Problemseminar im WS 03/04 - Prof. Dr. E. Rahm

Peer-to-Peer (P2P) Data Management

Quality of Service / Robustheit in P2P-Netzwerken

Bearbeiter: Oliver Schmidt

Betreuer: Dr. Sosna

Inhaltsverzeichnis

1. Einleitung

Aktuelle Peer-to-Peer-Netzwerke (P2P) teilen sich in zwei Lager auf: Auf der einen Seite stehen die puren P2P-Netze wie z.B. Gnutella, die aufgrund fehlender zentraler Server weniger Angriffsfläche für Service-Attacken bieten und deshalb relativ robust sind. Allerdings ist die Suche in diesen Netzen meist nicht sehr effektiv; Routing-Techniken wie das Fluten finden meist nur die Peers innerhalb ihres durch die TTL-Flags vorgegebenen Suchhorizonts und somit nicht die gesamte Antwortmenge (siehe Vortrag 3).
Auf die Suche spezialisiert sind jedoch die hybriden P2P-Systeme, bei denen zentrale Server die Daten oder Dienste der untergeordneten Clients in einer Datenbank zusammentragen und so einen größeren Teil der Antwortmenge finden. Dafür sind diese Server aber auch Single Points of Failure, denn scheidet einer der Server aus beliebigen Gründen aus, so werden die Daten oder Dienste der ihm untergeordneten Clients nicht mehr im Netz gefunden und die Clients müssen sich einen neuen Server suchen.
Das Ziel wäre es also, die Vorteile der beiden Systeme zu verbinden – die guten Suchergebnisse des hybriden Modells mit der Robustheit der puren P2P-Netze. Dieser Vortrag beschäftigt sich mit Ansätzen welche versprechen, die Servicegüte des Netzwerkes zu erhöhen und es dabei so robust wie möglich zu gestalten.

2. Kriterien für Quality of Service und Robustheit in P2P-Netzwerken

2.1 Robustheit

Zuerst muss jedoch geklärt werden, was Servicegüte und Robustheit in einem P2P-Netz bedeuten und welche Bewertungsmaßstäbe angelegt werden können.
Wenn ein Gegenstand mit der Eigenschaft „robust“ belegt wird so verbindet man damit dass er stabil ist, selten defekt geht und auch einmal eine falsche Behandlung übersteht. Bei Netzwerken ist dies ähnlich. Die wichtigste Forderung in Zusammenhang mit der Stabilität ist die Fehlertoleranz. Ein Netz aus tausenden von Peers mit der unterschiedlichsten Hardware darf nicht beim ersten kleinen Fehler den Dienst verweigern. Vielmehr muss es solche Ereignisse wie spontan das Netz verlassende Peers oder gar das Zusammenbrechen ganzer Teilnetze verarbeiten und dabei im

günstigsten Fall ohne Einschränkung oder Beeinträchtigung der angebotenen Dienste weiter funktionieren. Ständige Fehler sind bei einem P2P-Netz die Regel und nicht die Ausnahme und so muss ein robustes Netz diese erkennen und geeignete Maßnahmen ergreifen um weiterhin seinen Zweck zu erfüllen. Laufende Operation wie z.B. ein Datentransfer sollten im Falle einer Unterbrechung automatisch fortgeführt werden und globale Funktionen des Netzes (z.B. das Routing) sollten nicht ausfallen.
P2P-Netze besitzen auch die Eigenschaft, dass die Peers sich in sehr schnellen Abständen an das Netz anmelden und sobald sie ihr Ziel erreicht haben sich sofort wieder daraus entfernen. Das Netz darf also keine starre Struktur haben und muss auch bei tausenden von Peers korrekt skalieren und seine Dienste weiterhin in der selben Qualität zur Verfügung stellen. Dazu muss es selbständig seine Effizienz optimieren und sich an die neuen Gegebenheiten anpassen.
Dies alles kann jedoch nur in reinen P2P-Netzen erreicht werden, da dessen Peers die nötige Autonomie besitzen. Hybride Lösungen bieten dagegen zu viele zentrale Angriffspunkte. Sie sind meist von bestimmten Servern abhängig, die für die Dienste des Netzes verantwortlich sind. So konnte das File-Sharing-System Napster nur aufgrund seiner ausgeprägten Server-Client-Struktur so schnell aus dem Verkehr gezogen werden, während das vollkommen dezentralisierte Netz Gnutella noch immer aktiv ist.

2.2 Quality of Service (QoS)

Ein P2P-Netzwerk soll immer einen bestimmten Zweck erfüllen, den z.B. das Internet nicht bietet. Da wäre zuerst die Performanz. Um einen Dienst oder Daten im Internet für eine große Anzahl von Clients bereit zu stellen bedarf es leistungsstarker und teurer Server. Und trotzdem sinkt bei hoher Netzlast oft die Performanz unter ein Niveau, welches man eigentlich garantieren will. In P2P-Netzen kann dieser Flaschenhals umgangen werden, indem die Daten oder Dienste auf verschiedenen Peers verteilt werden, die oft auch eine bessere Lokalität bieten.
Des weiteren sollte der Anbieter hoch verfügbar sein (also praktisch keine Ausfallzeit haben) und über genügend Bandbreite verfügen, was jedoch bestimmten Grenzen unterliegt. Ein zur Absicherung zusätzlich installierter Server und eine schnelle Anbindung an das Datennetz sind nicht zu unterschätzende Posten, von den Kosten zur Administration ganz zu schweigen. Verteilte Netzwerke können hier kostengünstig eine höhere Dienstgüte anbieten, als ein Peer alleine in der Lage wäre bereitzustellen. Dazu sollten sie aber auch in ihren Operationen nicht zu stark skalieren

und auch bei einem plötzlichen Ansturm an Queries in der selben Geschwindigkeit antworten und Dienste erbringen.
Auch an die Anfragen in P2P-Netzen werden hohe Ansprüche gestellt. So sollte ein jedes im Netz verfügbare Objekt und jeder Peer auf eine Suche hin auch gefunden werden. Des weiteren hat das Netz auch für die Qualität und Nutzbarkeit der Ergebnisse zu sorgen, denn was nützen schon hoch verfügbare Dienste oder Daten, die gar nicht der Vorstellung entsprechen oder gar fehlerhaft sind. Und zuletzt sollte ein Netz die Dauerhaftigkeit der Ressourcen garantieren, so dass Daten und Dienste auch nach einer kurzen Zeit mit hoher Nachfrage nicht komplett aus dem Netz verschwinden, sondern nach einem längeren Zeitraum zumindest noch verfügbar sind.

2.3 Bewertungskriterien

Um die Dienstgüte richtig einzuschätzen bedarf es deshalb bestimmter Kriterien. Leider stellen Nutzer von P2P-Netzen meist unterschiedliche Anforderungen, nach denen verschieden optimiert werden kann. Die Kriterien lassen sich nach [RH01] grob in 2 Bereiche einteilen: Informations- und Speichermanagement.
Unter Informationsmanagement fallen z.B. die Ergebnisse von Suchanfragen. Der eine User legt nur Wert auf eine möglichst große Anzahl von Treffern, um dann eine Auswahl daraus zu treffen, während andere Benutzer korrekte, relevante und vor allem aktuelle Ergebnismengen bevorzugen, wobei sie vorher genau wissen müssen wonach sie suchen. Da die Anfrage an ein P2P-Netz auch als Information Retrieval über eine Datenmenge (die von den Peers gehosteten Daten und Dienste) aufgefasst werden kann, können die dafür geltenden Bewertungskriterien verwendet werden: *Precision* gibt an, welcher Anteil der gefundenen Antwortmenge überhaupt relevant ist und *Recall* ist ein Maßstab dafür, wie viele der relevanten Ergebnisse vom System gefunden werden. Um möglichst beide oben beschriebenen Usergruppen zu befriedigen sollte ein QoS-unterstützendes P2P-Netz also eine große Teilmenge der im Netz zu dem Zeitpunkt verfügbaren Antwortmenge finden (Recall gegen 1) und falsche Ergebnisse vorher heraus filtern (Precision gegen 1).
Wichtig im Speichermanagement dagegen ist die Zeit, die eine Anfrage braucht, um Ergebnisse zu liefern und diese auf den suchenden Peer zu übertragen. Dies betrifft die Lokalisation von Daten oder Diensten, die so schnell wie möglich erfolgen soll und dabei aber im bestmöglichen Fall auch alle zu diesem Zeitpunkt verfügbaren Ressourcen findet. Ist ein Ergebnis dann gefunden zählt auch noch wie schnell der Suchende das Objekt oder den Dienst dann an seinem Peer zur Verfügung hat,

also ob die Ressource mit der höchsten Verfügbarkeit und Nähe gefunden wurde. Dazu gehört auch eine möglichst schnelle Wiederaufnahme eines Transfers nach einem Abbruch, was den Zusammenhang zwischen QoS und Robustheit verdeutlicht.

3. Probleme mit Datenbeständen und Services in P2P-Netzwerken

Viele der aktuellen P2P-Netze dienen zum Austausch oder zu Verwahrung von Daten, wobei die Spanne von Textdateien über Computerprogramme bis hin zu Filmen reicht. Alle diese Daten haben ähnliche Probleme, die von den Netzen gelöst werden wollen. Das Grundproblem ist dabei die nicht vertrauenswürdige Infrastruktur. Keiner der Peers weiß wer am anderen Ende der Leitung sitzt und ob dieser in den selben Maße bereit ist für die Daten oder den Dienst zu garantieren. So könnte ein Peer einfach alle über ihn laufenden Anfragen einfach fallen lassen anstatt sie weiter zu leiten.
Vertraut man einem Peer allein bestimmte Ressourcen an so sind verschiedene Szenarien möglich:

- Optimaler Fall: der Peer erfüllt seinen Dienst und ist für alle anderen erreichbar.
- Der Peer könnte aber auch die Daten oder den Dienst unfreiwillig manipulieren, z.B durch fehlerhafte Hardware oder weil er Opfer eines Angriffs geworden ist (z.B. Viren oder Hacker). Er würde die Ressourcen dann weiterhin für alle anbieten, diese wären jedoch für die anderen Peers nutzlos, da sie nicht korrekt wären.
- Der Peer könnte die Daten oder den Dienst mutwillig manipulieren. So könnte er Viren auf andere Peers übertragen oder Denial-of-Service-Attacken durchführen. Der harmloseste Fall wäre noch die einfache Verschwendung der Bandbreite mit nutzlosen Ressourcen.
- Der Peer könnte aber auch einfach nicht mehr erreichbar sein. So könnte sein Teilnetz vom restlichen Netz abgetrennt sein, oder der Peer hatte einen Hardwarefehler oder die bereitstellende Partei hat sich aus dem Netzwerk ausgeklinkt. Somit stünden die kompletten Daten oder Dienste den anderen Peers nicht mehr zur Verfügung.

Für alle diese Szenarien muss ein P2P-Netzwerk entsprechende Regeln einführen und Gegenmaßnahmen ergreifen, will es Robustheit und QoS garantieren.

4. Lösungsansätze für QoS und Robustheit

4.1 Data Replication

Nachdem nun die Kriterien und die Probleme bekannt sind soll hier eine Auswahl von Lösungsansätzen vorgestellt werden. Diese sind zum Teil schon in P2P-Netzwerken implementiert, doch weniger in den bekannten FileSharing-Netzen mit vielen Benutzern, so dass nicht immer genaue Aussagen über die Effektivität der Methoden gemacht werden können.
Ist eine Ressource nur in einer kleinen Menge im Netz vertreten und wird sie häufig angefordert, so entstehen Engpässe und nicht zumutbare Performanzeinbrüche bei der Nutzung. Deshalb liegt es nahe die Daten zu vervielfältigen und sie damit auch längerfristig verfügbar zu machen. Dafür gibt es zwei unterschiedliche Ansätze der Data Replication, nämlich das simple Kopieren der Daten (Redundanz) und die aus der Kryptographie bekannte redundante Kodierung (Erasure Codes).

4.1.1 Redundanz

Redundanz bedeutet die einfache Vervielfältigung einer Ressource, z.B. einer Datei. Dazu werden Kopien davon angefertigt und an unterschiedlichen Orten im Netz platziert. Dies erhöht zum einen die Dauerhaftigkeit [CGM02], da die Wahrscheinlichkeit dass alle Kopien aus dem Netz entfernt werden sinkt, zum anderen werden auch die Anfrage- und Bereitstellungszeiten gesenkt, da nun die Wahrscheinlichkeit höher ist dass sich eine Kopie in der Nähe des eigenen Peers befindet. Damit verbessert sich auch die Fehlertoleranz, denn erstens muss die Datei nicht mehr so weit im Netz übertragen werden, was die Anzahl der Übertragungsfehler senkt, und zweitens ist es durch die unterschiedlichen Speicherorte und Typen der datenhaltenden Peers auch schwerer, einen Angriff auf genau diese Ressource zu führen.
Beim Kopien-Management stellen sich dabei die Fragen, wie oft eine Ressource kopiert werden soll und vor allem wohin die entstandenen Kopien dann verteilt werden sollen.
Überlässt man die Verteilung den Nutzern des Netzes so ist festzustellen, dass häufig gefragte Dateien sich besonders schnell verbreiten, während selten angeforderte Daten aus dem verteilten Speicher verschwinden [CS02]. Da jedoch die Dauerhaftigkeit eine Qualitäts-Anforderung an ein P2P-Netz ist, muss man das Kopien-Management dem Netzwerk überlassen. Der naive Ansatz ist dass alle Ressourcen gleichermaßen oft kopiert werden, so dass für alle Daten die selbe

Dauerhaftigkeit garantiert ist. Bei starker Nachfrage nach bestimmten Dateien ergeben sich dann jedoch Probleme mit der Verfügbarkeit, so dass die meisten P2P-Netze mit Kopien-Management die Anzahl der Replikate dynamisch den Anforderungen anpassen, ohne jedoch ein bestimmtes Level an Kopien zu unterschreiten.
Die Frage nach der optimalen Platzierung auf den Peers hat sich als NP-vollständig herausgestellt [KRT02]. So werden meist Heuristiken angewandt um die Peers nicht zu stark zu belasten:

- Bei der ***Zufallswahl*** werden die erzeugten Kopien nach einem zufälligen Schema über das P2P-Netzwerk verteilt. Dabei können jedoch keine Aussagen über die Verfügbarkeit getroffen werden, so dass bei einer ungünstigen Verteilung keiner der Kopien-Halter eine bestimmte Dienstgüte garantieren kann und die Gesamtperformanz nur minimal steigt. Dafür ist die Aufteilung über das Netz aber recht gleichmäßig.
- Bei der ***Sortierung nach Verfügbarkeitskriterien*** (wie Bandbreite und Ausfallzeiten) werden die Kopien auf die Peers mit der besten Dienstgüte verschoben. Dies erfordert Wissen des Netzes über die eigene Topologie und hat zur Folge, dass die hoch verfügbaren Server den Großteil der Daten speichern und das Netz wieder zu einer hybriden P2P-Struktur mutiert und damit die selben Nachteile wie Single Point of Failure und Bandbreitenmangel besitzt.
- Den meisten Aufwand bereitet die ***Verteilung nach Bedarf***, denn das Netz muss ständig sämtliche Anfragen überwachen und bei entsprechenden Engpässen Kopien an den erforderlichen errechnete Punkten im Netz hinterlegen und diese auch wieder entfernen. Durch die relativ optimale Verteilung wird jedoch auch die Netzlast verringert, da keine großen Routing-Strecken zurückgelegt werden müssen.

Trotz der Unterschiede in Effizienz und Aufwand erhöhen alle drei Verfahren nach [OSS03] die Geschwindigkeit der Anfragen und der Datenübertragung, wobei sich Nähe als das wichtigere Kriterium heraus gestellt hat, da mit steigender Routenlänge auch die Übertragungsfehler zunehmen.

4.1.2 Erasure Codes

Eine andere Form der Replizierung ist die so genannte M-von-N-Codierung. Dabei werden die Daten in n Fragmente aufgeteilt, von denen aber nur beliebige m Fragmente ($m < n$) zur

Wiederherstellung benötigt werden. Diese Fragmente werden schließlich nach dem Zufallsschema auf andere Peers verteilt. Dadurch dass nicht alle Fragmente auch wieder gebraucht werden wird die Dauerhaftigkeit der Daten gewährleistet, denn selbst beim Ausfall von $n - m$ der Fragmente können die Daten wieder hergestellt werden. In gewissen Abständen werden die Fragmente dann wieder zusammengetragen, dekodiert und die Daten erneut fragmentiert und verteilt. Damit wird eine bei gleichem Speicherplatzbedarf höhere Haltbarkeitsdauer gegenüber normaler Redundanz erreicht (ca. 4000fach), die jedoch mit erhöhtem Aufwand beim Warten (Bandbreitenverlust durch Kommunikation) und Fragmentieren (mehr Rechenzeit) erkauft wird [WK02].

4.2 Nomadic Data

Das Konzept des Nomadic Data bedeutet dass die im Netz gespeicherten Daten nicht mehr auf einem bestimmten Peer gespeichert werden, sondern selbständig im Netz umher wandern und sich dort aufhalten wo sie gebraucht werden. Daten werden so nur noch nach dem Namen identifiziert und können keinem Ort im Netz direkt zugeordnet werden. Dies hat zur Folge dass die Suche im Netz zum Routing-Problem wird [KU03], dem der nächste Abschnitt gewidmet ist.
Der Vorteil dieses Ansatzes ist dass die Daten sich durch das selbständige Bewegen immer dort befinden, wo der meiste Bedarf nach ihnen besteht und dass zu einem bestimmten Zeitpunkt häufig gefragte Daten nicht auf schlecht verfügbaren Peers die Performanz drücken. Des weiteren verringert sich die Wahrscheinlichkeit eines erfolgreichen Denial-of-Service-Angriffs, da nie genau festgestellt werden kann wo sich der Service gerade befindet und auf welchem Weg sich die Anfrage durch das Netzwerk bewegt. In Kombination mit dem Konzept der Data Replication ist es sogar nicht einmal sicher, dass die selbe Datei angeroutet wird. Dies erhöht die Fehlertoleranz und damit die Robustheit des P2P-Netzes, während gleichzeitig die User von der erhöhtem Übertragungsperformanz und Suchgeschwindigkeit profitieren.

4.3 Routing

In dem meisten aktuellen P2P-Netzen wird zwischen dem Peer, welcher den Dienst anbietet, und dem diesen Dienst nutzenden Peer eine direkte Verbindung aufgebaut, zu der meist das Internet genutzt wird. Doch gerade Verbindungen mit IP-Routing sind nicht in der Lage, die Güte eines Dienstes zu garantieren. Deshalb ist es wünschenswert, dass auch die Datenübertragung über die

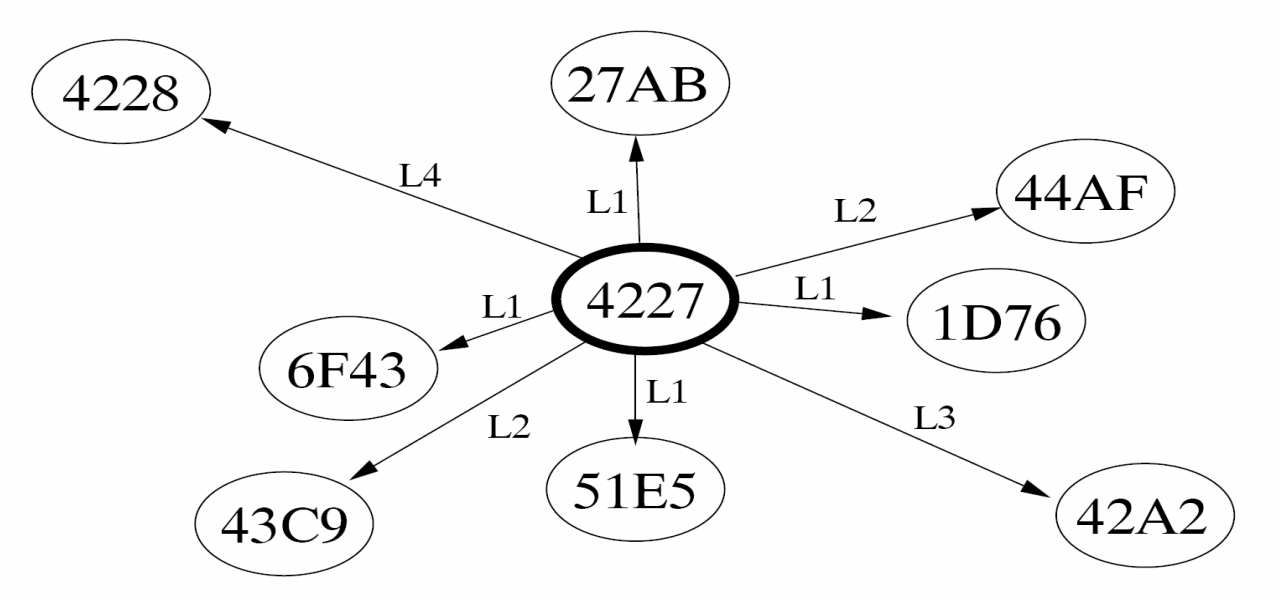

Abbildung 1 - Der Knoten 4227 und seine Nachbarknoten (aus [HK02])

Routingschicht des P2P-Netzes erfolgt, da sie mit verschiedenen Optimierungen ein höheres Level an QoS zur Verfügung stellen kann. Ein wichtiger Faktor dabei ist das deterministische Routing, d.h. dass ein mit dem Netz verbundener Peer auch immer gefunden wird.

Verfahren wie das Fluten sind nicht in der Lage, diese Eigenschaft zu erfüllen, weshalb auch die Anfrageergebnisse nicht die gewünschte Qualität aufweisen. Trotzdem kann man auch bei nichtdeterministischen Verfahren noch die Ergebnisse verbessern. So muss man die Anfragen nicht wahllos an alle verfügbaren Nachbarn des Peers verschicken, sondern kann eine Vorauswahl treffen. Diese könnte sich nach der Anzahl der vorhandenen Daten oder Dienste auf dem anderen Peer bzw. dessen Verfügbarkeit richten oder alle Peers könnten Nachbarlisten mit Interessengebieten führen und Anfragen dann nur an die Teilnehmer im Netz schicken, von denen bekannt ist, dass sie auch diese Interessen vorweisen [SUV03]. So wird der Kommunikationsaufwand für Suchen im Netz gesenkt und die Precision erhöht. Nachteil des Verfahrens ist, dass der entsprechende Client solche Listen stets aktuell halten muss, ohne über den aktuellen Zustand des entsprechenden Peers Bescheid zu wissen. Auch könnte die Einteilung in fest vorgegebene Interessengebiete bei bestimmten Daten und Diensten Schwierigkeiten bereiten. Also darf man diese Aufgabe nicht den Peers überlassen sondern muss dies automatisch im Netzwerk durchführen, zum Beispiel nach dem Plaxton / Rajamaran / Richa Schema [PRR97].

Hierbei bekommen die Netzknoten (Node-ID) und die Daten (GUID) im Netz eindeutige Namen zugewiesen, z.B. durch eine Hash-Funktion, die Zahlen mit einer festen Bitlänge erzeugt oder eine Zufallsfunktion, die mit sehr geringer Wahrscheinlichkeit gleiche Werte produziert. Danach werden

alle Ressourcen auf die Knotenmenge abgebildet, d.h. jede GUID wird einer möglichst übereinstimmenden Node-ID zugeordnet. In diesen Knoten werden dann die IDs der datenhaltenden Peers abgespeichert, so dass die Suche nach einer GUID zum Routing-Problem wird.
Dieses Routen erfolgt dann über Nachbartabellen. In diesen Tabellen speichert jeder Knoten die Übereinstimmung der Präfixe seiner Node-ID mit denen seiner Nachbarn. Hat der Knoten also die ID 4227, so speichert er in der ersten Zeile seiner Nachbartabelle diejenigen Nachbarn, deren ID nicht mit 4 anfängt in der entsprechenden Spalte. Auf dem zweiten Level folgen dann die Links zu den Nachbarknoten mit einer 4 am Anfang, z.B. die Knoten 44AF und 43C9. Die Tabelle des Knotens aus Abbildung 1 sieht dann aus wie in Abbildung 2:

	1	2	3	4	5	6	7	8	9	0	A	B	C	D	E	F
Level 4								4228								
Level 3											42A2					
Level 2			43C9	44AF												
Level 1	1D76	27AB			51E5	6F43										

Abbildung 2 - Nachbartabelle des Knotens 4227 in Abbildung 1

Sucht der Knoten 4227 nun einen anderen Knoten mit der ID 8734, so stellt er fest dass die beiden Node-IDs keine Präfix-Übereinstimmung haben und sucht nun in den Level 1-Einträgen seiner Nachbartabelle nach einem Knoten, der den Präfix 8 besitzt. Dieser wiederum sucht dann einen Nachbarknoten auf dem Level 2, der nächste auf dem Level 3 usw. Enthält die Tabelle keinen Eintrag auf dem nächsthöheren Level wird auf der aktuellen Stufe weiter gesucht, bis ein Eintrag bei einem Knoten gefunden wird. In dem Fall dass bei allen Knoten dieses Levels der Eintrag in der Tabelle leer ist gibt es die entsprechenden Node-ID auch nicht (da die Tabellen als vollständig angenommen werden) und das Netz routet automatisch den „ähnlichsten“ Knoten an. So ist garantiert, dass das Routing auch in jedem Fall erfolgreich verläuft.
Wird nun eine neue Ressource im Netzwerk veröffentlicht (Publishing), so sucht der daten- oder diensthaltende Knoten per Routing den Peer mit der selben ID wie die GUID des Objekts. Auf dem Weg zu dieser zentralen Speicherstelle für Verweise auf das Objekt hinterlässt er auf jedem Knoten ebenfalls einen Verweis. Wird die Ressource nun im Netz gesucht und kreuzt das Suchrouting den Pfad des Publishings, so entdeckt es dort den Verweis und kann sich den weiteren Weg bis zur Wurzel sparen, da es einen Link gefunden hat.
Dieser deterministische Ansatz hat jedoch den Nachteil, dass durch die Zentralisierung der Speicherung aller Zeiger auf ein Objekt wieder ein Single Point of Failure entsteht, der zum Ausfall

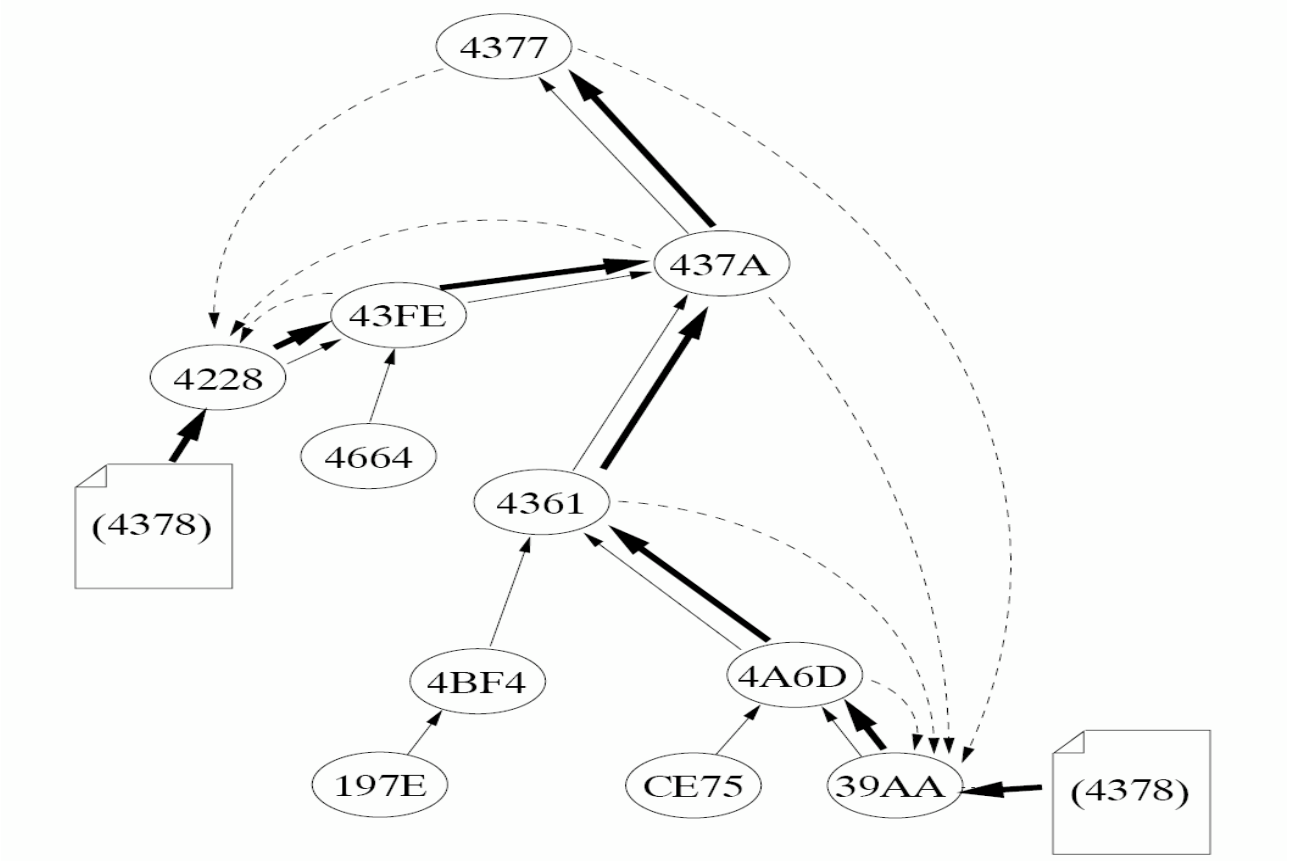

Abbildung 3 - Routing und Publishing nach dem PRR-Schema (aus [HK02]). Die gestrichelten Linien sind Verweise auf Ressourcen, die fetten Linien zeigen den Routing-Pfad.

der kompletten Ressource führen kann, wenn die Wurzel aus dem Netz verschwindet. Außerdem entsteht für das Publizieren einer Ressource ein höherer Kommunikationsaufwand als z.B. in hybriden P2P-Netzen.

Plaxton / Rajamaran / Richa gingen bei der Entwicklung ihres Schemas von einem stabilen Netz aus, welches aber in einem P2P-Netzwerk meist nicht gegeben ist. Dass der Ansatz jedoch mit einigen Modifikationen dennoch auch für sich schnell ändernde, verteilte Netze brauchbar ist, wird in Kapitel 5.2 bei der Vorstellung von Tapestry gezeigt [ZH03].

In [LDA03] wird dagegen ein Routing vorgestellt, welches bei der Anfrage bereits auf der gesamten Route Bandbreite für die Übertragung reserviert. Diese Qualität kann jedoch nur in statischen Netzen angeboten werden, wo garantiert wird dass diesselbe Route auch während der kompletten Übertragung konstant bleibt. P2P-Netzwerke leben jedoch von dem ständigen Einfügen und Entfernen einer großen Menge von Peers, weshalb dieses Routing dort nicht eingesetzt werden kann.

4.4 Trading

Ein großes Problem der aktuellen P2P-Netzwerke in Bezug auf die Robustheit ist dass so genannte Free-Riding: Die User melden ihre Clients im Netz an, ohne eigene Dienste oder Daten anzubieten, und nutzen die angebotenen Ressourcen des Netzes um sich danach gleich wieder auszuklinken. Unter solchen Umständen kann kein hohes QoS-Level garantiert werden, da das Netz sich ständig umstrukturieren muss und die wenigen Peers mit hoher Verfügbarkeit und Speicherkapazität praktisch zu Servern für den Rest der Clients verkommen. Ein kollaboratives Verhalten aller Nutzer ist also erforderlich, wobei jedoch berücksichtigt werden muss dass die Autonomie jedes Peers nicht verloren geht – jeder soll weiterhin eigene Entscheidungen bezüglich den Partnern, der Bereitstellung eigener Ressourcen und dem Zugang dazu treffen.
Die Teilnehmer an einem P2P-Netzwerk habe im allgemeinen 3 verschiedene Funktionen:

- Sie sind Server, die Daten oder Dienste hosten und diese dem Netz zur Verfügung stellen.
- Sie können aber auch die Rolle des Clients übernehmen, der diese Ressourcen nutzt.
- Sie müssen dem Netz passiv bereitstehen, um Anfragen und Ressourcen weiterzuleiten.

Nutzer des Netzes, die nur eine der Funktionen wahrnehmen, müssen also dazu angeregt werden, auch die restlichen Aufgaben zu übernehmen. Dafür bietet sich ein ökonomisches Modell an, welches die Leistungen eines Peers in einem globalen Maßstab bewertet und dem entsprechend „erfolgreiche“ Peers mit besonderen Vorteilen belohnt. Diese Belohnungen könnten z.B. ein höheres Level an QoS sein, indem Anfragen dieser Peers in den Warteschlangen bevorzugt behandelt werden, oder Zugang zu ansonsten nicht zur Verfügung stehenden Objekten.
Bei Netzen in denen die Speicherung von Daten im Vordergrund steht bietet sich ein Tauschprinzip an. Die in [CGM01] beschriebenen Trading-Verfahren gehen davon aus, dass ein Peer seine Daten auf möglichst vielen anderen Peers verteilen will um die Verfügbarkeit und Dauerhaftigkeit zu erhöhen. Dazu bietet der Nutzer zum Ausgleich für die Benutzung der Speicherkapazität des anderen Teilnehmers eigenen Speicherplatz (Data Trading) oder Nutzungsrechte (Deed Trading) dafür an. Nimmt der Trading-Partner das Angebot an, so werden die Daten ausgetauscht und beide Teilnehmer sind (hoffentlich) zufrieden. Dieses System erhöht die Fairness innerhalb des Netzes, wobei jedoch Clients mit begrenztem Speicherplatz es schwerer haben ihre Daten verteilt zu bekommen. Dafür wird aber jeder Nutzer angeregt, sich aktiv und passiv zu beteiligen, was die

Qualität des Netzwerkes erhöht, da mehr Speicherplatz und Bandbreite zur Verfügung steht und durch die größere Suchmenge Ergebnisse schneller gefunden und übertragen werden. Gleichzeitig steigt auch die Robustheit, da der Ausfall einzelner Peers oder eines Teilnetzes sich durch die weite Verbreitung der Ressourcen nicht mehr so stark auf die Verfügbarkeit auswirkt.

4.5 Load Balancing

Je stärker ein Netzwerk wächst, also je mehr Peers sich ständig an- und abmelden, umso größer wird der administrative Aufwand um diese Netz unter Beibehaltung des QoS-Levels und der Robustheit zu skalieren. Dies kann keine Institution übernehmen, da sie ja ebenso dynamisch wachsen müsste. Also kann diese Aufgabe nur das Netzwerk selber erledigen.
Dazu gehört jedoch ein globales Wissen über den aktuellen Zustand des Netzwerkes, welches jedoch nicht auf zentralen Servern gespeichert werden soll. Also müssen die Peers alle einzeln die Daten über die Netztopologie und den Datenverkehr in ihrer Nähe sammeln und auswerten. Die Ergebnisse könnten dann z.B. entlang einer temporären Hierarchie einen Baum hinauf wandern und von den Vaterknoten in größerem Kontext ausgewertet werden.
Aber es können einige Entscheidungen auch schon lokal auf den Knoten getroffen werden. Fällt von einem Peer ein Nachbarknoten aus oder ist dieser überlastet, so kann er Anfragen und Datenverbindungen auf einen anderen Knoten umleiten. Des weiteren kann sich der Peer in dem Ausfallszenario (spontanes Verschwinden eines Nachbarknotens aus dem Netz) selber neue Nachbarn suchen.
In einem größerem Kontext können Routen so auch weitläufiger abgeändert werden (zum Beispiel beim Ausfall eines ganzen Teilnetzes oder bei Denial-of-Service-Attacken) oder lokale Kopien dort erstellt werden, wo sie gerade benötigt werden.
Des weiteren muss sich ein neu angemeldeter Peer auch automatisch in das Netz einbringen können ohne gleich die Performanz der benachbarten Knoten oder gar des gesamten Netzwerkes negativ zu beeinflussen. Dazu ist es notwendig dass das Netz seine Leistungsfähigkeit einzuschätzen weiß und ihm die nötigen Daten zur Eingliederung übermittelt.
Ein weiterer positiver Aspekt des automatische Load Balancings ist dass keine Institutionen mehr Einfluss auf die Gestaltung der Topologie haben. So gibt es keine Vor- oder Nachteile für die verschiedensten Peers, alle werden gleich behandelt und können sich keine Begünstigungen verschaffen, was wiederum durch die Ausgeglichenheit die Robustheit des Netzes fördert.

4.6 File / Service Filtering

Da in vielen der aktuell sehr populären P2P-Netzwerke die Ressourcen nur nach einem Namen identifiziert werden, ist der Anteil der absichtlich oder unabsichtlich falsch benannten Objekte relativ hoch. Dies wird jedoch meist erst nach dem Datentransfer bemerkt und so verbrauchen nutzlose Übertragungen Bandbreite des Netzes, die anderen Ressourcen nicht mehr zur Verfügung steht. Des weiteren besteht für den User eine Gefahr, die sich durch gefährlich veränderte Ressourcen ergibt. So könnte die Sicherheit des eigenen Clients nicht mehr gewährleistet sein, wenn sich Viren, Würmer oder andere Programme mit schädlicher Wirkung über das P2P-Netz verbreiten.
Deshalb muss es Mechanismen zur Prüfung der Integrität und Authentizität der Daten und Dienste geben. Mit Hilfe von Signaturen kann man zum Beispiel herausfinden, ob das gerade übertragene Objekt mit dem übereinstimmt, welches der Halter der Ressource besitzt. Dies nützt jedoch nichts wenn bereits der Anbieter des Objekts die Daten manipuliert hat.
Aber wie findet man heraus dass ein Ergebnis einer Anfrage auch genau das Objekt ist, welches man sucht? In [DY03] werden dazu Bewertungsmöglichkeiten vorgestellt:

- Das **Expertenurteil**. Hierbei wird jede Ressource durch einen als Experten auftretenden und sich mit einer Signatur ausweisenden Peer beglaubigt und dieser Maßstab auf die Objekte einer Suchanfrage angewendet. Voraussetzung ist jedoch dass es genügend Experten im Netz gibt um alle Ressourcen zu bewerten und dass man diesen Experten vertrauen kann.
- Das **Votingschema**. Jeder Besitzer eine Datei gibt über deren Qualität eine Wertung ab. Diese Wertungen werden zusammengetragen und auf die Ergebnismenge abgebildet. Hat eine Datei eine hohe Anzahl positiver Wertungen, so kann sie als authentisch angenommen werden. Leider sind solche Votes leicht manipulierbar. So könnten sich Peers mit gemeinsamen Interessen zusammenschließen und durch ihre Anzahl die Bewertungen manipulieren, so dass zum Beispiel auch gefährliche Objekte gute Votings erhalten.
- Das **Reputationssystem**. Dies ist die Verknüpfung der beiden soeben vorgestellten Verfahren. Jeder Peer darf dabei eine beliebige Datei bewerten, diese Bewertung wird jedoch in Relation zu seiner Reputation im Netz gesetzt. Gibt zum Beispiel ein Peer eine falsche Bewertung über ein Objekt ab, so kann derjenige der dies entdeckt eine negative Bewertung über den Peer abgeben, was dessen Glaubwürdigkeit herabsetzt. User werden so motiviert, korrekte Angaben über

Ressourcen zu machen und eine Gruppe von Peers mit unmoralischen Absichten bekommt nicht mehr so viel Einfluss auf die Bewertung.

Alle diese Verfahren haben Probleme, die Authentizität zu garantieren. Eine Precision von 1 wird so auch in absehbarer Zeit nicht zu erreichen sein, jedoch wird bereits bei der Filterung weniger nicht authentischer Ergebnisse die Bandbreite des Netzes geschont und eine höhere Anfragequalität erreicht, weshalb vor allem die bandbreitenhungrigen aktuellen P2P-Netzwerke gut daran tun dies zu implementieren.

5. Beispielimplementation: OceanStore

5.1 Konzepte von OceanStore

Verschiedene der bisher vorgestellten Verfahren finden bereits Verwendung in „OceanStore“[1], dem von der Uni Berkeley entwickelten P2P-Netzwerk zur Verwaltung von Daten (weniger zum Tausch). Bei dessen Entwicklung wurde besonderen Wert auf ein garantiertes Level von Dienstgüte und Robustheit gelegt, wobei jedoch von einem unsicheren und sich schnell ändernden Netz ausgegangen wird. Ein flexibles Update- und Versionenmanagement mit Benutzerrechten ermöglicht auch Anwendungen mit verteilten Groupware-Lösungen, genaueres dazu findet sich in [KU00].

Die Grundannahme von OceanStore ist das „Untrusted Network”. Dabei wird von einem Netz ausgegangen, in dem sich sehr viele unterschiedliche Clients in schnellen Zeitabständen mit dem Netzwerk verbinden und sich auch wieder daraus entfernen. Aufgrund der unterschiedlichen Architekturen und der verschiedenen Software kann keine genaue Aussage über einen Peer getroffen werden, weshalb einem mit dem Netzwerk verbundenen Client grundsätzlich kein Vertrauen entgegengebracht wird. Dies äußert sich in den verwendeten Techniken wie Soft State (Abschnitt 5.2), Erasure Codes mit kryptografischen Verschlüsselung (Abschnitt 5.3), Nomadic Data oder Introspective Monitoring (beide Abschnitt 5.4).

Zum Routing wird eine auf dem Internet aufliegende Verbindungsschicht namens Tapestry (folgender Abschnitt) benutzt, die das Routen und Querying vollkommen von der unterliegenden Schicht kapselt und so erst Quality of Service ermöglicht.

5.2 Tapestry als Routing- und Lokalisationsschicht

Tapestry basiert auf dem PRR-Schema und benutzt deshalb GUIDs für die Kennzeichnung der Daten und Node-IDs für die Peers. Damit kapselt es das eigentliche Routing vollkommen von dem unterliegenden Internet; das Routing von Peer zu Peer übernimmt Tapestry, welches intern aber IP-Routing verwendet. Da die Datenübertragung ausschließlich über von Tapestry gefundenen Routen erfolgt ist OceanStore robuster als das Internet, denn es findet selbst dann noch Routen, wenn der direkte Weg im Internet abgeschnitten ist (siehe Abbildung 4).

Dabei erweitert Tapestry das PRR-Schema an einigen Stellen um die Anforderungen eines sich schnell ändernden Netzes zu erfüllen. So speichert nicht nur ein Knoten die Informationen über ein bestimmtes Objekt mit einer ähnlichen GUID, sondern jede Ressource-ID wird auf eine Menge von Knoten-IDs abgebildet welche einen gleichen Präfix und nur Unterscheidungen in den letzten Bits besitzen (Surrogate Routing, siehe [HK02]). Tapestry legt jeweils dynamisch fest, wie lang der gleiche Präfix sein muss um es einem Knoten zu erlauben, der Wurzelmenge (The Inner Ring) anzugehören. Fällt die Anzahl der Knoten in der Wurzelmenge unter einen bestimmten Wert, so wird die Längenanforderung des Präfixes gesenkt und neue Knoten werden in die Menge

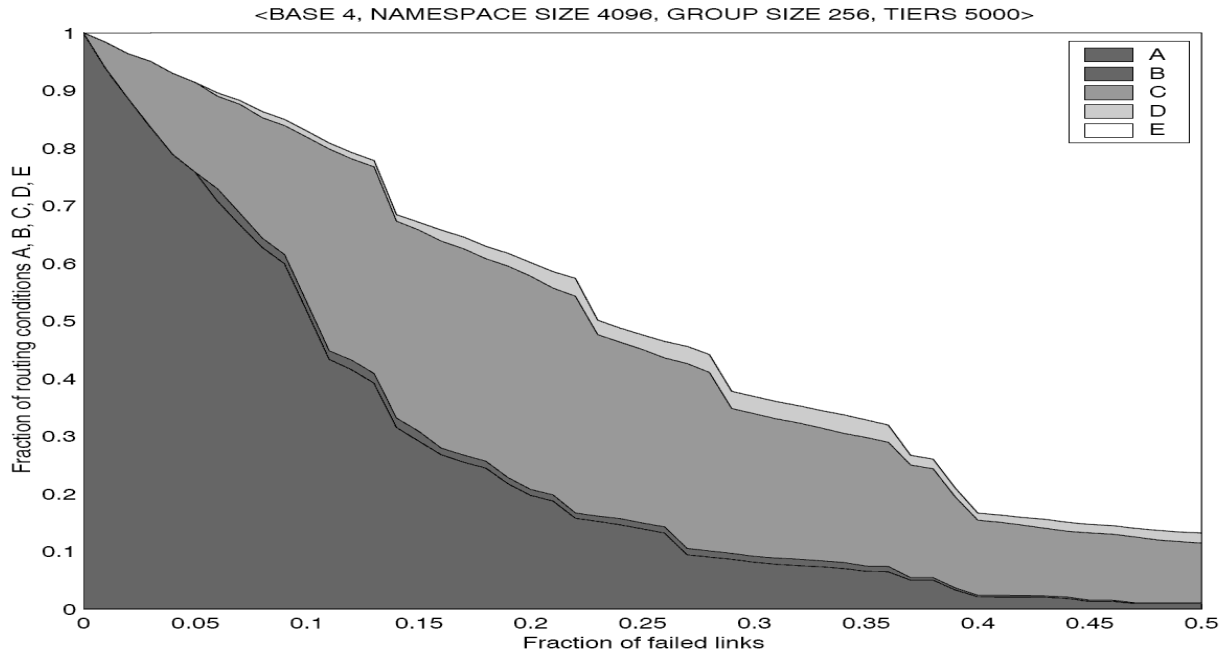

Abbildung 4 - Die Abbildung zeigt wie viele Pakete noch erfolgreich das Ziel erreichen wenn bis zu 50% aller Verbindungen im Internet nicht mehr verfügbar sind (aus [ZH01])

A – Tapestry und IP finden zum Ziel *B – nur IP findet zum Ziel*

C – nur Tapestry findet das Ziel *D – beide versagen obwohl das Ziel erreichbar ist*

E – das Ziel ist vollkommen abgeschnitten und nicht mehr erreichbar

aufgenommen. Diese tauschen untereinander die Pointer auf die ressourcenhaltenden Peers aus. So ist der Ausfall einer einzelnen Wurzel zu verschmerzen, da die Informationen nicht verloren gehen sondern weiterhin verfügbar sind und ein Denial-of-Service-Angriff auf eine Ressource von mehreren Knoten aufgefangen wird (insofern der Angriff überhaupt die Wurzelmenge erreicht).
Die Nachbartabellen aus dem PRR-Schema werden ebenfalls erweitert. So werden zu jedem Eintrag zwei Ersatzlinks eingetragen, die bei einem Ausfall des ersten Eintrags zum Einsatz kommen. Merkt ein Peer nämlich, dass der Nachbar nicht mehr reagiert (Tapestry registriert Time-Outs der IP-Pakete), so wird er diesen Eintrag kennzeichnen und fortan einen der Ersatzlinks verwenden. Nach einer bestimmten Zeitspanne schickt der Knoten jeweils eine Hello-Message an den nicht mehr reagierenden Nachbarn, und erst wenn dieser wiederholt nicht antwortet wird auch der Link komplett gestrichen und ein neuer Kandidat angefordert. Dazu wird der Knoten mit der „ähnlichsten“ ID zu der des gestrichenen Eintrags angeroutet und befragt. Dieses Verfahren nennt sich Second Chance und hat den Vorteil, dass bei kurzfristigen Ausfällen oder zu großer Belastung des Nachbarn nicht sofort wieder nach neuen Einträgen gesucht werden muss sondern dies erkannt wird und die Tabellen so relativ stabil bleiben.
Generell arbeitet aber Tapestry nach dem Soft-State-Prinzip, welches auf die Anforderungen eines sich schnell ändernden verteilten Netzes abgestimmt ist. Dabei wird jede Information, welche im Netzwerk vorhanden ist, nur als temporär gültig angesehen. Nach einer bestimmten Zeitspanne muss diese Information erneuert werden, ansonsten wird sie automatisch gelöscht. Alle Knoten schicken deshalb nach der festgelegten Zeitspanne Informationen über sich an die Knoten aus der Nachbartabelle und die datenhaltenden Peers schicken die Verweise auf die eigenen Ressourcen den Baum hinauf bis zur Wurzelmenge (Re-Publishing). Findet eine Query heraus dass ein auf einem Peer gespeicherter Verweis nicht mehr aktuell ist (Knoten kann nicht angeroutet werden), so schickt er eine Information darüber an die Wurzelmenge, die daraufhin alle ressourcenhaltenden Knoten auffordert ihre Verweise neu zu publizieren. Auf den Routen zu den Ressourcen werden dabei vorher alle alten Verweise gelöscht. Ist diese Prozedur beendet wird die Anfrage, die den toten Verweis entdeckte, neu gestartet.
Zum Eintritt in das Netz braucht ein Peer nur die Adresse eines einzigen aktiven Knotens zu wissen. Nach der Erstellung der persönlichen Knoten-ID versucht er dann über den aktiven Knoten sich selber im Netz zu finden. Dabei findet er jedoch nur einen Nachbarn mit großer Präfix-Übereinstimmung. Von diesem übernimmt er die Nachbartabelle und stimmt sie mit den neuen Nachbarn ab, so dass diese die ID des neuen Knotens ebenfalls übernehmen falls der Abstand dazu kürzer ist als zu den bisherigen Einträgen. Gehört der neue Knoten zu einer bestehenden

Knotenmenge, so informieren ihn die Nachbarn aus dieser Menge darüber und er kopiert sich die Informationen über die Ressourcenverweise von ihnen.
Scheidet ein Peer koordiniert aus dem Netz aus, so informiert er vorher die Einträge aus seiner Nachbartabelle darüber und bietet Ersatzeinträge aus seiner eigenen Tabelle an. Die Nachbarn löschen dann den Peer aus ihren Tabellen und versuchen mit dessen Informationen die Lücken in den Tabellen wieder aufzufüllen. Ansonsten können auch die noch vorhandenen Nachbarn nach deren Tabelleninhalten befragt werden. Erfolgt das Ausscheiden jedoch ohne Ankündigung, kommt wieder das Ersatzeintrag-Schema zum Einsatz.
Bei den Anfragen verwendet Tapestry ein zweistufiges System. Als erstes wird an alle nahen Nachbarn eine Anfrage nach dem Prinzip des Flutens geschickt (mit geringer Time-to-Live). Erst danach wird das deterministische Routing angewendet. Im Durchschnitt werden so die nahen Verweise als erstes aufgespürt, so dass die geringe Zeit und die zusätzlichen Messages nicht weiter ins Gewicht fallen, da das etwas länger dauernde (aber mit log(N)-begrenzte) deterministische Routing gespart wird und kurze Übertragungswege als erstes entdeckt werden.
Tapestry bietet also als Routing- und Lokalisationsschicht über dem Internet eine viel höhere Dienstgüte: Knoten werden nach dem erweiterten PRR-Schema deterministisch mit nach oben begrenzter Zeit gefunden (Recall = 1), bei den Anfragen werden die nahen Objekte als erstes lokalisiert. Weiterhin erfolgt das Einfügen und Entfernen von Peers vollautomatisch (skaliert im log(N)-Bereich und nicht linear [ZH03]) und veraltete Informationen im Netz werden ebenfalls erkannt und aktualisiert. Zusammen mit der Einführung der Wurzelmenge wurde die Robustheit des Netzwerks also auch bei sich schnell verändernden Netzen gegenüber dem PRR-Schema erhöht.
Dafür dauert das Publizieren neuer Informationen im Netz ungefähr 3mal so lange wie in zentralisierten Netzwerk-Datensystemen wie NFS [RH03] und das Routen ist durchschnittlich um den Faktor 2 langsamer als IP-Routing, was jedoch die schnellen und qualitativ hochwertigeren Suchergebnisse ausgleichen [RRK03]. Dadurch und dank der Replikationstechnik, welche fast ausschließlich örtlich nahe Transfers bewirkt, wird auch Bandbreite des Netzes gespart, die auf der anderen Seite jedoch für die eine hohe Kommunikation erzeugende Fehlertoleranz wieder verbraucht wird.
Allerdings ist OceanStore durch die Verwendung von Hashwerten der Ressourcen als GUIDs nicht geeignet, um unbekannte Objekte im Netzwerk zu finden. Zwar ist es möglich, über eine auf dem System aufliegende Applikation nur die Namen der Ressourcen zu hashen, doch OceanStore selber sieht dies nicht vor, so dass die Informationen über die Objekte andersweitig publiziert werden müssen.

5.3 Data Replication in OceanStore

Zur Erhöhung der Dauerhaftigkeit der gespeicherten Daten verwendet OceanStore beide der im Kapitel 4.1 beschriebenen Verfahren. Zum einen werden durch das im nächsten Abschnitt beschriebene Konzept des Introspective Monitoring die Daten repliziert (Secondary Replicas) und im Netz dorthin verteilt, wo sie benötigt werden (Redundanz in Verbindung mit Nomadic Data), zum anderen wird aber auch eine Form der Erasure Codes genutzt um Daten wieder zu rekonstruieren (Primary Deep Archival Replicas). Hierauf basiert auch das Update- und Versionenmanagement, welches in [KU00] genau beschrieben wird.
Dabei wird jede zu archivierende Datei von der an der Datenhaltung interessierten Partei in Blöcke zerlegt, die dann in n kryptographisch (mit Privatschlüsseln) geschützte Fragmente zerlegt werden. Von diesen n Fragmenten werden später zur Rekonstruktion nur beliebige m ($m < n$) benötigt. Diese n Fragmente werden zusammen mit ihren Hashwerten zufällig über das Netzwerk verteilt. Um eine dauerhafte Speicherung zu garantieren müssen die Daten regelmäßig überprüft werden. Dieser „Sweep“ genannte Vorgang wird in Abständen von einem halben Jahr automatisch initialisiert und kontrolliert mit Hilfe der Hashwerte die Datenintegrität. Fällt das Verhältnis von m zu n unter einen bestimmten Wert, muss die datenhaltende Partei die Fragmente wieder einsammeln, die Datei zusammensetzen und den Vorgang erneut starten – aufgrund der damit verbundenen Rechenlast erledigt OceanStore dies nicht automatisch.
Für einem erhöhtem Rechenaufwand durch die Codierung bietet OceanStore somit als Ausgleich eine garantierte Dauerhaftigkeit der zu schützenden Daten, die auch nicht durch fremde Parteien manipuliert werden kann.

5.4 Introspective Monitoring

„Introspective Monitoring with Adaption” [KU00] haben die Entwickler von OceanStore ihre Form des Load Balancings getauft. Dabei wird jede Aktion des Netzes mit Eventhandlern in lokalen Datenbanken auf jedem Peer einzeln gespeichert. Diese Meta-Informationen des Netzes werden dann sowohl lokal genutzt als auch regelmäßig in temporären Baum-Hierarchien untereinander getauscht.
Eine erhöhte Nachfrage nach einer bestimmten Ressource mit verbundenem Verlust an Dienstgüte (Performanz und Verfügbarkeit) veranlasst das Netz dann dazu, eine lokale Kopie bei der

Wurzelmenge anzufordern. Wenn eine Anfrage eine lange Strecke bis zu dem angeforderten Objekt zurücklegen muss oder der Sender schlecht verfügbar ist, können Vorschläge für eine neue Kopie gemacht werden. Für lange Zeit ungenutzte Ressourcen werden automatisch gelöscht [RH01]. Die Mechanismen in OceanStore sind sogar in der Lage, zeitabhängige Datennutzungen an bestimmten Punkten des Netzes zu erkennen und das Kopienmanagement danach auszurichten – so können aktuelle Daten über den Tag auf dem Arbeitsplatzrechner verfügbar sein und am Abend auf dem Privatrechner.
Des weiteren wird aus der Datenbank ein Graph der näheren Umgebung erstellt und bei erkennender Clusterbildung versucht, diesen durch Änderungen an den Nachbartabellen der betroffenen Peers aufzulösen [KU00]. So werden von vornherein Schwachstellen in der Topologie vermieden und die Robustheit erhöht.

5.5 Fazit

OceanStore kann im Vergleich mit aktuellen P2P-Netzwerken die Robustheit erhöhen und vor allem Quality of Service garantieren. So werden Peers und Objekte, insofern sie innerhalb des Netzes existieren, fast deterministisch gefunden. Außerdem reagiert das Netz nicht so empfindlich auf Ausfälle von ganzen Teilnetzen oder einer großen Anzahl von Peers. Selbst wenn die Teilnehmer über IP-Routing nicht mehr erreichbar sind besteht noch eine Chance, dass Tapestry, die Routing-Schicht von OceanStore, sie findet. Durch das ständige Anpassen an die aktuelle Topologie erhöht das Netz automatisch den Durchsatz und die verfügbare Bandbreite und kann auf verstärkte lokale Anfragen an Ressourcen reagieren, so dass die Performanz und Verfügbarkeit erhalten bleibt.
Für die Daten kann OceanStore Dauerhaftigkeit garantieren und über Erasure Codes auch die Integrität. Diese Vorteile werden mit einem erweiterten Kommunikationsaufwand erreicht, der jedoch mit log(N) sehr gut skaliert und durch die anderern Einsparungen wie die effektive Suche wieder ausgeglichen wird.

6. Zusammenfassung

Im Gegensatz zu bekannten Lösungen für Quality of Service muss in einem P2P-Netz davon ausgegangen werden, dass die Netzstruktur uneinheitlich und schnell veränderlich ist. Dennoch kann mit Verfahren wie der Data Replication, den Nomadic Data, Trading und einem intelligenten Routing ein global stabiler Zustand erreicht werden. Diese Robustheit ist die Grundvoraussetzung für garantierte Service-Qualitäten wie das Auffinden jeder Ressource und hohe Anfrage- und Übertragungsperformance. Zur Bereitstellung dieser Merkmale wird aber ein höherer Aufwand an Kommunikation und Speicherplatz verlangt. So stabil wie die Telefonleitung wird ein P2P-Netz zwar niemals werden, aber schon gegenüber dem Internet ergeben sich Vorteile die erklären, warum immer mehr Daten in verteilten Netzen gespeichert werden und nicht auf einzelnen Servern.

7. Quellenverzeichnis:

[1] Oceanstore - http://oceanstore.cs.berkeley.edu/

[CGM01] Brian Cooper, Hector Garcia-Molina, “Creating Trading Networks of Digital Archives”, Proc. of the First ACM+IEEE Joint Conference on Digital Libraries, 2001

[CGM02] Brian Cooper, Hector Garcia-Molina, “Peer-to-peer resource trading in a reliable distributed system”, International Workshop on Peer-to-Peer Systems 2002

[CS02] Edith Cohen, Scott Shenker, “Replication Strategies in Unstructured Peer-to-Peer Networks”, The ACM Conference of the Special Interest Group on Data Communication (SIGCOMM), August 2002

[DY03] Neil Daswani, Beverly Yang, Hector Garcia-Molina, “Open Problems in Data-Sharing Peer-to-Peer Systems”, The 9th International Conference on Database Theory (ICDT), 2003

[HK02] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, Ben Y. Zhao, “Distributed Object Location in a Dynamic Network”, Proceedings of the 14. ACM Symposium on Parallel Algorithms and Architectures (SPAA), August 2002

[KU00] J. Kubiatowicz et al., “OceanStore: An Architecture for Global-Scale Persistent Storage”, Proc. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX), ACM Press, 2000.

[KU03] John Kubiatowicz, “Extracting Guarantees from Chaos”, Communications of the ACM February 2003/Vol. 46, No. 2

[KRT02] Jussi Kangasharju, Keith W. Ross, David A. Turner, “Optimal Content Replication in P2P Communities”, Manuskript 2002 http://www.tk.informatik.tu-darmstadt.de/Forschung/publikationen/abstracts.php3?lang=;paperID=193

[LDA03] Christoph Loeser, Michael Ditze, Peter Altenbernd, ”Architecture of an intelligent Quality-of-Service aware Peer-to-Peer Multimedia Network”, Proc. of the 7th World of Multiconference on Systemics, Cybernetics and Informatics (SCI-2003), 2003

[OSS03] Giwon On, Jens Schmitt, Ralf Steinmetz, “QoS-Controlled Dynamic Replication in Peer-to-Peer Systems”, Praxis der Informationsverarbeitung und Kommunikation, April 2003

[PRR97] Greg Plaxton, Rajmohan Rajaraman, Andréa Richa, “Accessing Nearby Copies of Replicated Objects in a Distributed Environment”, Proc. of ACM Symposium on Parallel Algorithms and Architectures (SPAA), June 1997

[RH01] Sean Rhea, Chris Wells, Patrick Eaton, Dennis Geels, Ben Zhao, Hakim Weatherspoon, John Kubiatowicz, “Maintenance-Free Global Data Storage“, IEEE Internet Computing, September / October 2001

[RH03] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, John Kubiatowicz, “Pond: the OceanStore Prototype“, Proceedings of the 2nd USENIX Conference on File and Storage Technologies 2003

[RRK03] Sean C. Rhea, Timothy Roscoe, John Kubiatowicz, “Structured Peer-to-Peer Overlays need Application-driven Benchmarks”, Proceedings of 2nd International Workshop on Peer-to-Peer Systems 2003

[SUV03] Lingling Sun, Yamini Upadrashta, Julita Vassileva, “Ensuring Quality of Service in P2P File Sharing through User and Relationship Modelling”, Workshop on User and Group models for web-based adaptive collaborative environments, June 2003

[WK02] Hakim Weatherspoon, John D. Kubiatowicz, “Erasure Coding vs. Replication: A Quantitative Comparison“, Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002

[ZH01] Ben Yanbin Zhao, “A Decentralized Location and Routing Infrastructure for Fault-tolerant Wide-area Network Applications“, Ph.D. Qualifying Examination, University of California, Berkeley, 2001
http://www.cs.berkeley.edu/~ravenben/tapestry/html/papers.html

[ZH03] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, John D. Kubiatowicz, “Tapestry: A Resilient Global-scale Overlay for Service Deployment“, IEEE Journal on Selected Areas in Communications 2003