ändern.

Im Freenet werden die vorhandenen Dateien mit ihren Keys gespeichert. Ein Beispiel hierfür ist: /music/year1979/pinkfloyd/wall.mp3

Nur wer den Klartext Key kennt, kann die dahinterstehende Information auch abfragen. Von diesem Key wird eine Prüfsumme gebildet, beispielsweise PF97XY3AX.

Eine Anfrage nach einer Datei wird an den unmittelbaren Nachbarn des Anfragenden geschickt. Jeder Teilnehmer hat eine Tabelle, die aus Prüfsummen und den dazugehörigen IP Adressen besteht. Wenn in der Tabelle zu einer Prüfsumme keine IP Adresse vorhanden ist, bedeutet dies, daß die betreffende Datei lokal vorhanden ist. Besitzt der Nachbar die Datei nicht, so schickt er die Anfrage an den lexikographisch nächsten Host weiter. Das bedeutet daß er die Anfrage mit den Einträgen in der Tabelle vergleicht und an die IP Adresse der lexikographisch nächsten Prüfsumme weiterleitet. Dadurch werden lexikographische Cluster erzeugt. Thematische Verwandtschaft besteht hingegen nicht. Die Informationen sind weiterhin über das ganze Netz verstreut.

Die Anfrage wird weitergeleitet bis eines der folgenden Ereignisse eintritt (vgl. [HEISE1]):

- Der Time to Live Zähler ist abgelaufen: hier wird die Suchanfrage abgebrochen und mit einer Mißerfolgsmeldung quittiert. Ein weiterer Versuch mit einem höheren TTL ist möglich.
- Die Datei wurde gefunden: hier geht die Datei auf dem gleichen Weg zurück, den auch schon die Anfrage genommen hat. Dabei wird die Datei in jeder Zwischenstation im Cache gespeichert. Falls der Cache voll ist, werden die ältesten Dateien gelöscht um wieder Platz für die neuen Dateien zu schaffen.
- Es sind auf diesem Weg im Netz keine weiteren Hosts: falls der TTL Zähler noch nicht abgelaufen, wird der lexikographisch nächste Host ausgewählt.

Juristisch gesehen könnte man den betreffenden Host für die Dateien, die er speichert, verantwortlich machen. Er könnte verpflichtet werden Keys mit bestimmten Inhalten nicht zu speichern, beispielsweise /filme/kinder/pornographie.avi.

Allerdings ist es eine Tatsache daß der Host selber nicht weiß was er speichert, da er zwar die Prüfsummen, aber nicht den für den Zugriff nötigen Klartext kennt. Die Prüfsummen lassen sich nämlich nicht in Keys zurückverwandeln. Vor dem Upload werden die Dateien mit ihrem Original Key verschlüsselt und können nur mit Kenntnis des exakten Keys wieder dekodiert werden. Die Prüfsumme ist zu diesem Zweck nutzlos. Der Host weiß nur dann was er zwischenspeichert, wenn er selber zufällig die betreffende Datei anfordert und sie in seinem eigenen Cache findet. Eine Zensur wird auf diese Weise erheblich erschwert.

Dateien lassen sich auch in das Netz einfügen. Zu diesem Zweck wird zuerst eine Verbindung hergestellt. Dann wird der Key der Datei (nur die Prüfsumme) mit einem TTL an einen Server geschickt. Dieser leitet den Key an den lexikographisch nächsten Host weiter. Tritt eine Namenskollision auf, wird die Datei an den Sender zurückgeliefert und dieser wird damit informiert, daß der gewählte Name bereits vergeben ist. Durch diese Vorgehensweise wird die Möglichkeit der Zensur weiterhin verhindert. Der Versuch eine Datei durch eine andere Datei zu ersetzen führt dazu daß diese Datei weiterverbreitet wird, da sie in jeder Zwischenstation zwischengespeichert wird. Ist der Key noch nicht vorhanden, signalisiert der Nachbarhost seine Empfangsbereitschaft. Dann erfolgt der Upload zu allen Hosts innerhalb des TTL. Aufgrund der erzeugten Netzlast können Uploads mit sehr hohen TTLs ignoriert werden. Die Hosts tragen dann den Key und die Senderadresse in ihre Routingtabellen ein.

Problematisch ist das Fehlen der Suchmöglichkeiten. Das wird dadurch verstärkt durch den Umstand daß die Hosts nicht wissen was sie speichern.

## 4.4. Blocks

Bei Blocks bilden die einzelnen Teilnehmer ein verteiltes Netz. Jeder Teilnehmer ordnet seinen unmittelbaren Nachbarn einen zufällig ausgesuchten Buchstaben zu. Des Weiteren besitzt jeder Teilnehmer einen Cache. Hier werden eigene Dateien gespeichert, unterteilt in Blöcken zu je 64 KB. Den Nachbarn werden dann Listen dieser Dateien einschließlich Name und Größe zugeschickt. Diese Liste wird als Advert bezeichnet. Der Empfänger eines solchen Adverts hängt diesem den Buchstaben des Absenders an und speichert das Ergebnis. Beim Download werden die Buchstaben berücksichtigt. Falls eine Route unterbrochen wird, werden Alternativrouten gesucht. Es ist also nicht festzustellen, ob die genommene Route die Originalroute ist, oder ob sie auch dem Cache eines Teilnehmers berechnet wurde. Der TTL Zähler startet immer mit dem Wert TTL = 6.

Stammen die Adverts nicht vom Benutzer selber, werden sie immer wieder neu angekündigt. Die Adverts verbreiten sich vom Original Absender weg. Die Indizes der Teilnehmer vervollständigen sich damit langsam.

Bei der ersten Verbindung tauschen die Server eine Liste der 1024 neuesten Adverts aus. Es ist aus diesem Grunde nicht festzustellen ob das Advert nun vom Anbieter selber stammt oder von einem Empfänger des Adverts.

Eine Suche ist im lokalen Index möglich, hat keine Mengenbegrenzung ind erzeugt keine Zeitverzögerung im Netz. Der Dateitransfer ist verschlüsselt. Jeder Teilnehmer weiß welche Daten er speichert. Blocks erzeugt einen Datenstrom, der anderen verschlüsselten Protokollen

sehr ähnlich ist. Daher ist es schwierig das Betreiben eines Block Servers zu verbieten, da die Datenströme auch im Bereich E-Commerce in ähnlicher Form anzutreffen sind.

Auf festen Verbindungen arbeitet Blocks am besten. Wenn ein Server ausfällt werden viele Routen ungültig. Der gesamte Cache des Servers wird in einem solchen Fall gelöscht, einschließlich der eigenen Dateien.
Fährt der Server wieder hoch, muß mühsam wieder alles eingespeist werden. Diese Eigenschaft wird in Zukunft aber geändert werden.

Blocks ist open Source und damit noch entwicklungsfähig. Die Skalierbarkeit wurde bisher noch nicht überprüft.

# 5. Sicherheit in P2P-Netzen

Die Sicherheit in Peer to Peer Netzen sollte nicht vernachlässigt werden. Die Vergangenheit hat gezeigt daß Peer to Peer Netze, wie auch viele andere Strukturen, die über öffentliche Telekommunikationsnetze verbunden sind, Ziel von Angriffen von außen geworden sind.

Die Sicherheitseinrichtungen auf den Knoten unterliegen bestimmten Bedingungen. Die Kosten zu Gewährleistung der Sicherheit sollen überschaubar bleiben. Zudem ist eine möglichst kleine Entschlüsselungszeit wünschenswert. (vgl. auch [AGARWAL1]) Der Einsatz von weit verbreiteten Sicherheitsmechanismen ist eine Notwendigkeit. Nur dann ist eine Zusammenarbeit zwischen verschiedenen Systemen in heterogenen Umgebungen möglich. Eine geringe Entschlüsselungszeit aus ergonomischen Gründen sollte nicht zu einer zu einfachen Verschlüsselung führen. Eine gute Verschlüsselung sollte das unbefugte Entschlüsseln einer Information weitestgehend unmöglich, oder zumindest extrem aufwendig machen, damit es einem unbefugten Dritten nicht mehr lohnenswert erscheint.

Registrierungsprozeduren sollten auch nicht zu kompliziert sein. Weniger erfahrene Nutzer sind ansonsten schnell überfordert und können sie daher nicht durchführen. Komplizierte Registrierungsprozeduren werden von den Nutzern oft nicht akzeptiert und daher nicht verwendet.

Um ein P2P System zu sichern, steht eine Beschäftigung mit diesem System an erster Stelle. Wichtig ist vor allem die Charakterisierung der Eigenschaften des zu sichernden Systems. Eine Schwierigkeit in P2P Systemen ist die Eigenschaft, daß es dynamische Gruppen gibt. Die Mitglieder wechseln ständig. Es ist durchaus berechtigt anzunehmen, daß zwischen den verschiedenen Gruppen ein gewisses Mißtrauen vorausgesetzt werden kann (vgl. auch [AGARWAL1]).

Neben unterschiedlichen Latenzzeiten und Bandbreiten gibt es auch noch Firewalls und sonstige Barrieren. Gemeinsame Verzeichnisdienste gibt es nicht. Zur Gewährleistung von Sicherheit ist unter anderem eine sichere Gruppenkommunikation nötig. Außerdem ist eine gleichartige und glaubwürdige Zertifizierung erforderlich, die nicht zu kompliziert sein sollte. Besonders wichtig ist, daß die verschiedenen Sicherheitsrichtlinien und Sicherheitsmodelle untereinander interoperabel sein müssen.

Ein Beispiel für eine Sicherung in einem P2P System bietet die Anwendung SETI@home. Hier werden verschiedene Berechnungen an die Teilnehmer weitergegeben. Die Teilnehmer schicken die Ergebnisse ihrer Berechnungen wieder zurück. Das sicherheitsrelevante Problem ist, ob die Teilnehmer die Berechnungen korrekt ausführen, und korrekte Ergebnisse zurückgeben. Fehler können hier sowohl beabsichtigt als auch unbeabsichtigt auftreten. Die Lösung für dieses Problem ist verhältnismäßig einfach. Jeder Berechnungsauftrag wird zugleich an mehrere Teilnehmer abgegeben. Mittels Prüfsummen werden die Berechnungen dann überprüft. (vgl. auch [AGARWAL1])

Die Sicherheit in einem Computernetz beschäftigt sich mit verschiedenen Sichtweisen. Diese sollen im Folgenden kurz erläutert werden.

## 5.1. Vertraulichkeit

Vertrauliche Daten, die über ein öffentliches Medium gesendet werden, zeichnen sich dadurch aus, daß ein unbeteiligter Dritter diese Daten nicht lesen kann. Es darf nicht möglich sein, daß außer dem Sender und dem Empfänger, eine dritte Partei in der Lage ist den Inhalt der Daten einzusehen.

Es ist eine Tatsache daß die Daten, die über ein öffentliches Medium geleitet werden, für jeden empfangbar sind, der auch an dieses Medium angeschlossen ist. Werden diese Daten im Klartext über die Leitungen geschickt ist es dementsprechend offensichtlich, daß jede dritte Partei diese Daten auch lesen kann. Das ist im Sinne der Vertraulichkeit unerwünscht.

Zur Sicherung der Vertraulichkeit gibt es relativ einfache Möglichkeiten. Das Problem in der öffentlichen Kommunikation liegt in der allgemeinen Lesbarkeit der Daten in dem Medium. Die Lesbarkeit der Daten seitens Unbefugter läßt sich durch eine Verschlüsselung der Daten vermeiden. Dazu werden Operationen für die Verschlüsselung von Daten benötigt.

Mögliche Verschlüsselungsverfahren für Daten sind beispielsweise Public Key Verfahren, Private Key Verfahren, Symmetrische Verschlüsselung oder Asymmetrische Verschlüsselung. Auf die genauen Vorgehensweisen dieser Verfahren soll im Rahmen dieser Arbeit nicht weiter eingegangen werden.

## 5.2. Integrität

Die Integrität von Daten ist in Peer to Peer Systemen ist durchaus gefährdet. Es ist nicht nur möglich über ein öffentliches Telekommunikationsmedium gesendete Daten „mitzulesen“, sondern es besteht prinzipiell auch die Möglichkeit die mitgelesenen Daten zu manipulieren. Diese Manipulationen können schwerwiegende Folgen haben, wie beispielsweise Datenverluste oder auch den Befall von Computer Schädlingen.

Es ist nicht leicht festzustellen welche Daten auf dem Weg zwischen Sender und Empfänger verändert worden sind. Daher ist es wichtig Mechanismen einzuführen, die Veränderungen an Daten sichtbar machen. Ein möglicher Mechanismus ist die Verwendung von Hash Funktionen. Der Hashwert kann dann verschlüsselt zwischen den Teilnehmern ausgetauscht werden.

## 5.3. Authentifikation

Authentifikation ist der Prozeß festzustellen, ob eine Einheit tatsächlich ist, die sie vorgibt zu sein. Das ist in einem zentralisierten Netz relativ einfach. Hier wird ein zentraler Server in Anspruch genommen, der über eine zentrale Datenbank verfügt. Manipulationen sind hier nur unter großen Schwierigkeiten möglich. Schließlich müßte diese Manipulation direkt am zentralen Server stattfinden.

In einem dezentralen Netz ist die Authentifikation schwieriger. Hier ist der Einsatz von digitalen Signaturen notwendig (vgl. auch [HEISE1]). Eine digitale Signatur ist die Verschlüsselung der Prüfsumme der zu signierenden Information mit dem privaten Schlüssel des Absenders. Wer den passenden öffentlichen Schlüssel besitzt, kann die Authentizität der Nachricht überprüfen. Zu diesem Zweck ist es hinreichend zu schauen, ob die Prüfsumme stimmt.

Auf diese Weise wird auch die Verwendung von Pseudonymen erleichtert. Pseudonyme werden vor allem aus Gründen der Anonymität verwendet. Ein Nutzer möchte anonym bleiben, aber trotzdem sicher und verbindlich im Netz handeln können. Das gewünschte Pseudonym ist allerdings nur dann benutzbar, wenn niemand sonst unter diesem Namen einen öffentlichen Schlüssel betreibt.

Im Gegensatz zur Authentifikation ist die Authorisation der Prozeß, der einer authentifizierten Entität die Erlaubnis gibt etwas zu tun oder auf eine Ressource zuzugreifen.

# 6. Replica Management

Es ist sicherlich ratsam häufig angeforderte Dateien oder Informationen mehrfach in einem Rechnernetz abzuspeichern. Werden diese Mehrfachkopien, oder Replicas, an „strategisch günstigen“ Stellen gespeichert, so kann die allgemeine Verfügbarkeit dieser Dateien oder Informationen erhöht werden, bei evtl. gleichzeitiger Verringerung der Netzlast.

Es gibt verschiedene Arten von Replication. Zuerst ist die „owner replication“ zu nennen. Hier speichert nur der Nachfrager der Datei eine Kopie derselben. Diese Vorgehensweise findet bei Gnutella Anwendung. Eine weitere Art der Replication ist die „path replication“. Die Kopie wird hier entlang des Weges in jedem Knoten zwischen Nachfrager und Anbieter gespeichert. Bei der „random replication“ werden die Replicas zufällig auf den besuchten Sites verteilt (vgl. auch [LV1]).

Bei der Betrachtung von Replicas stellen sich Fragen, deren Antworten für die Verwaltung von Replicas unerläßlich sind:

Wann sollen Replicas erzeugt werden ?
Welche Daten sollen repliziert werden ?
Wo sollen die Replicas abgelegt werden ?

Die Antworten auf diese Fragen legen letztendlich die Strategie für die Replizierung der Daten fest.

Die Begriffe „Replication“ und „Caching“ werden häufig synonym verwendet. Tatsächlich gibt es aber Unterschiede. Beim Caching fragt der Client eine Datei nach und speichert eine Kopie davon lokal, damit sie zur zukünftigen Nutzung zur Verfügung steht. Jeder nahe Knoten kann die gecachte Kopie nachfragen.
Bei der Replication entscheidet dagegen nur der Server wann und wo eine Kopie seiner Datei erzeugt wird. Diese Entscheidung kann zufällig oder auch auf Grundlage des Verhaltens der Clienten erfolgen.

Für das Anlegen von Replicas gibt es verschiedene Strategien. Im Folgenden sollen einige dieser Strategien vorgestellt werden (vgl. auch [LV1]).

# 6.1. Best Client

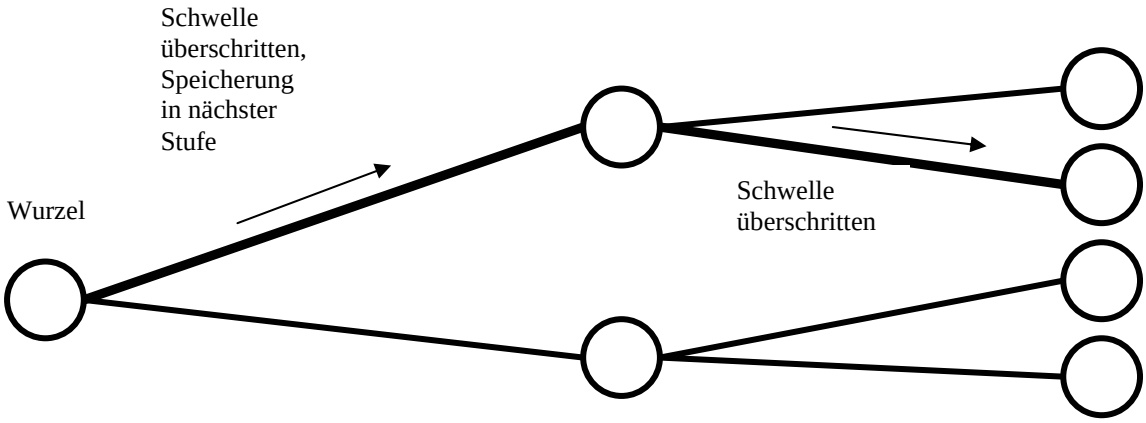
In der Best Client Strategie führt jeder Knoten eine „History“ für jede Datei die er erhält. In dieser History sind sowohl die Anzahl der Anforderungen für die Datei als auch die nachfragenden Knoten enthalten. Nach einem gegebenen Zeitintervall wird geprüft, ob die Anzahl der Anforderungen für eine Datei einen gegebenen Schwellenwert überschritten hat. Wenn das der Fall ist, so wird der „Best Client“ für die Datei ermittelt. Der Best Client ist derjenige Client, der die meisten Anforderungen für die betreffende Datei generiert hat. Der Knoten erzeugt dann eine Replica dieser Datei beim Best Client.

Die Histories werden regelmäßig gelöscht. Auf diese Weise ist es möglich Änderungen in den Zugriffsmustern zu erkennen. Da die Nutzer ihre Gruppenzugehörigkeit wechseln können ist die Effektivität der Strategie abhängig davon, wie gut das Zeitintervall an das Zugriffsverhalten angepaßt worden ist.

# 6.2. Cascading Replication

Wird in der Wurzel (Abb. 6.2.(1)) eine Schwelle überschritten, wird eine Replica auf dem Pfad zum Best Client auf der nächsten Stufe erzeugt. Der neue Ort der Replica ist dementsprechend ein Vorgänger des Best Clients. Wird auf der nächsten Stufe die Schwelle erneut überschritten, wird eine Replica auf der übernächsten Stufe erzeugt, und so weiter. Dadurch wird eine populäre Datei letztendlich im Client selber repliziert.

Der Vorteil dieser Strategie ist eine Speicherplatznutzung auf allen Stufen. Außerdem bringt eine Replizierung auf einer höheren Stufe als der aktuellen Quelle der Anfragen, die Daten näher zu den Knoten im gleichen Unterbaum.




| Abb. 6.2.(1) | Darstellung der Baumstruktur des Netzes |
|---|---|

# 6.3. Plain Caching

In dieser Strategie speichert der Client, der die Datei nachgefragt hat, lokal eine Kopie der Datei. Da der Client in der Regel jedoch nur über einen begrenzten Speicherplatz verfügt, werden die auf diese Art gespeicherten Dateien schnell durch andere Dateien ersetzt.

## 6.4. Caching kombiniert mit Cascading Replication

Diese Stategie kombiniert die Strategien Caching und Cascading Replication. Der Client speichert die Datei lokal. Der Server erkennt regelmäßig populäre Dateien und gibt diese Dateien weiter in die Hierarchie herunter.

Die Clienten sind dabei immer Blätter im Baum. Die Server werden als Knoten im Baum betrachtet. Geschwister sind Knoten mit den gleichen Eltern, bzw. Vorgängern. Der Client kann für seine Geschwister auch zum Server werden.

## 6.5. Fast Spread

Die Datei wird in jedem Knoten auf dem Weg vom Server zum Clienten gespeichert. Das führt zu einer schnelleren Verbreitung der Datei. Wenn ein Knoten keinen zusätzlichen Speicherplatz für weitere Replicas hat sucht er in seinem Speicher nach der am wenigsten populären Datei, die er dann löscht. Für den Fall daß er über mehrere gleich wenig populäre Dateien verfügt, löscht er zuerst die älteste dieser Dateien.

## 6.6. Fazit

Die Strategie Best Client hat die ungünstigsten Eigenschaften. Allerdings wird bei den übrigen Strategien oft viel Overhead erzeugt. Dieser Overhead kann die Vorteile der Strategie zunichte machen und dann zu einer schlechteren Performanz führen, als wenn keine Replicas erzeugt worden wären.

Die Strategie Fast Spread hat bessere Antwortzeiten und spart mehr Bandbreite ein als Caching. Der Nachteil ist allerdings daß diese Strategie große Anforderungen an den Speicherplatz stellt.

Das Cascading Replication benötigt zwar mehr Bandbreite als Fast Spread, kommt aber mit weniger Speicher aus. Jede Replica muß einzeln gespeichert werden, anders als beim „Speichern entlang des Weges“. (vgl. auch [LV1])

# 7. Suche in Peer-to-Peer Netzen

Peer-to-Peer Netze bieten Zugriff auf alle möglichen Arten von Informationen oder Ressourcen, die sonst für die meisten Teilnehmer unerreichbar wären. Der Wert eines Peer to Peer Netzes ist abhängig von der Menge und von der Qualität der verfügbaren Ressourcen. Je mehr Teilnehmer oder Peers das Netz hat, umso größer ist der Wert des Netzes. Je populärer das Netz ist, umso mehr Peers beteiligen sich daran.

Dennoch kann es passieren, daß die Leistungsfähigkeit des Netzes sinkt, wenn sich mehr als eine bestimmte Anzahl von Peers daran beteiligen. In einem großen Netz werden die Lokalisierungsroutinen komplizierter und damit teurer. Dementsprechend ist der Wert eines Netzes auch abhängig von der Effizienz der Suche nach Informationen in diesem Netz.

Leistungsfähige Suchstrategien benötigen zusätzliche Informationen über die zu suchenden Daten. Diese Informationen werden als Metadaten bezeichnet und umfassen beispielsweise Dateiname, SHA-1 Hashwert oder auch sonstige Angaben. Die Aufgabe von Metadaten besteht in der Steigerung der Effektivität der Suche. Diese Effektivität der Suche beeinflußt auch die Popularität des Netzes. Damit ist eine breite Unterstützung von Metadaten und Suchoptionen ein Indikator für den Wert eines P2P Netzes (vgl. auch [ALPINE1])

## 7.1. Überblick über verschiedene Suchstrategien

Die Suche nach Informationen in Peer-to-Peer Netzwerken wird zunehmend schwieriger, je schneller diese Netzwerke anwachsen. Schließlich entwickelten sich verschiedene Ansätze wie eine Suche in einem Peer-to-Peer Netz durchzuführen ist. Einige dieser Ansätze sollen im Folgenden vorgestellt werden.

### 7.1.1. Fluten von Netzen

Ein Beispiel für das Fluten von Netzwerken ist die ursprüngliche Gnutella Implementation. Bei der Suche wird die Suchanfrage an alle Nachbarn geleitet, die sie dann wieder an alle ihre Nachbarn weiterleiten, und so weiter.

Der Vorteil einer solchen Suchstrategie liegt in der Flexibilität der Bearbeitung der Suchanfrage. Jeder Teilnehmer kann selber entscheiden wie er die Anfrage bearbeiten will. Allerdings ist diese Art der Suche nur in kleinen Netzwerken anwendbar. Das ist genau dann ein Nachteil wenn das Netz wächst und es eine große Anzahl von Teilnehmern gibt. Bei einer großen Popularität kann der Sättigungspunkt in dieser Beziehung schnell überschritten werden. Es kann daher zu einer Aufspaltung des Netzes in kleinere Gruppen kommen. Dann nimmt sowohl die Anzahl der Peers als auch die Menge an Ressourcen ab.

Problematisch sind auch böswillige Angriffe auf das Netzwerk. Böswillige Teilnehmer können große Mengen von sinnlosen Anfragen freisetzen und damit das Netz belasten oder sogar lahmlegen.

### 7.1.2. Selektives Weiterleiten

Die Methode des Selektiven Weiterleitens ist leichter skalierbar als das Fluten. Hier wird die Anfrage nur an ausgewählte Teilnehmer geschickt, von denen angenommen wird, daß bei ihnen die benötigte Ressource mit der größten Wahrscheinlichkeit zu finden ist.

Des Weiteren ist die Sicherung gegen Angriffe besser geregelt. Da an der Bearbeitung der Anfrage weniger Teilnehmer beteiligt sind, ist es besonders wichtig, daß gerade diese Teilnehmer auch vertrauenswürdig sind. Böswillige Teilnehmer können sich dennoch an verschiedenen Punkten mit dem Netz zu verbinden und Anfragen falsch weiterleiten oder sogar ganz zu verwerfen. Auf diese Weise können Resultate verfälscht werden. Die Genauigkeit der Suche sinkt. Bei der entsprechenden Intensität der Angriffe kann die Leistungsfähigkeit des Netzwerkes sinken.

Es besteht weiterhin die Gefahr des Mißbrauches oder der Ausbeutung des Netzes. Der Grund ist daß das Netzwerk sowohl dezentral als auch offen ist.

Um empfangene Anfragen weiterleiten zu können, müssen die Teilnehmer zusätzliche Informationen erhalten. Für kleine Netzwerke ist dieser Overhead noch gering, wächst aber bei größeren Netzwerken immens an.

### 7.1.3. Dezentrale Hash Indizes

Jedes Dokument erhält eine eindeutige ID, typischerweise einen SHA-1 Hashwert des Inhaltes. Dieser Hashwert wird benutzt um die Ressource zu finden. Dabei werden Netzwerk und Teilnehmer derart angeordnet, daß ein gegebener Schlüssel unabhängig von der Größe des Netzwerkes gefunden werden kann.

Nachteilig ist anzumerken, daß die Daten nur über ihre ID erkannt werden. Eine ungenaue Suche oder Schlüsselwortsuche in diesem Netzwerk ist nicht möglich, da das Einfügen und Wiederauffinden von Informationen nur über die ID möglich ist.

Angriffe auf das Netzwerk bleiben auch weiterhin möglich. Böswillige Teilnehmer können Anfragen falsch weiterleiten, sinnlose Daten einfügen und damit den Schlüsselraum überlasten, oder auch das Netzwerk durch viele sinnlose Anfragen belasten.

## 7.1.4. Zentralisierte Indizes

Die zentralen Indizes schienen bisher die leistungsfähigste Methode zur Informationssuche zu sein. Nachteilig zu erwähnen sind allerdings die hohen Kosten, da Bandbreite und Hardware für große Netze sehr teuer sind.

Die aktuelle Rechtsprechung in diesem Bereich läßt zudem den Schluß zu daß Schadensersatzklagen in zentralisiert verwalteten Systemen zunehmend erfolgreicher zu werden scheinen. Beispielsweise erzwingen die Gesetze eine Überwachung der Aktivitäten im Netzwerk, um Urheberrechtsverletzungen zu identifizieren.

## 7.1.5. Verteilte Indizes

Bei der Verwendung von verteilten Indizes sind zentralisierte Server überflüssig. Die Teilnehmer übernehmen die Aufgaben der Indizierung. Die Verwundbarkeit gegenüber der Gerichtsbarkeit nimmt ab. Es gibt keine zentrale Kontrolle über die Operationen der Indizierung.

Diese Dezentralisierung führt auch zu Problemen. Die Eindeutigkeit der indizierten Daten ist gefährdet. Der Overhead die Informationen beim Teilnehmer aktuell zu halten ist groß, da die Fluktuation von Teilnehmern und Daten immens ist.

Aber es gibt noch weitere Probleme:

- „The Number of peers supporting the index network is large.“
  (Die Anzahl an Teilnehmern, die das Index-Netzwerk unterstützen ist groß)
- „Many peers join and depart the network maintaining the index“
  (Viele Peers, die den Index verwalten, schließen sich dem Netzwerk an oder verlassen es)
- „The amount of data to be indexed is significant“
  (Die Menge an zu indizierenden Daten ist groß)
- „The meta data for the indexed data is very diverse“
  (Metadaten für indizierten Daten sind vielfältig)
- „Malicious peers exploit the trust implicit in a shared index“
  (Bösartige Peers nutzen das Vertrauen, das in einem verteilten Index impliziert wird, aus) (Zitat aus [ALPINE1])

Bei Angriffen können bösartige Peers große Mengen von sinnlosen Daten einfügen, und damit den Schlüsselraum der verteilten Indizes belasten und damit die Genauigkeit der Suche beeinträchtigen.

Es ist schwierig eine Vielfalt der Metadaten zu unterstützen. Durch ein XML Schema können die Metadaten greifbar gemacht werden. Allerdings vergrößern diese Metadaten zusätzlich zu Namen und Schlüsseln den Overhead für die Indizierung. Die Skalierbarkeit sinkt dadurch.

## 7.1.6. Relevanzbezogene Netzwerk Crawler

Der relevanzbezogene Netzwerk Crawler verwendet eine Datenbank von Informationen, die der Teilnehmer schon gesammelt hat. Damit legt er fest ob die Ressourcen, denen er begegnet, für ihn relevant sein können. Mit der Zeit werden große Mengen von Informationen analysiert. Es werden dann auch gemeinsame Dinge gefunden, die der Teilnehmer als

interessant eingestuft hat. Der Crawler „kriecht“ praktisch durch das Netz und sucht nach neuen Informationen, die in das Profil der früheren Peer Informationen passen.

Auch bei diesem Ansatz gibt es Probleme. Beispielsweise bietet der relevanzbezogene Netzwerk Crawler keine Unterstützung für Anfragen nach speziellen neuen Informationen, da er nur von den vorhergehenden Informationen geleitet wird.

Der relevanzbezogene Netzwerk Crawler bietet auch keine Unterstützung für eine breite Vielfalt an Ressourcen, denn die Suchmaschine arbeitet nur auf bestimme Arten von Daten, meistens HTML oder andere Textdokumente.

### 7.1.7 Kataloge und Meta Indizes in Netzwerken mit Hash Indizes

Im Freenet selber werden Katalogdokumente gespeichert, die eine Ressource beschreiben, die durch ihren Hash Key repräsentiert wird. In diesem Katalog sind Suchanfragen durchführbar, damit man die Ressourcen im Netzwerk schnell und effizient finden kann. Ähnliche Methoden werden auch in anderen Netzwerken verwendet.

Diese Kataloghaltung führt auch zu Problemen. Eine ständige Wartung dieses Kataloges ist unerläßlich um die Daten immer aktuell zu halten. Auch das erstmalige Auffinden des Kataloges kann problematisch sein.

### 7.1.8. Hybride Netzwerke und Super Peers

KaZaa oder Morpheus sind Beispiele für Hybridnetzwerke im Bereich Media Sharing. Hier gibt es keinen zentralen Server. Dieser wurde durch eine große Anzahl sogenannter Super Peers ersetzt.

Ein Super Peer ist ein Teilnehmer, der über überdurchschnittliche Bandbreite und Verarbeitungskapazität verfügt. Diese Super Peers übernehmen zusätzliche Last, büßen deshalb aber nicht nennenswert an Leistung ein.

Soll eine Anfrage bearbeitet werden, so wendet sich der Nutzer an einen solchen Super Peer. Kriterien für die Auswahl der Super Peers sind Bandbreite, Speicherkraft und CPU Leistungsfähigkeit. Findet der Super Peer lokal keine Treffer für die Anfrage, arbeitet er häufig mit anderen Super Peers zusammen.

Diese Lösung scheint auf den ersten Blick relativ praktikabel zu sein. Die Verwendung zentraler Server wird vermieden. Das bietet einen besseren Schutz gegen gerichtliche Auseinandersetzungen. Allerdings ist KaZaa seinerseits in gerichtliche Auseinandersetzungen verwickelt. Der Ausgang dieser Auseinandersetzungen ist noch ungewiß (vgl. auch [ALPINE1]).

Leider gibt es auch bei den hybriden Netzwerken gewisse Nachteile. Die Super Peers stellen noch immer kleine zentralisierte Server dar. Aktuelle Gerichtsurteile fordern von diesen Servern ihre Inhalte zu überwachen und zu filtern um Urheberrechtsverletzungen zu vermeiden. Diese Filterung scheint fast unmöglich durchzuführen zu sein. Die Größe der aktuellen Filter erschwert dies sehr.

Weiterhin ist die Unterstützung von Metadaten schwierig. Jeder Knoten müßte jede Art von Metadaten unterstützen, die in den Anfragen benutzt werden. Es wird ein großer Overhead zur Synchronisierung der Metadaten in den Super Peers benötigt.

Die Super Peers bieten böswilligen Teilnehmern eine gute Angriffsfläche. Ein Client kann große Mengen von sinnlosen Anfragen oder falsche Indexinformationen an einen Super Peer senden. Damit wird er handlungsunfähig gemacht. Das Ausmaß der Schäden hängt von der genauen Implementation des Super Peers ab. Mögliche Schäden sind übermäßiger Speicherverbrauch, zerstörte Indizes oder geringe Leistung.

### 7.1.9. Adaptive soziale Suchmechanismen für große Netzwerke

„Social discovery implies a direct, continued interaction between peers in the network“ (Zitat aus [ALPINE1])
(Soziale Suche bedeutet direkte, kontinuierliche Interaktion zwischen Peers im Netzwerk)

Daraus ergeben sich gewisse Unterschiede zu einigen vorher betrachteten Ansätzen. Es wird nun eine direkte Verbindung zwischen jedem Peer und den Peers mit denen kommuniziert wird, angenommen. Damit kontrolliert jeder Teilnehmer jeden anderen Teilnehmer, mit dem er kommuniziert, was zu einer Kontrolle über den Verbrauch an Bandbreite erlaubt. Auch die Kontrolle über den Gebrauch des Netzwerkes ist nun möglich. Damit ist eine gute Kontrollmöglichkeit gegen den Mißbrauch des Netzes gegeben.

Jede Kommunikation wird so lange aufrechterhalten, wie die beteiligten Teilnehmer es wünschen. Damit wird diese Kommunikation längerlebig als eine typische TCP-Verbindung. Wenn ein Nutzer einer Wählverbindung seine IP-Adresse ändert, kann die Verbindung wieder aufgebaut werden. Das erlaubt es eine „History“ der Interaktionen unter den Teilnehmern zu erstellen. Der Teilnehmer kann nun auch eine Reputation, einen Ruf, aufzubauen. Dadurch können Suchaktionen im Netzwerk verbessert werden.

„Simple, low overhead messaging forms the foundation of peer communication“ (Zitat aus [ALPINE1])
(Leichtes Messaging mit geringem Overhead bildet das Fundament der Peer Kommunikation)

Die Basis der Kommunikation zwischen den Teilnehmern ist das UDP-Protokoll, welches für einfache Datenpakete mit wenig Overhead konzipiert ist. Die Kommunikation zwischen den Teilnehmern verläuft über ein UDP-Socket. Ein Multiplexing Protokoll in der Anwendungsschicht erlaubt es eine große Anzahl direkter Verbindungen zu verwalten ohne viel Overhead in Kauf nehmen zu müssen.

Das Auffinden einer Ressource erfordert eine gewisse Menge an Kommunikation zwischen den einzelnen Teilnehmern. In großen dezentralisierten Netzwerken wird ein großer Anteil der verfügbaren Bandbreite genau hierfür verbraucht. Also soll das „Messaging“-Protokoll so leicht, aber auch so kompakt wie möglich gestaltet werden. Das wird möglich, wenn der Overhead für das Senden einer Nachricht verkleinert wird.

„Connection persistance allows profile and perfomance tracking of peers.“ (Zitat aus [ALPINE1])
(Die Persistenz der Verbindung der Verbindung erlaubt das Verfolgen der Profile und der Leistungsfähigkeit der Teilnehmer)

Die Teilnehmer bleiben so lange miteinander verbunden wie sie es wünschen. Die Verbindungen werden entsprechend wieder aufgebaut, wenn eine Anwendung neu gestartet wird, eine Modemverbindung unterbrochen wurde oder wenn sich die Ports einer Firewall ändern. Der Grund hierfür ist, daß für jeden Teilnehmer eine „History“ erstellt werden soll. Das ist ein Profil, welches beschreibt wie wertvoll der Teilnehmer bei Suchoperationen ist, und wie viele Ressourcen er hat. Dann können böswillige Teilnehmer schnell erkannt werden, da sie zwar kaum Wert haben, aber dafür viel Bandbreite oder andere Ressourcen verbrauchen. Ihre Verbindung wird dann unterbrochen. Auch Teilnehmer, die zwar konsumieren, aber selber keine Ressourcen teilen, sind daher schnell zu identifizieren. Die Verbindung zu diesen Teilnehmern wird ebenfalls getrennt.
So läßt sich der Mißbrauch des Netzwerkes relativ gut vermeiden. Außerdem werden die Teilnehmer durch diese Vorgehensweise dazu angehalten selber Ressourcen zur Verfügung zu stellen, was den Wert des Netzwerkes erhöht.

„Past query responses are used to optimize resource discovery“ (Zitat aus [ALPINE1])
(Vergangene Suchantworten werden benutzt um das Auffinden von Ressourcen zu optimieren.)

Die Suche nach Ressourcen läuft folgendermaßen ab: ein einzelnes kompaktes Anfrage-Paket wird an jeden Teilnehmer der zu befragenden Gruppe geschickt. Dieses Paket wird linear weitergeleitet, bis entweder eine angemessene Anzahl von Ressourcen gefunden wurde, oder bis der Nutzer die Anfrage abbricht.

Die Effizienz der Suche wird dadurch gesteigert, daß jedem Teilnehmer ein Profil zugeordnet wird, mit dem festgestellt wird, in welcher Reihenfolge die Teilnehmer das Anfragepaket erhalten. Dabei erhalten diejenigen Teilnehmer, die in der Vergangenheit relevante Antworten oder Ressourcen geliefert haben einen entsprechend hohen Qualitätswert. Fragt man zuerst die Teilnehmer mit einem hohen Qualitätswert, steigt die Wahrscheinlichkeit eine Ressource zu finden. Die für die Suche benötigte Bandbreite wird verringert.

„Social discovery and profiling encourages sharing and good behavior“ (Zitat aus [ALPINE1])
(Soziale Erkennung und Profilierung ermutigt zum Teilen und zu gutem Benehmen)

Die meisten Such-Netzwerke bieten den Teilnehmern kaum Anreize selber Ressourcen anzubieten. In einem sozialen Netzwerk ist das völlig anders. Wenn der Teilnehmer keine Ressourcen anbietet riskiert er daß seine Verbindung mit dem Netzwerk getrennt wird. Wird eine Verbindung mit Teilnehmern mit hohem Qualitätswert gewünscht, so ist es nötig zunächst selber einen hohen Qualitätswert zu erlangen. Durch die laufende Umstrukturierung der Teilnehmer Gruppen werden böswillige Teilnehmer oder Teilnehmer mit schlechten Qualitätswerten aus dem Netzwerk herausgetrennt und durch neue Teilnehmer ersetzt, von denen angenommen wird, daß sie bessere Werte erzielen können. Hier wird also gutes Benehmen und das Bereitstellen einer guten Qualität von Ressourcen belohnt.

„Distinct groups of peers are supported for distinct types of discovery“ (Zitat aus [ALPINE1])
(Bestimmte Gruppen von Teilnehmern werden für bestimmte Arten der Suche unterstützt.)

In einem Netzwerk wird meistens nach vielen verschiedenen Arten von Ressourcen gesucht. Daraus folgt daß ein Teilnehmer hohe Qualtitätswerte für die Suche nach einer bestimmten Art von Ressourcen haben kann, aber bei der Suche nach einer anderen Art von Ressource sehr schlecht arbeitet. Aus diesem Grund werden Gruppen von Teilnehmern unterstützt, so

daß man den Teilnehmer fragen kann, der am effektivsten für eine bestimmte Anfrage erscheint. Damit erhalten Teilnehmer mit guten Qualitätswerten keine schlechteren Bewertungen, wenn Anfragen laufen, die sie nicht unterstützen können.

Die Effizienz der Suchoperationen wird durch die Unterteilung in Gruppen unterstützt, wobei die Teilnehmer in jeder Gruppe spezielle Suchoperationen unterstützen.
Zum Beispiel betrachtet man einen Teilnehmer X, der in Gruppe 1 eine gute Bewertung erhalten hat. Dabei soll Gruppe 1 die Gruppe sein, die sich vornehmlich mit zeitgenössischer Literatur beschäftigt. Gruppe 2 hingegen soll sich den Sonetten von W. Shakespeare widmen. Es scheint evident, daß der Teilnehmer X bei einer Anfrage nach einem bestimmten Sonett kaum verwertbare Lösungen liefern kann.

„Adaptive social discovery relates directly to the interaction of a user with his/her peers“ (Zitat aus [ALPINE1])
(Adaptive soziale Suche hängt direkt ab von der Interaktion des Nutzers mit den anderen Teilnehmern)

Wie auch in der realen physikalischen Welt organisieren sich Teilnehmer mit ähnlichen Interessen in Gruppen. Sie erhalten ihre Beziehungen untereinander so lange aufrecht, wie sie ihnen wertvoll erscheinen. Wenn ein Teilnehmer seine Ressourcen nicht teilen will oder sogar Angriffe durchführt, wird dieser Teilnehmer ausgesperrt bis er sein unerwünschtes Verhalten ändert.

Durch diese Art der Interaktion können die Ansprüche für Qualität, Performanz und Flexibilität, die für die Suche nach Ressourcen in dezentralisierten peer-basierten Netzwerken benötigt werden, erfolgreich implementiert werden.

# 7.2. Konkrete Verfahren

Für die Suche in Peer-to-Peer Netzwerken gibt es auch konkretere praktische Ansätze. Sie enthalten die Details, auf die die vorherigen Ansätze verzichtet haben einzugehen. Diese Ansätze sollen in diesem Teilabschnitt vorgestellt werden.

## 7.2.1. Routing Indizes

In Peer-to-Peer Netzen kann man auf verschiedene Art und Weise suchen. Es gibt die Suche ohne Index, die Suche mit spezialisierten Index Knoten und die Suche mit Indizes in jedem Knoten (vgl. auch [CRESPO1]).

### 7.2.1.1. Suche ohne Index

Bei der bei Gnutella verwendeten Suchmethode haben die Teilnehmer keinen Suchindex. Die Anfragen werden von Knoten zu Knoten weitergeleitet bis ein Treffer gefunden wird. Dieser Suchmechanismus wird als Fluten bezeichnet.

Der Vorteil ist daß der Suchmechanismus einfach und stabil ist. Nachteilig sind die Kosten zu erwähnen. Das Netzwerk wird bei jeder Anfrage geflutet.

### 7.2.1.2. Suche mit spezialisierten Index Knoten

In zentralisierten Systemen besitzen spezielle Teilnehmer einen Index oder einen Katalog von den im System verfügbaren Dokumenten. Sucht ein Nutzer nach einem Dokument, so schickt er seine Anfrage an den Teilnehmer, der den Katalog besitzt. Dieser sucht dann die Dokumente von Interesse heraus. Hier liegt der Vorteil in der Effizienz. Es wird nur eine Suchanfrage verschickt und damit wenig Bandbreite verschwendet.

Es ist allerdings problematisch, daß das System gegenüber Angriffen leicht verwundbar ist. Schließlich müssen nur die Teilnehmer, die Kataloge verwalten, angegriffen werden. Die Kataloge selber sind auch nur durch einen erheblichen Wartungsaufwand aktuell zu halten.

### 7.2.1.3. Suche mit Indizes in jedem Knoten

Eine andere Möglichkeit die Suche in Netzwerken zu verbessern, ist es einen Index bei jedem Teilnehmer zu hinterlegen. Diese verteilten Indizes müssen klein sein. Daher werden statt der traditionellen „Destination“ Indizes, wo das Ziel der Anfrage vorherbestimmt ist, jetzt „Routing“ Indizes verwendet. Dieser Index gibt statt dem Zielpunkt die Richtung des Weges an.

Zwischen dem Destination Index und dem Routing Index gibt es Unterschiede, die im Folgenden erklärt werden sollen.

Beim Destination Index werden Multicast Algorithmen verwendet. Diese benötigen einen relativ stabilen Multicast Baum. Der Pfad von der Senderadresse zur Zieladresse wird entlang dieses Baumes beschritten.

Der Routing Index arbeitet ohne einen Multicast Baum. Der Fortschritt entlang eines solchen Baumes ist in einem P2P Netzwerk nicht unbedingt berechenbar, und wenn, dann nur unter erheblichen Kosten. Aus diesem Grund wird die Anfrage an einen Teilnehmer geschickt, der voraussichtlich die besten Ergebnisse liefert. Der Zielpunkt für das Datenpaket steht bei der Versendung der Anfrage noch nicht fest, sondern hängt eng mit der im Datenpaket nachgefragten Information zusammen. Der lokale Index erkennt die Inhalte der Anfragen und gibt dann Zeiger zu den Dokumenten mit dem gewünschten Inhalt aus.

Das Problem der Indizierung in P2P Netzwerken sieht der Indizierung in verteilten Datenbanken sehr ähnlich. Hier werden Annahmen getroffen um das Problem leichter greifbar zu machen. In einer verteilten Datenbank sind die Teilnehmer ständig miteinander verbunden. Fallen sie aus, werden sie wieder repariert und wieder dem System zugefügt. Die Anzahl der Teilnehmer ist in einer verteilten Datenbank relativ klein. Diese Annahmen sind in P2P Netzen aber nur sehr begrenzt einsetzbar, da diese Netzwerke in der Regel Zehntausende oder sogar Hunderttausende Teilnehmer haben können.

Ein Beispiel für diese verteilten Indizes in Peer-to-Peer Netzen ist das Freenet. Hier baut jeder Teilnehmer einen Index auf, der den Ort der zuletzt nachgefragten Dokumente enthält. Werden diese Dokumente später ein weiteres Mal nachgefragt, ist es möglich sie mit relativ geringen Kosten wieder zu beschaffen.

Leider ist dieses System nicht ohne Nachteile. Die Anfragen werden durch die ID eines einzelnen Dokumentes begrenzt. Außerdem ist eine gewisse Zeit nötig um einen brauchbaren Index in einem Teilnehmer aufzubauen.

## 7.2.2. Suche in dezentralen Hashtabellen

Verteilte Hashtabellen verbessern die Skalierbarkeit und die Treffer Genauigkeit des Systems. Der große Nachteil des Systems ist, daß nur exakte Treffer möglich sind. Komplexe Suchoptionen gibt es nicht.

Einfache Suchsprachen, die meist im Bereich des Filesharing eingesetzt werden, sollen bestimmte Daten finden. Sie suchen nach dem Prinzip: „Finde alle Dateien, die <**> enthalten“.

Komplexe Suchsprachen geben sich damit nicht zufrieden. Sie wollen Kombinationen zwischen den gefundenen Gegenständen erlauben.

Beispiel: Bei Gnutella ist es möglich eine Suche zu starten mit dem Inhalt: „Suche nach Musik von J.S. Bach“; allerdings ist die Suche „Suche alle Choräle von J.S. Bach“ ist unmöglich, weil das Wort Choral im Namen nicht immer vorkommt. (vgl. auch [HARREN1])

Verteilte Hashtabellen (DHT) unterstützen ein Hashtabellen Interface von put(key,value) und get(key). Die verteilten Hashtabellen lösen zwar das Problem der Skalierbarkeit, lassen aber nur exakte Anfragen zu, da in den Hashtabellen gesucht wird. Dadurch wird die Suchsprache weiter vereinfacht und sogar verarmt. Das bedeutet: Hashtabellen lösen zwar ein Problem, schaffen dafür aber ein anderes Problem.

Allerdings gibt es auch hierfür eine Lösung. Die Eigenschaft der Hash Indizes nur exakte Anfragen bearbeiten zu können, läßt sich auch nutzen um Suchen mit Textähnlichkeiten durchzuführen. Zu diesem Zweck wird jeder zu indizierende String in sogenannte n-Gramme zerteilt. N-Gramme sind Substrings der Länge n.

Beispiel: Der Titel „Dark Side of the Moon“ von Pink Floyd kann in folgende 3-Gramme aufgespalten werden:
„Dar“, „ark“, „rk%“, „k%S“, „%Si“, „Sid“, „ide", „de%“ und so weiter.
(Das Zeichen % steht für das Leerzeichen)

Für jeden Index X muß jedes n-Gramm $n_i$ in den Hashindex das Paar ($n_i$,X) eingefügt werden. Dann wird ein Index über die n-Gramme für verschiedene Werte von $n_i$ gebildet. In der Praxis wird vor allem eine Mischung aus 2-Grammen und 3-Grammen verwendet (vgl. auch [HARREN1]).

Fortsetzung des Beispiels: Bei der Suche nach dem Substring „Side“ wird unterteilt in die 3-Gramme „Sid“ und „ide“. Im Index soll jetzt nach jedem 3-Gramm der Anfrage gesucht werden. Als Resultat entsteht eine Liste von Treffern. Diese wird nach Datei ID gruppiert. Des Weiteren wird die Anzahl der Kopien jeder Datei ID ausgerechnet. Für die strenge Substringsuche werden nur die Dateien zurückgegeben, wo die Anzahl der Kopien identisch ist mit der Anzahl der verwendeten n-Gramme. In diesem Beispiel also 2.

Es ist nicht wahrscheinlich, daß die n-Gramme in der korrekten, geordneten Reihenfolge erscheinen. Es kommt also zu einer Übermenge der richtigen Antwort. Dieses Problem ist

allerdings leicht zu beheben. Die Liste wird einfach nachträglich verarbeitet und dann direkt auf Substrings getestet.
Es ist zu beachten, daß die im Beispiel verwendete Anfrage fast direkt in SQL übersetzt werden kann (vgl. auch [HARREN1]):

```
SELECT  H.DateiID, H.Dateiname
FROM  Hashtabelle H
WHERE  H.text  IN <Liste der zu suchenden n-Gramme>
GROUP BY  H.DateiID
HAVING COUNT(*) >= <Anzahl der n-Gramme in der Suchliste>
AND  H.Dateiname  LIKE <Substring>
```

Durch die Verwendung von SQL als Suchsprache wird gezeigt wie universell die Operatoren einsetzbar sind. Sie arbeiten nicht nur auf Datenbankanfragen, sondern auch auf Textsuchen, sogar bei Hash Indizes. Wenn also die relationale Algebra in P2P Netzwerken mit verteiltem Hash Index verarbeitbar ist, dann ist auch der Spezialfall, die Suche nach Substrings möglich.

Allerdings ist es nicht notwendig eine Datenbank zur Suche in P2P Netzen einzusetzen. Die Bearbeitung von relationalen Anfragen an eine Datenbank ist wesentlich mächtiger als die Möglichkeiten, die von P2P Filesharing Tools für die Suche angeboten werden. Auch die Eigenschaften einer Datenbank, wie die transaktionale Speicherung oder das strenge relationale Datenbankmodell, werden in P2P Netzen nicht gefordert. Es wird hier nur wenig Wert auf eine perfekte Speichersemantik gelegt oder auf die gute Verwaltung der Daten. Der Schwerpunkt in einem Peer-to-Peer Netzwerk liegt vielmehr in der leichten Nutzbarkeit des Systems. Hinreichend sind hier eine gute Skalierbarkeit, eine Robustheit gegen Fehler und eine Best Effort Funktionalität. In einem P2P Netz führen die Nutzer meistens keine „Hochleistungssuche“ durch, sondern sie erwarten schnelle und zufriedenstellende Ergebnisse. Diese müssen nicht notwendigerweise vollständig sein.

Es gibt auch im Bereich P2P Anwendungen, bei denen eine transaktionelle Speichersemantik entweder notwendig oder vorteilhaft wären. Leider kennen sich viele Endbenutzer in dieser Materie nicht besonders gut aus. Viele Nutzer haben auch gar kein Interesse daran eine Datenbank aufzubauen oder zu verwalten. Weil das relationale Datenbankmodell hochgradig universell ist und Details vor dem Nutzer verborgen werden können, ist das vielleicht auch nicht nötig. Der Nutzer muß keine komplexe Datenmodulierung beherrschen. Diese Aufgabe kann vom System erfüllt werden. Es muß nur noch eine Suchmaschine oder Eingabemaske implementiert werden, in die der Nutzer wie gewohnt seine natürlichen und bekannten Attribute eingibt.

Das Ziel der Bearbeitung von Suchanfragen im P2P Bereich ist vor allem eine breite Anwendbarkeit. Die Bearbeitung dieser Anfragen soll breit gefächert aber dennoch praktisch nutzbar sein und sowohl mit den P2P Filesharingsystemen als auch mit dem Dateisystem des Nutzers zusammenarbeiten.

Daraus ergeben sich Anforderungen, die an die APIs der verteilten Hashtabellen gestellt werden müssen. Der verteilte Hash Index soll die APIs nicht mit ausführlichen Spezifika von Suchanfragen verkomplizieren. Die Verarbeitungstechnologie der Suchanfragen sollte portabel sein, da es im Bereich Design der verteilten Hashtabellen noch keinen klaren Sieger gibt. Außerdem wird von den APIs noch erwartet so dünn wie möglich zu sein.

Die Architektur dieser Technologie soll hier kurz erläutert werden:
Die DHT Schicht wird für die Mechanismen der Indizierung, aber auch für Routing Mechanismen verwendet.

Die unterste Schicht, der lokale Datenspeicher muß einen Iterator unterstützen. Dieser arbeitet mit einem Interface zusammen, um eine Menge von Objekten zu durchsuchen. Für jedes Objekt muß es Zugriffsmöglichkeiten für die Attribute geben. Die local IC kann ein Byte Array sein und ist ein speicherweiter, eindeutiger Identifier für das Objekt, der den „Inhalt“ des Objektes darstellt. Ein Metadaten Interface soll zusätzliche Attribute der Objekte im Speicher herausfinden. Für diese zusätzlichen Attribute werden dann noch Zugriffsmöglichkeiten benötigt. Die Details der Datenspeicherung sollen hier nicht betrachtet werden, und das Interface als read-only angesehen werden.
In der DHT Schicht wird ein put/get Interface unterstützt. Dieses wird durch einen Iterator (lscan) unterstützt, der sich mit den DHT Bibliotheken verbindet und es dem lokalen Code erlaubt durch alle auf der Maschine gespeicherten DHT Einträge zu iterieren. Die Vorrichtung newData benachrichtigt höhere Schichten wenn es neue Einträge in die lokale verteilt Hashtabelle gibt.
Die oberste Schicht ist die Suchverarbeitungsschicht, oder Query Processing Layer (QP). Hier werden Anfrage Operatoren implementiert. Es gibt eine Unterstützung zum Spezifizieren von Anfragen und zum Iterieren durch die Anfrage Ergebnisse.

In Systemen mit verteilten Hashtabellen wird ein breiter Raum für Namen und Tabellen benötigt. Tabellen, temporäre Tabellen, Tupel in einer Tabelle oder auch Felder in einem Tupel müssen benannt werden können. Es muß daher ein hierarchischer Namensraum implementiert werden, der auf dem flachen von DHT zur Verfügung gestellten Identifier Space aufsetzt. Dazu werden die Identifier in mehrere Felder zerteilt und jedes Feld kann dann Objekte identifizieren. Das Routing Protokoll muß dann auch verändert werden, da ein hierarchischer Namensraum komplexere Routingprimitiven benötigt als das normale Multicast.

## 7.2.3. Verbesserung des Mechanismus des Flutens

Das Fluten von Netzwerken ist problematisch, da eine große Last erzeugt wird. Eine Verbesserung des Mechanismus ist also unumgänglich. Die Suche in einem unstrukturierten Netzwerk soll ermöglicht werden und dennoch skalierbar sein. Ein Ansatzpunkt für Verbesserung ist der Time to Live (TTL) Zähler. Es ist nicht leicht zu bestimmen, wie hoch der TTL Zähler sein sollte.

Ein erster Versuch ist, daß die Empfänger einer Anfrage eine Rückfrage an den Originalsender senden, bevor sie die Nachricht weiterleiten. Das führt aber leider zu einer Explosion der Anzahl der Nachrichten am Anfrageknoten. Daher ist dieser Lösungsversuch leider nicht weiter brauchbar.

Der zweite Versuch ist es eine sogenannte „successive flood“ zu initiieren. Das funktioniert wie folgt:
Ein Teilnehmer beginnt mit dem Fluten und nimmt dazu einen kleinen TTL. Er wartet dann ab, ob seine Sucher erfolgreich ist. In diesem Fall ist das Ziel erreicht und die Suche kann beendet werden. Ist die Suche nicht erfolgreich, vergrößert der Teilnehmer den TTL und flutet das Netz erneut. Diese Methode löst das Problem der Suche nach einem geeigneten TTL. Das Problem der Nachrichtenvermehrung beim Fluten bleibt jedoch ungelöst. Um auch dieses Problem zu lösen ist eine weitere Verbesserung nötig.

Der sogenannte „random walk“ ist eine Technik, die die Anfragenachricht an einen zufällig ausgewählten Nachbarn weiterleitet. Dies wird so lange fortgeführt bis das gesuchte Objekt gefunden wird. Die Anfragenachricht wird in diesem Kontext als „walker“ bezeichnet. Die Verwendung von genau einem walker reduziert den Overhead der Nachricht stark.

Der Anfrageknoten schickt hier nur eine Nachricht. Es können aber auch mehrere walker parallel eingesetzt werden. Der Anfrageknoten schickt in diesem Fall n Anfragen. Jede dieser n Anfragen absolviert dann einen random walk. Das bedeutet: je mehr walker eingesetzt werden, umso schneller kann das Objekt gefunden werden. Anderseits wird bei mehreren walkern auch mehr Last erzeugt. Es gibt einen Sättigungspunkt hinsichtlich der Anzahl der walker. Das heißt, ab einer bestimmten Anzahl von walkern ist ein zusätzlicher walker nicht mehr nutzbringend, sorgt aber für erheblich mehr Netzverkehr.

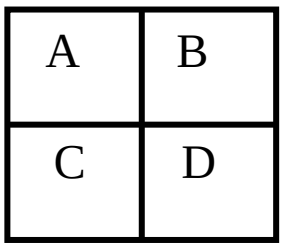
Werden mehrere walker eingesetzt, müssen Methoden gefunden werden diese „walks“ zu beenden. Hierzu gibt es zwei Möglichkeiten, nämlich die Verwendung von TTL oder das „checking“.

Bei der Verwendung von TTL bei den walkern, halten die walker nach einer bestimmten Anzahl von hops an.

Beim checking nimmt der walker regelmäßig Rückfragen beim Absender der Anfrage vor, bevor er zum nächsten Knoten weitergeht. Auch hier wird ein TTL verwendet. Dieser ist jedoch sehr groß und soll vor allem Zyklen vermeiden. Auf diese Weise wird eine Nachrichtenexplosion beim Anfrageknoten vermieden. Da pro check aber ein Nachrichtenpaar erzeugt wird, wird auch viel Overhead erzeugt (vgl. auch [LV1]).

## 7.2.4. Inhaltsadressierbare Netzwerke

Das Inhaltsadressierbare Netzwerk (Content Addressable Network, CAN) arbeitet auf dem Prinzip der Hashtabellen. Die Basisoperationen umfassen das Einfügen, Suchen und Löschen von (key,value)-Paaren. Das Netzwerk besteht aus vielen Knoten, die in einem karthesischen Raum angeordnet sind (siehe Abb. 7.2.4.(1)). Jeder Knoten speichert einen Teil der gesamten Hashtabelle, genannt zone. Außerdem speichert jeder Knoten Informationen über eine kleine Anzahl von benachbarten zones in einer Tabelle. Anfragen wie insert (Einfügen), lookup (Suchen) und delete (Löschen) für einen speziellen key (Schlüssel) werden von den dazwischenliegenden CAN Knoten zu demjenigen CAN Knoten geroutet, dessen zone diesen Key enthält. Damit lernt und speichert der Knoten die IP Adressen von den Knoten, deren zones mit der eigenen zone benachbart sind.
Jeder CAN Knoten enthält eine Koordinaten Routingtabelle, die IP-Adresse und virtuelle Koordinatenzone von jedem seiner Nachbarn im Raum enthält.

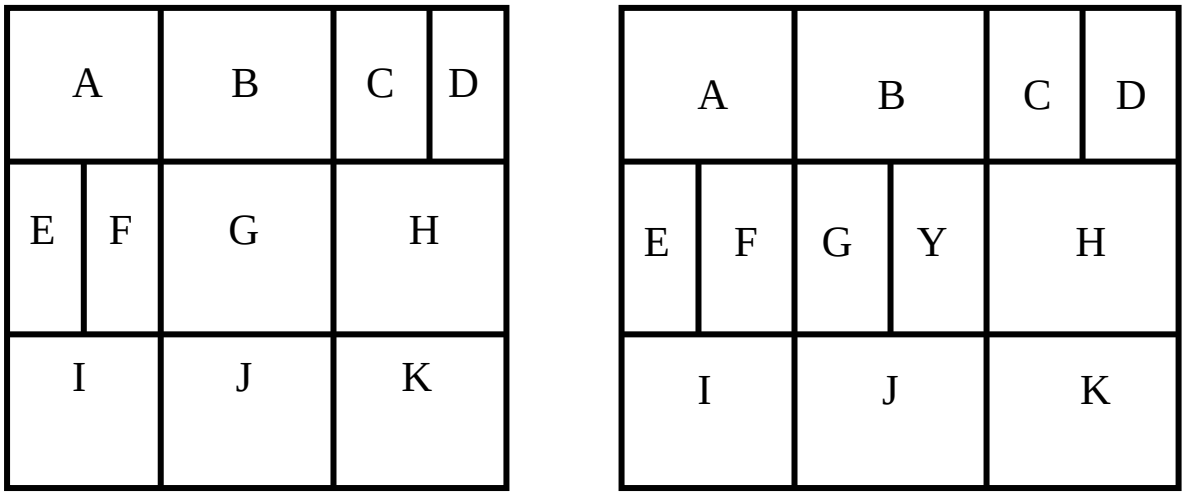



| | |
|---|---|
| Abb. 7.2.4.(1) | Darstellung 2D CAN Raum |

## 7.2.4.1. Konstruktion des CAN

Der CAN Raum wird zwischen den sich im System befindlichen Knoten aufgeteilt. Wenn das Netz wachsen soll, muß jeder neu eingefügte Knoten seinen eigenen Anteil am Gesamtraum erhalten. Verbindet sich also ein neuer Knoten mit dem Netzwerk, muß ein bereits existierender Knoten seine zone halbieren und dem neuen Knoten eine Hälfte davon übergeben.

Das Einfügen eines neuen Knotens vollzieht sich in drei Schritten. Im ersten Schritt sucht der neue Knoten einen bereits im CAN befindlichen Knoten. Im zweiten Schritt wird unter Benutzung der CAN Routing Mechanismen ein Knoten gefunden, der seine zone teilt. Im dritten Schritt werden die Nachbarn der geteilten zone benachrichtigt, damit sie ihre Routing Tabellen aktualisieren können. Das Hinzufügen eines Knotens ist in Abb. 7.2.4.1.(1) dargestellt.

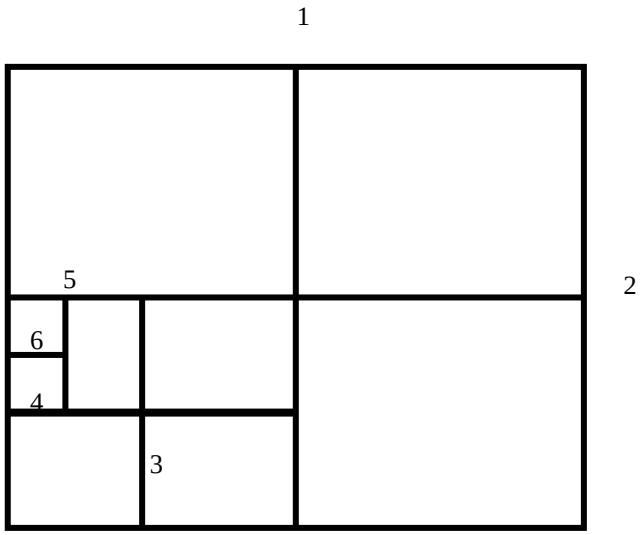



| Abb. 7.2.4.1.(1) | Darstellung: Einfügen eines CAN Knotens |
|---|---|

In der Abb. 7.2.4.1.(1) wird gezeigt wie sich der Knoten Y in den CAN Raum einfügt. Links ist der aktuelle Zustand des CAN Raumes vor der Ankunft von Y dargestellt. Rechts befindet sich der CAN Raum nach dem Einfügen von Y.
In diesem Beispiel teilt der Knoten G seine zone und übergibt eine Hälfte davon an den Knoten Y. Der Knoten Y lernt dann nach Erhalt seiner zone die IP Adressen seiner Nachbarn. Die erforderlichen Angaben erhält er von Knoten G. Der Knoten G aktualisiert die Liste seiner Nachbarn und streicht die ehemaligen Nachbarn aus der Liste, die nicht mehr seine Nachbarn sind. Danach werden sowohl die Nachbarn von Knoten G als auch die Nachbarn von Knoten Y über die Neuverteilung des Raumes informiert.

Die Nachbarliste von Knoten G vor der Teilung der zone:
F, B, J, H
Die Nachbarliste von Knoten G nach der Teilung der zone:
F, B, J, Y
Die Nachbarliste von Knoten Y nach dem Einfügen in den CAN Raum:
B, J, H, G

Der neue Knoten Y sucht sich zufällig einen Punkt im Raum und schickt hierhin einen JOIN-Request. Diese Nachricht wird von einem existieren CAN Knoten in das CAN geschickt. Jeder CAN Knoten verwendet die CAN Routingmechanismen um die Nachricht weiterzuleiten bis sie Knoten G erreicht, in dessen zone sich der zufällig gewählte Punkt befindet. Der Knoten G teilt seine zone derart in zwei Häften, daß diese wieder zusammengeführt werden können, wenn ein Knoten das System wieder verläßt. Dabei wird die zone zuerst entlang der X-Achse geteilt, dann entlang der Y-Achse und so weiter. Das Prinzip der Teilung der zones ist in Abb. 7.2.4.1.(2) dargestellt.




| Abb. 7.2.4.1.(2) | Reihenfolge der Teilung der CAN zones |
|---|---|

## 7.2.4.2. Funktionsweise des CAN

Jeder Knoten sendet regelmäßig Nachrichten mit den Daten seiner aktuell zugeteilten zone an seine Nachbarn. Auf diese Weise erkennen die Nachbarn Änderungen schnell und können damit schnell die Liste ihrer Nachbarn aktualisieren. Neue Knoten beeinflussen daher nur wenig andere Knoten im Raum und haben so nur einen geringen Einfluß auf den Gesamtraum.

Die Anzahl der Nachbarn eines Knotens hängt sowohl von der Dimension des karthesischen Raumes ab, als auch von der Gesamtzahl der Knoten im Raum. Ein CAN Raum muß nicht 2-Dimensional sein, wie im Beispiel aus Kapitel 7.2.4.1, sondern kann auch mehrdimensional sein. Wichtig für CANs mit vielen Knoten ist daß das Einfügen eines neuen Knotens nur O(Anzahl der Dimensionen des Raumes) viele Knoten beeinflußt (vgl. auch [RATNASAMY1]).

Ein Knoten kann das System jederzeit verlassen. Das kann freiwillig sein, oder auch aufgrund eines Ausfalls eines Knotens. Es müssen also für das Ausscheiden eines Knotens aus dem System Vorkehrungen getroffen werden.

Im Normalfall übergibt der ausscheidende seine zone inklusive seiner Datenbank explizit an seinen Nachbarn. Das funktioniert genau dann, wenn die zone des ausscheidenden Knoten und die zone seines Nachbarn zusammen eine legale zone ergeben. Ist die entstehende Ergebniszone nicht legal, übergibt der ausscheidende Knoten seine zone an den Nachbarn mit

der kleinsten zone. Dieser Nachbar muß dann beide zones verwalten. Es gibt natürlich auch den Fall, daß ein Knoten ausfällt und deshalb seine zone nicht mehr an einen Nachbarn ordnungsgemäß übergeben kann. In diesem Fall greift der sogenannte Takeover Algorithmus. Dieser stellt sicher daß einer der Nachbarn die zone des ausgefallenen Knoten übernimmt. Die Datenbank mit den (key,value)-Paaren des ausgefallenen Knotens sind verloren, bis sie durch die Datenbesitzer wieder aktualisiert werden.

Es gibt auch eine Methode den Ausfall eines Knotens zu erkennen. Der Knoten sendet normalerweise regelmäßig Aktualisierungsbotschaften an seine Nachbarn. Diese Botschaften enthalten die zone-Koordinaten des Knotens und die Liste seiner Nachbarn und deren Koordinaten. Der Ausfall eines Knotens wird genau dann erkannt, wenn die erwarteten Botschaften über einen längeren Zeitraum ausfallen. Wenn ein Nachbar einen Ausfall erkennt, sendet er eine Takeover Nachricht an alle Nachbarn des ausgefallenen Knotens und beginnt mit dem Takeover Mechanismus. Damit wird die Raumaufteilung des Gesamtraumes gewährleistet.

Die Koordinatenräume können auch mehrdimensional sein. Je mehr Dimensionen vorliegen, umso kürzer werden die Routing Wege. Allerdings steigt mit der Anzahl der Dimensionen auch die Anzahl der Nachbarknoten. Damit steigt die Fehlertoleranz beim Routing, da jeder Knoten mehr potentielle next hop Knoten hat, über die Nachrichten geroutet werden, falls einer oder mehrere Nachbarknoten zusammenbrechen.
Steigt die Anzahl der Dimensionen weiter, werden mehrere unabhängige Koodinatenräume erstellt, die realities genannt werden. Jeder Knoten hat in jedem Raum seine zone. In einem CAN mit n realities hat jeder Knoten n zones, genau eine pro reality. Dadurch entstehen dann pro Knoten auch n unabhängige Nachbarmengen. Der Inhalt der Hashtabelle wird über jede reality repliziert und damit wird auch die Datenverfügbarkeit erhöht.

Es ist auch möglich die Koordinatenzones zu überladen. Bisher wurde nur der Fall betrachtet, daß es nur einen Knoten pro zone gibt. Aber es ist auch möglich, daß sich mehrere Knoten, auch termed peers genannt, sich die gleiche zone teilen. Jeder Knoten besitzt dann zusätzlich zur Liste seiner Nachbarn auch eine Liste seiner Peers. Er kennt zwar alle Knoten in seiner eigenen zone, muß aber nicht alle Knoten seiner Nachbarzones kennen. Er kann sich aus jeder Nachbarzone einen bekannten Knoten auswählen.

Weiterhin ist in jeder zone der Systemparameter MAXPEERS enthalten. Dieser gibt die maximale Anzahl von Peers in einer zone an. Das Überladen einer zone steigert nicht die Menge an Informationen, die ein Knoten über seine Nachbarn kennen muß. Allerdings muß zusätzlich der Zustand für bis zu MAXPEERS Peerknoten enthalten sein.

Wenn sich ein $\text{Knoten}_{(neu)}$ an das System anschließen will, sucht er sich einen $\text{Knoten}_{(alt)}$, dessen zone besetzt werden kann. Der $\text{Knoten}_{(alt)}$ teilt seine zone nicht, sondern prüft seinen MAXPEERS Zustand. Ist das Maximum noch nicht erreicht, schließt sich der $\text{Knoten}_{(neu)}$ ohne Raumaufteilung an und erhält eine Liste der Peers und der Nachbarn von $\text{Knoten}_{(alt)}$. Die regelmäßigen Aktualisierungen von $\text{Knoten}_{(neu)}$ informieren die Nachbarn daß $\text{Knoten}_{(neu)}$ mit dem System verbunden ist.
Falls MAXPEERS bereits erreicht ist, wird der Raum aufgeteilt. Der $\text{Knoten}_{(alt)}$ informiert seine Peers über die Teilung der zone. Die Peers teilen sich auf die beiden beteiligten zones auf und jeder neue Knoten erhält vom alten Knoten eine Liste der Peers und Nachbarn.

Das Überladen der zones hat Vorteile. Die Pfadlänge wird verkleinert, da es jetzt mehr Knoten pro zone gibt. Das wirkt sich aus wie eine Reduzierung der Knotenzahl im System.

Der Knoten hat nun eine größere Auswahl in der Wahl der Nachbarknoten und kann nun einen Nachbarknoten mit geringerer Auslastung zur Weiterleitung der Nachrichten auswählen. Das Überladen der zones führt auch zu einer höheren Fehlertoleranz. Die zone kann nur dann leer werden, wenn alle Knoten in der zone gleichzeitig ausfallen.
Nachteilig wirkt sich allerdings aus, daß es nur ein einfaches Interface für die Speicherung und das Wiederauffinden der (key,value)-Paare gibt. Außerdem braucht der Nutzer für das Auffinden von Dokumenten die genaue Dokumenten ID, den key. Das ist aber wenig praktikabel, da die Inhalte von vielen Quellen unabhängig voneinander erzeugt werden, und die genaue ID nicht immer verfügbar ist (vgl. auch [RATNASAMY1]).

## 7.2.5. Peer Search

Eine weitere Suchmöglichkeit bietet das auf CAN aufsetzende Peer Search. Es ist völlig dezentralisiert und es gibt weder eine komplexe Hierarchie noch einen single point of failure. Peer Search unterstützt die inhaltliche und semantische Suche, die anders als die Schlüsselwortsuche in natürlicher Sprache durchgeführt werden kann. Bei der Suche und der Indizierung wird das gefürchtete Fluten des Netzwerkes vermieden. Peer Search ist skalierbar und effizient.

Die Effizienz wird durch eine Kombination aus Indexplazierung und Anfragerouting erreicht. Bei der Suche muß nur eine geringe Anzahl von Knoten besucht werden.

Die Dokumente und Anfragen werden als Vektor dargestellt. Der Kosinus des Winkels zwischen ihren Vektordarstellungen stellt das Maß der Ähnlichkeit zwischen den Dokumenten dar. Der Dokumentenindex wird im CAN gespeichert. Dazu wird die Vektordarstellung des Dokumentes als Koordinaten im CAN verwendet. Das Ergebnis ist, daß Indizes, die nahe beieinander liegen, sich auch semantisch sehr nahe sind.

Im CAN wird der logische Raum als n-dimensionaler karthesischer Raum aufgefaßt und in zones aufgeteilt. Pro zone kann es einen oder mehrere Besitzer geben. Der Objektschlüssel, oder object key, ist ein Punkt im Raum und das Objekt wird in einem Knoten gespeichert, in dessen zone der Punkt liegt. Das Routing vom Quellknoten zum Zielknoten entspricht daher de, Routing von der Quellzone zur Zielzone. Das kann allerdings ungünstig sein, da die darunterliegende physikalische Netzwerktopologie vernachlässigt wird. Daher werden bei Peer Search die sogenannten Expressways definiert. Diese Expressways zeichnen sich dadurch aus, daß hier nur Routen verwendet werden, die sich möglichst dicht an die darunterliegende Internet Topologie annähern.

Um das System von Peer Search besser erläutern zu können, sind einige Begriffe zu erklären.

### 7.2.5.1. Vector Space Model

Um Dokumente und Anfragen als Vektoren darstellen zu können, ist ein Vektorraum nötig. Dieser Vektorraum wird Vector Space Model (VSM) genannt. Jede Komponente des Vektors stellt die Wichtigkeit eines Wortes, oder term, in einem Dokument oder einer Anfrage dar. Zur weiteren Präzisierung läßt sich auch das Gewicht einer Komponente festlegen. Die Wichtigkeit eines terms im Dokument wird dadurch bestimmt wie oft ein term in dem Dokument erscheint. Dieser Wert soll als TF bezeichnet werden. Wichtig ist aber auch wie oft dieser term in anderen Dokumenten vorkommt. Dieser Wert wird hier als IDF bezeichnet. Das Gewicht der Komponente berechnet sich dann wie folgt:

Gewicht der Komponente = TF * IDF

Wenn ein term häufig in einem Dokument auftaucht, bestehen gute Chancen daß man diesen term verwenden kann, um dieses Dokument von anderen Dokumenten zu unterscheiden. Wenn ein term allerdings in anderen Dokumenten ebenfalls häufig auftaucht, z.B. das Wort „Technik“ in technisch orientierten Schriften, sollte die Wichtigkeit des terms verringert werden.

Im Allgemeinen wird die euklidische Norm des Vektors eines Dokumentes verwendet um die Unterschiede in der Länge der verschiedenen Dokumente auszugleichen. Bei einer Anfrage wird der Anfragevektor mit allen Dokumentenvektoren verglichen. Die Dokumentenvektoren, die dem Anfragevektor am nächsten liegen, werden als ähnlich klassifiziert und an den Initiator der Anfrage ausgegeben. Eine verbreitete Meßart für die Bestimmung der der Ähnlichkeit zwischen Vektoren ist der Kosinus des Winkels zwischen den Vektoren (vgl. auch [TAN1]).

Bei der Schlüsselwortsuche ergeben sich neue Probleme. Die Existenz von Synonymen und Polysemen, aber auch das Rauschen in den Dokumenten, wirken sich nachteilig auf die Sucheingenschaften aus.

### 7.2.5.2. Latent Semantic Indexing und Singular Value Decomposition

Das Latent Semantic Indexing (LSI) könnte das Problem der Synonyme lösen. LSI verwendet die sogenannte Singular Value Decomposition (SVD) um eine Matrix von Dokumentenvektoren zu transformieren und zu trunkieren, um die darunterliegende Semantik in den terms zu erkennen.

Dokumentenvektoren mit vielen Dimensionen werden in sogenannte Semantikvektoren mit mittelmäßig vielen Dimensionen transformiert, indem der Dokumentenvektor in einen semantischen Unterraum von mittlerer Dimension projiziert wird. Die Basis dieses semantischen Unterraumes wird unter Verwendung von SVD berechnet. Die Semantikvektoren werden normalisiert und ihre Ähnlichkeiten wie oben beschrieben gemessen. Es wird also festgestellt, daß sowohl VSM als auch LSI Dokumente und Anfragen als Vektoren darstellen. Die Ähnlichkeit einer Anfrage zu einem Dokument wird als Kosinus des Winkels zwischen ihren Vektordarstellungen gemessen. Diese Eigenschaften sind auch schon die einzigen Eigenschaften für die Algorithmen, auf die sich Peer Search verläßt.

Die Frage stellt sich, wie VSM und LSI Algorithmen erweitert werden müssen, um mit CAN zusammenzuarbeiten.

### 7.2.5.3. Peer-to-Peer Vector Space Model

Um eine Zusammenarbeit mit dem CAN zu erreichen wird das Peer-to-Peer Vector Space Model (P-VSM) eingeführt. Der Inhalt eines Dokumentes kann durch eine kleine Anzahl von Schlüsselworten beschrieben werden. Beim P-VSM ist jeder Knoten selber dafür verantwortlich dafür die Indizes, die spezielle Schlüsselworte enthalten, zu speichern.

Dazu ist ein Dokument gegeben. Das VSM wird dann verwendet um die wichtigen Schlüsselworte automatisch zu erkennen. Der Index des Dokumentes wird dann in dem Knoten veröffentlicht, der für diese speziellen Schlüsselworte verantwortlich ist. Aus diesem Dokument wird dann unter Benutzung seine Vektordarstellung berechnet. Die m schwersten Vektorkomponenten, oder terms, $t_i = 1 \ldots m$ werden abhängig von der Peerbasis festgelegt

und alle ($h(t_i)$,index)-Paare werden im DHT unter Verwendung des Expressway Routing gespeichert, wobei h die Hash Funtion ist, die Strings in Punkte des karthesischen Raumes abbildet.

Bei einer Suchoperation wird die Anfrage an die Knoten weitergeleitet, die für die in der Anfrage enthaltenen Schlüsselworte verantwortlich sind. Diese führen dann die Suche unter Verwendung von VSM lokal durch. Die passenden Indizes werden als Ergebnis ausgegeben. Jeder term in der Anfrage wird in einem Punkt unter der Verwendung von h gehasht. Dann wird die Anfrage zu den Knoten geroutet, deren zones diese Punkte enthalten. Jeder gefragte Knoten sucht dann lokal und schickt die Ergebnisse an den anfragenden Knoten zurück. Der anfragende Knoten ordnet die erhaltenen Ergebnisse nach einem Rangsystem. Die Ergebnisse mit den niederen Rängen werden dann verworfen, der Rest dem Nutzer präsentiert.

Leider können Synonyme und Polyseme hier noch immer zu Problemen führen. Beispielsweise kann das Wort „Laster“ in einem Dokument sowohl einen Lastkraftwagen als auch eine schlechte Angewohnheit bezeichnen. Ein Thesaurus kann hier jedoch Abhilfe schaffen.

Weiterhin soll das Peer to Peer Latent Sematic Indexing (P-LSI) eingeführt werden. Hier gibt es zwei Versionen: den vereinfachten P-LSI, oder simplified P-LSI, und den vollständigen P-LSI, oder full P-LSI.

Zuerst soll der vereinfachte P-LSI vorgestellt werden. Im LSI Semantikraum werden $\alpha$ und $\kappa$ zur Beschreibung benutzt. Die Dimensionen der Räume sollen als l und k bezeichnet werden. Jedes Dokument wird auf einem Punkt in $\kappa$ abgebildet, indem $l = k^3$ und der Semantikvektor als seine Koordinaten in $\kappa$ betrachtet wird. Der Semantikvektor S des Dokumentes wird mittels LSI berechnet und das Paar (S,index) wird mittels Expressway Routing im DHT gespeichert. (vgl. auch [TAN1]).

In einer Suchoperation wird der Semantikvektor Q einer Anfrage berechnet. Die Anfrage wird geroutet indem man Q als DHT Schlüssel verwendet. Bei der Ankunft am Zielort flutet die Anfrage nur Knoten innerhalb eines vorab berechneten Radius r, basierend auf einer Ähnlichkeitsschwelle, die vorher durch den Nutzer festgelegt wurde. Alle Knoten, die die Anfrage erhalten führen unter Verwendung von LSI eine lokale Suche durch und geben die Resultate wie im P-VSM an den Nutzer zurück. Weil die Indizes von Dokumenten, die der Anfrage ähnlich sind und innerhalb der Schwelle liegen, nur innerhalb des Radius liegen können und in dieser Umgebung eine umfassende Suche durchgeführt wird, erreicht der P-LSI die gleiche Leistung wie der LSI. Normalerweise ist der Radius r klein und die beteiligte Knotenzahl gering im Vergleich zur Gesamtknotenzahl. Dennoch gibt es Probleme. Beispielsweise sind die Dokumentenvektoren in $\alpha$ nicht gleichverteilt. Damit leidet dieses Verfahren unter hot spots, auch wenn die Knoten selber gleichverteilt sind.

Der vollständige P-LSI versucht dieses Problem zu lösen. Der Semantikvektor $S = (s_1, s_2, ..., s_l)$ in $\alpha$ wird in den Parametervektor $(\theta_1, \theta_2, ... , \theta_{l-1})$ im (l-1)-dimensionalen polaren Unterraum P transformiert, indem man die Gleichung $\Theta_j = \arctan(\frac{s_{j+1}}{\sqrt{\sum_{i=1}^{j} s_i^2}})$ verwendet.

Nach der Transformation sind die Parametervektoren in P noch immer nicht gleichverteilt. Das macht weitere Änderungen des vereinfachten P-LSI Algorithmus nötig. Zunächst wird $l-1 = k$ gesetzt. Dann wird bei einem gegebenen Dokument oder gegebener Anfrage noch der obigen Gleichung der Parametervektor berechnet und anstelle des ursprünglichen

Semantikvektors als Schlüssel für das DHT Routing benutzt. (vgl. auch [TAN1]) Bei dem Verbindungsaufbau eines Knotens wird ein Dokument betrachtet, das der Knoten veröffentlichen will, und der Parametervektor des Dokumentes als Zufallspunkt benutzt, zu dem der JOIN Request gerouted wird.

Es werden drei Ziele erfüllt. Jeder Knoten speichert in etwa die gleiche Anzahl von Indizes, was als load balancing bezeichnet wird. Bei einer großen Anzahl von Knoten nähert sich die Knotenverteilung $\kappa$ der Indexverteilung an. Zweitens sind die Indizes, die in einem Knoten gespeichert werden, dem Inhalt, den der Knoten selber veröffentlicht, sehr ähnlich. Das wird als index locality gespeichert. Es ist wahrscheinlich daß der Knoten die Indizes für seine eigenen Inhalte speichert. Die Veröffentlichung der Inhalte ist daher sehr effizient. Zuletzt gilt die Annahme daß die eigenen Dokumente eines Knotens die Interessen dieses Knotens wiederspiegeln. Daraus folgt dann daß die Anfragen zumeist von den Nachbarknoten verarbeitet werden, was als query location bezeichnet wird. Es ergeben sich daher fast konstante Kosten für das DHT Routing.

#### 7.2.5.4. Fazit

Die Flexibilität der unterliegenden Technologien von Peer Search erlaubt den Einsatz von vielen Anwendungen. Es gibt keine Beschränkung auf die Suche nach Dokumenten. Beispielsweise kann Peer Search auch für die Suche nach Audiodateien oder Videodateien eingesetzt werden. Algebraische Eigenschaften können aus Bildern extrahiert werden oder Frequenzvektoren aus Musik berechnet werden. Der Einsatz von Vektoren bei Peer Search bewirkt daß das Suchen und das Routing zu ein und demselben Problem werden. Damit ist Peer Search allgemein betrachtet auf jedes Medium anwendbar, welches auf Vektoren abstrahiert werden kann und wo die Objektähnlichkeit als eine Distanz im Vektorraum gemessen werden kann.

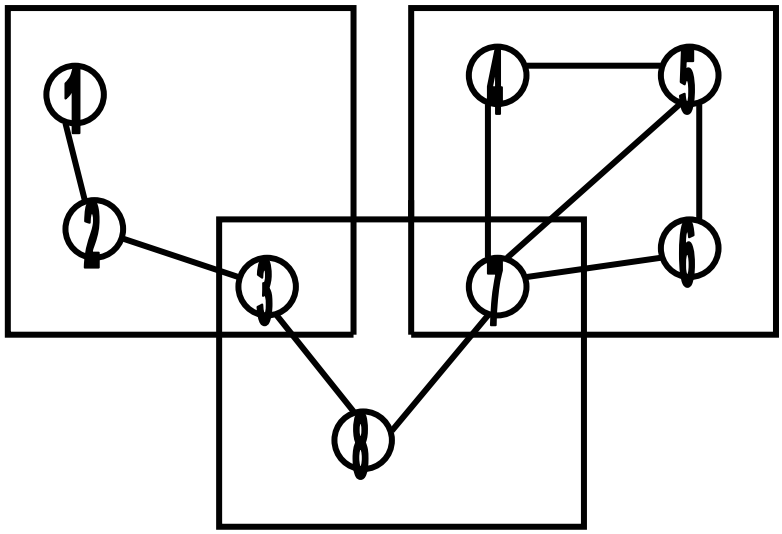
### 7.2.6. Semantic Overlay Networks

Das Semantic Overlay Network (SON) besteht wird aus semantisch verwandten Knoten gebildet. Anfragen an das Netzwerk werden zum passenden SON oder zu den passenden SONs geroutet. Die Chancen steigen damit Dateien schneller zu finden. Gleichzeitig wird die Suchlast für Knoten mit nichtverwandten Inhalten reduziert. Diese stehen dann weiterhin für andere Operationen zur Verfügung. Auf diese Weise kann die Performanz für Anfragen verbessert werden.

Dem Nutzer wird gleichzeitig erlaubt zu entscheiden, welche Inhalte auf dem eigenen Rechner gespeichert werden dürfen und mit wem Verbindungen zugelassen werden.

In Random Overlay Netzen werden die Anfragen blind von Knoten zu Knoten weitergeleitet. Häufig werden Inhalte in Knoten basierend auf Hashfuntionen dargestellt. Das erleichtert es Inhalte wiederzufinden und bietet eine gute Performanz für Punktanfragen, d.h. Anfragen für exakt bekannte Schlüsselworte. Leider ist die Arbeitsweise ineffizient, wenn Textanfragen oder „ungenaue“ Anfragen gestellt werden.

SON ist eine flexible Netzwerkorganisation. Die Knoten sind hochgradig autonom. Sie werden mit Knoten mit ähnlichen semantischen Inhalten zusammengebracht.




| Abb. 7.2.6.(1) | Darstellung einer SON Organisation |
|---|---|

In Abb. 7.2.5.(1) wird eine SON Organisation beispielhaft vorgestellt. Jedes Viereck stellt einen semantisch ähnlichen Bereich, ein SON dar. Die Knoten 1, 2 und 3 sollen die Komödie darstellen. Die Knoten 4, 5, 6 und 7 stellen die Tragödie dar. Die Knoten 3, 7 und 8 befassen sich mit Lyrik. Die Knoten 3 und 7 liegen in zwei Bereichen, nämlich der Komödie und Lyrik im Falle von Knoten 3 und im Bereich der Tragödie und Lyrik im Falle des Knoten 7.
Es ist nicht erforderlich daß die Knoten innerhalb eines semantischen Bereiches alle vollständig miteinander verbunden sind. Schließlich gibt es keine Vorschriften über die Verbindungen innerhalb eines SON. Außerdem können die Knoten zu mehr als zu einem SON gehören. Eine weitere Partitionierung der SONs ist auch möglich.

Für die Abarbeitung der Anfragen wird zunächst festgestellt welches SON oder welche SONs zum Finden der Antwort am besten geeignet ist oder sind. Erst danach wird die Anfrage an einen Knoten im betreffenden SON geschickt. Dieser leitet die Anfrage dann nur an die Knoten innerhalb seines SONs weiter.

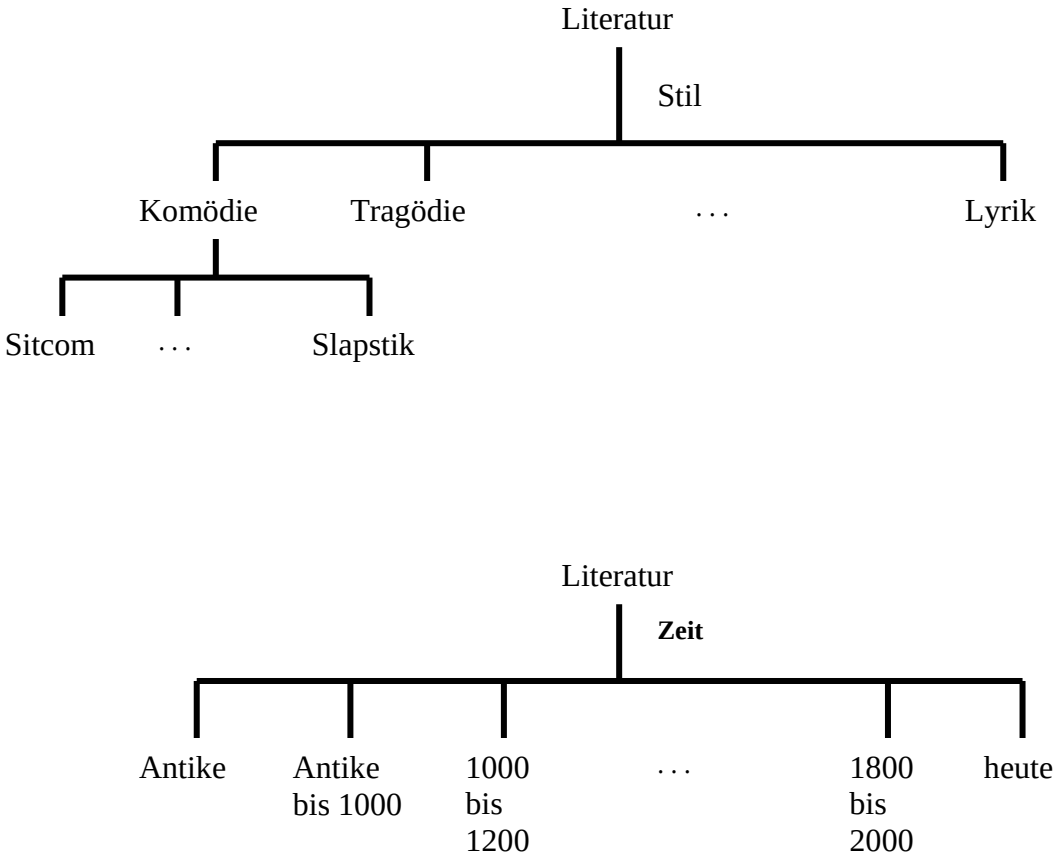
So werden Anfragen zum Thema Lyrik nur an Knoten weitergeleitet, die Lyrikinhalte besitzen, denn hier ist das Auffinden einer richtigen Antwort auf die Anfrage am wahrscheinlichsten. Die Zeit, die für das Finden der Antwort benötigt wird, wird damit verkürzt. Es werden nicht mehr alle Knoten des Gesamtnetzwerkes abgesucht, sondern nur diejenigen Knoten, die inhaltlich zur Anfrage passen könnten. Knoten außerhalb des Lyrik SONs werden von der Anfrageoperation nicht beeinflußt. Sie können ihre Ressourcen zur Steigerung der Performanz für andere Anfragen zur Verfügung stellen.

Beim Aufbau von SONs gibt es allerdings einige Herausforderungen zu meistern. Es ist extrem wichtig eine gute Klassifizierungsstrategie zu finden. Das ist für die spätere Performanz des Netzes bei den Anfragen wichtig. Im obigen Beispiel gilt es zu klären was überhaupt unter Lyrik verstanden werden soll. Außerdem müssen innerhalb der Klasse die Feinheiten der Ausprägungen festgelegt werden. Im Beispiel der Lyrik ist eine weitere Unterteilung beispielsweise in Balladen, Sonette, Schüttelreime, etc. möglich.

Auch zu klären ist die Frage wann sich ein Knoten in einem SON anschließen kann, bzw. wie viele Dokumente ein Knoten zu einem bestimmten Themenbereich haben muß, um sich einem

bestimmten SON anschließen zu dürfen. Eine Klassifizierung soll das Problem lösen, daß bei einer gegebenen Anfrage eine kleine Menge von SONs, deren Knoten eine hohe Trefferzahl haben, gefragt werden. Das hat gewisse Vorteile. Bei einer derart geeigneten Klassifizierung werden die Knoten, die eine Anfrage erhalten, viele Treffer zurückgeben können. Das verkürzt die Antwortzeit für die Anfrage. Knoten, die nur wenig Resultate für diese Anfrage bieten können, erhalten diese Anfrage gar nicht erst. Auf diese Weise wird die Verschwendung von Ressourcen auf diese Anfrage vermieden. Andere Anfragen können dadurch schneller bearbeitet werden. (vgl. auch [CRESPO2])

Die Klassifikationshierarchie sollte als Basis zur Erstellung von SONs verwendet werden.




| Abb. 7.2.6.(2) | Darstellung einer Klassifikationshierarchie |
|---|---|

Ein Dokument oder eine Anfrage wird durch ein oder mehrere Blattkonzepte der Hierarchie klassifiziert. Dazu kann auch Abb. 7.2.6.(2) als Beispiel betrachtet werden. Leider können in der Praxis können die Klassifikationsprozeduren unpräzise ein, weil möglicherweise nicht immer exakt feststellbar ist, in welches Konzept das Dokument oder die Anfrage gehört. Es kann passieren, daß ein Dokument oder eine Anfrage zu mehr als einem Nachfolger des Konzeptes paßt, aber nicht feststellbar ist, welche.

Beispielsweise kann ein Dokument aus dem Bereich Lyrik auch als Komödie klassifiziert werden, auch wenn nicht mehr entscheidbar ist, in welcher Untermenge (Sitcom, Slapstic, etc.) es gehört.

In P2P Systemen werden Dokumente von Knoten gehalten. Damit macht es Sinn die Knoten zu klassifizieren, und weniger die Dokumente. Semantisch verwandte Knoten werden dann als SON angeordnet. Auf diese Weise wird das SON mit dem Konzept der Klassifikationshierarchie in Verbindung gebracht. Ein SON mit dem Konzept x wird dann als $SON_x$ bezeichnet.

Da der Aufbau eines SONs nicht trivial ist, soll die notwendige Vorgehensweise kurz skizziert werden, da der Aufbau eines SONs direkt auf die Performanz des Netzwerkes einwirkt.

### 7.2.6.1 Aufbau eines SONs

Zuerst müssen alle potentiellen Klassifizierungshierarchien gesammelt und bewertet werden. Dann muß eine geeignete Hierarchie ausgewählt werden. Diese Hierarchie muß in einigen oder allen Knoten gespeichert werden, die dann verwendet werden sollen um die SONs zu definieren.

Wenn sich ein Knoten mit dem System verbinden will, muß das Netzwerk zuerst mit Requests für die Hierarchie auf Gnutella-Art geflutet werden. Dann wird der Dokumentenklassifizierer gestartet, der auf der Hierarchie über alle verfügbaren Dokumente basiert. Der Klassifizierer weist dem Knoten dann ein spezielles SON oder mehrere spezielle SONs zu. Der Knoten verbindet sich dann mit dem SON, indem ein Knoten gefunden werden muß, der dem gewünschten SON angehört. Das kann durch das Fluten des Netzwerkes oder durch den Gebrauch eines zentralen Verzeichnisdienstes erfolgen.

Stellt der Knoten eine Anfrage, wird diese Anfrage zunächst klassifiziert und erst dann an die geeigneten SONs geschickt. Die Knoten im SON werden dann aufgefunden wie beim Verbindungsaufbau. Ist die Anfrage im richtigen SON angekommen, können die Knoten innerhalb des SONs unter Nutzung geeigneter Weiterleitungsmechanismen Treffer finden.

Hier zeigt sich die Wichtigkeit einer intensiven Auseinandersetzung mit den Klassifikationshierarchien. Diese Klassifikation ist gleichermaßen wichtig für den Aufbau des Netzwerkes, aber auch für die Suche im Netzwerk. Je besser die Klassifizierung schon beim Aufbau des Systems durchdacht wurde, um so einfacher gestaltet sich die Suche. Eine schlechte Klassifizierung kann die Vorteile des Semantic Overlay Netzwerkes verringern oder sogar zunichte machen.

### 7.2.6.2. Klassifizierung

Gute Klassifizierungshierarchien zeichnen sich durch verschiedene Eigenschaften aus. Sie erzeugen Sammlungen mit Dokumenten, die zu einer möglichst kleinen Anzahl von Knoten gehören. Die Knoten haben ihre Dokumente in einer kleinen Anzahl von Sammlungen. Das erlaubt dann den Gebrauch von Klassifikationsalgorithmen, die leicht zu implementieren sind. Die Fehlerwahrscheinlichkeit sinkt mit der Verbesserung der Klassifizierung.

Die Klassifizierung von Dokumenten ähnelt der Klassifizierung von Anfragen. Allerdings können die Voraussetzungen unterschiedlich sein. Knoten werden mit relativ kleinen Zuwachsraten hinzugefügt. Die Auftrittshäufigkeit von Anfragen ist hingegen deutlich höher. Die Klassifizierung von Knoten erfolgt häufig in Bursts, da evtl. sehr viele Dokumente klassifiziert werden müssen wenn sich ein Knoten dem Netzwerk anschließen will. Anfragen kommen regelmäßiger , müssen aber nur je einmal die Klassifizierung durchlaufen.

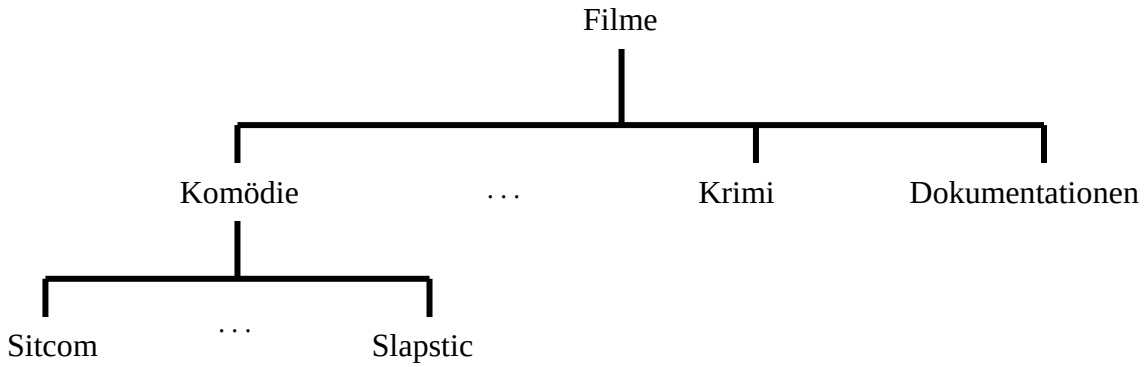
Klassifizierer für Dokumente können einen möglichst genauen Algorithmus verwenden, auch wenn das zeitintensiver ist. Dieser genaue Algorithmus kann große Mengen von Daten verarbeiten. Ein Klassifizierer für Anfragen darf hingegen weniger präzise sein, da bei Anfragen die Schnelligkeit der Antwort wichtiger ist als die Genauigkeit. Schließlich will der Nutzer nicht lange auf eine Antwort warten.

Eine Klassifikation kann sowohl automatisch als auch manuell vorgenommen werden. Im traditionellen SON schließt sich der Knoten häufig nur einem SON an. Das ist dann das SON, welches in der Relevanz prozentual am nächsten liegt.

Eine Weiterentwicklung dazu bietet das sogenannte Layered SON, das geschichtete SON. Hier wird bei der Klassifizierung des Knotens etwa festgelegt mit wieviel Prozent der betreffende Knoten relevanzmäßig zu den verschiedenen SONs paßt. Die so entstehende Liste wird sortiert, und der Knoten schließt sich den SONs an, deren Relevanz einen vorgegebenen Schwellenwert überschreitet.

## 7.2.6.3. Suche in Layered SONs

Bei der Suche in Layered SONs wird zuerst die Anfrage klassifiziert. Dann wird die Anfrage zu dem SON geschickt, welches mit dem Basiskonzept der Anfrageklassifikation übereinstimmt. Die Anfrage wird dann progressiv in höhere Ebenen der Hierarchie geschickt bis genügend Resultate gefunden wurden (siehe auch Abb. 7.2.6.3.(1)).




| Abb. 7.2.6.3.(1) | Darstellung einer Hierarchie |
|---|---|

Wenn beispielsweise in dem SON $_{Filme}$ gesucht werden soll, kann die Suche im $SON_{Slapstic}$ beginnen. (siehe auch Abb. 7.2.6.3.(1)) Gibt es in diesem SON nicht genügend Treffer, dann kann im SON $_{Komödie}$ weitergesucht werden. Wenn es hier auch nicht genügend Treffer gibt, kann im $SON_{Filme}$ weitergesucht werden.

Der Suchalgorithmus kann keine vollständige Suche garantieren. Wenn es zu Klassifikationsfehlern gekommen ist, können nicht alle Dokumente gefunden werden. Eine Lösung hierfür bietet eine umfassende Suche in allen Knoten des Netzwerkes. Eine exakte und standardisierte Klassifizierung kann die Fehlerwahrscheinlichkeit senken und damit die Vollständigkeit erhöhen. Ein solches Klassifizierungssystem wird im Patentwesen bereits verwendet, wo eine vollständige Suche essentiell ist.

# 8. Suche mittels Agenten

Bevor es möglich ist über eine Suche mittels Agenten zu sprechen, gibt es noch ein kleines Problem zu lösen. Was ist eigentlich ein Agent ?

Für die Begriffsdefinition eines Agenten gibt es verschiedene Meinungen. Festzuhalten sind an dieser Stelle vor allem die Gemeinsamkeiten dieser Definitionen.

Ein Softwareagent ist eine Rechenentität, die einen Auftrag oder mehrere Aufträge übernimmt. Arbeitet der Softwareagent nur in dem System, in dem er gestartet wurde, so wird er auch als „stationärer“ Agent bezeichnet.

Der mobile Agent hat zusätzliche Eigenschaften, die der stationäre Agent nicht hat. So ist der mobile Agent nicht darauf beschränkt nur in dem System zu arbeiten auf dem er gestartet wurde. Er kann stattdessen auch in andere Computer des Netzwerkes transferiert werden. Auch ist der mobile Agent autonom. Das bedeutet er kann selber entscheiden wo er hingeht, was er dort tut und wie lange er existiert. Er kann durch seine Umwelt, aber auch durch andere Agenten beeinflußt werden. Dennoch lösen die mobilen Agenten nicht ein vorher unlösbares Problem. Dennoch haben sie Vorteile gegenüber anderen Technologien. Mobile Agenten können allerdings Anwendungsgebiete vereinfachen, wie z.B. den E-Commerce, das verteilte Information Retrieval, Telekommunikationsnetzwerkdienste, etc.

Neben den Agenten gibt es auch die sogenannten Aglets. Diese unterscheiden sich dadurch von den Agenten, daß die Aglets bereits eine javabasierte Implementation eines Agenten sind. Aglets wurden ursprünglich von IBM in Japan entwickelt und wurden später open source und werden nun von der Aglets Community entwickelt und erhalten.

Die Programmiersprache Java scheint aus mehreren Gründen geeignet um Aglets zu entwickeln. Die große Popularität von Java macht die Entwicklung von Aglets nicht nur schneller, sondern auch billiger. Außerdem ist Java Code hochgradig portabel. Zusätzlich hat die Java Virtual Machine feingranulare und vor allem konfigurierbare Sicherheitsmechanismen. Eine eingebaute Unterstützung für Netzwerkprogrammierung ist auch vorhanden. (vgl. auch [DUNNE1])

# 8.1. Warum mobile Agenten benutzen ?

Mobile Agenten bieten viele Vorteile im Vergleich mit herkömmlichen Methoden. Sie reduzieren den Bedarf an Bandbreite. Peers, die ein verteiltes Protokoll benutzen, erstellen oft einen Kommunikationskanal von einem Peer zum anderen Peer. Über diesen Kanal interagieren sie dann. Mobile Agenten erlauben das Zusammenpacken dieser Interaktionen und den Versandt als diskrete Teile des Netzwerkverkehrs. Die Interaktion zwischen den Peers kann dann lokal stattfinden, da der mobile Agent bei seiner Ankunft auf dem Computer bereits alle Daten dabei hat. Er muß also nicht mehr über das Netzwerk mit anderen Rechnern kommunizieren.

Ohne Verwendung eines Agenten müssen alle Rohdaten durch das Netz reisen um verarbeitet zu werden, auch wenn nur eine kleine Untermenge dieser Daten benötigt werden. Hier werden also die Daten zu Verarbeitung gebracht.
Wird ein Agent verwendet, werden die Rohdaten nicht mehr bewegt, sondern nur die Verarbeitungsroutinen, so daß die Verarbeitung dann lokal auf den Daten durchgeführt werden kann. Der Netzverkehr wird so reduziert.

Mobile Agenten können sehr klein sein. Allerdings haben sie die Fähigkeit daß sie stark anwachsen können, je mehr Daten sie aufnehmen müssen.

Mobile Agenten sind asynchron. Wird er freigesetzt muß der Nutzer nicht auf seine Rückkehr warten. Er braucht noch nicht einmal mit dem Netz verbunden sein, während der mobile Agent im Netz unterwegs ist. Der mobile Agent wartet ab bis sich der Nutzer wieder mit dem Netz verbunden hat und kehrt erst dann wieder zurück.

Ein weiterer Vorteil der mobilen Agenten ist ihre Autonomität. Der mobile Agent lernt das Netzwerk kennen wenn er es durchschreitet. Damit kann er Peers besuchen, die ihm unbekannt waren, als der Agent gestartet wurde. Bei jedem Peer trifft der Agent Entscheidungen, die auf seiner Geschichte der bereits besuchten Peers und dem aktuellen Peer basieren.

Der Besuch eines Agenten bietet den Peers viele Vorteile. Der Agent hat in der Regel neue oder aktuellere Informationen über Ressourcen hat als der Peer. Aber auch der Agent profitiert von einem Besuch bei einem Peer. Der Agent lernt bei seinen Besuchen Informationen über neue oder aktualisierte Ressourcen. Wenn der mobile Agent keine neuen Informationen hat, dann kann er zerstört werden. Will ein Peer den Besuch eines mobilen Agenten akzeptieren, benötigt der Peer Ressourcen wie Speicher und Rechenzyklen. Wenn seine Ressourcen beschränkt sind, kann der Peer weitere Anfragen von Agenten ablehnen und erst dann wieder akzeptieren wenn die benötigten Ressourcen wieder zur Verfügung stehen.

Es ist auch möglich mobile Agenten zu klonen. Diese Klone werden dann in verschiedene Richtungen verschickt und können parallel arbeiten. Dann sind mehrere Agenten im Netzwerk aktiv, und die Suchanfrage im Netzwerk kann schneller beendet werden. Der einzelne mobile Agent verbringt somit weniger Zeit im Netzwerk.

Es ist auch zu erwähnen daß die Verwendung von mobilen Agenten zu einer größeren Fehlertoleranz führt. Sogar wenn mobile Agenten zerstört werden, können die überlebenden Agenten weiter positive Resultate einfahren. Auch über die zerstörten Agenten ist zu sagen daß sie den Peers, die sie besucht haben, bis zu ihrer Zerstörung gedient haben.

Auch können die mobilen Agenten mit den Eigenschaften von anderen Peer to Peer Systemen kombiniert werden. Auf diese Weise kann möglicherweise eine bessere Komplettlösung für das Problem gefunden werden.

## 8.2. Der Peer Discovery Algorithmus

Im Hauptalgorithmus wird geklärt wie sich der mobile Agent bei der Suche nach anderen Peers verhält. Der Algorithmus kann nicht die ganze Netzwerkerforschung leisten und versucht dies auch nicht, da diese Aufgabe in großen Netzwerken nicht immer skalierbar ist. Deswegen versucht der Algorithmus einen Kompromiß zu finden zwischen einer detaillierten und einer effizienten und praktischen Netzwerkerforschung.

Nach der Beendigung der Arbeit des mobilen Agenten hat der Peer, der diesen Agenten beauftragt hat, eine Sicht auf den Teil des Netzwerkes, den er nunmehr kennt. Diese Sicht ist, graphentheoretisch gesehen, ein ungerichteter Graph. Daher ist es nicht unwahrscheinlich, daß in der Graphentheorie ein besserer Algorithmus gefunden werden kann (vgl. auch [DUNNE1])

1. Schritt: Ein Peer, der am Netzwerk teilnehmen möchte, erzeugt einen mobilen Agenten. Dieser Peer ist der „Creator Peer“ für diesen mobilen Agenten und allen seinen zukünftigen Klonen. Der mobile Agent übernimmt Informationen vom Creator Peer. Er vermerkt sich dessen Adresse als seine Heimatadresse. Für den mobilen Agenten wird außerdem eine sogenannte Journey Time gesetzt, die festlegt wie viele Peers der Agent maximal besuchen soll. Nach Ablauf dieser Zeit muß der Agent zu seinem Creator Peer zurückkehren. Als weitere Größe wird der Branching Factor festgelegt. Hier wird angegeben wie oft ein mobiler Agent in einem Peer geklont werden darf. Diese beiden Parameter kontrollieren die Tiefe und Breite der Suche und damit auch die maximale Anzahl der zu besuchenden Peers.

2. Schritt: Der mobile Agent erhält zunächst die Adressen einiger der am Netzwerk teilnehmenden Peers. Diese Anzahl der Adressen sollte kleiner sein als der Wert des Branching Factors. Um die besten Resultate zu erreichen sollten die Adressen aus verschiedenen Quellen kommen. Das erhöht die Chancen daß die Peers aus unterschiedlichen Unternetzen stammen und sich vorher noch nicht kannten.

3. Schritt: Der mobile Agent klont sich mehrfach, damit er einen Klon an jeden Peer aus der im 2. Schritt erhaltenen Adressenliste schicken kann.

4. Schritt: Der mobile Agent kommt beim Peer an und verringert seine Journey Time. Er aktualisiert den Peer über seinen Creator Peer und über die anderen Peers, denen er auf seiner Reise schon begegnet ist. Wird ein Peer innerhalb einer festgelegten Zeitspanne von mehreren Agenten des gleichen Creator Peer besucht, dann wird der zweite mobile Agent zerstört. Das erfolgt auch mit evtl. folgenden Agenten. Das kann zu einer Verringerung der Informationen führen, die über das Netzwerk gesammelt werden. Allerdings werden durch dieses Verhalten Zyklen auf den Reisen der Agenten vermieden.

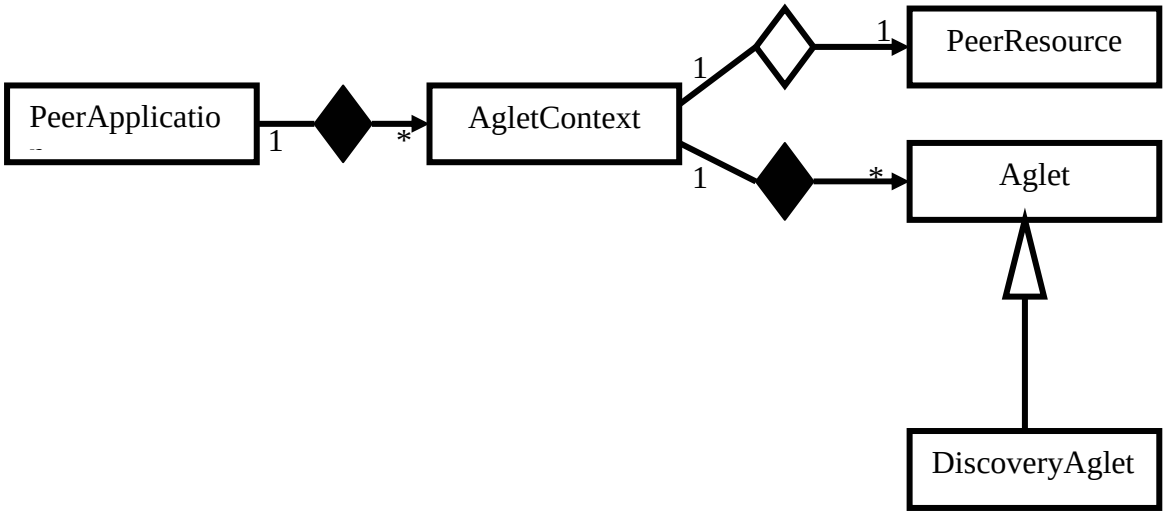
5. Schritt: Der mobile Agent kehrt zu seinem Creator Peer zurück wenn seine Journey Time abgelaufen ist. Er gibt seine gesammelten Informationen dann an seinen

Creator Peer weiter. Er wird ansonsten so häufig geklont daß er an jeden bekannten Peer geschickt werden kann.

6. Schritt: Fortsetzung in 4. Schritt

## 8.2.1. Implementation und Architektur

Die Implementation erfolgt mit dem Aglet Software Development Kit (ASDK)




| Abb. 8.2.1.(1) | Klassendiagramm der Architektur (aus [DUNNE1]) |
|---|---|

Im Klassendiagramm (Abb. 8.2.1.(1)) finden sich Objekte, die einer Erklärung bedürfen. Da ist zunächst das DiscoveryAglet. Das ist ein mobiler Agent, der durch das Netzwerk reist. Es erweitert die Aglet Klasse, die vom ASDK bereitgestellt wird. Das DiscoveryAglet implementiert den in Kapitel 8.2. vorgestellten Algorithmus. Die Methoden hierfür sind nicht überschreibbar. Das ASDK implementiert die Mobilität seiner mobilen Agenten durch Serialisierung. Das Aglet erhält seinen Zustand durch seine Instanzvariablen, die vom Typ Boolean, String oder Vector sind um das DiscoveryAglet so klein wie möglich zu halten. Die Klassen sind Teil der Java Virtual Machine und daher müssen ihre Bytecodes nicht mit dem DiscoveryAglet übertragen werden. Hier wird Bandbreite gespart.

Auch der AgletContext ist Teil des ASDK. Er wird benötigt um Aglets erhalten und verwalten zu können. Die Aglets können nur mit Objekten kommunizieren, die auch im AgletContext liegen. Der AgletContext stellt Methoden zur Verfügung, die es möglich machen andere Objekte darin zu plazieren und aus ihm wiederzugewinnen.

Die PeerResource ist ein Objekt, die Informationen über alle Ressourcen im aktuellen Peer ernthält. Sie wird nach ihrer Initialisierung im AgletContext plaziert. Die mobilen Agenten können so im AgletContext darauf zugreifen. Die PeerResource enthält Methoden, die nicht überschrieben werden können, um die notwendige Funktionalität einer Ressource zur Verfügung zu stellen.

Die PeerApplication ist eine Java Anwendung. Sie kapselt die vorher erwähnten Software Komponenten ein.

Das DiscoveryAglet stellt die benötigte Funktionalität zur Verfügung.

## 8.3. Wünschenswerte Charakteristika von Peers

Wenn agentenbasierte Lösungen verwendet werden sollen, ist es auch nötig sich mit den Peers zu beschäftigen. Peers können sehr unterschiedliche Eigenschaften haben und es ist durchaus wünschenswert zu untersuchen welche Eigenschaften die Peers in einer mit Agenten versetzten Umwelt haben sollten, damit eine Zusammenarbeit optimiert werden kann.

Der Peer sollte autonom entscheiden können, wann er sich mit dem Netzwerk verbindet, oder wann er es verläßt. Er entscheidet weiterhin ob und wann eine Dienstanfrage befriedigt werden soll und ob er eine Bezahlung für seinen Dienst verlangt.

Auch sollte der Peer soziale Fähigkeiten besitzen. Die Konzepte von Vertrauen und Reputation sollten umgesetzt werden, wie auch Verhaltensweisen wie Kooperation oder Wettbewerb bei gegebenen sozialen Aufgaben.

Anpassungsfähigkeit und Beweglichkeit erlauben dem Peer einen Lernalgorithmus zu implementieren, wie z.B. für das Reisen durch das Netz oder das Entdecken von Verbindungen oder Links und Ressourcen. Damit kann die globale Performanz erhöht werden.

Neben wünschenswerten Eigenschaften gibt es aber auch Einschränkungen, die in diesem Rahmen auch betrachtet werden sollen. Diese Eigenschaften sind zwar nicht wünschenswert, aber für das Netzwerk und seine Lösungen von Wichtigkeit sind.

Die Ausdrucksstärke der Nachrichten sind eingeschränkt, da die Nachrichtensprachen einfach sind und nur definierte Arten von Daten und Objekten kommunizieren. Informationen mit komplexer Semantik sind nicht in Nachrichten auszudrücken.

Auch sind die Datenmodelle einfach und basieren auf dem Modell von Dateien und Verzeichnissen.

Die Peer-Umgebung zieht Heterogenitäten, Inkonsistenzen und andere ähnliche Probleme leider nicht immer in Betracht. In verteilten Umgebungen treten diese Phänomene häufig auf.

## 8.4. Agentenbasierte Peers

Die Knoten in einem Peer to Peer System stellen Dienste oder Referenzen auf Dienste zur Verfügung. Wenn ein Knoten eine Anfrage stellt, weiß er nicht ob der Dienst bereitgestellt wird oder bereitgestellt werden kann. Die Konzepte von Vertrauen und Reputation sind für Agenten bereits bekannt. Sie sind auch gut geeignet für die Definition des Sozialverhaltens von Peers. Das Konzept des Vertrauens steht im Gegensatz zum Konzept der strengen Sicherheit, das oft durch Passworte oder Kryptographie durchgesetzt wird. Die strenge Sicherheit erlaubt es die Identität eines Agenten zu kennen, stellt aber nicht sicher ob der Agent auch erwartungsgemäß reagiert oder einen angemessenen Dienst zur Verfügung stellt.

Bei der Betrachtung von Vertrauen in Multi-Agenten-Systemen müssen verschiedene Gesichtspunkte betrachtet werden. Es ist wichtig zu wissen wie sich ein Agent entscheiden kann wann er seine Reputation offenbaren will. Es stellt sich auch die Frage wie ein Agent

vertrauenswürdige Agenten finden kann, wenn vorher keine Kontakte damit bestanden. Die Beschleunigung der Weiterleitung von Informationen durch das soziale Netzwerk ist außerdem interessant.

Ein möglicher Mechanismus bedient sich dafür des Konzeptes von Belohnung und Strafe. Auf eine Anfrage antwortet der Agent nur dann, wenn er sich sicher ist daß er die Anfrage beantworten kann. Die Antwort wird auf Basis von direkter Erfahrung, Bewertungen von Nachbarn und dem Vertrauen in die Nachbarn bewertet. In diesem Sinne soll der Ansatz sozial sein und Vertrauensinformationen können weiterreisen. Für die Kooperationen mit anderen Agenten werden positive und negative Bewertungen erstellt. Schlechtes Verhalten soll vermieden werden dadurch, daß eine gute Reputation nur langsam aufgebaut aber sehr schnell wieder zerstört werden kann. (vgl. auch [MORO1])

## 8.4.1. Die Hauptkomponenten eines agentenbasierten Peers

Die Hauptkomponenten in der Architektur eines agentenbasierten Peers sind der Assistant, der Contract Net Agent, die Yellow Pages und der Search Agent. Diese Komponenten sollen im Folgenden kurz vorgestellt werden.

Der Assistant arbeitet mit Nutzerinteressen und behält ein dynamisches Profil. Die Nutzeranfragen werden in eine echte Kommunikationssprache übersetzt und das Ergebnis dann dem Contract Net Agent übergeben.

Der Contract Net Agent arbeitet mit Anfragen. Er überprüft ob sie lokal oder im Netzwerk beantwortet werden können. Ist die Beantwortung einer Anfrage im Netzwerk notwendig, werden die Yellow Pages befragt. Wenn hier nichts gefunden wurde, schickt er eine Anfrage nach einem gegebenen Routingalgorithmus ab.

Die Yellow Pages überwachen die Ressourcen, welche verwendet werden können um Anfragen zu beantworten. Dies schließt auch die Beschreibung der generellen Interessen der Knoten ein.

Der Search Agent wird vom Contract Net Agent gestartet. Er führt die Anfrage dann tatsächlich durch. Er „lernt“ um seine Suchleistung zu verbessern. Außerdem kann er Interessenbeschreibungen in den Yellow Pages aktualisieren.

Die Kommunikationssprachen der Agenten verwenden Nachrichten um Sprache von einem Agenten zum anderen Agenten zu bringen. Jede Nachricht enthält einen Absender, einen Empfänger und einen Inhalt. Wenn sich zwei Agenten bestehen sollen ist eine allgemeine Beschreibung der Welt als auch ein Interaktionsprotokoll nötig. Dieses Protokoll legt fest welche Kommunikationsmuster legal sind.

## 8.4.2. Die Kommunikationssprache für Agenten

Die Kommunikationssprache für Agenten (ACL) basiert auf der Voraussetzung daß kommunikationswillige Agenten sich eine gemeinsame Basis teilen müssen. Das soll sicherstellen daß die Agenten den jeweils gleichen Symbolen auch die gleiche Bedeutung in der Nachricht zuweisen. Nur so ist eine Kommunikation verständlich und möglich.

Es gibt einige Aktionen, die besonders häufig vorkommen und daher kurz erwähnt werden sollten:

JOIN: Hier bittet ein Peer einen anderen Peer um Erlaubnis sich dem Netzwerk anschließen zu dürfen.

DISJOIN: Hier bittet ein Peer einen anderen Peer um Erlaubnis sich vom Netzwerk zu trennen. Das ist gegenüber den anderen Teilnehmern ein respektvolleres Verhalten als wenn sich der Peer einfach nur vom Netzwerk trennt. Eine Bitte um Erlaubnis zur Trennung vom Netzwerk trägt zu einer besseren Reputation des Teilnehmers bei, weil das Peer to Peer Netzwerk mit dem Wissen einer bevorstehenden Trennung seine Struktur neu organisieren kann.

LOOK UP: Der Peer stellt die Anfrage um etwas mittels Search Agent zu finden.

UPDATE: Ein Peer informiert einen anderen Peer über seine Änderungen in seinen lokalen Daten.

REQUEST: Ein Peer stellt einem anderen Peer eine Frage, z.B. um einen Dienst zu benutzen.

### 8.4.3. Dienste intelligenter Agenten

Peer to Peer ist eine verteilte Rechentechnologie. Diese kann einen großen Einfluß auf die Suchfähigkeiten im Internet haben. Intelligente Agenten sind Software Werkzeuge. Sie werden auch als „bots“ bezeichnet, was eine Kurzform von Robots ist. Diese Software Werkzeuge führen eine Suche durch um Informationen zu finden. Es gibt zwei Anwendungen, wo bots ein großes Potential haben. Das sind laut [MORO1] insbesondere healthcare knowledge discovery und Data-Mining (vgl. auch [MORO1]). Aber auch andere Anwendungen können von dieser Technologie profitieren.

Beim healthcare knowledge discovery wird eine Peer to Peer Technologie mit den neuesten Erkenntnissen der künstlichen Intelligenz kombiniert. Basierend auf Klassifikation und Indizierung soll eine komplette Gesundheitsvorsorge geliefert werden. Das wird vor allem von professionellen Mitarbeitern der Gesundheitsvorsorge gewünscht.

Das data mining benötigt oft eine Reihe von Suchschritten. Die bots können hier Arbeit sparen solange sie sich auf der Suche befinden. Bots treffen Entscheidungen. Diese basieren auf vergangene Erfahrungen, die dann für das data mining zu einem wichtigen Werkzeug wurden um komplexe Suchen zu perfektionieren.
(vgl. auch [MORO1])

## 8.5. Das SEWASIE Projekt

SEWASIE ist die Abkürzung von Semantic Webs und AgentS in Integrated Economies. Es handelt sich hier um ein Forschungsprojekt, das von der EU gegründet wurde. (http://www.sewasie.org)

Das Ziel von SEWASIE ist es eine fortschrittliche Suchmaschine zu entwickeln und zu implementieren. Es soll ein intelligenter Zugriff auf heterogene Datenquellen im Internet über semantische Beziehungen ermöglicht werden. Als Basis soll eine strukturierte, sichere, webbasierte Kommunikation dienen.

Der SEWASIE Client besitzt einen Suchclienten mit einem leicht bedienbaren Suchinterface. Dieses ist in der Lage die gewünschte Information aus dem Internet zu gewinnen und es in einem benutzerfreundlichen Format darzustellen. Der SEWASIE Prototyp soll dann einen Suchclienten und einen Indizierungsserver bereitstellen.
Es soll eine agentenbasierte, sichere, skalierbare und verteilte Systemarchitektur für eine semantische Suche entwickelt werden.

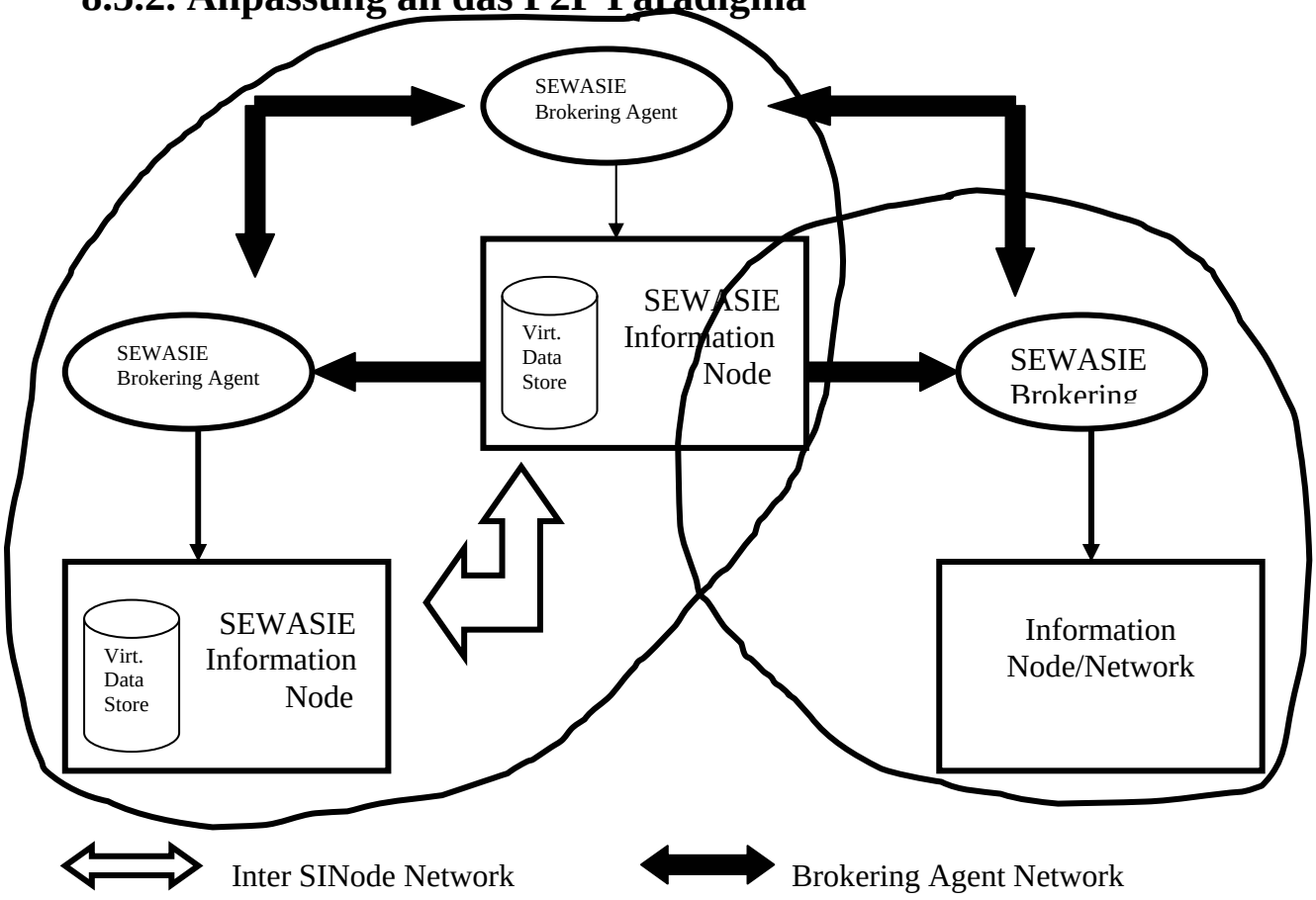
SEWASIE soll den europäischen Firmen die Möglichkeit bieten im globalen Marktwettbewerb zu bestehen und strategische Allianzen auf europäischer Ebene zu bilden. Es soll ein fortschrittliches Information Retrieval und ein fortschrittlicher Kommunikationsdienst, basierend auf der semantischen Webtechnologie, angeboten werden.

## 8.5.1. SEWASIE in einer P2P Architektur

Im Allgemeinen besitzt ein P2P System kein zentralisiertes Schema und keine zentrale Verwaltung. Die SEWASIE Architektur hingegen verläßt sich auf zwei zentralisierte Aspekte: den brokering agent, der das Wissen im Gesamtnetzwerk hält, und das global schema oder das data repository des Netzwerkes. (vgl. auch [GUERRA1])

Die SINodes sind passive Elemente, deren Daten und Metadaten von den Suchagenten und dem brokering agent extrahiert werden. Es ist also offensichtlich, daß die SEWASIE Architektur verändert werden muß, um dem P2P Paradigma zu entsprechen.

## 8.5.2. Anpassung an das P2P Paradigma




| Abb. 8.5.2.(1) | Die zwei alternativen P2P Netzwerke (aus [GUERRA1]) |
|---|---|

In Abb. 8.5.2.(1) werden die zwei alternativen P2P Netzwerken dargestellt.

Brokering Agents Network:
Es können mehr Brokering Agents zugeteilt werden, einen zu jedem SINode. Dieser enthält dann sowohl das SINode Wissen als auch das Koordinationswissen. Im Brokerimg Agents Network kommuniziert jeder Brokering Agent mit anderen Peers um Informationen über andere Information Nodes zu erhalten.

Inter SINodes Network:
Ein SINode liefert anderen SINodes das Wissen über beteiligte Informationsquellen. Die Koordinationsformeln können spezifiziert werden. Sie erklären die nötigen Beziehungen der Daten in einem Peer mit den Daten in einem bekannten Peer.

Das Brokering Agents P2P Netzwerk erzeugt ein verteiltes Wissen über die beteiligten Informationsquellen und können eine Unterstützung für das Generieren von Koordinationsformeln bieten.

Das Brokering Agents Netzwerk unterstützt die Bildung eines Anfrageplans um festzustellen welche SINodes befragt werden müssen. Das Netzwerk kann Interessengruppen aus Knoten mit einem ähnlichen Inhalt bilden. Außerdem kann das Netzwerk bei der Anfrageoptimierung helfen. Das geschieht durch das Bereitstellen von Informationen für den query agent über das „Data Placement“. Ein Peer weiß wie die Daten über die Knoten verteilt sind und der Anfrageplan kann die existierenden Ressourcen und Bandbreitenbeschränkungen berücksichtigen.

Das SINodes Netzwerk ist ein alternativer Ansatz, in dem das Peer to Peer Paradigma direkt von den SINodes unterstützt wird. In diesem Fall gibt es weiterhin einen einzelnen Brokering Agent, der das Wissen über die Netzwerktopologie enthält. Außerdem wird eine P2P Schicht in jedem SINode, der folgende Funktionalitäten enthält:

Die P2P Schicht benötigt ein Protokoll um Bekanntschaften zu machen. Sie kann eine halbautomatische Unterstützung bieten um Koordinationsformeln zu generieren. Für die Anfragebarbeitung können multi database systems benutzt werden. Hierfür werden aber auch Richtlinien um Unterabfragen durch Ketten von P2P Verbindungen zu leiten.
Außerdem sollte die P2P Schicht eines SINodes in der Lage sein, beispielsweise durch Verzeichnisdienste, seine Inhalte anzuzeigen. Diese Informationen sind nützlich um Bekanntschaften zu machen und sich mit Knoten mit ähnlichen Inhalten zusammenzuschließen. (vgl. auch [GUERRA1])

# 8.6. Ein agentenbasiertes Suchsystem

Wenn in Peer-to-Peer Netzwerken eine Suche durchzuführen ist, dann ist zu entscheiden ob die Suche synchron oder asynchron durchgeführt werden soll.

## 8.6.1. Synchrone vs. Asynchrone Suche

Bei der synchronen Suche werden die Suchergebnisse auf dem gleichen Weg zurückgeschickt, den auch schon die Anfrage genommen hat. Der Peer muß warten bis seine Nachbarn ihre Resultate zurückgegeben haben. Erst dann kann er seine eigenen Resultate hinzufügen und das Ergebnis weiterleiten bis die Anfrageantwort beim Peer ankommen kann,

der die Anfrage ursprünglich abgeschickt hat. Die Suche ist genau dann beendet wenn die Suchergebnisse beim Peer ankommen, der die Anfrage gestellt hat.
Außerdem wissen die Peers bei der synchronen Suche nicht woher die Anfrage oder die Anfrageergebnisse kommen.

Bei der asynchronen Suche kann jeder beteiligte Peer seine Resultate direkt zum Ausgangspunkt der Anfrage zurückschicken, sobald sie verfügbar sind. Da diese Resultate beim anfragenden Peer nicht in einem Zuge, sondern in Stücken ankommen, sind Mechanismen notwendig, damit das System das Ende einer Suche anzeigt. Die Anonymität des Anfragestellers geht allerdings verloren, da jeder beteiligte Peer wissen muß von wo die Anfrage losgeschickt wurde, damit er die Antwort zurückgeschickt werden kann.

## 8.6.2. Die verteilte Beendigung der Suche

Bei der asynchronen Suche wird ein Suchergebnis zum Anfrageknoten zurückgeschickt, sobald ein Peer ein Suchergebnis findet. Aus der Sicht des Anfrageknotens sind die Resultate eine Stromgröße. Das ermöglicht dem Nutzer einen schnellen und frühen Zugriff auf Informationen. Eine weitere Verarbeitung in Verbindung mit der laufenden Suche wird ermöglicht.

Wichtig ist auch eine Mitteilung an den Nutzer, wenn alle Resultate produziert wurden. Nur dann erfährt der Nutzer daß die Suche beendet ist und er nicht länger auf weitere Resultate warten muß. Um das zu erreichen ist ein zusätzlicher Mechanismus in der asynchronen Suche nötig, der in der synchronen Suche nicht benötigt wird. Dieser Mechanismus ist der termination detector. Er benachrichtigt den Anfrageknoten wenn

- es keinen weiteren Peer gibt, der die Anfrage bearbeiten kann
- es keine Botschaft mehr gibt, die unterwegs ist und die Anfrage enthält
- es keine Botschaft mehr gibt, die unterwegs ist und Resultate enthält.

Weil es nicht wünschenswert ist daß ein Peer eine Nachricht im Kreis herumschickt sollte jede Nachricht einen Mechanismus enthalten, der einem Peer verrät daß er eine Nachricht schon einmal erhalten hat.

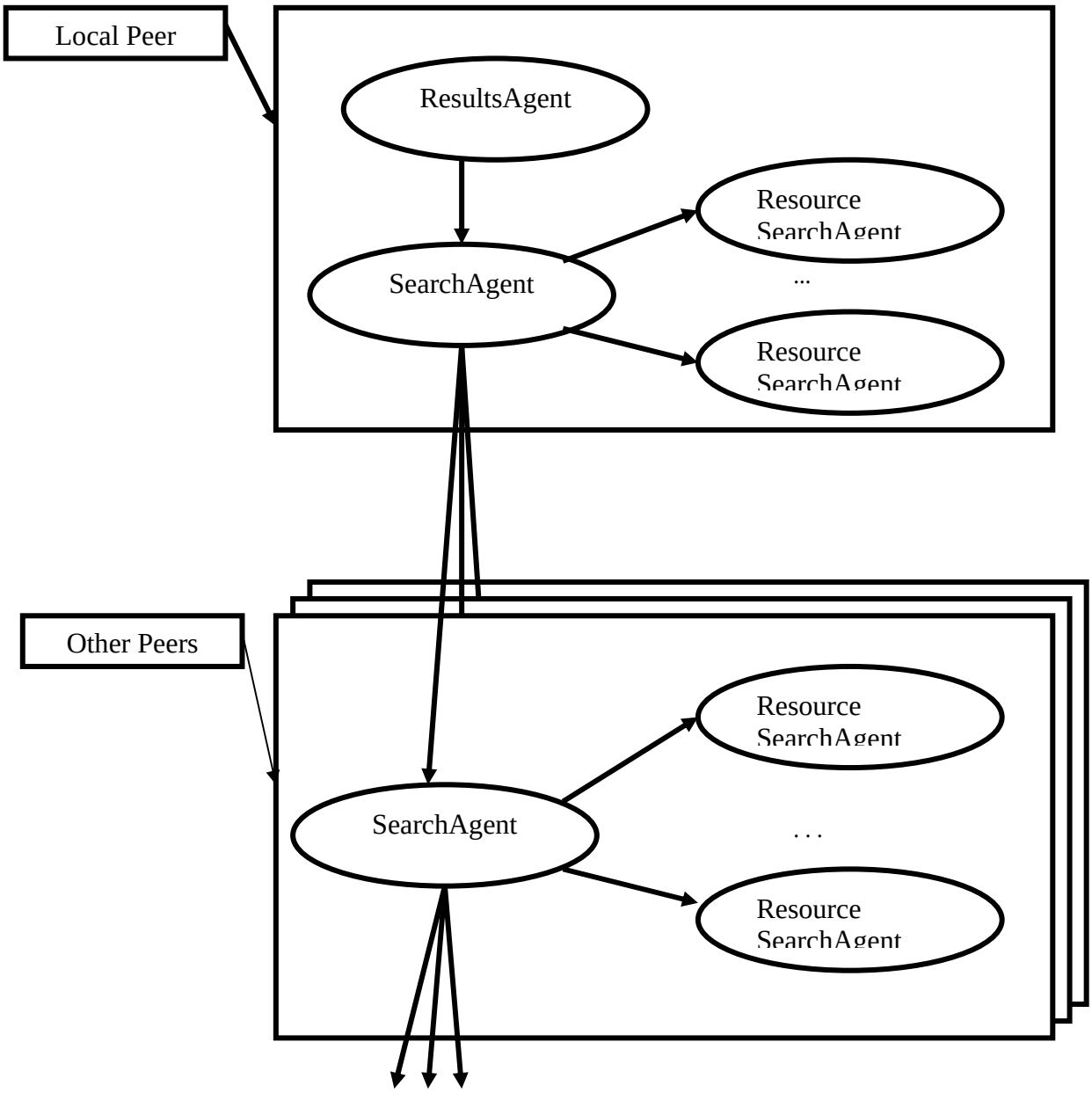
## 8.6.3. Deskriptoren und Anfragesprache

Die Ressourcen müssen derart beschrieben werden, daß ein Agent der eine Suche durchführt sie befragen kann. Hierzu werden für jeden Aspekt der Ressource Beschreibungsmöglichkeiten gefunden, die dann gruppiert und mit den Ressourcen „verbunden“ werden können.

Die Beschreibungen der Ressourcen müssen erweiterbar bleiben, damit die Nutzer in der Lage sind, ihre eigenen Beschreibungen zu erstellen. Das System muß neue Beschreibungen unterstützen können ohne daß Änderungen in den Suchprozeduren notwendig werden.

Die Beschreibung einer Ressource ist nur ein Aspekt um Ressourcen wiederfinden zu können. Die Suchsprache muß aber einfach genug sein um für jeden Nutzer benutzbar zu sein. Eine graphische Schnittstelle ist besonders für Nutzer wünschenswert, die sich mit ihrem Computer nicht besonders gut auskennen.

## 8.6.3. Architektur




| Abb. 8.6.3.(1) | Allgemeine Architektur (aus [SMITHSON1]) |
|---|---|

Die Agenten für das System fallen in zwei Kategorien, diejenigen, die Suchanfragen schicken und erhalten und diejenigen, die sie bedienen.

Ursprünglich gibt es einen Agenten, der eine Suchanfrage abschickt. Er ist in Abb. 8.6.3.(1) als ResultsAgent bezeichnet. Er ist als Endpunkt für die zurückerhaltenen Ergebnisse definiert, entweder als Array von Resultaten oder als Resultat von den Peers, die Resourcen besitzen.

Ein Suchagent erhält Anfragen und leitet sie weiter an die Peers die er kennt. Gleichzeitig führt er die Suche auf allen Agenten für jede der Ressourcen durch, die der Peer bereit ist zu teilen durch. Sie werden hier als SearchAgent bezeichnet und können sowohl mit anderen

Peers im Netzwerk kommunizieren als auch mit anderen Agenten, die im lokalen Peer ablaufen.

Die anderen Agenten, die im System gefunden werden können, werden als ResourceSearchAgent bezeichnet. Ihre Aufgabe ist es die Anfragen mit ihnen bekannten Ressourcen in Verbindung zu setzen. Dabei spielt es keine Rolle ob es sich um eine Menge von Dateien oder um andere dem Nutzer zur Verfügung stehenden Ressourcen handelt. Die Ergebnisse werden dem SearchAgent zurückgegeben. Das geschieht im synchronen Fall zum gleichen Peer und im asynchronen Fall zurück zum Anfrageursprung.

## 8.6.4. Spezifikationen der Anfrage

Sowohl bei der synchronen als auch bei der asynchronen Suche wird ein Time to Live Argument benötigt und auch einen weiteren eindeutigen Identifier, dessen Rolle später beschrieben werden soll.

Die Anfrage wird in einer Suchsprache ausgedrückt. Es wird eine Menge von Deskriptorenmuster und Einschränkungen, die die Ressourcendeskriptoren einhalten müssen definiert. Anfragen arbeiten auf Deskriptoren, die zur Laufzeit definiert werden, vorausgesetzt sie drücken Einschränkungen über eine Unterklasse von Deskriptoren aus.

Das Time to Live Argument ist als Integer definiert und kontrolliert die Tiefe der Suche. Eine Suche beginnt stets mit einem gegebenen Time to Live Wert. Wenn ein Peer die Anfrage weiterleitet wird der Wert jeweils dekrementiert. Erhält ein Peer eine Anfrage mit dem Time to Live Wert Null, so leitet er die Anfrage nicht mehr weiter. In der synchronen Suche ist dies der Zeitpunkt an dem die Resultate wieder auf der Anfrageroute zurückgeschickt werden.

Der oben erwähnte eindeutige Identifier soll Zyklen vermeiden. Dieser Identifier muß für alle Peers im Netzwerk eindeutig sein. Für jede eingehende Anfrage entscheidet der SearchAgent ob er die Anfrage schon einmal gesehen hat. Wenn die Anfrage noch nicht bearbeitet wurde, wird der Identifier gespeichert und die Anfrage bearbeitet.

Für eine asynchrone Suche werden zwei weitere Parameter benötigt. Der Ursprung der Anfrage muß definiert werden und der Ort an den die Resultate zurückgeschickt werden sollen. Dieser Parameter identifiziert des Agenten, der die Anfrage abgeschickt hat. Der zweite zusätzliche Parameter wird benutzt um die Beendigung der asynchronen Suche zu erkennen.

Die Resultate der Suche enthalten eine Menge von Deskriptoren von jeder Ressource und den Ort wo die Ressource zu finden ist als eine URL. Diese URL kann dann benutzt werden um auf die Ressource zuzugreifen.

Die Knoten kommunizieren nur durch den Austausch von Nachrichten. Im asynchronen Fall wird eine Berechnung als beendet angesehen, wenn jeder Knoten seine lokalen Berechnungen abgeschlossen hat und keine Nachricht mehr „unterwegs“ ist. Eine termination detection ist die Fähigkeit festzustellen daß das System seine Berechnungen beendet hat.

Für eine asynchrone Suche erschafft der Ursprungsknoten einen Token. Dieser ist ein neu geschaffenes Objekt, dessen Referenzen gezählt werden. Auf dem Ursprungsknoten wird eine Methode mit dem Token assoziiert, die getriggert wird wenn der Referenzzähler des Tokens Null erreicht hat. Initiiert der Anfrageknoten eine Anfrage, so gibt er den Token als Teil der

Anfrage mit. Jeder Host, der die Anfrage weiterleitet, muß den gleichen Token mit einfügen. Beendet ein Peer seine Suche lokal, muß er seine Referenz zum Token lösen. Das dekrementiert den Zähler, der mit dem Ursprungsknoten assoziiert wird. Hält kein anderer Peer mehr eine Referenz zum Token, und ist keine Anfrage mit dem Token unterwegs, erreicht der Referenzzähler des Tokens den Wert Null. Für das Zählen der Referenzen können verschiedene Algorithmen verwendet werden.

## 8.6.5. Implementation der Agenten

Der SearchAgent soll Anfragen von anderen Peers erhalten, sie lokal verarbeiten und sie an andere Peers, die er kennt, weiterleiten.

Eine synchrone Suche wird durch den query_ref implementiert. Dieser gibt ein Array von Deskriptoren zurück, die zur Suche passen.

Die asynchrone Suche wird durch request unterstützt. Dieser gibt keine Resultate zurück.

Ein Peer behält eine Sammlung von Proxies zu Agenten, die auf dem lokalen System laufen und für die anderen Peers, die der Agent kennt. Kommt eine Suchanfrage an, so startet der Agent zwei Threads um die Anfrage an die Agenten in beiden Sammlungen weiterzuleiten.

In der synchronen Suche blockiert der Agent bis beide Threads fertig sind. Zu dieser Zeit werden dann die Resultate an die Suchkette zurückgegeben.

In der asynchronen Suche werden die beiden Threads gestartet. Der MainAgent hat in der Ausführung keine weitere Aufgabe zu erfüllen. Der SearchAgent erlaubt auch daß andere ResourceAgents dynamisch zum System zugefügt oder aus dem System entfernt werden. Das wird durch einen Mechanismus der Registration erledigt.

Der FileSearchAgent ist ein Beispiel für einen Agenten, der Dateiressourcen verwaltet. Er akzeptiert eingehende Suchanfragen und schickt die Informationen zum Anfrageknoten entlang der spezifizierten Route zurück. Der Agent hält eine persistente Datenbank, die Dateinamen mit Ressourcenbeschreibungen verbindet. Ein Datenbankeintrag paßt nur zur Anfrage, wenn die Beschreibungen, die im Eintrag erhalten sind jede Konjunktionen in der Anfrage erfüllt.

Das zurückgegebene Resultat ist ein FileNameDescriptor, der den Dateinamen und die URL enthält, von der aus auf die Ressource zugegriffen werden kann, z.B. unter Benutzung der Remote Method Invocation RMI oder im Falle des Downloads unter Verwendung von HTTP oder FTP. Das Rahmenwerk der Agenten wird nur benutzt um Ressourcen zu finden. Zum eigentlichen Zugriff auf die Ressource werden spezialisiertere Protokolle verwendet.

Zum Schluß werden die Resultate an den ResultsAgent übergeben. Dieser behandelt sie auf die vom Nutzer festgelegte Weise. Er benutzt den ResultsProcessor, der folgende drei Methoden implementiert:

```
void add(Predicate[] results)
void add(Predicate result)
void alert()
```

Die add-Methoden werden aufgerufen, wenn die Resultate dem ResultsAgent übergeben werden. Die alert-Methode wird aufgerufen, wenn die asynchrone Suche beendet wird.

Die erste add-Methode schiebt die Resultate in den Standard Output Stream. Nachdem ein Fenster geöffnet wurde werden die Resultate in das Fenster geschrieben. Ein alert-Fenster wird erzeugt, wenn die asynchrone Suche beendet wird.

Wenn die Resultate in das Fenster gedruckt werden, kann der Nutzer sie durch einen Doppelklick auswählen. Mit der Auswahl kann eine Aktion programmiert werden. Wird ein Dateiresultat ausgewählt, wird dem Nutzer eine Dialogbox präsentiert um einen Ort zur Speicherung der Datei auszuwählen. Dann wird die Datei vom Peer Computer zum Nutzer Computer übertragen. (vgl. auch [SMITHSON1])

## 8.7. Fazit

Konventionelle Suchstrategien in Peer-to-Peer Netzen sind leider nicht besonders zufrieden stellend. Deshalb gab es die Notwendigkeit diese Suchstrategien maßgeblich zu verbessern. Es gab auch im konventionellen Bereich weitreichende Fortschritte (vgl. auch Kapitel 7), aber die gesetzten Erwartungen wurden nicht immer erfüllt. Daher stellte sich die Frage sich anderen Technologien als den konventionellen Möglichkeiten zuzuwenden.

Die Verwendung von mobilen Agenten wurde in Betracht gezogen. Sie sollten die Suche effizienter gestalten können als die konventionellen Methoden. Die agentenbasierten Systeme unterscheiden sich daher auch von den anderen Systemen.

In den konventionellen Systemen wurde eine Suchanfrage durch ein Netz geschickt. Jeder beteiligte Peer leitete die Suchanfrage weiter und lieferte seine Resultate ab, soweit er welche liefern konnte. Das so entstandene Kommunikationsbedürfnis der Teilnehmer stieg rasch an und belastete das Netz. Ständig waren Verbindungen zu anderen Peers für die verschiedensten Aufgaben nötig.

In einem agentenbasierten System trägt der Agent alle benötigten Verarbeitungsroutinen mit sich herum. Damit ist er relativ klein, im Gegensatz zu den Botschaften in konventionellen Systemen, wo große Mengen an Rohdaten verschickt werden müssen, um damm Berechnungen auf einer kleinen Untermenge dieser Daten durchzuführen. Das macht eine großangelegte Kommunikation mit anderen Teilnehmern unnötig. Der Agent reist von einem Peer zum nächsten, und die Berechnungen zur Erfüllung seiner Aufgabe werden jeweils lokal durchgeführt und bedürfen keiner weiteren Verbindung zu anderen Hosts.

Der entscheidende Unterschied zwischen den konventionellen Systemen und den agentenbasierten Systemen ist also, dass in den agentenbasierten Systemen die Berechnungsvorschriften von einem Teilnehmer zum nächsten Teilnehmer transportiert werden. Aus diesem Grund ist die eigentliche Aufgabe jeweils lokal in einem Peer durchführbar. Verbindungen zu anderen Teilnehmern werden also für die Berechnung nicht benötigt. Der Agent kennt seinen Absender und findet ihn selbstständig wieder.
In den konventionellen Systemen ist eine Kommunikation zwischen den Teilnehmern unumgänglich. Es wird hier nur eine Anfrage verschickt, und der Peer muß sich um Berechnungen und zusätzliche Informationen kümmern. Er kann die Berechnungen nicht lokal vornehmen und ist auf Verbindungen mit anderen Peers angewiesen, sowohl für die Berechnung als auch für die Beantwortung der Anfrage.

Der mobile Agent arbeitet autonom und hat die Fähigkeit sowohl die besuchten Peers mit neuen Informationen aus dem von ihm bereits gesammelten Fundus auszustatten, als auch die Fähigkeit seinen eigenen Vorrat an Informationen um die beim aktuell besuchten Peer

gefundenen Daten zu aktualisieren. In den konventionellen Methoden ist dies wenn überhaupt, dann nur mit einem großen Aufkommen an Kommunikation machbar. Diese Kommunikation belastet das Netz dann zusätzlich.

# 9. Fazit

Nach der Beschäftigung mit den verschiedenen Suchmethoden in Peer to Peer Netzen ist es unumgänglich eine Evaluation durchzuführen. Dafür sollen alle bisher betrachteten Mechanismen auf ihre Vorteile und Nachteile untersucht werden. Später soll dann ein Fazit gezogen werden.

## 9.1. Die Klassiker

Zuerst sollen die in Kapitel 2 vorgestellten Klassiker untersucht werden. Sie sind zwar keine Peer to Peer Systeme im eigentlichen Sinne, sind aber auch zum Tauschen von Dateien und Informationen geeignet. Einige dieser Systeme stellen auch Suchmechanismen zur Verfügung.

### 9.1.1. Das Usenet

Im Usenet werden in der Hierarchie alt.binaries Möglichkeiten zum Anbieten von Binärdateien geboten. Es ist auf Clientseite unnötig komplette Dateien herunterzuladen. Statt dessen läd sich der Nachfrager einer Datei oder Information eine Liste herunter. Diese Liste enthält Kopfzeilen und Informationen über Inhalte und Absender. Der Nutzer durchsucht dann diese Liste, die unter Umständen mehrere Tausend Dateieinträge enthalten kann, selber und markiert diejenigen Dateien, die er dann herunterladen möchte.

Der Vorteil ist, daß aufgrund der interessenbedingten Gruppenbildung im Usenet diese Listen relativ gut sortiert sind. Es ist sehr wahrscheinlich daß die gewünschte Information vorhanden ist und auch gefunden wurde. Außerdem wird dem Nutzer neben der nachgefragten Information auch noch weiteres, möglicherweise relevantes, Material zur Verfügung gestellt wird, ohne daß eine weitere Suchaktion nötig wird.

Der Nachteil ist, daß insgesamt weniger Daten zur Verfügung stehen als in anderen Filesharing Systemen. Dieser Nachteil ist aber möglicherweise nicht sehr gravierend, da es viele unterschiedliche Gruppen mit unterschiedlichen Inhalten gibt. Die Inhalte innerhalb einer Gruppe sind auf die Interessen der Mitglieder abgestimmt, und wenn alle Gruppen zusammengenommen werden, werden sich die Informationen schnell aufsummieren.

Ein gravierender Nachteil ist es, daß der Nutzer sich durch eine relativ umfangreiche Liste von Dateieinträgen durcharbeiten muß. Das kann sehr mühselig sein. Inwieweit dieser Nachteil aber nun zu der Entscheidung führt auf ein anderes System zurückzugreifen, das bliebt auch weiterhin im Ermessensspielraum des einzelnen Nutzers.

### 9.1.2. Der Internet Relay Chat

Im Internet Relay Chat kann man nicht direkt Dateien austauschen. Als textbasiertes Medium bietet der Internet Relay Chat den Nutzern mit gleichen Interessen zusammenzufinden. Der Datentransfer wird dann über eine Verbindung zwischen Anbieter und Nachfrager direkt abgewickelt. Mit modernen Clienten ist es möglich beispielsweise die lokale Datenhierarchie mittels Textoberfläche zugänglich zu machen. Über Trigger ist eine Kontaktaufnahme mit Nutzern mit gleichen Interessen möglich. Hier können Filesharing Aktivitäten verabredet werden. Als Vorteil ist zu erwähnen daß der Internet Relay Chat nur wenig Bandbreite benötigt, da nur Textzeilen ausgetauscht werden.

Der Nachteil ist offensichtlich. Es gibt keine echten Suchroutinen. Kontakte zu anderen Nutzern können zwar geknüpft werden, aber eine effektive Suche nach bestimmten Dateien oder Informationenen kann dabei sehr mühselig sein.

### 9.1.3. File Transfer Protocol und Tauschen im World Wide Web

Mit dem File Transfer Protocol können Dateien ausgetauscht werden. Die Daten werden in Archiven gespeichert, für die ein Nutzer verschiedene Rechte zum Upload und Download besitzen. Spezielle Suchmaschinen können FTP-Server auffinden.

Leider ist FTP nicht sehr komfortabel. Damit ist eine Benutzerfreundlichkeit nicht gegeben.

Es ist auch möglich Dateien im World Wide Web zu tauschen. Webspace Anbieter bieten Nutzern Platz Daten zu speichern. Diese können dann von jedermann per Browser erreicht werden. Hier ist die Suchmöglichkeit abhängig von der verwendeten Internet Suchmaschine, bzw. auf deren Fähigkeit diese Dateien zu suchen und anzuzeigen.

Speicherplatzanbieter bieten Platz zur Speicherung von eigenen Inhalten an. Der Eigentümer dieser Daten kann diese dann über ein Passwort erreichen. Dieses Passwort kann auch an andere Personen weitergegeben werden. Der Personenkreis, der auf diese Daten zugreifen kann, ist aber in der Regel klein. Suchmechanismen sind hier nicht unbedingt erforderlich.

## 9.2. Suche in P2P Systemen

Peer to Peer Systeme bieten die Möglichkeit Dateien und Informationen zu tauschen. Das führt zu der Notwendigkeit Suchmechanismen anzubieten. Nur so können gewünschte Dateien aus der Vielzahl von Dateien herausgefunden werden. Es gibt mittlerweile verschiedene Suchsysteme mit verschiedenen Eigenschaften.

### 9.2.1. Fluten des Netzwerkes

Beim Fluten eines Netzwerkes wird vom Absender eine Anfrage an alle verfügbaren Nachbarn geschickt. Diese schicken die Anfrage weiterhin an alle ihre Nachbarn und so weiter.

Der Vorteil dieser Strategie ist die Einfachheit. Jeder Teilnehmer kann selber entscheiden wie er die Anfrage bearbeiten will.

Allerdings ist diese Suchstrategie nur in kleinen Netzwerken effizient. In großen Netzwerken wird durch die durch das Fluten erzeugte Nachrichtenexplosion die Last im Netzwerk zu groß. Ein effektiver Betrieb des Netzwerkes ist dann nicht mehr möglich.

Weiterhin ist die Gefahr von böswilligen Angriffen groß. Durch die Versendung vieler unsinniger Anfragen ist es leicht das Netzwerk lahmzulegen.

Der Mechanismus das Netzwerk bei Anfragen mit Nachrichten zu fluten wurde daraufhin verbessert. Es wurde ein Time to Live Zähler eingeführt und das Konzept des random walk vorgestellt.

Der Time to Live Zähler wird zunächst auf einen niedrigen Wert gesetzt. Dann startet der Nutzer eine Anfrage Flut. Erhält er keine zufriedenstellende Antwort setzt er den TTL Zähler höher und startet eine neue Flut. Die Anfrage bricht jeweils nach der so vorgegebenen Anzahl von Schritten ab.

Der Vorteil ist, daß bei einer Anfrage zunächst nicht das ganze Netzwerk geflutet wird. Die Flut wird lokal begrenzt und dann schrittweise weiter ausgedehnt.
Nachteilig ist allerdings zu erwähnen, daß die Nachrichtenexplosion einer Flut weiterhin stattfindet.

Der random walk soll diese Nachteile beseitigen. Die Anfragebotschaft wird jeweils nicht an alle, sondern nur an einen Nachbarn weitergeleitet. Der Nachbar leitet die Nachricht auch jeweils nur an einen weiteren Nachbarn weiter. Durch den Einsatz zusätzlicher walker kann die Performanz gesteigert werden.

Hier ist das Problem der Nachrichtenexplosion beseitigt worden. Allerdings findet das Weiterleiten zufällig statt und die Antwortzeit könnte aufgrund der „Zufallsbefragung“ von Knoten suboptimal sein.

### 9.2.2. Dezentrale Hash Indizes

Hier werden die Daten nur über ihre ID erkannt. Ungenaue Suchen oder Schlüsselwortsuchen werden unmöglich gemacht.

Böswillige Teilnehmer können den Schlüsselraum belegen oder das Netzwerk mit sinnlosen Anfragen fluten.

### 9.2.3. Zentralisierte Indizes

Die zentralisierte Indizes waren bisher die leistungsfähigste Methode der Informationswiederauffundung. Hier übernehmen zentrale Server die Verwaltung der Daten. Leider sind die Kosten für ein solches System sehr hoch. Bandbreite und Hardware für große Netze sind teuer.

Gerichtsurteile haben in der Vergangenheit gezeigt, daß Schadensersatzklagen in zentral verwalteten Netzwerken immer erfolgversprechender werden.

### 9.2.4. Verteilte Indizes

Verteilte Indizes machen zentralisierte Server überflüssig. Die Teilnehmer übernehmen die Aufgabe des zentralen Servers.

Die Verwundbarkeit gegenüber der Gerichtsbarkeit sinkt damit zwar, allerdings kann die Vereinbarkeit und Eindeutigkeit der indizierten Daten darunter leiden. Durch die große Fluktuation der Teilnehmer und Daten kann der Overhead, der benötigt wird um diese Informationen aktuell zu halten sehr umfangreich werden. Auch die Unterstützung von einer Vielzahl von Metadaten kann problematisch sein. Metadaten werden benötigt als Informationen über indizierte Daten. Durch ein XML Schema können diese Daten erschlossen werden. Es wird allerdings ein Overhead erzeugt, der die Skalierbarkeit beeinflußt.

### 9.2.5. Relevanzbezogene Netzwerk Crawler

Der relevanzbezogene Netzwerk Crawler benutzt eine Datenbank. Diese enthält Informationen, die der Teilnehmer schon gesammelt hat. Er bestimmt damit ob eine Information, die ihm begegnet, für den Teilnehmer relevant sein könnte.

Der Vorteil ist, daß der Crawler selbsttätig Informationen für den Teilnehmer findet und klassifiziert. Das ist allerdings gleichzeitig auch der Nachteil. Der Teilnehmer kann nur Anfragen stellen, die seinem erstellten Profil entsprechen. Fremdartige oder spezielle Informationen können vom Crawler nicht zufriedenstellend bearbeitet werden, da er aufgrund der von ihm schon im Vorfeld gesammelten Informationen arbeitet.

Weiterhin unterstützt der Crawler nur sehr wenige verschiedene Arten von Daten, meistens HTML Dokumente oder andere Textdokumente.

### 9.2.6. Kataloge und Metaindizes

In einigen Netzen werden Katalogdokumente gespeichert. Diese beschreiben eine Ressource, die durch ihren Hash Schlüssel dargestellt werden. In diesem Katalog können Suchanfragen durchgeführt werden.

Das Problem ist, daß eine ständige Wartung und Aktualisierung der Kataloge unumgänglich ist. Das ist aber auch mit nicht unerheblichen Kosten verbunden.

### 9.2.7. Hybride Netzwerke und Super Peers

Der Zentrale Server zum Indizieren von Inhalten wurde hier abgeschafft. Statt dessen gibt es nun eine Anzahl sogenannte Super Peers. Diese übernehmen gemeinsam die Aufgaben des ehemaligen zentralen Servers.

Von Vorteil ist, daß zentrale Server vermieden werden. Damit sollen die Nachteile einer zentralen Architektur vermieden werden.

Nachteilig ist zu sagen, daß diese Super Peers noch immer kleine zentralisierte Server darstellen. Sie stehen dem Problem gegenüber viele Arten von Metadaten unterstützen zu müssen. Jeder Super Peer muß alle Arten von Metadaten unterstützen, die in Anfragen verwendet werden können. Das erzeugt einen großen Overhead. Weiterhin ist die Synchronisierung aller Super Peers kostspielig.

### 9.2.8. Adaptive soziale Suchmechanismen

Da die Kommunikation der Teilnehmer untereinander so lange aufrecht erhalten wird, wie diese es wünschen, ist es möglich für jeden Teilnehmer eine History zu führen. Daraus kann die Reputation eines Teilnehmers hergeleitet werden. Suchaktionen im Netzwerk können dadurch verbessert werden.

Böswillige Teilnehmer werden so schnell erkannt und ausgeschlossen. Der Schaden, den sie anrichten können hält sich so in Grenzen.

Durch die Reputation und den selbstbestimmbaren Verbindungszeiten wird ein gutes Benehmen im Netzwerk unterstützt. Die Teilnehmer werden zum Teilen ihrer Ressourcen motiviert.

### 9.2.9. Komplexe Suche in dezentralisierten Hashtabellen

Hashtabellen ermöglichen nur die Suche nach einem exakt bekannten Namen, der hinter der Hashfunktion steht. Dieser Nachteil kann ausgeglichen werden indem man den zu indizierenden String in Substrings der Länge n, genannt n-Gramme, unterteilt. Dann ist die Kenntnis des genauen Strings nicht mehr erforderlich und eine weniger strenge Suche möglich.

Hervorzuheben ist die Eigenschaft dieser Methode sich an die Suche in relationalen Datenbanksystemen anzulehnen. Es entsteht eine transaktionelle Speichersemantik, die aber nicht von jeder Anwendung gefordert wird.

### 9.2.10. Content Addressable Networks

Jeder Knoten wird in einem Koordinatenraum angeordnet und erhält einen Teil der gesamten Hashtabelle des Netzwerkes. Dieser Tabellenteil wird zone genannt.

Es gibt leider nur ein einfaches Interface für die Speicherung und das Wiederauffinden von Daten. Für das Wiederauffinden von Dateien ist die genaue Dokumenten ID nötig. Da aber Inhalte von vielen unabhängigen Quellen erzeugt werden, ist die genaue ID nicht immer verfügbar. Das ist der größte Schwachpunkt dieses Systems.

## 9.2.11. Peer Search

Peer Seach setzt auf CAN auf. Anfragen und Dokumente werden in Vektoren abgebildet. Mit der Berechnung des Kosinus der Winkel der Vektoren untereinander wird eine Ähnlichkeit der Vektoren und damit der Dokumente festgestellt. Die Vektoren sind genormt und die Wichtigkeit der Schlüsselworte im Dokument werden in die Vektoren mit einbezogen.
Indizes die sich semantisch nah sind werden auch im Netzwerk nahe beieinander gespeichert. Außerdem werden die Knoten nahe der für sie interessanten Indizes angeordnet

Ein Vorteil ist, daß die Teilnehmer ihre Interessengebiete in ihrer unmittelbaren Nähe finden. Daher findet auch die Suche in den meisten Fällen lokal begrenzt statt.

Ein weiterer Vorteil ist die Flexibilität der Technologie. Diese Technologie kann angewendet auf jede Art von Informationen, die sich auf Vektoren abstrahieren lassen.

## 9.2.12. Semantic Overlay Networks

Ein SON wird aus semantisch verwandten Knoten gebildet. Die Anfragen werden dann direkt an das passende SON geroutet. Dort angekommen wird die Anfrage nur innerhalb dieses SONs abgearbeitet. Damit werden semantisch fremde Knoten nicht berührt und können andere Aufgaben übernehmen. Das steigert die Leistungsfähigkeit dieses Modells. Weiterhin können Teilnehmer in mehreren SONs Mitglied sein, wenn sie semantisch in mehrere SONs passen. Innerhalb der SONs gibt es keine Vorschriften wer mit wem verbunden sein darf.

Die SON Architektur verteilt die Arbeit dorthin, wo sie mit der größten Wahrscheinlichkeit mit Erfolg erledigt werden kann. Das ist ein Vorteil dieser Architektur, denn die übrigen Knoten werden nicht beteiligt und können sich gleichzeitig anderen Aufgaben widmen.

## 9.2.13. Suchsysteme mit Agenten

Ein Agent ist ein Stück Software, welches zur Erfüllung seiner Aufgabe durch das Netzwerk reist. Er ist zu Beginn so klein wie möglich gehalten, kann aber im Zuge seiner Reise durch das Netz Daten sammeln und dementsprechend größer werden.

Agenten sind durch ihre Eigenschaften wie Autonomie und Mobilität sehr flexibel. Sie können nicht nur als Suchsysteme arbeiten, sondern können auch auf andere Aufgabengebiete programmiert werden.

Ihre Flexibilität und große Programmierbarkeit machen Agenten zu einem sehr mächtigen Werkzeug. Dieses Werkzeug kann bei richtiger Programmierung sehr gezielt eingesetzt werden. Dadurch sind die Agenten den konventionellen Suchsystemen auf Dauer überlegen. Allerdings ist noch eine tiefgreifende Beschäftigung mit dem Thema und eine umfangreiche Erstellung der benötigten Software nötig.

## 9.3. Fazit und Ausblick

Die Klassiker sind prinzipiell in der Lage als Tauschbörsen zu fungieren, auch wenn sie nicht dafür konzipiert wurden. Hier ist das Usenet wohl am besten geeignet. Hier findet sich eine Unterteilung in Gruppen und die Gruppen besitzen jeweils eine Dateiliste. Diese muß der Nutzer dann selber abarbeiten. Hier wäre eine Automatisierung wünschenswert.

In den P2P Netzwerken gibt es verschiedene Suchstrategien. Das Fluten des Netzwerkes ist dabei wohl eine der schlechtesten Methoden. Zu schnell kann durch viele Suchanfragen ein großes Netzwerk beeinträchtigt oder sogar lahmgelegt werden. Die Verbesserungen des Flutens, wie TTL oder random walk, bringen zwar eine Verbesserung, sind aber nicht wirklich praktikabel.

Zentralisierte Indizes lösen ihre Aufgabe recht gut, aber mit zunehmender Größe des Netzwerkes leiden sie unter den Nachteilen die ein zentralisiertes System mit sich bringt. Sie sind also für große Netzwerke nicht besonders gut geeignet.

Die komplexe Suche in dezentralen Hashtabellen, Peer Search und Semantic Overlay Netzwerks scheinen bisher die besten Kandidaten zu sein.

Die Suche in dezentralen Hashtabellen zeichnet sich wegen seiner Nähe zu den Relationalen Datenbanken aus. Das transaktionale Speichermodell ist sicher nicht für alle Anwendungen erforderlich. Dennoch könnte es Anwendungen geben, für die das Datenbankmodell wichtig oder sogar essentiell ist. Hier können die dezentralen Hashtabellen mit der vorgestellten komplexen Suchmethode gute Dienste leisten.

Auch die Flexible Methode des Peer Search erscheint vielversprechend. Sie ist skalierbar und effizient. Viele verschiedene Arten von Informationen können in Vektoren abstrahiert werden. Das macht Peer Search für eine breite Palette von Anwendungen einsetzbar.

Das Semantic Overlay Netzwerk erscheint zunächst auch vielversprechend. Die Suche wird nur in den betroffenen SONs durchgeführt ohne die übrigen Knoten zu beeinträchtigen. Ein gravierender Nachteil ist allerdings die große Abhängigkeit von einer Klassifikationshierarchie. Die Erstellung dieser Klassifikationshierarchie ist schwierig und vor allem teuer. Wenn sie fehlerhaft ist kann es zu Ungenauigkeiten bei der Suche kommen. Sind die Fehler gravierend kann eine Suche auch fehlschlagen, weil die betreffenden Dokumente falsch eingeordnet werden und damit unauffindbar sind. Eine teure und gewissenhafte professionelle Klassifizierung lohnt sich wahrscheinlich nur bei Anwendungen, bei denen es auf eine extreme Genauigkeit ankommt, und Fehler zu großen finanziellen Verlusten führen können. Eine dieser Anwendungen wird im Patentwesen verwendet.

Agentenbasierte Systeme erscheinen besonders vielversprechend. Sie sind sehr flexibel und können sogar weiterarbeiten, wenn sich der Peer vorübergehend vom Netzwerk trennt. Agenten stören den Netzwerkverkehr auch nur geringfügig. Sie reisen zwar durch das Netz, führen ihre Berechnungen aber lokal und ohne Kommunikation mit entfernten Rechnern durch.

Bei der Beschäftigung mit der Suche in P2P Netzwerken ist klar geworden, daß hier noch viel Forschung investiert werden muß. Einige der Suchmechanismen sind zu aufwendig, zu teuer oder schlichtweg ungeeignet. Aufbauend auf den positiv bewerteten Methoden ist eine weitere Entwicklung nötig. Anderseits ist seit der Erfindung des Flutens des Netzwerkes

schon einiger Fortschritt erreicht worden. Wenn die Entwicklung und Forschung in gleicher Weise weitergeführt wird, können die Methoden weiter verbessert werden und die Suche in Netzwerken effizienter gestaltet werden.

Es ist festzustellen daß agentenbasierte Systeme besonders vielversprechend zu sein scheinen. Sie erscheinen besonders leistungsstark und gut handhabbar. Dennoch ist auch hier noch eine umfangreiche Beschäftigung mit der Materie nötig. Erst wenn die Entwicklungsphase und Erprobungsphase abgelaufen ist, und sich die Agenten bewährt haben, werden sich die agentenbasierten Systeme in der „freien Natur“ durchsetzen. Eine Kombination aus den klassischen Peer to Peer Technologien und den Agenten kann hier sehr effektiv und gut einsetzbar sein.

# Abbildungsverzeichnis

# Literaturverzeichnis


| | |
|---|---|
| [AGARWAL1] | Manish Agarwal,<br>Security Issues in P2P Systems |
| [ALPINE1] | Decentralized Ressource Discovery in Large Peer Based Networks<br>http://peertech.org/alpine/discovery.html (Abruf am 26.5.2004) |
| [CRESPO1] | Arturo Crespo, Hector Garcia-Molina,<br>Routing Indices For Peer-to-Peer Systems<br>http://www-db.stanford.edu/~crespo/publications/crespora_ri.ps<br>(Abruf am 26.5.2004) |
| [CRESPO2] | Arturo Crespo, Hector Garcia-Molina,<br>Semantic Overlay Networks for P2P Systems<br>http://www-db.stanford.edu/~crespo/publications/op2p.pdf (Abruf am 26.5.2004) |
| [DUNNE1] | Using Mobile Agents for Network Ressource Discovery in Peer-to-Peer Networks,<br>Cameron Ross Dunne School of Computer Applications<br>Dublin City University, Dublin 9, Ireland |
| [GUERRA1] | S. Bergamaschi, F. Guerra,<br>First International Workshop AP2PC 2002<br>Peer-to-Peer Paradigm for a Semantic Search Engine |
| [HARREN1] | Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon T. Loo, Scott Shenker and Ion Stoica,<br>Complex queries in DHT-based peer-to-peer networks, in Procedings of the First International Workshop on Peer-to-Peer Systems (IPTPS), March 2002 |
| [HEISE1] | Erik Möller<br>http://www.heise.de/tp/deutsch/inhalt/te/8504/1.html (Abruf am 11.3.2004) |
| [LV1] | Qin Lv, Pei Cao, Edith Cohen, Kai Li, Scott Shenker<br>Search and Replication in Unstructured Peer-to-Peer Networks<br>http://parapet.ee.princeton.edu/~sigm2002/papers/p258-lv.pdf (Abruf am 26.5.2004) |
| [MINAR1] | Nelson Minar, Topologies<br>http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html (Abruf am 26.5.2004) |
| [MORO1] | Gianluca Moro, Manolis Koubarakis,<br>First International Workshop AP2PC 2002<br>Agents and Peer-to-Peer Computing: A Promising Combination of Paradigms |

[RANGANATHAN1] Kavitha Ranganathan, Ian Foster,
Identifying Dynamic Replication Strategies for a High Performance Data Grid
http://people.cs.uchicago.edu/~krangana/papers/p_grid_computing.ps (Abruf am 26.5.2004)

[RATNASAMY1] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker
A Scalable Content-Addressable Network
http://www.ovmj.org/GNUnet/papers/can.pdf (Abruf am 26.5.2004)

[SMITHSON1] Andrew Smithson, Luc Moreau,
First International Workshop AP2PC 2002
Engineering an Agent-Based Peer-to-Peer Resource Discovery System

[TAN1] Chunqiang Tan, Zhichen Xu, Mallik Mahaligam
Peer Search: Efficient Information Retrieval in Peer-to-Peer Networks
http://hpl.hp.com/techreports/2002/HPL-2002-198.pdf (Abruf am 26.5.2004)