eXamen.press

eXamen.press ist eine Reihe, die Theorie und Praxis aus allen Bereichen der Informatik für die Hochschulausbildung vermittelt.

Peter Mahlmann · Christian Schindelhauer

Peer-to-Peer-Netzwerke

Algorithmen und Methoden

Mit 130 Abbildungen

Springer

Peter Mahlmann
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn
mahlmann@upb.de

Christian Schindelhauer
Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 51
79110 Freiburg
schindel@informatik.uni-freiburg.de

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISSN 1614-5216
ISBN 978-3-540-33991-5 Springer Berlin Heidelberg New York

Springer ist ein Unternehmen von Springer Science+Business Media
springer.de

Satz: Druckfertige Daten der Autoren
Herstellung: LE-TEX, Jelonek, Schmidt & Vöckler GbR, Leipzig
Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg
Gedruckt auf säurefreiem Papier 33/3180 YL – 5 4 3 2 1 0

Für Tina, Tom und Lena

Vorwort

Grundlage dieses Buches sind die Vorlesungen „Algorithmen für Peer-to-Peer-Netzwerke“ aus dem Jahr 2004 an der Universität Paderborn und „Peer-to-Peer-Netzwerke“ an der Albert-Ludwigs-Universität Freiburg. Beide Vorlesungen wurden durch Publikationen einer Reihe namhafter Wissenschaftler aus dem Bereich der Algorithmen inspiriert. Als das Buch konzipiert wurde, war der Begriff der Peer-to-Peer-Netzwerke erst fünf Jahre alt.

In dieser kurzen Zeit hatten sich die Peer-to-Peer-Netzwerke schon grundlegend gewandelt. Der 19-jährige Shawn „Napster“ Fanning entwarf ein einfaches Netzwerk namens Napster, das den Zugriff auf Dateien anderer Teilnehmer direkt über das Internet zuließ. Dieser Zugriff zwischen gleichen Teilnehmern wird Peer-to-Peer genannt. Das Zugreifen auf Dateien über das Internet war schon lange zuvor mit dem *File Transfer Protocol* (FTP) möglich. Nur musste man hierfür die Dateien auf einem FTP-Server ablegen, bevor sie jemand anders dort wieder abholen konnte. Napster machte diesen Umweg überflüssig, so dass die Dateien direkt von *Client* zu *Client* — also von *Peer* zu *Peer* — verteilt werden werden können.

Napster bestand aus einer *Client-Server*-Komponente und einer *Peer-to-Peer*-Komponente. Wenig später folgte mit Gnutella das erste echte Peer-to-Peer-Netzwerk, das vollständig ohne zentrale Steuerungsmechanismen arbeitet. Aber Gnutella ist langsam und unzuverlässig.

Die Ineffizienz von Gnutella motivierte viele Informatiker, bessere Netzwerke zu entwerfen. 2001 gab es mit CAN und Chord die ersten Peer-to-Peer-Netzwerke mit effizienter Suche und bald sollten weitere folgen. Während sich Wissenschaftler mit immer besseren Netzwerkstrukturen übertrumpften, veröffentlichten Programmierer immer bessere Software, die ihre Ideen umsetzen. Die Schattenseite dieser Entwicklung ist, dass die meisten Peer-to-Peer-Netzwerke von den Benutzern dazu missbraucht werden, massenhaft das Urheberrecht zu verletzen: Nutzer tauschen uneingeschränkt Musik und Filme aus. Schätzungen zufolge sollen mittlerweile mehr als 50% des gesamten Transportvolumens im Internet diesem Zweck dienen. Dadurch entstand ein bis heute ungelöstes Spannungsfeld zwischen massenhafter illegaler Nutzung von Peer-to-Peer-Netzwerken und den Leidtragenden dieser Entwicklung, den Künstlern und deren Vertretern in Musik- und Filmindustrie. Diese

gesellschaftlichen Probleme wurden andernorts schon ausführlich und fachkundig besprochen.

Ziel dieses Buches ist es, dem Leser ein grundlegendes Verständnis der Techniken hinter den aktuellen Peer-to-Peer-Netzwerken aufzuzeigen und Algorithmen zu vermitteln, die vielleicht erst in einigen Jahren umgesetzt werden. Das Buch richtet sich in erster Linie an Studenten der Informatik ab dem fünften Semester. Daher werden allgemeine Kenntnisse der Grundlagen der Informatik und Mathematik vorausgesetzt. Aber auch der interessierte Nichtinfomatiker soll von diesem Buch profitieren können. Ohne Kenntnisse aus dem Bereich der Mathematik und Informatik wird sich diesem Leser das Buch wohl nicht ganz erschließen. Jedoch sollten die Ziele, Kernaussagen und Ergebnisse jedem Leser auch ohne akademischen Hintergrund klar werden.

Dieses Buch beschäftigt sich allgemeiner mit Algorithmen und Methoden und will einen Einblick in die aktuelle Forschung geben. Dieses Forschungsgebiet ist immer noch sehr lebhaft, und daher kann in diesem Buch nur eine kleine Auswahl des Zwischenstandes dargestellt werden, die naturgemäß den Interessen der Autoren folgt.

Konkrete Programmbeispiele oder exakte Protokollspezifikationen kann man in diesem Buch nicht finden, dafür aber viele Ideen, Analysen, mathematische Werkzeuge und verteilte Algorithmen. Wir haben versucht, diese so verständlich wie möglich darzustellen, so dass der erfahrene Softwaredesigner hieraus eine Fülle von Anregungen und Techniken aus der aktuellen Forschung gewinnen kann.

Danksagung

Wir möchten uns bei unseren Familien für die aufgebrachte Geduld und Unterstützung bedanken. Des Weiteren bedanken wir uns bei allen Buchlektoren, insbesondere bei Kerstin Pfeiffer, Barbara Schindelhauer, Thomas Janson, Arne Vater und Mario Vodisek. Dank gebührt auch einer Reihe von Studenten, mit denen in Vorlesungen, Seminaren und Diplomarbeiten anregende Diskussionen geführt wurden. Stellvertretend seien hier Peter Bleckmann, Christian Ortolf, Markus Scherschanski und Markus Werner genannt.

Nicht unerwähnt bleiben sollen auch die vielen wissenschaftlichen Diskussionen, die in diesem Zusammenhang mit Kollegen aus Paderborn und dem Rest der Welt geführt wurden: James Aspnes, Pierre Fraigniaud, Miroslaw Korzeniowski, Dahlia Mahlki, Rajmohan Rajamaran, Christian Scheideler, Gunnar Schomaker, Scott Shenker, Richard Karp, Shay Kutten, Stefan Rührup, Eli Upfal, Arne Vater, Klaus Volbert, Berthold Vöcking, Roger Wattenhofer und Gerhard Weikum. Wir bedanken uns bei Andrew Parker von CacheLogic für die Erlaubnis, zwei Bilder verwenden zu dürfen.

Besonderer Dank gebührt Thomas Erlebach, der es ermöglicht hat, eine Woche in Bertinoro, Italien, an diesem Buch arbeiten zu können, sowie Friedhelm Meyer auf der Heide, der die Arbeit an diesem Buch mit unterstützt hat.

Waldkirch im Breisgau, *Christian Schindelhauer*

Löhne in Westfalen, *Peter Mahlmann*

Inhaltsverzeichnis

1

Ein kurzer Überblick

Peer, du lyver!
Henrik Ibsen (Peer Gynt)

1.1 Peer-to-Peer-Netzwerke im Jahr 2006

Es gibt sie noch, die Leute, die noch nie von Peer-to-Peer-Netzwerken gehört haben. Andere kennen Peer-to-Peer-Netzwerke nur aus der Zeitung als eine Schlangengrube von Raubkopierern, die illegal Musikstücke über das Internet vertreiben: Bisweilen wird dieses oder jenes Peer-to-Peer-Netzwerk stillgelegt, und so genannte Raubkopierer erhalten Besuch von der Polizei oder Post von der Staatsanwaltschaft. Und dann ist da noch die Gruppe der Peer-to-Peer-Netzwerk-Benutzer, die meist verschämt die Augen niederschlagen, wenn sie darauf angesprochen werden. Sie sind ausgestattet mit einem gehörigen Maß an Paranoia, da sie fürchten (oder wissen) irgendetwas Unrechtes getan zu haben und möglicherweise dafür belangt zu werden. Andererseits haben sie immer das neueste Video und das neueste Musikstück. Typischerweise legen sie größten Wert auf einen schnellen Netzwerkzugang und besitzen (für andere Zwecke) unangemessen große Festplattenkapazitäten.

Tatsächlich verwenden immer mehr Personen Peer-to-Peer-Netzwerke, die oft auch als „P2P Networks“ bezeichnet werden. Im Juni 2005 waren nach Angaben der Firma *Bigchampagne*, die Hitparaden aufgrund der Downloads in Peer-to-Peer-Netzwerken erstellt, rund neun Millionen Menschen gleichzeitig Teilnehmer an Peer-to-Peer-Netzwerken. Insgesamt sollen bereits 35 Millionen Europäer schon einmal ein Peer-to-Peer-Netzwerk verwendet haben. Nach Angaben der Firma *CacheLogic* wird mehr als 50% der durch das Internet beförderten Datenmenge durch Peer-to-Peer-Netzwerke verursacht, siehe Abbildung 1.1 und 1.2. Während früher der Großteil dieser Daten Musikdateien waren, die komprimiert im *MP3*-Format von Teilnehmern getauscht wurden, sind es in letzter Zeit gleich ganze Kinofilme, die übertragen werden. Während MP3-Dateien von Musikstücken in der Regel die Größe von einigen Megabytes besitzen, schlagen Kinofilme, die zumeist von DVDs kopiert werden, mit einigen Gigabytes zu Buche. Dadurch lässt sich die Datenmenge leicht er-

klären. Insbesondere lassen sich bestimmte Ereignisse in Peer-to-Peer-Netzwerken im Gesamtdatenvolumen des Internets ablesen, wie zum Beispiel die ungenehmigte Veröffentlichung einer Arbeitskopie des Films „Star Wars III“ in verschiedenen Peer-to-Peer-Netzwerken.

In der Öffentlichkeit hat sich der Begriff *Tauschbörse* für Peer-to-Peer-Netzwerke eingebürgert. Dieser Begriff ist jedoch in zweierlei Hinsicht irreführend. Mit Tausch oder Börse haben diese Übertragungen wenig gemein. Die meisten in solchen Netzwerken angebotenen Daten sind Verstöße gegen international geltende Urheberschutzrechte, die mittlerweile auch systematisch geahndet werden. Fast alle Netzwerke beruhen auf der kostenfreien Bereitstellung von Diensten aller Teilnehmer. Für das Herunterladen einer Datei entstehen somit nur die Verbindungskosten jedes Teilnehmers zum Internet.

Die Motive der Nutzer, die die Kosten für den Kauf oder das Leihen aktueller Musikstücke, Filme oder Software scheuen, kann man nachvollziehen. Jedoch erscheinen die Motive der Teilnehmer, die diese Formate zumeist unentgeltlich zur Verfügung stellen, rätselhaft. Vielleicht sind es Neugier und Geltungssucht. Vielfach aber auch sind es verabredete Übertragungen unter Freunden oder Internet-Bekanntschaften.

Man darf hierbei nicht übersehen, dass es auch legale Anwendungen gibt. Hierunter fällt zum Beispiel Software, die unter der *GNU General Public Licence (GPL)* [1] veröffentlicht worden ist. Daneben werden auch Filme, Musikstücke und Bücher zu Werbe- oder Informationszwecken kostenfrei zur Verfügung gestellt. Kleine unabhängige Musikgruppen nutzen dies zum Beispiel, um ihren Bekanntheitsgrad zu steigern. Manche Software-Unternehmen verteilen durch Peer-to-Peer-Netzwerke ihre Software-Aktualisierungen. Die Firma SUN-Microsystems hat für solche legalen Anwendungen die Open Source-Plattform *JXTA* zur Verfügung gestellt. Mittlerweile gibt es eine Vielzahl von Unternehmen [2], die auf dieser Plattform legale Anwendungen von Peer-to-Peer-Netzwerken benutzen. Eine weitverbreitete, ebenfalls legale Anwendung von Peer-to-Peer-Netzwerken ist das Internet-Telefon. So verwendet *Skype* [3] ein Peer-to-Peer-Netzwerk (siehe auch Seite 248). Mit dieser VoIP-Software (*Voice over Internet Protocol*) sind Ende 2006 durchgehend mehr als fünf Millionen Benutzer gleichzeitig erreichbar. Besonders wichtig ist aber auch die Verwendung von Peer-to-Peer-Netzwerken zur unzensierten und anonymen Verbreitung von Nachrichten und Informationen. Gerade Benutzer in Diktaturen können dadurch ihr Grundrecht auf freie Meinungsäußerung ungestraft wahrnehmen. Hierfür wurde unter anderen *Freenet* [4] entwickelt. Dieses Peer-to-Peer-Netzwerk ermöglicht es den Teilnehmern Dateien zu veröffentlichen, ohne dabei die Identität und den Ort des Autors preiszugeben. Die Autoren von Freenet wollten damit den Bürgern in Diktaturen ein sicheres und freies Podium zur Verfügung stellen. In der Praxis zeigt sich aber, dass die meisten Benutzer von Freenet ganz andere Ziele verfolgen als die Durchsetzung von Menschenrechten. Aus einer Untersuchung aus dem Jahr 2000 [5] geht hervor, dass von den Textdateien fast 60% das Thema Drogen behandeln; von den Grafikdateien und Videodateien waren über 75% pornografischen Inhalts und der Großteil der Audiodateien war überwiegend von Rockbands, die wohl kaum mit der kostenfreien Publikation in Freenet einverstanden waren.

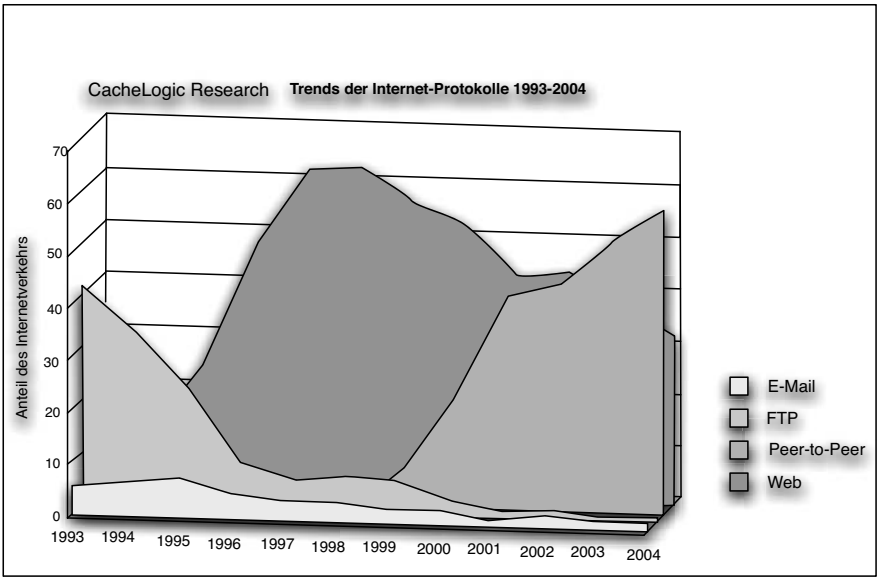


Abb. 1.1. Entwicklung des Datenverkehrs im Internet von 1993 bis 2004 nach der Darstellung der Firma CacheLogic [6].

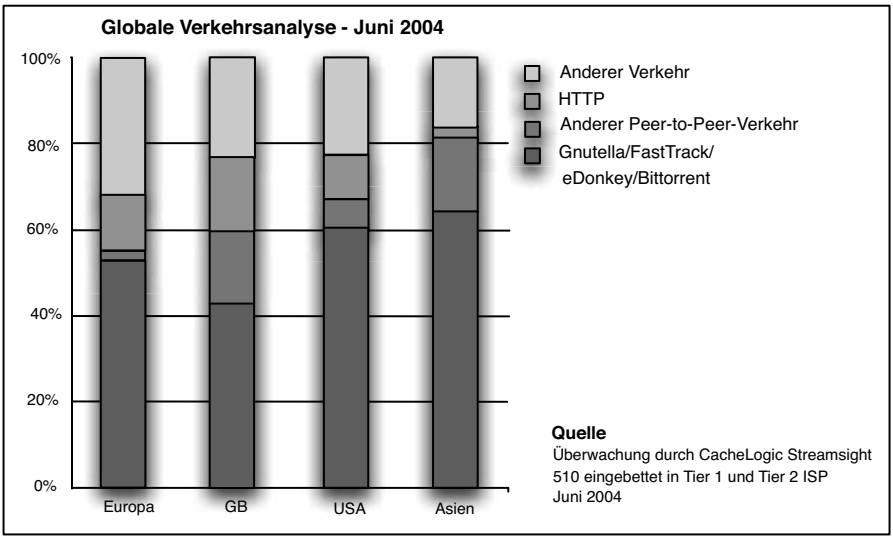


Abb. 1.2. Verteilung des Datenverkehrs im Internet Ende 2004 nach der Darstellung der Firma CacheLogic [6].

Vielen Autoren, Künstlern, Programmierern und Filmschaffenden sowie ihren Verlegern ist dieses illegale Kopieren naturgemäß ein Dorn im Auge. Mit verschiedenen Methoden bekämpfen sie dieses Phänomen. Am sichtbarsten sind Imagekampagnen, in denen ein illegaler *Download* in die Nähe von Schwerverbrechen gerückt wird, was man schon an der irreführenden Bezeichnung *Raubkopierer* sieht: Raub bezeichnet (sonst) einen gewaltsamen Diebstahl. Propagandafilmchen zeigen Raubkopierer, die mehrjährige Haftstrafen abbüßen. Darüber hinaus bemühen diese die Gerichte, die mittlerweile unter der Last der Klagen zu leiden haben, um ihre Rechte durchzusetzen. So gab es im Jahr 2005 Tausende von Klagen gegen Einzelnutzer. Hierbei hat sich auch ein lukratives Geschäftsfeld zur Verfolgung von Urheberschutzverletzungen entwickelt. So gibt es Unternehmen, wie beispielsweise die Logistep AG, die an Peer-to-Peer-Netzwerken teilnehmen, um dann zum Beispiel über die Internet-Adressen (IP-Adresse, siehe Seite 17) der Nutzer Klagen gegen diese anzustrengen.

In der Tat ist die *rechtliche Situation* für den Laien ziemlich unübersichtlich. Ein Grund ist, dass es das Internet möglich macht, global Daten zu übertragen, aber die Gesetzgebung einzelner Länder äußerst unterschiedlich ist. So kann zum Beispiel schon der Verkauf einer CD über eBay in der Originalverpackung in Deutschland eine Urheberrechtsverletzung darstellen. Dagegen wird seit Jahr und Tag in Schweden ohne jegliche juristische Konsequenzen eine zentrale Anlaufstelle für die Vermittlung von illegalen Dateiübertragungen im *Bittorrent*-Protokoll unterhalten (`thepiratebay.org`). Des Weiteren ist in einigen Ländern die Verwendung bestimmter kryptographischer Protokolle strafbar. Hierunter fallen auch Programme, die untrennbarer Bestandteil heutiger Rechner sind. Das betrifft nicht nur Länder wie China und Iran, sondern auch demokratische Staaten, wie z.B. Frankreich.

Ein weiterer Grund ist die Skrupellosigkeit der Vertreiber von Peer-to-Peer-Netzwerk-Software und Suchmaschinen für in Peer-to-Peer-Netzwerken indizierte Dateien. Diese klären die Benutzer über die juristischen Konsequenzen ihres Handelns nicht auf. Vielmehr versuchen sie, aus der massenhaften Nachfrage nach solcher Software eigene kostenpflichtige Plattformen zu bewerben.

Trotz dieser Gründe hat es sich sicher bei den meisten Benutzern von Peer-to-Peer-Netzwerken herumgesprochen, dass nicht alles, was technisch möglich ist, auch legal ist.

1.2 Peer-to-Peer-Netzwerke seit 1999

Schon das erste so genannte Peer-to-Peer-Netzwerk verdankte seinen Aufstieg und Fall diesem Spannungsfeld. *Shawn „Napster“ Fanning* (Jahrgang 1980) brachte im Juni 1999 eine Beta-Version seines mittlerweile legendären Peer-to-Peer-Netzwerks *Napster* heraus. Die ursprüngliche Aufgabe dieser Software war die Bereitstellung eines File-Sharing-Systems, mit dem man Dateien auf Rechnern, die sich an diesem System anmelden, lokalisieren und dann direkt von diesen Rechnern herunterladen kann. Erst durch das Engagement eines Freundes von Shawn Fanning wurde aus dem File-Sharing-System ein Musik-Portal, das hauptsächlich zur Verbreitung von

Musikdateien diente. Dadurch erhielt Napster schon im Herbst 1999 den Titel des Download des Jahres.

Wegen der Missachtung von Urheberschutzrechten der Musikautoren klagten einige Musikverlage gegen Fanning und drohten mit der Stilllegung des Systems. Tatsächlich geschah einige Zeit nichts, und durch die durch den Rechtsfall gesteigerte Publizität nahm die Anzahl der Teilnehmer sogar zu. In einer überraschenden Wendung ging Fanning Ende 2000 einen Kooperationsvertrag mit Bertelsmann eCommerce ein. Nach einigen weiteren juristischen Wirren wandelte er sein bislang kostenloses System schließlich in eine kommerzielle File-Sharing-Plattform um.

Bevor *Napster* vom Netz ging, zeigte sich schon, dass die zentrale Struktur von Napster nicht geeignet ist die eingehenden Anfragen zu bearbeiten. Aber die Tatsache, dass Millionen von Benutzern von einem Rechner abhängig waren, der dann (aus juristischen Gründen) tatsächlich ausfiel, motivierte den Gegenentwurf eines völlig dezentralen Netzwerks.

Gnutella [7] ist das erste Peer-to-Peer-Netzwerk, das diesen Namen auch verdient. Während in Napster ein zentraler Server die Indexdateien der Daten verwaltet hat, haben die Entwickler von Gnutella diese Aufgabe auf alle teilnehmenden Rechner verteilt. Jeder Rechner hat sowohl Daten als auch die Indexinformationen. Die Verbindungsstruktur, aber auch die Verteilung der Daten oder Indexinformationen sind vollkommen unorganisiert. Gnutella war seit seinem Start im Jahr 2000 praktisch unangreifbar. Denn solange nur ein Rechner aktiv teilnimmt, existiert das Netzwerk. Wird ein Rechner herausgenommen, so betrifft dies nur die Daten und Informationen, die dieser Rechner zur Verfügung stellte. Alle Operationen wie Anmelden, Suche und Bereitstellung geschehen nämlich ohne zentrale Koordination. Jeder Teilnehmer ist ersetzbar. Die Schattenseite ist das völlige Chaos in der Netzwerk- und Datenstruktur. Die Suche nach einem bestimmten Datum ist langwierig, ressourcenaufwändig und unzuverlässig. Nur Daten, die häufig vorkommen, werden schnell gefunden. Viele Nutzer stört das wenig, da zumeist die Musikhits sehr gut und schnell verfügbar sind. So wuchs die Gnutella-Gemeinde ebenso schnell wie zuvor diejenige von Napster. Dieses Massenphänomen, das zudem auch noch während des Internet-Booms stattfand, faszinierte viele Informatiker. Das hängt auch damit zusammen, dass ein so einfaches Protokoll (siehe Seite 57) eine so komplexe Netzwerkstruktur aufbaut (siehe Seite 174).

So starteten viele ihr eigenes verbessertes Peer-to-Peer-Netzwerk und im Jahr 2000 kam das ebenfalls populäre Netzwerk *eDonkey* heraus. Im Jahr 2001 folgte *FastTrack*, das vor allem durch die Client-Software *Morpheus*, *Kazaa* und *Grokster* bekannt wurde. Ebenfalls 2001 wurde eine veränderte Version von Gnutella (hier Gnutella-2 genannt) veröffentlicht, sowie die Open Source Peer-to-Peer-Netzwerk-Plattform *JXTA* von SUN Microsystems, mit der man sehr einfach funktionstüchtige, praxistaugliche Peer-to-Peer-Netzwerke entwickeln kann. All diese Netzwerke verbesserten die ursprünglichen Probleme von Gnutella und spielen heute noch eine große Rolle. Oftmals verzichten sie aber auf den Einsatz elementarer algorithmischer Techniken, die Informatiker schon zu Beginn ihres Studiums kennenlernen.

Effiziente Datenstrukturen werden, seitdem es Computer gibt, zur Beschleunigung der Suche nach Daten verwendet. Diese Datenstrukturen können aber nicht ein-

fach auf diese neuartigen Netzwerke angewendet werden. Zwar gab es schon länger parallele Rechennetzwerke und effiziente Algorithmen für Parallelrechner. Peer-to-Peer-Netzwerke stellten aber eine gewisse Herausforderung dar, weil kein Rechner eine hervorgehobene Rolle spielen darf. Außerdem erscheinen und verschwinden die Netzwerkteilnehmer ständig, so dass sich ein Peer-to-Peer-Netzwerk in ständiger Unruhe befindet.

Die erste effiziente Datenstruktur für ein Peer-to-Peer-Netzwerk wurde 2001 mit *CAN* (*Content Addressable Network Storage*) [8] vorgestellt. CAN verwendet eine Technik namens *Distributed Hash-Tables* (DHT), 1997 für die Verteilung von Daten und Verkehr für Web-Surfer entwickelt [9]. CAN beschrieb damals schon sehr weitgehende Konzepte zur Verbesserung der Suche in Peer-to-Peer-Netzwerken. Im selbem Jahr wurde mit *Chord* [10] eine effizientere Lösung vorgestellt. Chord realisiert die Suche effizienter als CAN und besitzt eine vergleichbar einfache Struktur. Noch heute ist Chord eine brauchbare Methode zur verteilten Indizierung. An CAN und Chord waren namhafte Algorithmiker beteiligt, und es entwickelte sich in dieser Gemeinde eine wahre Jagd nach der besseren verteilten Datenstruktur (Wie auch zeitgleich in der Gemeinde der Netzwerkpraktiker immer neue Peer-to-Peer-Netzwerke entwickelt wurden, wenn auch mit völlig anderer Zielrichtung). So kamen 2001 mit *Pastry* [11] und *Tapestry* [12] zwei eng verwandte, ebenfalls effiziente Netzwerke hinzu, die eine Routing-Methode von Plaxton, Rajamaran und Richa [13] auf Peer-to-Peer-Netzwerke übertrugen. Erwähnenswert ist auch das *Kademlia*-Netzwerk [14], das 2002 veröffentlicht wurde und ebenfalls eine einfache und effiziente Methode zur Suche nach Indexdaten zur Verfügung stellt.

In diesen Anfangsjahren der Peer-to-Peer-Netzwerke überschlugen sich die Ereignisse, und zahlreiche theoretische und praktische Konzepte wuchsen nebeneinander her. Diese Situation scheint sich nun zu ändern. So verwendet *Overnet*, eine Erweiterung von eDonkey, zum effizienten Routing von Suchanfragen die Methode von Kademlia. *Bittorrent*, eines der erfolgreichsten Peer-to-Peer-Netzwerke, wurde 2005 um eine DHT-Datenstruktur erweitert. Bis dahin hatte es sich seit seinem Entstehen 2001 dadurch ausgezeichnet, dass es zwar große Dateien unerreicht schnell übertragen kann, aber überhaupt keine Möglichkeit zum Finden von solchen Daten bereitgestellt hat. Diese Information musste zuvor anderweitig beschafft werden. Die theoretischen und praktischen Ansätze wachsen also nun zusammen und man kann gespannt sein, wie weit sich dieses Konzept noch weiterentwickeln wird.

1.3 Was ist ein Peer-to-Peer-Netzwerk?

Die häufigste Anwendung von Peer-to-Peer-Netzwerken war und ist momentan die Weitergabe von Musik- und Film-Dateien, und das zumeist noch im Zusammenhang mit der Verletzung von Urheberrechten. Dabei war jedoch von Anfang an klar, dass Peer-to-Peer-Netzwerke mehr als das können.

Das englische Wort *Peer* ['pir] bezeichnet einen Ebenbürtigen, Gleichrangigen (aber auch ein Mitglied des Hochadels). Dem Wörterbuch von Merriam-Webster zufolge geht dieses Wort über das Mittelenglisch zurück auf das lateinische Wort *par*,

das auch im Deutschen mit der Bedeutung „gleich“ verwendet wird. Eine *Peer-to-Peer*-Beziehung ist demnach ein Treffen unter Gleichen. Keiner steht über dem anderen. Niemand kontrolliert die Kommunikation oder gibt Anweisungen. Ein Peer-to-Peer-Netzwerk ist wortwörtlich ein Netzwerk, das Gleichrangige mit anderen Gleichrangigen direkt verbindet ohne zentrale Instanzen oder Kontrollen. Es gibt keine Teilnehmer, die die Rolle einer tonangebenden Ordnungsbehörde spielen.

Das zugehörige politische Äquivalent des Peer-to-Peer-Netzwerks ist vielleicht die Demokratie (oder Anarchie?), wo jeder Beteiligte gleiche Rechte und Pflichten hat. Nur gibt es in Peer-to-Peer-Netzwerken keinen gewählten Präsidenten, keine Richter und keine Polizei.

Tatsächlich behandelt auch das *Internet* (siehe Seite 11) Rechner gleich. In den verwendeten Protokollen *TCP*, *UDP* und *IP* gilt immer das Prinzip der Gleichheit. Es gibt auch keine privilegierten IP-Adressen. So gesehen setzte das Internet schon das Peer-to-Peer-Prinzip ein, lange bevor Peer-to-Peer-Netzwerke als solche bezeichnet wurden. Diese Gleichbehandlung geschah aber in den unteren, für die Benutzer nicht sichtbaren Schichten des Internet-Protokolls: der *Transportschicht* (Transport Layer), d.h. *TCP (Transmission Control Protocol)* und *UDP (User Datagram Protocol)*, und der *Vermittlungsschicht* (Network Layer) mit *IP (Internet Protocol)*. In der *Anwendungsschicht (Application Layer)* werden zumeist *Client-Server-Protokolle* verwendet. Die Software, die der Otto Normalverbraucher erhält, ist der sogenannte *Client*, beispielsweise ein *Web-Browser*, mit dem man HTML-Seiten abrufen kann. Auf der anderen Seite stehen die Web-Inhalte, die auf einem *Server* gespeichert werden. Der Client kontaktiert den Server, um die Web-Seite abzurufen und der Server schickt sie dem Client zu. Diese Beziehung ist völlig asymmetrisch. In der Praxis wird Server-Software zumeist auf besonders zuverlässigen und leistungsfähigen Rechnern betrieben, während Client-Software massenhaft auf Standardrechnern läuft. Solche Client-Server-Architekturen kennt man von *HTML (Hypertext Markup Language)*, *E-Mail (SMTP - Simple Mail Transmission Protocol)*, Internet-Suchmaschinen, Verzeichnisdiensten, Online-Radio, Internet-Dienste-Anbietern und vielen anderen. Die Client-Server-Architektur ist die vorherrschende Konstruktionsmethode für Anwendungen im Internet.

Bei einer Client-Server-Netzwerk-Struktur kontrolliert ein privilegierter Knoten, der Server, die anderen Teilnehmer, die Clients. Verweigert der Server einem Client den Dienst, so ist der Client machtlos. Fällt der Server aus, so kann kein Client mehr arbeiten. Hierzu reicht es schon, dass der Server kurzzeitig mit zu vielen Client-Anfragen überlastet wird.

In Peer-to-Peer-Netzwerken gibt es diese klare Aufteilung in Server und Client nicht mehr. Wenn man es in der Denkweise der Client-Server-Architektur ausdrücken will, so kann man sagen: In einem Peer-to-Peer-Netzwerk nimmt jeder Teilnehmer sowohl Client- als auch Server-Aufgaben wahr. Ein Definitionsversuch von Peer-to-Peer-Netzwerken der *Peer-to-Peer-Working-Group* [15] war: „In einem Peer-to-Peer-Netzwerk werden verteilte Rechenressourcen durch direkte Kommunikation gemeinsam genutzt.“ Diese Definition ist sicherlich zu allgemein. Letztlich haben sich Wissenschaftler nur darauf geeinigt, dass Peer-to-Peer-Netzwerke nicht Client-Server-Netzwerke darstellen.

Ironischerweise war ausgerechnet das erste Peer-to-Peer-Netzwerk, nämlich *Napster*, bis auf den Download ein klassisches Client-Server-Modell. Erst die darauf folgenden Netzwerke verfolgten den Peer-to-Peer-Gedanken vollständig. Bis jetzt hat sich die Fachwelt im Groben darauf geeinigt, dass ein Peer-to-Peer-Netzwerk ein *Overlay-Netzwerk* des Internets ist, in dem Rechner ebenbürtig ohne zentrale Koordination kommunizieren und gemeinsam eine Anwendung zur Verfügung stellen. *Overlay* bezeichnet die Tatsache, dass über dem bestehenden Netzwerk, dem Internet, ein weiteres Netz gesponnen wird, in dem ausschließlich die Peer-to-Peer-Teilnehmer vorhanden sind.

1.4 Ein Überblick über dieses Buch

Bevor wir wieder auf Peer-to-Peer-Netzwerke zu sprechen kommen, ist es ganz hilfreich das Netzwerk unter dem Overlay-Netzwerken kennenzulernen: Das Internet. In Kapitel 2 stellen wir dieses Netzwerk mit seinen Besonderheiten vor. Im Wesentlichen besteht das Internet aus dem Internet-Protokoll (IP) und den Transport-Protokollen TCP und UDP. Viele Design-Entscheidungen in Peer-to-Peer-Netzwerken werden erst klar, wenn man das Internet als Kommunikationsnetzwerk kennt.

Dann wenden wir uns in Kapitel 3 den ersten Peer-to-Peer-Netzwerken zu: Napster und Gnutella. Diese Netzwerke haben die weitere Entwicklung entscheidend geprägt. Insbesondere ihre Unzulänglichkeiten waren ein Ansporn für die Entwicklung der nachfolgenden Netzwerke.

Den Schwerpunkt dieses Buchs stellen die Kapitel 4 bis 8 dar. Hier werden Peer-to-Peer-Netzwerke besprochen, die die effiziente Suche nach Daten und die Netzwerkstruktur perfektioniert haben. Diese sind CAN in Kapitel 4, Chord in Kapitel 5, Pastry und Tapestry in Kapitel 6. Viceroy, Distance-Halving und Koorde optimieren neben der Suche auch die Anzahl der Nachbarn. Diese drei werden in Kapitel 7 vorgestellt. In Kapitel 8 werden effiziente Netzwerkstrukturen besprochen, die die Indexdaten sortiert abspeichern. Dies sind P-Grid und Skipnet.

In Kapitel 9 wenden wir uns dem Phänomen der Selbstorganisation in Peer-to-Peer-Netzwerken zu. So kann man in dem unorganisierten Verbindungsaufbau von Gnutella eine Struktur beobachten, die man mit den Begriffen Pareto-Verteilung und Small-World-Phänomen beschreiben kann. Im gleichen Kapitel wird beschrieben, wie einfache lokale Operationen bestimmte Grapheigenschaften (kleiner Durchmesser und Expander-Eigenschaft) herstellen können. Diese Mechanismen kann man zur Reparatur von Peer-to-Peer-Netzwerken wie Chord verwenden.

Peer-to-Peer-Netzwerke sind häufig das Ziel von Angriffen. In diesen Netzwerken ist das Problem der Abwehr solcher Angriffe besonders schwierig, da keine zentralen Strukturen vorhanden sind, die autoritär eingreifen können. Außerdem kennen und vertrauen sich die Teilnehmer nicht. Diese Sicherheitsprobleme werden in Kapitel 10 beschrieben.

Das Internet bietet a priori keine Anonymität, denn es arbeitet mit eindeutigen Adressen. Teilnehmer in Peer-to-Peer-Netzwerken möchten aber aus verschiedenen

Gründen nicht öffentlich auftreten. In Kapitel 11 werden wir zeigen, wie Peer-to-Peer-Netzwerke Anonymität zusichern können.

Peer-to-Peer-Netzwerke werden in der Praxis hauptsächlich zum Dateitransfer eingesetzt. In Kapitel 12 werden wir beschreiben, wie man diesen Prozess beschleunigen kann. Insbesondere beschreiben wir hier IP-Multicast, das Peer-to-Peer-Netzwerk Bittorrent und die Methode der Netzwerkkodierung.

In Kapitel 13 werden die am meisten verbreitetste Peer-to-Peer-Systeme vorgestellt. Dies sind Fasttrack, eDonkey, Overnet und Bittorrent.

Wer ein zeitgemäßes Buch über Peer-to-Peer-Netzwerke schreibt, der darf die juristische Situation nicht verschweigen. Diese ist äußerst unübersichtlich und im fortwährenden Wandel. Unter anderem wird in Kapitel 14 eine Hilfestellung für Netzwerk-Designer gegeben. (Alle juristischen Auskünfte in diesem Buch sind ohne Gewähr.) Außerdem werden in diesem abschließenden Kapitel bestehende und zukünftige Anwendungen für Peer-to-Peer-Netzwerke sowie offene Fragen diskutiert.

Eine Zusammenfassung der verwendeten mathematischen Notationen und Sätze findet sich im Anhang ab Seite 261.

2

Das Internet — unter dem Overlay

On the Internet, nobody knows you're a dog.
Peter Steiner. The New Yorker.

In diesem Kapitel befassen wir uns mit dem Internet, also dem Netzwerk unter allen Peer-to-Peer-Netzwerken. Das ist notwendig, da man die Konstruktionsprinzipien und die Geschichte von Peer-to-Peer-Netzwerken nur im Zusammenhang mit den besonderen Eigenschaften des Internets verstehen kann. Seinen Ursprung hatte das Internet im so genannten *ARPANET* (*Advanced Research Projects Agency Network*) Ende der sechziger Jahre. Das Ziel des ARPANETs war die großräumige und kontinuierliche Vernetzung von Großrechnern in den USA. Angefangen mit vier Rechnern im Jahr 1969, wuchs das ARPANET innerhalb von drei Jahren zu einem Netzwerk von einigen Dutzend Rechnern. Damals waren Rechenanlagen selten und diese waren zumeist mit einigen *Terminals* verbunden. Terminals sind Endgeräte für die Ein- und Ausgabe, mit denen Benutzer auf dem gastgebenden (daher *Host*) Großrechner arbeiten. Diese Tastatur/Bildschirm-Kombinationen sind sternförmig mit dem Großrechner verbunden.

Das ARPANET stellte eine Art Zwischennetzwerk (Inter-Net) dar, das einige der Großrechner vernetzte, so dass diese einen gleichberechtigten Verbund bildeten. Damals beteiligten sich hauptsächlich Universitäten am Aufbau von ARPANET. Aus diesem Netzwerk ging das Internet hervor, das mittlerweile zig Millionen Rechner verbindet. Wie in einem Peer-to-Peer-Netzwerk werden im Internet alle beteiligten Rechner gleich behandelt. Lange Zeit war nicht klar, was der Nutzen eines solchen großflächigen Netzwerks (*Wide Area Network*, *WAN*) für den Normalverbraucher sein könnte. Denn die Hauptanwendungen waren E-Mail, Newsgroups, der Austausch von wissenschaftlichen Daten und Spezialsoftware — Dienste, die damals in der Öffentlichkeit kaum bekannt waren.

Erst die Verbreitung des persönlichen Rechners (*PC: Personal Computer*), die Verbesserung der Netzanbindung von Privatpersonen (*Modem*: *Modulator Demodulator*, ISDN: *Integrated Services Digital Network*, DSL: *Digital Subscriber Line*) und vor allem das *World-Wide-Web (WWW)* ermöglichten den Siegeszug des Internets gegenüber bisherigen Kommunikationsmitteln wie Telefon, Telex, Fax, Radio,

Fernsehen, Zeitung und Briefverkehr. Manchmal werden die Begriffe Internet und World Wide Web (WWW) miteinander verwechselt. Das Internet ist das weltweite, paketorientierte Netzwerk zur Verknüpfung lokaler Rechnernetzwerke. Das World Wide Web dagegen ist eine Kollektion von *Hypertext*-Dokumenten, die über das Internet abrufbar sind. Der Zugriff erfolgt durch das *HTTP (Hypertext Transfer Protocol)*, das in der obersten Schicht des Internets, der Anwendungsschicht, liegt (siehe Kapitel 2.1).

Die Protokolle des Internets erlauben Computern unterschiedlicher Leistungsfähigkeit, Hersteller und Betriebssysteme miteinander zu kommunizieren. Der jetzige Aufbau dieser Protokolle ist weitestgehend das gewachsene Ergebnis eines Prozesses, der durch den enormen Zuwachs an Rechenkapazitäten, Datendurchsatz und Anzahl verbundener Rechner geprägt wurde. Ein historischer Überblick findet sich in [16].

2.1 Die Schichten des Internets

Wir geben hier zuerst einen kurzen Überblick über das Zusammenspiel der verschiedenen Protokolle. Eine ausführliche Beschreibung der TCP/IP-Protokolle im Einzelnen findet sich in [17].

Das Internet besteht aus vier Schichten: Anwendungsschicht (*Application Layer*), Transportschicht (*Transport Layer*), Vermittlungsschicht (*Network Layer*) und Verbindungsschicht (*Link Layer* oder *Host-to-Network*), siehe Tabelle 2.1. Das Internet ist kein eigenständiges Netzwerk. Es ist vielmehr ein Verbindungsnetzwerk zwischen lokalen Netzwerken, daher „Inter-Net“. Deswegen wird die unterste Schicht des Internets auch nicht gesondert spezifiziert. Hier kann man beliebige lokale Netzwerke (*LAN: Local Area Network*) einsetzen, wie zum Beispiel *Ethernet*, *Token Ring*, *WLAN* (drahtlose lokale Netzwerke: *Wireless Local Area Network*), Glasfasernetzwerke etc. Tabelle 2.1 und Abbildung 2.1 geben einen Überblick über diese Schichten des Internets.

Tabelle 2.1. Die Schichten von TCP/IP.

Anwendungsschicht	z.B. Telnet, FTP, HTTP, SMTP, ..., **Peer-to-Peer-Netzwerke**
Transportschicht	**TCP**, **UDP**
Vermittlungsschicht	**IP** +ICMP, +IGMP
Verbindungschicht	Gerätetreiber (z.B. Ethernet, Token Ring Driver)

Die wesentlichen Schichten des Internets sind die Transport- und die Vermittlungsschicht. Die Vermittlungsschicht besteht aus dem Hauptprotokoll *IP* (*Internet Protocol*) und den beiden Hilfsprotokollen *ICMP* (*Internet Control Message Protocol*) und *IGMP* (*Internet Group Message Protocol*). Diese Schicht kümmert sich um die Auslieferung der Datenpakete von einem Rechner zu einem anderen und

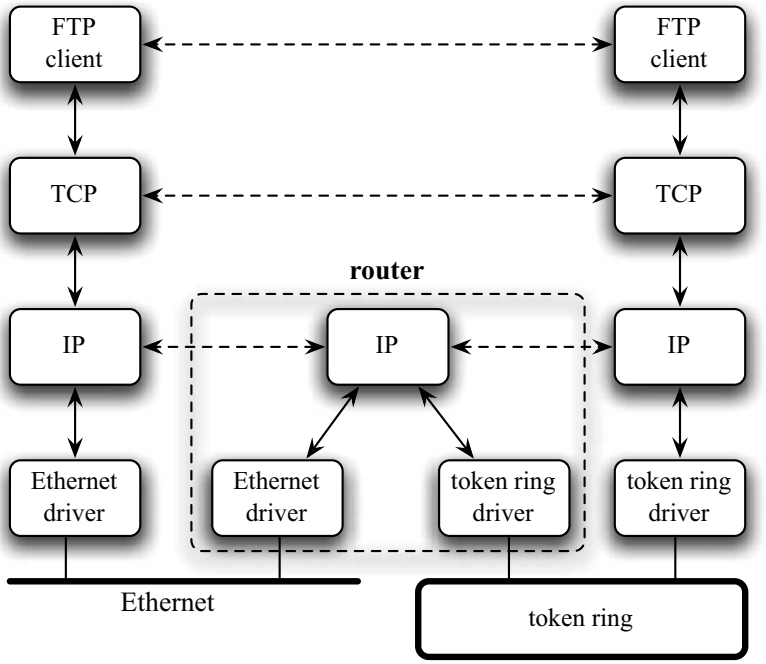


Abb. 2.1. Beziehungen der Schichten von TCP/IP [17].

darum, welche Routen dabei gewählt werden. Die Transportschicht besteht aus den Protokollen *TCP* (Transmission Control Protocol) und *UDP* (User Datagram Protocol). Dabei etabliert TCP eine zuverlässige Verbindung zwischen zwei Rechnern, während UDP nur eine Art unzuverlässiger Postkartenversand ist.

Im Moment gibt es zwei verschiedene Versionen des Internet-Protokolls IP, Version 4 (*IPv4*) und 6 (*IPv6*). (Die Versionsnummer 5 wurde übersprungen.) Die vorherrschende Form des Internet-Protokolls der Vermittlungsschicht ist im Moment IPv4. In bestimmten Teilen der Welt hat IPv6 schon die Überhand gewonnen. Wir werden hier zunächst IPv4 und dann die Neuerungen von IPv6 beschreiben.

Von den verschiedenen Protokollen des Internets sind die beiden wichtigsten TCP und IP. Daher wird die Transport- und Vermittlungsschicht mitunter auch als *TCP/IP-Layer* bezeichnet, was etwas irreführend ist.

Datenkapselung

In den jeweiligen Schichten tauschen die beteiligten Rechner Kontrollinformationen aus. Diese werden der Nachricht vorangestellt (*Header*). Diese Kontrollinformationen werden (zumeist) in den weiteren Schichten ausgewertet. In den unteren Schichten wie der Sicherungsschicht (ein Teilbestandteil der Verbindungsschicht), die hier nicht von Interesse sind, wird diese Kontrollinformation auch am Ende des Pakets als so genannter *Trailer* angefügt.

Der TCP-Header wird als Nutzdatenlast in IP mitbefördert und erst am Zielort in der Transportschicht gelesen. Ähnlich wird der IP-Header hinzugefügt und mit den Übertragungsprotokollen lokaler Netzwerke mitbefördert. Im Gegensatz zum TCP-Header wird der IP-Header in jedem Router verändert (insbesondere das *TTL*-Feld: *Time to Live*), und am Zielort wird der Transportschicht die Nachricht ohne IP-Header übergeben.

Natürlich können auch Protokolle der Anwendungsschicht Header-Informationen hinzufügen. Mit dieser Zwiebelstruktur wird das zu befördernde Datenvolumen vergrößert, siehe Abbildung 2.2.

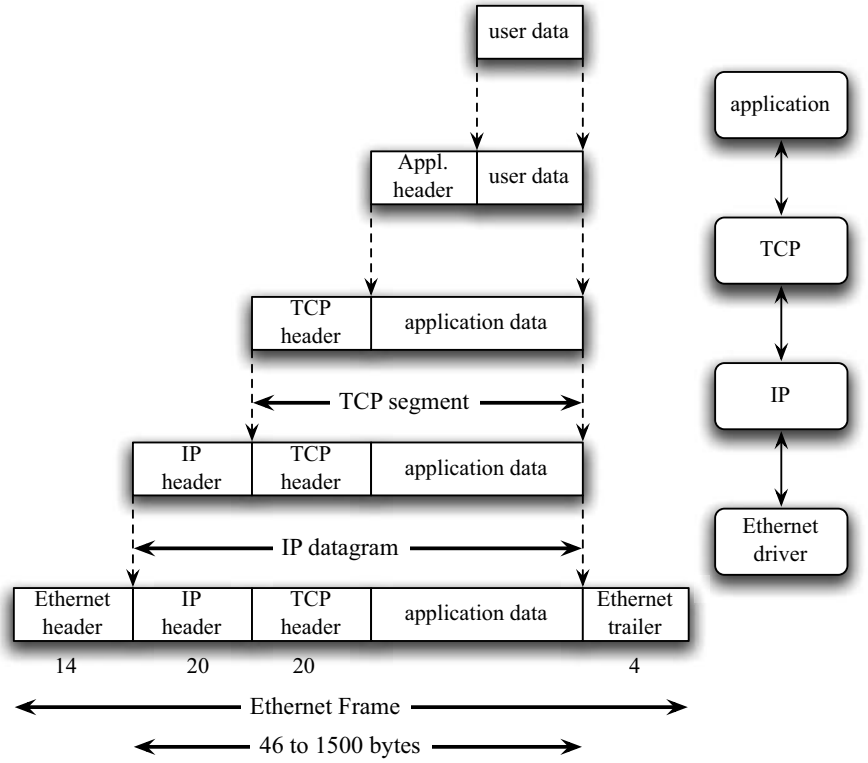


Abb. 2.2. Zwiebelstruktur der gekapselten Kontrollinformationen einer FTP-Übertragung [17].

2.2 Die Vermittlungsschicht: IPv4

Das *Internet Protocol* (IP) und seine Hilfsprotokolle *ICMP* (*Internet Control Message Protocol*) und *IGMP* (*Internet Group Message Protocol*) ermöglichen einen

Verbund von (lokalen) Netzwerken. Dabei ist IP ein unzuverlässiger verbindungsloser Datagrammauslieferungsdienst:

- IP ist ein *Datagrammauslieferungsdienst*.
  Ein *Datagram* (von *Data* und *Telegram*) besteht im Wesentlichen nur aus Datenpaket, Sender- und Empfängeradresse. Die Aufgabe von IP ist, dieses Datagramm unverändert vom Sender zum Empfänger zu befördern.
  Ist keine direkte Verbindung zwischen Sender und Zielrechner vorhanden, werden als Zwischenstationen Rechner benutzt. Diese Rechner werden als *Router* bezeichnet.
- IP ist *unzuverlässig*.
  Die Fehlerbehandlung von IP ist sehr einfach gehalten und arbeitet wie folgt: Tritt ein Problem bei der Auslieferung eines Datagramms auf, wird das Datagramm gelöscht und versucht eine ICMP-Nachricht zum Sender zu schicken. Tritt dabei ein Problem mit der Auslieferung einer ICMP-Nachricht auf, wird auch diese gelöscht (und keine neue ICMP-Nachricht erzeugt). Es werden insbesondere keine Kopien von Datagrammen erzeugt. Das bedeutet unter anderem: Fällt ein Router aus, so gehen mit ihm alle Datagramme verloren.
- IP ist *verbindungslos*.
  Die Weitergabe der Datagramme geschieht ausschließlich durch Weitergabe von Router zu Router, als *Hop-to-Hop*-Protokoll. Jedes Datagramm wird unabhängig behandelt. Deswegen können Datagramme in veränderter zeitlicher Reihenfolge am Zielrechner eintreffen.

IPv4-Header

Der in Abbildung 2.3 dargestellte IPv4-Header ist mindestens 20-Byte und besteht aus den folgenden Teilen:

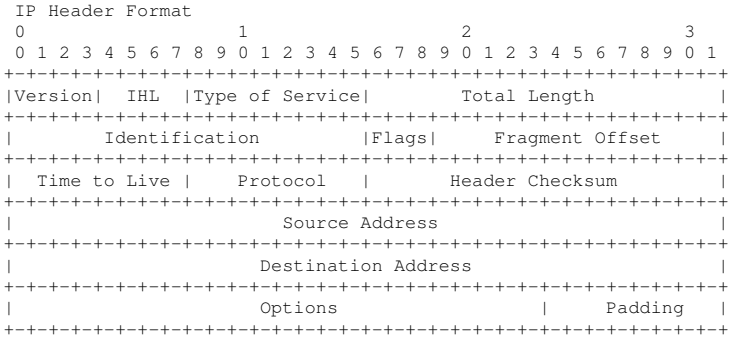


Abb. 2.3. IPv4-Header-Definition aus RFC 791 [18].

Version
Das Version-Feld enthält die Versionsnummer von IP. Momentan findet ein Übergang von IPv4 zu IPv6 statt, um eine Reihe von historisch bedingten Einschränkungen zu lösen. IPv4 und IPv6 werden aber noch sehr lange parallel existieren. Wir beziehen uns hier auf IPv4, so dass das Feld den Wert 4 enthalten würde.

IHL
Das Feld IHL (*IP Header Length*) enthält die Länge des IP-Headers in vielfachen von 32 Bit. Der Mindestwert ist 5.

Type of Service
Vom Type of Service Feld (TOS) werden nur vier Bits verwendet, um zu signalisieren, ob *delay*, *throughput*, *reliability*, oder *monetary cost* optimiert werden soll. Tatsächlich wird das gesamte Feld von den meisten Routern ignoriert.

Total Length
Das Total Length Feld gibt die Größe des IP-Pakets in Bytes an. Damit beschränkt IPv4 die maximale Paketgröße auf 64 kBytes. Tatsächlich werden hier in der Regel höchstens einige kBytes eingetragen.

Identification
Das Identification Feld identifiziert ein von einem Rechner gesendetes Paket (also ein Datagram) eindeutig. Der Identifikationswert wird in der Regel mit jedem gesendeten Paket um eins erhöht.

Time to Live
Das Time to Live Feld (TTL) gibt die maximale Anzahl von Routern an, die dieses Paket passieren darf. Jeder Router verringert diesen Eintrag. Ist er Null und ist das Paket noch nicht angekommen, wird eine ICMP-Nachricht an den Sender geschickt und das IP-Paket (Datagramm) verworfen.

Protocol
Dieses Feld wird zur Verteilung der Pakete im Rahmen des *Demultiplexing* an TCP, UDP, ICMP etc. benutzt. In ihm ist vermerkt, an welchen Dienst das Datagram übergeben werden soll.

Checksum
Die in diesem Feld enthaltene Prüfsumme bezieht sich nur auf den IP-Header und dient der Erkennung von Übertragungsfehlern sowie der Elimination sinnloser Datagramme.

Source Address
Dieses Feld enthält die 32-Bit-IP-Adresse des Quellknotens des Pakets.

Destination Address
Dieses Feld enthält die 32-Bit-IP-Adresse des Zielknotens des Pakets.

Options
Das Options Feld ermöglicht die Angabe zusätzlicher Optionen, wie zum Beispiel:

- Sicherheits- und Handhabungseinschränkungen für militärische Zwecke,
- Routeninformation (*record route*),
- Zeitstempel (z.B. für *Traceroute*),

- Zwischenstationen, die dieses Datagramm passieren darf (*loose source routing*),
- Zwischenstationen, die dieses Datagramm passieren muss (*strict source routing*).

Padding

Das Padding Feld wird lediglich verwendet um sicherzustellen, dass der Header an einer 32-Bit Grenze endet.

IPv4-Adressen und DNS

Ursprünglich bezeichnete jede IP-Adresse eine Schnittstelle (*Interface*) eines im Internet erreichbaren Rechners. Mit einer IP-Adresse kann (oder vielmehr, wie wir später sehen werden: konnte) jedem Rechner im Internet ein Paket zugestellt werden. Diese 32-Bit-Adressen werden byteweise notiert, zum Beispiel als `65.114.4.69`. Sie gliederten sich bis 1993 in eine Rechner-Identifikationsnummer (*Host-ID*) und eine Netzwerkadresse (*Net-ID*) sowie eine Klassenbeschreibung A, B oder C, die festlegen, welche Adresslängen *Net-ID* und *Host-ID* haben. Netzwerkadressen werden vom *Internet Network Information Center* vergeben, Host IDs von der jeweiligen Netzwerkadministration. Jede dieser Klassen war gekennzeichnet durch ein Präfix (Anfangsteilstück) fester Länge, das das *Teilnetz* (*Subnet)* beschrieb und ein Suffix (Endteilstück), das die *Host-ID* (Rechnername im Teilnetz) beinhaltete.

Seit 1993 ersetzt das so genannte *Classless Inter-Domain Routing* (*CIDR*) diese ineffiziente Form der Adressvergabe. Der Grund war, dass gewissen Teilnetzen Millionen von Adressen reserviert waren, während in anderen Teilnetzen mit weniger privilegierten Klassen die Host-IDs ausgingen. In CIDR wird die Aufteilung der Adresse in Subnetz und Host-ID variabel durch die so genannte Netzwerkmaske bewerkstelligt. Die Netzwerkmaske besteht aus 32 Bits, die mit einer Folge von Einsen beginnen und mit einer Folge von Nullen enden. Die Anzahl der führenden Einsen gibt an, wie viele Bits der IP-Adresse die Teilnetzwerkadresse bezeichnen und wie viele Bits zu der Host-ID gehören.

Beispiel:

Die Netzwerkmaske

`11111111.11111111.11111111.00000000`

besagt, dass die IP-Adresse

`10000100.11100110.10010110.11110011`

im Teilnetz mit der Bezeichnung

`10000100.11100110.10010110.`

den Host

`11110011`

adressiert (Die Punkte dienen nur der besseren Lesbarkeit).

Wie bereits angedeutet, ist nur noch ein Teil der IP-Adressen tatsächlich eindeutig und durch das IP-Routing direkt erreichbar. Wir werden später in den Abschnitten über *Port/Network Address Forwarding* (PAT/NAT), *Dynamic Host Configuration Protocol* (DHCP) und *Firewalls* darauf zu sprechen kommen.

Dem Internet-Benutzer ist nicht zuzumuten, diese IP-Adressen von Hand einzugeben. Daher wurde das *Domain Name System* (*DNS*) geschaffen. Hier kann man statt IP-Adressen Namen eingeben, wie etwa

```
cone.informatik.uni-freiburg.de
```

die dann durch das DNS in die entsprechenden IP-Adressen umgewandelt werden. Das Domain Name System ist eine verteilte robuste Datenbank, die regelmäßig aktualisiert wird (in Unix-Systemen ist die Datenbank abrufbar mit dem Befehl `host`).

Im Zusammenhang mit dynamischen IP-Adressen bei DHCP (siehe Seite 48) kann sich DNS als geeigneter Dienst erweisen, um die IP-Adressen wieder auffindbar zu machen.

2.2.1 Routing und Paketweiterleitung

Im Internet Protocol (IP) wird zwischen der Wegewahl (*Routing*) und der Paketweiterleitung (*Packet Forwarding*) unterschieden. Die Wegewahl muss vor der eigentlichen Beförderung eines Datagramms geschehen. Das Ergebnis der Ermittlung der Wegewahl wird in den so genannten *Routing-Tabellen* gespeichert. Diese Wegewahl kann manuell oder aber auch automatisch geschehen. Die Paketweiterleitung geschieht dann durch einen einfachen Algorithmus, der in Abhängigkeit der Zieladresse im Datagramm den nächsten Rechner auf dem Weg durch das Internet bestimmt.

Paketweiterleitung

Neben dem Fachbegriff *Packet Forwarding* wurde früher auch *Packet Routing* verwendet. Sowohl Router als auch Endgeräte benutzen Routing-Tabellen um den nächsten Knoten für die Weiterleitung des Pakets zu bestimmen. Hierbei wird der im Folgenden beschriebene Algorithmus verwendet, der nur die Routing-Tabelle und die IP-Adresse verwendet. Zur Erinnerung: Die IP-Adresse bezeichnet eine Kommunikationsschnittstelle (Interface) eines Rechners.

Wenn die Zieladresse eines ankommenden IP-Pakets die Adresse des entsprechenden Interfaces ist, dann wird die Nachricht nicht mehr weitergeleitet und das Paket der Transportschicht zur weiteren Verarbeitung übergeben. Andernfalls, d.h. wenn für die Adresse oder die Netzwerkadresse (siehe CIDR) des Paketes ein Eintrag in der Routingtabelle vorhanden und der dort zugeordnete Rechner direkt erreichbar ist, wird es an diesen weitergeleitet. In allen anderen Fällen sendet der Router das Paket an das *Default Gateway*, das ebenfalls in der Routing-Tabelle steht.

Doch bevor das Paket weitergeleitet wird, wird zunächst das TTL-Feld ausgewertet. Dieses gibt an, wie viele Stationen ein Paket auf seinem Weg benutzen darf, bevor es verworfen wird. Bei der Weiterleitung des IP-Pakets wird dieser Zähler um

Eins verringert, wenn dieser nicht Null war. Andernfalls wird eine ICMP-Nachricht mit dem Router als Startadresse und der Quelladresse als Zieladresse erzeugt, um den Sender über den Misserfolg zu informieren. Solche ICMP-Nachrichten werden nicht erzeugt, falls das IP-Paket selbst schon eine ICMP-Nachricht war.

Dieser TTL-Mechanismus verhindert, dass bei einer Fehlkonfiguration der Routing-Tabellen Pakete ewig weitergereicht werden. Dies ist insbesondere dann der Fall, wenn die Routing-Tabellen Kreise für die Paket-Route implizieren. In diesem Fall wandert das Paket so lange im Kreis, bis der TTL-Zähler abgelaufen ist. Dann wird eine ICMP-Nachricht erzeugt. Diese hat natürlich (wie alle IP-Pakete) ebenfalls einen TTL-Eintrag und so wird diese Fehlernachricht auch nur eine gewisse Strecke zurücklegen, bis sie gelöscht wird. Da dann kein weiteres IP-Paket erzeugt wird, ist dieser (erfolglose) Auslieferungsprozess damit abgearbeitet worden.

Der TTL-Mechanismus kann auch verwendet werden um Informationen über das Internet zu gewinnen. Er wird insbesondere beim so genannten *Traceroute* verwendet, siehe Seite 28.

Routing

Für kleine lokale Netzwerke werden oftmals feste Routing-Tabellen verwendet, die vom Netzwerkadministrator manuell erstellt werden. Diese Methode wird als *statisches Routing* bezeichnet. Es wird also explizit eingetragen, welche Pakete wohin geleitet werden sollen. Daher ist statisches Routing nur für kleine, stabile Netze sinnvoll.

In der Regel werden Routing-Tabellen jedoch automatisch erzeugt, was dann analog als *dynamisches Routing* bezeichnet wird. Die Routing-Tabelle wird hierbei fortwährend an neue Gegebenheiten angepasst, wie z.B. an den Ausfall von Rechnern oder das Erscheinen neuer Router. Beispiele für dynamische Routing-Algorithmen sind *RIP* (*Routing Information Protocol*) und *OSPF* (*Open Shortest Path First*).

Das Prinzip der Paketweiterleitung durch Routing-Tabellen ermöglicht nur die Beförderung eines Pakets auf einem gleichbleibenden Weg vom Startknoten zum Zielknoten. Dieser sehr intuitive Ansatz kann in der Praxis die Performanz eines Netzwerks beeinträchtigen: Wenn zum Beispiel zwei unabhängige Wege zwischen zwei Rechnern möglich sind, so könnte man mit einer alternierenden Benutzung dieser Wege den Datendurchsatz vergrößern. Die Frage des größtmöglichen Datendurchsatzes wird beim Routing im Internet bereits bei der Wahl der Routing-Tabellen-Einträge vernachlässigt. Hier wird der kürzeste Weg grundsätzlich allen anderen Möglichkeiten vorgezogen.

So versuchen alle Routing-Algorithmen das so genannte *kürzeste-Wege-Problem* in Graphen zu lösen (siehe auch Anhang zum Begriff Graph auf Seite 263):

Definition 2.1 (Kürzeste-Wege-Problem). *Gegeben sei ein gerichteter Graph $G = (V, E)$, ein Startknoten $s \in V$ und Kantengewichte $w_e \in \mathbb{R}_0^+$ für alle Kanten $e \in E$. Gesucht werden alle Pfade vom Startknoten zu allen anderen Knoten, die jeweils die geringste Summe an Kantengewichten aufweisen.*

Wir verwenden im Folgenden Kantengewicht und Entfernung synonym. Im Internet wird hier für jede Verbindung e zwischen Rechnern der Wert $w(e) = 1$ gesetzt. Für manche Probleminstanzen gibt es mehrere Lösungen. Man kann aber leicht zeigen, dass unter den Lösungen immer eine Menge von Pfaden existiert, die in einen gerichteten Baum eingebettet sind, der den Startknoten als Wurzel hat und dessen Kanten zu den Blättern hin gerichtet sind.

Dijkstra
Eingabe: $G = (V, E)$, $s \in V$, $w : E \to \mathbb{R}_0^+$

for all $v \in V$ **do**
    Distanz$[v] \leftarrow \infty$
    Vorgänger$[v] \leftarrow v$
Distanz$[s] \leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Element aus Q mit minimalem Wert Distanz$[u]$
    $S \leftarrow S \cup \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for all $v \in V : (u, v) \in E$ **do**
        if Distanz$[u] + w_{(u,v)} <$ Distanz$[v]$ **then**
            Distanz$[v] \leftarrow$ Distanz$[u] + w_{(u,v)}$
            Vorgänger$[v] \leftarrow u$
return $\{(\text{Vorgänger}(u), u) \mid u \in V$ und $u \neq$ Vorgänger(u)$\}$

Abb. 2.4. Dijkstras Algorithmus.

Eine sehr effiziente Lösung für das Kürzeste-Wege-Problem ist der in Abbildung 2.4 dargestellte Algorithmus von Edsger Wybe Dijkstra [19]. Das Ergebnis dieses Algorithmus, angewendet auf einen Graphen, der Wege von s zu allen anderen Knoten beinhaltet, ist die Tabelle Vorgänger. Diese Tabelle beschreibt den Kürzeste-Wege-Baum mit den Kanten $(\text{Vorgänger}(u), u)$ für alle Knoten $u \in V \setminus \{s\}$. In Abbildung 2.5 ist der Ablauf von Dijkstras Algorithmus an einem Beispiel beschrieben.

Der Dijkstra-Algorithmus baut den Kürzeste-Wege-Baum sukzessive ausgehend vom Startknoten auf. Hierzu wird in jedem Durchlauf der While-Schleife der Knoten ausgewählt, der dem Startknoten am nächsten und noch nicht Teil des Baumes ist. Daraus ergibt sich die Korrektheit des Algorithmus. Liefe der kürzeste Weg zu diesem Knoten über einen anderen, noch nicht aufgenommenen Knoten, dann wäre dieser andere Knoten näher am Startknoten. Das widerspricht der Bedingung, dass der aufzunehmende Knoten dem Startknoten am nächsten war.

Algorithmen wie der Dijkstra-Algorithmus, die die Lösungsmenge sukzessive durch das am besten geeignete Element erweitern, nennt man *Greedy-Algorithmen*.

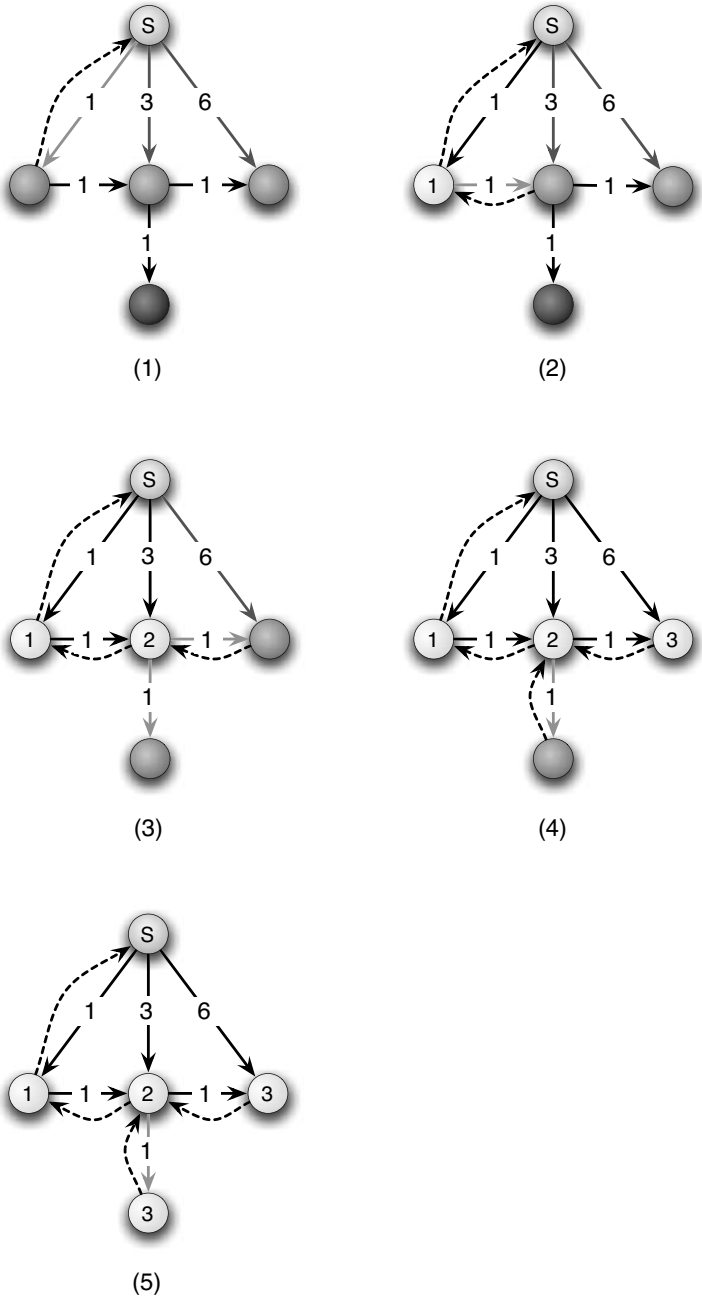


Abb. 2.5. Ablauf des Disjkstra-Algorithmus. Der resultierende Kürzeste-Wege-Baum wird durch gestrichelte Pfeile dargestellt.

Dijkstras Algorithmus besitzt eine enge Verwandtschaft zur Breitensuche im Baum. Wenn alle Kantengewichte den Wert Eins haben, entspricht die Reihenfolge in der die Knoten ausgewählt werden gerade derjenigen bei einer Breitensuche im Baum. Zudem ist Dijkstras Algorithmus sehr effizient: Für m Kanten und n Knoten muss der Algorithmus n-mal den nächsten minimalen Knoten bestimmen. Verwendet man für diesen Auswahlprozess eine geeignete Datenstruktur wie den Fibonacci-Heap [20], beträgt die Laufzeit des Algorithmus lediglich $\mathcal{O}(m + n\log(n))$, da die Aktualisierung der Distanzen in amortisierter konstanter Zeit möglich ist und nur die Auswahl und das Entfernen der Knoten logarithmische Zeit benötigen.

Link-State-Routing

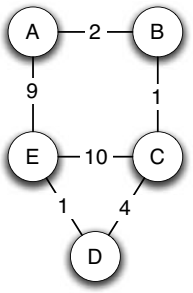
Eine praktische Umsetzung von Dijkstras-Algorithmus ist das so genannte Link-State-Routing. Bei diesem werden die Routing-Tabellen der Knoten bzw. Router analog zum Verfahren in Dijkstras Algorithmus bestimmt. Allerdings muss hierfür jeder Knoten alle Verbindungen des Netzwerks kennen. Dieses Problem wird dadurch gelöst, dass alle Knoten ihre Verbindungsinformation durch das Netzwerk fluten, was auch als *Broadcast* bezeichnet wird. Hierzu sendet jeder Knoten seine Informationen an alle seine Nachbarn. Diese reichen sie wieder an alle Nachbarn weiter usw. Somit erreicht jeden Rechner die gesamte Information des Netzwerks und er kann den Dijkstra-Algorithmus verwenden, um für jeden Rechner denjenigen Nachbarn zu bestimmen, der auf dem kürzesten Weg liegt. Bei diesem Verfahren muss jeder Knoten anfangs nur seine aktuellen Verbindungen verbreiten. Später genügt es, alle Teilnehmer über Änderungen zu informieren. Damit ist der Kommunikationsaufwand im laufenden Betrieb relativ gering.

Distance-Vector Routing

Der Distance-Vector-Routing-Algorithmus verwendet für jede bestehende Verbindung einen Entfernungseintrag für jeden Knoten. Diese Entfernungseinträge werden in so genannten Distance-Vector-Tabellen gespeichert. In der Distance-Vector-Tabelle eines Knotens H steht an der Position (A, B) (im Idealfall) die Länge des kürzesten Pfades von H zu A, der die Kante (H, B) als erste Kante verwendet. Distance-Vector-Tabellen für ein kleines Beispielnetzwerk finden sich in Abbildung 2.6.

Somit kann der Knoten H aus dieser Tabelle den kürzesten Weg bestimmen, indem er die Verbindung wählt, die den kleinsten Wert besitzt. Wenn H diese Information an alle Nachbarn weitergibt, können diese ihre Tabellen aktualisieren, da sie einfach die Entfernung zu H für den Eintrag hinzuaddieren müssen. Dieser verteilte Aktualisierungsprozess konvergiert zu den korrekten Einträgen.

Ein Problem, das beim Distance Vector Routing auftritt, ist das so genannte *Count-to-Infinity-Problem*. Dieses tritt zum Beispiel im folgenden Szenario auf: Wir betrachten drei Router A, B und C wobei Verbindungen zwischen A und B sowie B und C bestehen (siehe auch Abbildung 2.7 oben). Fällt nun der Router C aus, kann B ihn nicht mehr direkt erreichen. Router B meint jedoch, er könne C noch über den Router A erreichen und sendet seine geänderte Distance-Vector-Tabelle an A. Das



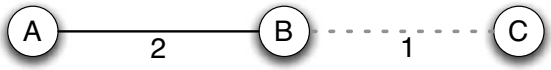
Distance Table für A

	über		Routing Tabellen
von A	B	E	Eintrag
nach B	**2**	11	B
C	**3**	19	B
D	**7**	10	B
E	13	**9**	E

Distance Table für C

	über			Routing Tabellen
von C	B	D	E	Eintrag
nach A	**3**	14	18	B
B	**1**	9	9	B
D	6	**4**	11	D
E	7	**5**	10	D

Abb. 2.6. Distance-Vector-Tabellen.



	über	Routing Tabellen
von A	B	Eintrag
nach B	2	B
C	3	B

	über		Routing Tabellen
von B	A	C	Eintrag
nach A	2	-	A
C	5	-	A

	über	Routing Tabellen
von A	B	Eintrag
nach B	2	B
C	7	B

	über		Routing Tabellen
von B	A	C	Eintrag
nach A	2	-	A
C	5	-	A

	über	Routing Tabellen
von A	B	Eintrag
nach B	2	B
C	7	B

	über		Routing Tabellen
von B	A	C	Eintrag
nach A	2	-	A
C	9	-	A

Abb. 2.7. Das Count-to-Infinity-Problem beim Distance-Vector-Routing.

hat zur Folge, dass auch Router A seine Tabelle anpasst (also den Eintrag für Routing nach C über B erhöht) und diese Information im Netzwerk verbreitet. Wenn Router B diese Nachricht erhält, wird er wiederum die Kosten erhöhen usw. Hier wird deutlich, dass sich schlechte Nachrichten, wie z.B. der Ausfall eines Routers, nur langsam verbreiten. Dagegen breiten sich gute Nachrichten sehr schnell aus. Das Count-to-Infinity-Problem lässt sich z.B. mit den im Folgenden kurz beschriebenen *Poisoned Reverse* oder *Split Horizon* Verfahren lösen.

Poisoned Reverse Wenn ein Router die Verbindung zu einem anderen Router verliert, teilt er dies den anderen Routern mit, indem er die verlorene Verbindung mit Länge unendlich angibt. So verbreitet sich die Information, dass der ausgefallene Router nicht mehr erreichbar ist, sehr schnell.

Split Horizon Bei diesem Verfahren wird ein Router daran gehindert, eine Route zu einem bestimmten Ziel zurück an den Router zu übermitteln, von dem er diese Route gelernt hat. Dadurch kann ein Router B, der die Verbindung zu einem anderen Router C verloren hat, nicht seinen Nachbarn A fragen, wie man nach C kommt, solange A davon ausgeht, dass dies über B geht.

Beide Verfahren haben jedoch das Problem, dass sie bei längeren Kreisen im Netzwerk nicht mehr zuverlässig arbeiten.

2.2.2 Autonome Systeme

Sowohl Link-State-Routing als auch Distance-Vector-Routing benötigen für jeden teilnehmenden Rechner mindestens einen Eintrag. Bei Link-State-Routing müsste jeder Rechner/Router sogar jede Verbindung im Netzwerk kennen. Nun gibt es Millionen von Router-Knoten im Internet, und die Anzahl der Endgeräte ist noch wesentlich höher. Daher ist es einfach nicht möglich, Routing-Tabellen nach diesen Verfahren zu erstellen. Auch statisches Routing ist kein gangbarer Weg.

Dieses Problem wird durch die Einführung von Hierarchien gelöst. Hierzu wird das Internet in so genannte *Autonome Systeme* (AS) partitioniert. Ein Autonomes System ist ein Teilnetz im Internet, das unter einer dedizierten Verwaltung steht. Dieses Netz wiederum kann sich aus Teilnetzen zusammensetzen. Üblicherweise gehört ein autonomes System einem Internetdienstanbieter, einer Universität, z.B. `uni-freiburg.de`, oder einem Unternehmen, das mit den übrigen Rechnern des Internets möglicherweise mehrfach verbunden ist. Diese Autonomen Systeme besitzen so genannte *Boundary Router*, die verschiedene Autonome Systeme verbinden, siehe Abbildung 2.8. Der Vorteil dieser Netzwerkstruktur ist, dass die Größe der Routing-Tabelle nur noch von der Größe des eigenen Autonomen Systems abhängt. Änderungen von Einträgen in den Routing-Tabellen werden nur innerhalb eines Autonomen Systems weitergegeben. Die Wegewahl im und durch das eigene Netz kann kontrolliert werden. Innerhalb eines Autonomen Systems wird ein einheitliches Routing-Protokoll verwendet. Die Routing-Protokolle verschiedener Autonomer Systeme müssen jedoch nicht identisch sein.

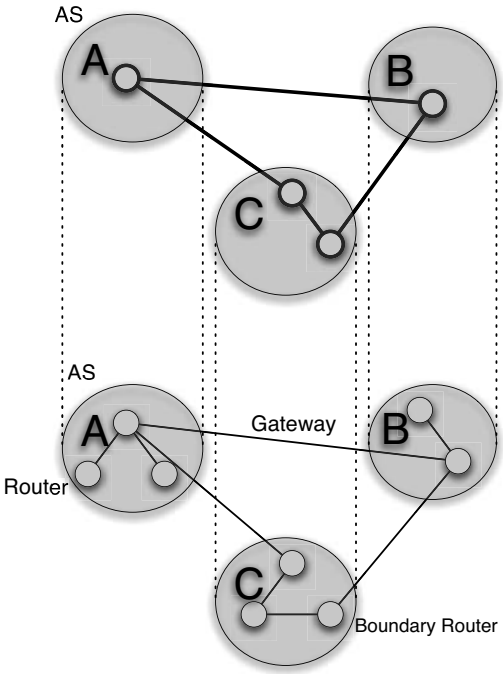


Abb. 2.8. Autonome Systeme.

Damit unterteilt sich die Ermittlung der Wege für Pakete in zwei Stufen: Routing innerhalb eines Autonomen Systems (*Intra-AS-Routing*) und die Wegewahl durch die verschiedenen Autonomen Systeme (*Inter-AS-Routing*).

Intra-AS-Routing

Für das Intra-AS-Routing stehen unter anderem die Routing-Protokolle Routing Information Protocol (RIP), Open Shortest Path First (OSPF), Interior-Gateway-Routing-Protocol (IGRP) zur Verfügung, die wir im Folgenden kurz beschreiben.

RIP: Routing Information Protocol

Das Routing Information Protocol (RIP) [21] (oder genauer gesagt *RIPv2*) ist ein Routing-Protokoll auf Basis des Distance-Vector-Routing-Algorithmus. Beim Starten eines Routers kennt dieser nur die direkt mit ihm verbundenen Router und sendet die daraus resultierende Routing-Tabelle an die benachbarten Router. Gleichzeitig fordert er von seinen benachbarten Routern deren Routing-Tabelle an. Mit diesen Informationen ergänzt der Router seine Routing-Tabelle und erkennt so, welche Netzwerke jeweils über welchen Router erreicht werden können und welche Kosten damit verbunden sind. Um Änderungen im Netzwerk (Ausfall oder Start eines Routers) zu erkennen, werden benachbarte Routing-Tabellen regelmäßig ausgetauscht,

unabhängig davon, ob sich Veränderungen ergeben haben oder nicht. Das Anbieten neuer Routen wird als *Advertisement* bezeichnet.

Falls nach einer gewissen Zeit kein Advertisement eingetroffen ist, also keine Änderungen an einem Eintrag in der Routing-Tabelle vorgenommen wurden, wird dieser gelöscht. Damit wird die Route über den Nachbarn für ungültig erklärt und eine schnelle Reaktion auf Topologie-Änderungen gewährleistet. Zur Überprüfung einer doch noch möglichen Verbindung werden neue Advertisements zum Nachbarn geschickt. Falls sich Änderungen an der Tabelle ergeben haben, schickt dieser Nachbar dann ebenfalls sein Advertisement und wird erneut in die Tabelle aufgenommen.

Bei RIP breitet sich die Routing-Information nur sehr langsam im Netzwerk aus. Außerdem tritt das *Count-to-Infinity-Problem* auf. Daher ist dieses Protokoll kaum für den Einsatz in großen Netzwerken geeignet.

OSPF: Open Shortest Path First

Das beim RIP bemängelte Verhalten in großen Netzwerken wird durch das *Open-Shortest-Path-First-Routing*-Protokoll (*OSPF*) gelöst. Kernstück von *OSPF* ist *LSD* (*Link-State-Database*), das eine Liste aller benachbarten Router enthält, zu denen Verbindungen bestehen. Sie spiegelt die Topologie des Netzes wider und wird in jedem Knoten gespeichert. Die Routenberechnung erfolgt auf Basis dieser Daten mit Hilfe Dijkstras Algorithmus. Damit die Datenbank aufgebaut oder bei Topologieänderungen aktualisiert wird, ist der Austausch von Routing-Informationen notwendig. Diese werden wie bei RIP als Advertisements durch das gesamte Autonome System geflutet.

Bei größeren Netzwerken wird das Netzwerk in zwei Ebenen unterteilt. Eine Ebene bildet das lokale Gebiet (*Local Area*) und eine weitere das Rückgrat des Netzwerks, das so genannte *Backbone*. Spezielle *Local-Area-Border-Router* fassen die Distanzen im eigenen lokalen Gebiet zusammen und bieten diese den anderen *Area-Border-Routern* per Advertisement an. Zur Verwaltung des gesamten Netzwerks werden ein oder mehrere *Backbone-Router* eingesetzt. Dieser verwendet gleichermaßen das OSPF-Protokoll, beschränkt sich aber auf das *OSPF-Backbone*.

OSPF wurde erstmals 1989 als Standard vorgeschlagen und existiert heute in der dritten Version als Routing-Protokoll für IPv4 und IPv6 [22].

IGRP: Interior-Gateway-Routing-Protocol

IGRP [23] ist ein 1980 von Cisco entwickeltes proprietäres Distance-Vector-Routingprotokoll, das von Routern verwendet wird, um innerhalb eines Autonomen Systems Routing-Informationen auszutauschen. Die Ziele bei der Entwicklung von IGRP waren vor allem eine Verbesserung der Skalierbarkeit sowie ein Überwinden der von RIP vorgegebenen maximalen Anzahl von 15 Netzwerkknoten, die ein Zielnetzwerk entfernt sein darf, bis das Netz als nicht erreichbar gilt (Dies ist bei IGRP 255). Bei RIP und OSPF gilt die Anzahl der Hops als Kostenmetrik. IGRP kann außerdem die zur Verfügung stehende Bandbreite, die auf dem Pfad entstehende Verzögerung, die Leitungszuverlässigkeit und die Leitungsauslastung zur Erstellung von Kostenmetriken verwenden. Standardmäßig wird die Kostenmetrik einer Route

aus der Bandbreite und der Leitungsverzögerung bestimmt. Routing-Updates werden alle 90 Sekunden versendet. Zur Vermeidung von Routingschleifen kommen Techniken wie der *Hold-down Timer*, das *Split-Horizon*-Verfahren sowie *Poisoned-Reverse* zum Einsatz. Dieses Routing-Protokoll ist mittlerweile vom verwandten EIGRP [24] (*Enhanced Interior Gateway Routing Protocol*, ebenfalls von Cisco) abgelöst worden.

Inter-AS-Routing

Für das Routing zwischen den Autonomen Systemen verwendete man früher das *Exterior-Gateway-Protocol* (*EGP*), das mittlerweile durch das *Border Gateway Protocol* (*BGP*) verdrängt wurde.

Damit ist das Border-Gateway-Protokoll das wichtigste Routing-Protokoll des Internets. Es unterhält eine Tabelle mit Einträgen von Teilnetzwerken oder auch Einträgen mit einem IP-Adress-Präfix und einem Pfad zu einem Rechner in dieser Menge. Ein Adress-Präfix vereinigt alle Teilnetze, die mit dieser Bitfolge beginnen, zu einem komprimierten Eintrag. Im Kern verwendet es ein *Path-Vector-Protokoll*. Dieses Protokoll ist ähnlich wie das Distance-Vector-Protokoll, nur dass statt Entfernungen zum jeweiligen Zielknoten die gesamte Pfadinformation gespeichert wird. Dadurch wird das Count-to-Infinity-Problem einfach vermieden, da dieses nur dann entstehen kann, wenn ein Knoten mehrfach in einem Pfad vorkommt.

BGP ist ein sehr umfangreiches und kompliziertes Protokoll, das in seiner Wegewahl auch physikalische und übertragungstechnische Parameter sowie politische und strategische Regeln berücksichtigt. Im Moment ist weltweit die Version 4 des BGP im Einsatz [25].

2.2.3 ICMP, Ping und Traceroute

Neben der Wegewahl und der Paketweiterleitung werden durch IP noch eine Reihe wichtiger Dienste angeboten.

ICMP

Das *Internet Control Message Protocol* (*ICMP*) sendet Kontrollnachrichten im Internet-Protokoll. Dieses sind Fehlermeldungen oder Informationen über das Netzwerk. ICMP-Daten werden in den Datenteil eines IP-Paketes geschrieben und im *Service-Type*-Feld des Headers angezeigt. Ein Beispiel für ICMP-Pakete sind die Fehlermeldungen, die von einem Router zum Absender zurückgesendet werden, falls der TTL-Zähler vor dem Ziel abgelaufen ist.

Ping

Ping ist ein Programm, das ursprünglich 1983 von Mike Muus entwickelt wurde, um die Erreichbarkeit von Computern zu testen. Es sendet ein *ICMP-Echo-Request*-Paket an den Empfänger, der darauf mit einem *ICMP-Echo-Reply*-Paket antwortet. Der Name Ping stammt aus dem militärischen Bereich. Beim Einsatz von Sonar zur Unterwasserortung wird ein Schallsignal ausgesandt, das sich in einem U-Boot wie ein Glasklang anhört. Lautmalerisch wird dieses Geräusch als „ping“ bezeichnet.

Traceroute

Traceroute ist ein Algorithmus, der die Route zwischen zwei Rechnern des Internets inklusive aller Zwischenstationen bestimmen kann. Außerdem führt Traceroute Laufzeitmessungen zwischen dem aufrufenden Rechner und allen Rechnern entlang der Route durch.

Beim Aufruf von Traceroute werden hintereinander ICMP-Pakete zu einer Zieladresse gesendet. Danach wird auf eine ICMP-Fehler-Nachricht gewartet. In Gruppen von jeweils drei Paketen wird dabei ein TTL-Wert von 1, 2, usw. gesetzt. Anschließend werden die ICMP-Nachrichten der einzelnen Router ausgewertet, so dass am Ende eine Liste aller Router auf dem Weg zum Ziel zustandekommt. Neben den IP-Adressen werden auch die Laufzeiten ausgegeben. Manche Internet-Router senden allerdings kein Fehlerpaket zurück. In diesem Fall kann Traceroute weder eine Zeitmessung noch die IP-Adresse des Routers ausgeben. Nach einem *Timeout* sendet Traceroute dann einfach das nächste Paket (und gibt das Zeichen $\star$ auf dem Bildschirm aus).

2.3 Die Transportschicht: TCP und UDP

Wie wir in Kapitel 2.1 gesehen haben, besteht die Transportschicht aus den Protokollen TCP (Transmission Control Protocol) und UDP (User Datagram Protocol), welche wir im Folgenden näher betrachten werden. Den Schwerpunkt werden wir dabei auf TCP legen, so dass wir uns UDP nur kurz zuwenden.

2.3.1 UDP: User Datagram Protocol

UDP ist ein einfacher Dienst, der Pakete von einem Rechner zu einem anderen Rechner ohne Erfolgsgarantie verschickt. Im Gegensatz zu TCP muss die Anwendungsschicht die Paketgröße vorgeben. Aus diesen Datenpaketen wird ein Datagramm gebildet und durch die Netzwerkschicht versendet. Es sind weder Bestätigungen noch eine Datenflusskontrolle vorhanden. Es fällt daher schwer, UDP als eigenständiges Protokoll zu bezeichnen, da es nur die direkte Erzeugung von Datagrammen und den Eingang in der höheren Schicht zur Verfügung stellt. Es ist so wenig verlässlich wie der Pakettransport im Internet.

2.3.2 TCP: Transmission Control Protocol

Die ursprüngliche Spezifikation von TCP ist in RFC 793 [26] zu finden. Im Gegensatz zu UDP erzeugt TCP einen zuverlässigen Datenfluss zwischen zwei Rechnern. Hierfür werden die Datenströme aus der Anwendungsschicht in Pakete angemessener Länge unterteilt. Diese Pakete, wie auch Bestätigungsnachrichten (*Acknowledgments*) für erhaltene Pakete, werden dann durch die Vermittlungsschicht versandt. Die Zielsetzung von TCP wird durch folgenden Satz beschrieben: TCP ist ein verbindungsorientierter und zuverlässiger Dienst für bidirektionale Byteströme. Die drei in diesem Satz zusammengefassten Eigenschaften erläutern wir im Folgenden näher:

- *TCP ist verbindungsorientiert.*
  TCP bewerkstelligt ein so genanntes *Unicast*. Das ist eine Punkt-zu-Punkt-Verbindung zwischen zwei Parteien, d.h. TCP unterstützt keine *Multicast*-Kommunikation (siehe IP-Multicast auf Seite 52 und Seite 225). Bevor Daten übertragen werden, wird die Verbindung aufgebaut. Während des Verbindungsaufbaus wird sichergestellt, dass die Parteien sich gegenseitig erreichen können. Außerdem werden elementare Verbindungsparameter ausgetauscht. Solange die Verbindung nicht ordentlich beendet wird, bleibt sie bestehen (Aus Sicherheitsgründen wird hier mittlerweile mit einem *Timeout*-Mechanismus gearbeitet).
- *TCP ist zuverlässig.*
  Für die Bereitstellung einer zuverlässigen Verbindung wirken in TCP die folgenden Mechanismen. TCP unterteilt die Anwendungsdaten in *Segmente*, die auch *TCP-Pakete* genannt werden. Empfängt TCP ein Paket, sendet es dafür ein Bestätigungssegment (Acknowledgment), evtl. kombiniert mit einem Datensegment. Wird die Bestätigung eines TCP-Pakets nicht empfangen, wird das Paket erneut versendet. TCP erstellt für Daten- und Headerbereich des Segments eine Prüfsumme (*Checksum*). Stellt der Empfänger Unstimmigkeiten zwischen Segment und mitgelieferter Prüfsumme fest, wird das TCP-Paket ersatzlos und ohne weitere Aktion verworfen. TCP-Pakete werden als IP-Datagramme versandt. Da IP nicht immer Pakete in der richtigen Reihenfolge ausliefert, stellt TCP die richtige Reihenfolge der Pakete wieder her. TCP löscht automatisch Pakete, die doppelt auftauchen. Solche Kopien können durch IP entstehen.
- *TCP ist ein Dienst für bidirektionale Byteströme.*
  Zwischen den beiden TCP-Anwendungen werden Daten auf 8-bit-Basis (Bytes) scheinbar kontinuierlich ausgetauscht. Es gibt keine Übertragungsmarker, die TCP veranlassen, Daten an bestimmten Stellen zusammenzufassen oder zu trennen. Es findet grundsätzlich keine Interpretation der Anwendungsdaten statt. Auch das Zeitverhalten wird durch TCP nicht weitergegeben. Sendet beispielsweise die Anwendungsschicht zuerst 10 Bytes und später 70 Bytes, so können auf der Empfangsseite beispielsweise alle 80 Bytes auf einmal oder als vier 20-Byte Stückchen ankommen.
  TCP ermöglicht einen *Full-Duplex*-Dienst für die Anwendungsschicht. Das heißt, die Daten fließen in beiden Richtungen völlig unabhängig, so als ob sie durch zwei völlig unabhängige Verbindungen realisiert werden würden. Wir werden sehen, dass die hin- und rückfließenden Pakete tatsächlich gleichzeitig für beide Kommunikationsrichtungen benutzt werden.

Der TCP-Header

Jedes TCP-Segement besteht aus einem *Header* und den eigentlichen Daten, die übermittelt werden sollen. Der (mindestens) 20-Byte-Header eines TCP-Pakets besteht, wie in Abbildung 2.9 dargestellt, aus den folgenden Teilen.

Source Port
16 Bit Port-Nummer des Absenders. Diese Port-Nummer ist erforderlich, um

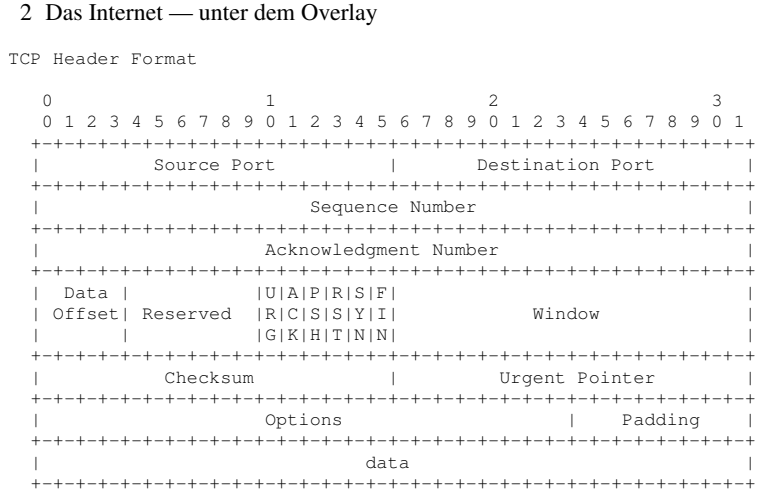


Abb. 2.9. TCP-Header-Definition aus RFC 793 [26].

über eine IP-Adresse gleichzeitig TCP-Verbindungen zu verschiedenen Diensten aufrecht zu halten. Entsprechend dieser Portnummer werden die TCP-Pakete den Anwendungen auf einem Rechner zugeordnet. Die eigentliche Absender-IP-Adresse findet sich im IP-Header. Die Kombination aus Port und IP-Adresse wird als *Socket* bezeichnet.

Destination Port

16 Bit Port-Nummer des Empfängers. Der Zweck dieser Port-Nummer ist analog zum Source Port, jedoch auf Empfängerseite. Ziel- und Quell-Socket bilden zusammen das so genannte *Socket-Pair*. Dieses Paar beschreibt eine TCP-Verbindung eindeutig.

Sequence Number

Anhand dieser 32 Bit Zahl wird die Reihenfolge der TCP-Pakete wiederhergestellt. In jeder Kommunikationsrichtung wird jedes Byte durchnummeriert (mit Umbruch bei $2^{32} - 1$ auf 0), und das erste Byte des Segments bestimmt die Sequenznummer des gesamten TCP-Pakets.

Acknowledgment Number

Mit der 32 Bit Bestätigungsnummer wird dem Sender einer Flussrichtung die Nummer des vom Empfänger als nächstes erwarteten Datenbytes mitgeteilt. Dieses Feld wird erst mit gesetztem *Acknowledgment*-Flag gültig.

Mit der Bestätigungsnummer wird jedes Paket bestätigt, für das die Sequenznummer plus Paketlänge kleiner ist als diese Bestätigungsnummer. Dadurch ist es insbesondere nicht (unmittelbar) möglich, einzelne Pakete zu bestätigen, die nach einem fehlenden Paket empfangen worden sind. Erscheint also ein bestimmtes Paket nicht in einem Datenstrom, so müssten eigentlich alle Pakete, die danach noch übermittelt worden sind, mit dem fehlenden Paket (das durch die Bestätigungsnummer beschrieben wird) erneut verschickt werden. Der *Fast-Retransmit*-Algorithmus umgeht dieses Problem, wie wir später sehen werden.

Data Offset
Das *Data Offset*-Feld gibt die Länge des TCP-Headers in 32 Bit Blöcken an und liefert somit die Startadresse der Nutzdaten des TCP-Segments.

Reserved
Das *Reserved*-Feld wird zurzeit nicht verwendet.

Flag-Bits
Die Flag-Bits sind binäre Variablen zur Kennzeichnung bestimmter für die Kommunikation und Weiterverarbeitung der Daten wichtiger Zustände. Die sechs Flags haben folgende Bedeutung:

URG
Das *Urgent*-Flag aktiviert den *Urgent Pointer*. Mit diesem Mechanismus wird der Empfänger über die Dringlichkeit von Daten informiert. Das Flag wird zumeist in Telnet und Remote Login-Verbindungen benutzt (Dieses Flag beschleunigt nicht etwa den Datenfluss, sondern spielt eine Rolle im interaktiven Zusammenspiel der gegenläufigen Datenflüsse).

ACK
Das *Acknowledgement-Number*-Flag aktiviert die Bestätigungsnummer.

PSH
Ein gesetztes *Push*-Flag zeigt dem Empfänger an, dass er die Daten so schnell wie möglich an die Anwendungsschicht weitergeben soll.

RST
Das *Reset*-Flag wird gesetzt, um anzuzeigen, dass ein empfangenes Segment einer TCP-Verbindung nicht korrekt zugeordnet werden konnte. Dies geschieht beispielsweise, wenn ein UDP-Paket an einem TCP-Port ankommt oder ein Port an einem Rechner nicht verwendet wird.

SYN
Mit dem *Synchronize*-Flag werden Sequenznummern beim Verbindungsaufbau synchronisiert.

FIN
Das *Finish*-Flag zeigt an, dass der Sender einer Flussrichtung aufhört, Daten zu senden. In der Gegenrichtung kann weiter übertragen werden.

Window
Das *Window*-Feld gibt an, wie viele Bytes mit noch nicht eingegangener Bestätigung maximal in einer Flussrichtung versendet werden dürfen. Die variierende Fenstergröße wird von TCP verwendet um die Datenrate anzupassen. Eine geringe Fenstergröße verursacht eine geringe Datenrate. Eine großes Fenster hingegen ermöglicht hohen Datendurchsatz, kann diesen jedoch nicht erzwingen.

Checksum
Die Prüfsumme dient zur Erkennung von Übertragungsfehlern und wird über den Header und die Daten berechnet.

Urgent-Pointer
Zusammen mit der Sequenz-Nummer gibt der *Urgent-Pointer* die genaue Position der Urgent-Daten im Datenstrom an.

Options
Das *Options*-Feld soll eine Möglichkeit bieten Funktionen bereitzustellen, die

im normalen TCP-Protokollkopf nicht vorgesehen sind. In TCP sind drei Optionen definiert: End of Option List, No-Operation und Maximum Segment Size. Die wichtigste dieser drei Optionen ist die Maximale Segmentgröße. Mit dieser Option kann ein Host die maximale Anzahl Nutzdaten übermitteln, die er annehmen will bzw. annehmen kann.

Padding
Das *Padding*-Feld wird lediglich verwendet um sicherzustellen, dass der Header an einer 32-Bit Grenze endet und die Daten an einer 32-Bit Grenze beginnen.

Data
Im *Data*-Feld stehen die eigentlichen Anwendungsdaten. Dieses Feld ist optional, und tatsächlich gibt es bei Verbindungsaufbau, Beendigung und gelegentlich bei Bestätigungen TCP-Pakete ohne dieses Datenfeld. Die Länge dieses Datenfelds ergibt sich aus der Gesamtlänge desjenigen Segments, das im Header des umgebenden IP-Datagramms mitgeteilt wird.

Verbindungsaufbau

Zumeist handelt es sich bei TCP-Verbindungen um eine *Client-Server*-Verbindung. Natürlich wird TCP auch in Peer-to-Peer-Netzwerken eingesetzt. Dann übernimmt ein Peer die Rolle des *Clients* und ein Peer die Rolle des *Servers*. Nach dem Verbindungsaufbau verhält sich TCP absolut symmetrisch.

Eine TCP-Verbindung wird durch drei Segmente aufgebaut, wie in Abbildung 2.10 dargestellt.

1. Im ersten Segment sendet der Client ein SYN-Segment mit der initialen Sequenznummer (und natürlich mit dem gewünschten Ziel-Port).
2. Der Server antwortet mit einem SYN-Segment mit seiner initialen Sequenznummer und bestätigt mit diesem Segment durch ACK-Flag und Bestätigungsnummer (= Sequenznummer des *Clients* $+1$) das erste Segment des Clients.
3. Der Client bestätigt das SYN-Segment des Servers durch ein ACK-Segment mit Bestätigungsnummer (= Sequenznummer des Servers $+1$).

TCP legt hierbei mit dem SYN-Segment auch die maximale Segmentgröße (*MSS*: *Maximum Segment Size*) fest. Dies geschieht dadurch, dass jeder Empfänger dem Sender die Segmentgröße, die er empfangen möchte, mitteilt. Problematisch ist bei diesem einfachen Protokoll, dass nicht getestet wird, ob diese Paketgröße auch tatsächlich übermittelt werden kann. Beispielsweise kann ein Router oder ein dazwischen genutztes Netzwerk hierbei gewissen Einschränkungen unterliegen. Es sei hier erwähnt, dass IPv4 Datagramme, deren Länge die maximale Übertragungseinheit einer Verbindung (*MTU: Maximum Transmission Unit*) überschreiten, zerlegt (fragmentiert) und am Zielrechner wieder zusammengesetzt werden. Ein solcher Vorgang verringert natürlich den Datendurchsatz und erhöht die Fehleranfälligkeit. In IPv6 werden zu große Datagramme gleich ganz gelöscht.

Es sei angemerkt, dass TCP neben dieser Art des Client-Server-Verbindungsaufbaus auch andere Varianten des Verbindungsaufbaus erlaubt [17], auf die wir hier

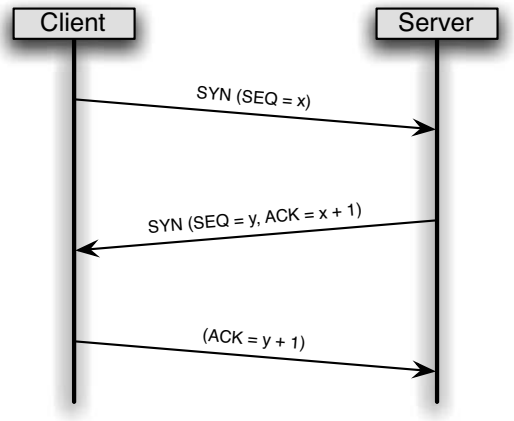


Abb. 2.10. Schematischer Verbindungsaufbau einer Client-Server TCP-Verbindung.

jedoch nicht eingehen. Stattdessen wenden wir uns nun dem Beenden von TCP-Verbindungen zu.

Verbindungsende

TCP erlaubt einen so genannten *Half-Close*, d.h., die gegenläufigen Datenströme können unabhängig voneinander beendet werden. Ist ein Datenstrom durch einen Half-Close beendet, können noch Daten in der Gegenrichtung übermittelt werden.

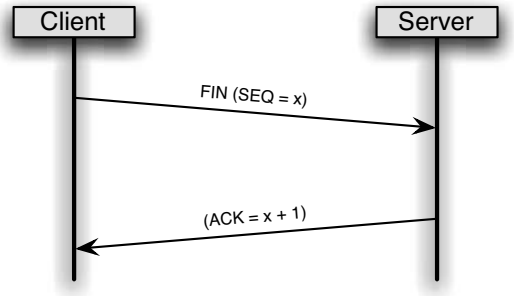


Abb. 2.11. Half-Close einer TCP-Verbindung.

Für einen solchen Half-Close werden zwei Segmente benötigt, siehe Abbildung 2.11:

1. Der Sender (einer Datenrichtung) verschickt ein FIN-Segment.
2. Der Empfänger bestätigt dies durch ein ACK-Segment.

Für das vollständige Schließen einer TCP-Verbindung werden wegen zweier Half-Close-Operationen daher vier Segmente benötigt, siehe Abbildung 2.12.

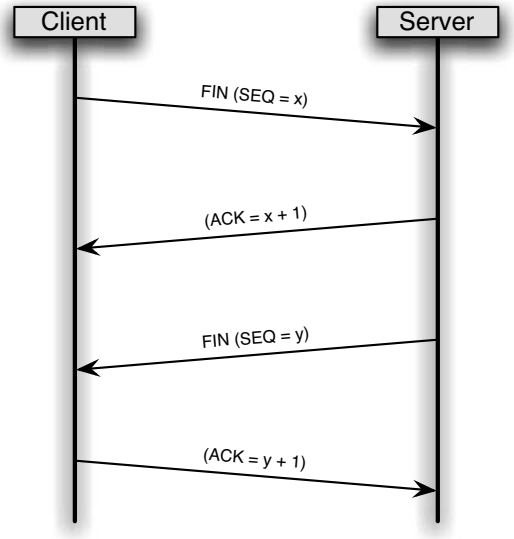


Abb. 2.12. Zwei entgegengesetzte Half-Close beenden eine TCP-Verbindung.

Bestätigungen und Fenster

Wie schon bei der Beschreibung des TCP-Headers erwähnt wurde, werden die Anwendungsdaten durch die Sequenznummern durchnummeriert. Jedes TCP-Paket mit Daten muss bestätigt werden. Pakete, die keine Daten tragen und nur aus Bestätigungen bestehen, werden nicht bestätigt. Diese verursachen natürlich auch keine Erhöhung der Sequenznummer.

TCP nutzt nun die gegenläufigen Datenströme, um Bestätigungen in einer Richtung per *Huckepack* mit Daten aus der Gegenrichtung zu befördern. Ferner müssen Segmente nicht einzeln bestätigt werden. Sind alle Segmente bis zum Datenbyte mit Sequenznummer i korrekt eingetroffen, wird lediglich eine einzige Bestätigungsnummer $i + 1$ zurückgeschickt, unabhängig davon, wie viele Pakete seit der letzten Bestätigung eingetroffen sind. Tatsächlich wird bei jedem Segment (außer bei Verbindungsaufbau und Ende) das ACK-Flag gesetzt und mit der Bestätigungsnummer das als nächstes erwartete Datenbyte mitgeteilt.

Um weitere Segmente einzusparen, arbeitet TCP mit verzögerten Bestätigungen (*Delayed Acknowledgements*). Hat etwa eine Seite keine Daten zu senden, wartet sie zwischen den reinen Bestätigungspaketen eine Mindestzeit (z.B. 200 ms), um einkommende Segmente zu sammeln.

Das *Packet-Forwarding* Protokoll von IP kümmert sich nicht um eine faire und effiziente Ausnutzung der Verbindungsressourcen. Diese Aufgabe muss die Transportschicht übernehmen. Wir stellen nun die verschiedenen Mechanismen hierfür vor.

Exponentielles Zurückweichen

TCP unterhält eine *Retransmission-Timer*-Variable, (*RTO – Retransmission Timeout*), die den minimalen Zeitraum (von z.B. einer Sekunde) vorgibt, nach dem ein verlorenes Segment als solches erkannt wird und es erneut verschickt werden darf. Bleibt also die Bestätigung für ein Paket aus, wartet TCP mindestens den in dieser Variablen beschriebenen Zeitraum und sendet dann das Paket nochmals.

Das Besondere hierbei ist, dass dieser Wert nach jedem erfolglosen Senden eines Pakets verdoppelt wird, bis er eine obere Grenze von 64 Sekunden erreicht hat und dort verbleibt. Erreicht die Bestätigung des Segments den Sender, wird die RTO-Variable auf ihren Anfangswert zurückgesetzt.

Mit diesem Mechanismus wird verhindert, dass bei einem Verbindungsverlust die Router mit Paketen bombardiert werden, die sowieso nicht ausgeliefert werden können. Ist der Verbindungsverlust nur kurzfristig, wird die TCP-Verbindung gehalten, ohne dass unnötig viele Pakete verschickt werden.

Der Algorithmus von Nagle

Eine weitere Technik, um die Paketanzahl auf Verbindungen mit geringem Datenfluss (wie z.B. in Telnet-Sitzungen) zu verringern, ist der *Algorithmus von Nagle*. Der Algorithmus verhindert den Versand von zu kleinen TCP-Paketen (kleiner als die Segmentgröße), solange noch Bestätigungen von verschickten TCP-Paketen ausstehen. Diese Daten werden gesammelt, bis sie die vereinbarte Segmentgröße erreichen oder bis die ausstehenden Bestätigungen eingetroffen sind.

Nagles Algorithmus führt so zu einer Selbsttaktung. Für schnelle Verbindungen werden mehr kleinere TCP-Pakete verschickt, während für langsame Verbindungen die Paketanzahl verringert wird. Nagles Algorithmus lässt sich deaktiveren, was in bestimmten Situationen Sinn ergibt (wenn z.B. die Bewegungen einer Computer-Maus übertragen werden müssen).

Schätzung der Umlaufzeit

Ein Paketverlust lässt in mehr als 99% aller Fälle den Schluss zu, dass es zu einem Datenstau (*Congestion*) auf einem Router gekommen ist [27]. Diese Router puffern nur sehr wenige Nachrichten und wenn diese kleinen Puffer voll sind, dann werden zusätzlich eintreffende Pakete gelöscht. Hier stellt sich die Frage, woher man weiß, dass ein Paket verloren ist und nicht etwa nur das ACK-Segment verzögert worden ist. Es gibt keine sichere Methode, Letzteres auszuschließen. TCP unterhält hierfür eine Stoppuhr, die die Zeitdauer zwischen Paket und Bestätigung überprüft. Dies ist

die so genannte Umlaufzeit (*RTT: Round Trip Time*). Dauert es wesentlich länger als die Umlaufzeit, bis ein Paket bestätigt wird, so gilt das Paket als verloren. Die Umlaufzeit unterliegt gewissen Schwankungen. Wichtig ist, sie nicht zu unterschätzen, damit Pakete, die etwas länger brauchen, nicht als verloren gemeldet werden. Ursprünglich [26] wurde der *Retransmission Timeout Value (RTO)* folgendermaßen bestimmt.

Zuerst wird, sobald ein neuer Messwert M für RTT verfügbar ist, ein geglätteter Schätzwert von RTT berechnet:

$$R \leftarrow \alpha R + (1 - \alpha) M \ ,$$

wobei $\alpha = 0,9$ gewählt wird. Mit Hilfe von R bestimmt sich RTO als

$$\text{RTO} \ = \ \beta R \ ,$$

wobei β als Varianzfaktor mit $\beta = 2$ empfohlen wird.

In [27] wird beschrieben, dass dieser Mechanismus nicht in der Lage ist, mit den enormen Umlaufzeitschwankungen fertig zu werden, was unnötigen Wiederversand von Daten verursacht. Daher wird folgende Methode vorgeschlagen: Wie zuvor wird mit jeder neuen Messung der Umlaufzeit M folgende Berechnung durchgeführt (nach geeigneter Initialisierung von A und D):

$$\begin{aligned} A &\leftarrow A + g\,(M - A) \ , \\ D &\leftarrow D + h\,(|M - A| - D) \ , \\ \text{RTO} &\leftarrow A + 4D \ . \end{aligned}$$

Hierbei werden die Werte $g = 1/8$ und $h = 1/4$ verwendet. In A wird eine geglättete Abschätzung für RTT gespeichert, während D die durchschnittliche Abweichung der Umlaufzeit RTT vom Schätzwert geglättet darstellt. Bei der Beschreibung dieser Berechnung standen eine Menge praktischer Fragestellungen im Vordergrund. Zum Beispiel sind alle Konstanten Zweier-Potenzen. Die Standardabweichung wurde nicht verwendet, weil sonst eine Quadratwurzel berechnet werden müsste.

Diese Aktualisierung darf nicht angewendet werden, wenn Pakete mehrfach übertragen worden sind, weil möglicherweise zwei Bestätigungen für ein Paket existieren und somit ein falscher Zeitraum M der Berechnung zugrunde liegen würde [28]. Dann wird der durch exponentielles Zurückweichen gewählte Wert von RTO für das nächste Paket noch einmal zugrunde gelegt.

Fenster

Die Datenratenanpassung von TCP basiert auf der Methode der gleitenden Fenster (*Sliding Windows*). Ein Parameter hierfür ist die Fenstergröße (*WND: Window Size*), mit der der Empfänger die Datenmenge des Senders steuern kann, wie in Abbildung 2.13 gezeigt wird.

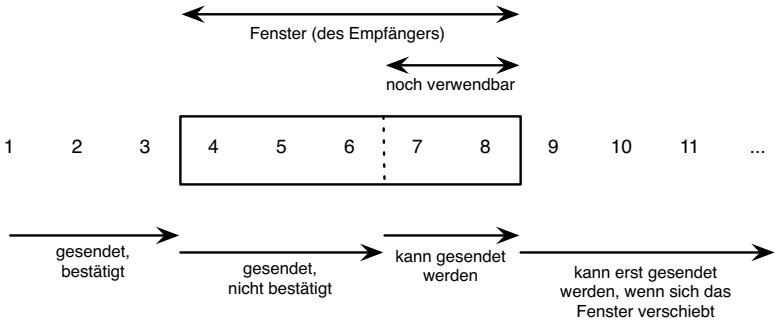


Abb. 2.13. Datenratensteuerung durch Fenster.

Wenn zum Beispiel die Eingangswarteschlange eines Empfängers voll ist, weil der Empfänger langsamer arbeitet als der Sender, dann wird mit dem nächsten Bestätigungspaket die Fenstergröße auf Null gesetzt. Der Sender darf dann kein weiteres Datenpaket senden, weil noch die Bestätigungen für die versandten TCP-Pakete ausstehen. Hat der Empfänger seine Warteschlange entsprechend abgearbeitet, wird er eine weitere Bestätigung schicken, die sich von der ersten nur durch die erhöhte (Anfangs-) Fenstergröße unterscheidet.

Wie in Abbildung 2.13 zu sehen, bezeichnen wir die Menge der nicht bestätigten Daten in einer Flussrichtung nun als „Fenster“. Die Fenstergröße wird vom Empfänger gesteuert und kann folgendermaßen verändert werden:

- Das Fenster *schließt* sich, wenn das linke Ende nach rechts wandert. Dies passiert, wenn Daten bestätigt werden.
- Das Fenster *öffnet* sich, wenn das rechte Ende nach rechts weiterläuft. Damit können mehr Daten gesendet werden. Dies geschieht insbesondere, wenn der Empfänger TCP-Pakete aus dem Puffer ausgelesen und damit Platz für neue Pakete geschaffen hat.
- Folgender Fall ist auch erlaubt, sollte aber nur in Ausnahmesituationen auftreten [29]: Das Fenster *schrumpft*, wenn sich das rechte Ende nach links bewegt. Der Empfänger verringert mit einer Bestätigung die Fenstergröße um mehr als die damit bestätigte Datenmenge. Eben hat der Empfänger noch signalisiert, dass er diese Pakete erhalten kann, und dann signalisiert er das Gegenteil.
  Von dieser Art der Fensterveränderung wird abgeraten. Dennoch müssen TCP-Protokolle sie unterstützen.

Slow-Start

Tatsächlich darf der Sender die angebotene Fenstergröße einer TCP-Verbindung nicht von Anfang an voll nutzen. In lokalen Netzwerken wäre dies in der Regel kein Problem, in größeren Netzwerken kann das jedoch auf den Routern zu Netzwerkstauungen und damit Datenverlust kommen.

Damit kommt ein weiterer Fenstermechanismus ins Spiel, nämlich das *Congestion-Fenster (CWND: Congestion Window)*, das in Vielfachen der Segmentgröße verwaltet wird. Im Gegensatz zum Fenster wird das Congestion-Fenster vom Sender verwaltet. Auch das Congestion-Fenster beschränkt die Anzahl der gesendeten unbestätigten Daten. Damit kann der Sender maximal $\min\{cwnd, wnd\}$ an unbestätigten Daten versenden oder schon versandt haben.

Der *Slow-Start*-Mechanismus arbeitet wie folgt:

- Beim Verbindungsaufbau wird das Congestion-Fenster auf die Paketgröße (gegeben durch die maximale Segmentgröße MSS) gesetzt.

$$cwnd \leftarrow \text{MSS}$$

  Damit kann anfangs nur ein (volles) TCP-Paket verschickt werden.
- Bei Bestätigung eines Pakets (nach Erhalt eines ACK-Segments), wird das Congestion-Fenster um die maximale Segmentgröße vergrößert:

$$cwnd \leftarrow cwnd + \text{MSS} .$$

Die maximal mögliche Fenstergröße ergibt sich aus der Kapazität, dem Produkt aus Bandbreite (Bytes/Sek.) und der Umlaufzeit (Sek.) (RTT):

$$\text{Maximale Fenstergröße (Bytes)} = \text{Bandbreite (Bytes/Sek.)} \times \text{Umlaufzeit (Sek.)} .$$

Zur Veranschaulichung dieses Algorithmus bezeichnen wir mit x die Anzahl der Pakete, die unter Berücksichtigung von *cwnd* innerhalb einer Umlaufzeit versendet werden dürfen, d.h.

$$x = \frac{cwnd}{\text{MSS}} .$$

Demnach gliedert sich *Slow-Start* in zwei Teile.

1. Zu Beginn setzen wir
$$x \leftarrow 1 .$$
2. Mit jedem Paket setzen wir $x \leftarrow x + 1$. Da dies bis zu x-mal geschehen kann, setzen wir im besten Fall:
$$x \leftarrow 2x .$$

Damit realisiert Slow-Start im optimalen Fall (d.h. alle Pakete werden bestätigt und *wnd* ist ausreichend groß) exponentielles Wachstum der Datenrate.

Stauvermeidung

Es genügt aber nicht nur, die Datenrate am Anfang langsam zu steigern. Wenn große Datenmengen übertragen werden müssen und damit die maximale Segmentgröße immer voll ausgeschöpft wird, passt TCP die Datenrate kontinuierlich an, damit in den Routern keine Staus entstehen. Wir stellen diesen Mechanismus zunächst vor und beschreiben dann, warum er gerade so gewählt worden ist.

Die entscheidenden Parameter sind die Congestion-Fenstergröße *cwnd* und ein Slow-Start-Schwellwert *ssthresh* (*Slow Start Threshold Size)*. Wir bezeichnen mit x wieder die Anzahl der vollen Pakete pro Umlaufzeit und mit y den Slow-Start-Schwellwert in Vielfachen der maximalen Segmentgröße, d.h. $y = \mathit{ssthresh}/\text{MSS}$.

1. Beim Verbindungsaufbau wird das Congestion-Fenster auf die maximale Segmentgröße (MSS) gesetzt
$$\mathit{cwnd} \leftarrow \text{MSS}$$
und der Slow-Start-Schwellwert auf die maximale Fenstergröße:
$$\mathit{ssthresh} \leftarrow 2^{16} - 1 = 65535\ .$$
In der Notation mit x und y bedeutet dies:
$$\begin{aligned} x &\leftarrow 1\ , \\ y &\leftarrow \max\ . \end{aligned}$$

2. Kommt es zu einem Paketverlust, d.h., eine Bestätigung eines Pakets erreicht den Sender nicht innerhalb des aktuell berechneten Zeitraums RTO oder eine Reihe von Acknowledgements desselben Pakets trifft ein, dann wird ein Datenstau (*Congestion*) angenommen.
Nun werden folgende Aktualisierungen durchgeführt:
$$\begin{aligned} \mathit{ssthresh} &\leftarrow \max\left\{2\ \text{MSS}, \frac{\min\{\mathit{cwnd}, \mathit{wnd}\}}{2}\right\}\ , \\ \mathit{cwnd} &\leftarrow \text{MSS}\ . \end{aligned}$$
Wenn wir wieder annehmen, dass *wnd* ausreichend groß ist und dass *cwnd* mindestens 4MSS ist, bedeutet das:
$$\begin{aligned} x &\leftarrow 1\ , \\ y &\leftarrow \frac{x}{2}\ . \end{aligned}$$

3. Werden Daten bestätigt und ist $\mathit{cwnd} \leq \mathit{ssthresh}$, wird Slow Start durchgeführt. Wird also ein Paket bestätigt, erhalten wir
$$\mathit{cwnd} \leftarrow \mathit{cwnd} + \text{MSS}\ .$$
Werden alle Pakete innerhalb der Zeit RTO bestätigt, verdoppelt sich das Fenster innerhalb der Umlaufzeit (RTT). Setzen wir $x = \mathit{cwnd}/\text{MSS}$, stellt sich also nach $\mathcal{O}(\log x)$ Umlaufzeiteinheiten (Vielfachen von RTT)
$$x \leftarrow \frac{x}{2}$$
ein.

4. Werden Daten bestätigt und ist $cwnd > ssthresh$, befindet sich der Algorithmus in der *Congestion-Avoidance*-Phase. Hier wird die Congestion-Fenstergröße langsamer erhöht:

$$cwnd \leftarrow cwnd + \frac{\text{MSS}^2}{cwnd} .$$

Dabei handelt es sich um eine lineare Zunahme der Datenmenge, wie folgende Überlegung zeigt: Die Gesamtmenge an Daten, die innerhalb der Umlaufzeit RTT übertragen wird, ist *cwnd*. Die Datenrate ist damit $w = cwnd/\text{RTT}$. Diese Menge wird nun für jedes der *cwnd* TCP-Pakete innerhalb der Umlaufzeit RTT um

$$\frac{\text{MSS}^2}{cwnd} \cdot \frac{cwnd}{\text{MSS}} = \text{MSS}$$

Bytes erhöht.[1] Damit wiederum erhöht sich die Datenrate um

$$d = \text{MSS} \,/\text{RTT} ,$$

d.h. pro Umlaufzeit nur um ein Paket. Wir erhalten also

$$x \leftarrow x + 1 .$$

Dieses additive Erhöhen und multiplikative Verringern der Datenrate nennt man auch *AIMD* (*Additive Increase/Multiplicative Decrease*). Auf Seite 41 werden wir die Motivation für dieses (auch unter dem Namen *TCP Tahoe* bekannte) Verfahren kennenlernen.

Geht nun nur ein einziges TCP-Paket verloren, so hat das drastische Auswirkungen auf das Übertragungsverhalten einer TCP-Verbindung. So wird nicht nur die Datenrate auf den kleinstmöglichen Wert gesetzt. Auch müssen alle TCP-Pakete, die nach dem Paket verschickt worden sind, wegen des Bestätigungsmechanismus als verloren angesehen werden. Deswegen wurde im Jahr 1990 in [30] das *Fast-Retransmit*- und *Fast-Recovery*-Verfahren vorgeschlagen.

Der Bestätigungsmechanismus in TCP erlaubt es nicht Pakete zu bestätigen, die nach einem verlorenen Paket liegen. Jetzt möchte der Empfänger dennoch signalisieren, dass sinnvolle Informationen nach der Lücke eingegangen sind. Daher wiederholt der Empfänger für jedes eingehende Paket die Nummer des Bytes des ersten fehlenden Pakets (was als Bestätigungsnummer für das letzte Paket vor der Lücke interpretiert werden kann). Der *Fast-Retransmit*-Mechanismus tritt in Aktion, wenn drei Bestätigungen derselben Bestätigungsnummer (*Triple Duplicate ACK*) beim Sender eingegangen sind. Dies signalisiert dem Sender, dass nach einem fehlenden TCP-Paket weitere TCP-Pakete empfangen wurden. Der Sender schickt daher als nächstes Paket nochmals jenes, das der dreimal wiederholten Bestätigungsnummer entspricht. Ansonsten verschickt der Sender keine weiteren Kopien, sondern schickt neue TCP-Pakete, sofern die Fenstergröße dies noch zulässt.

[1] Natürlich müsste man hier eigentlich verschiedene Werte für *cwnd* zugrunde legen. Hiermit würde sich das Ergebnis aber nur um einen kleinen konstanten Faktor ändern.

Der *Fast-Recovery*-Mechanismus verhindert nun nach dem Empfangen eines *Triple-Duplicate-Ack* einen *Slow-Start*. Vielmehr wird die Congestion-Fenstergröße direkt auf die Hälfte der ursprünglichen Fenstergröße gesetzt. Außerdem wird jede weitere Kopie einer Bestätigung als Bestätigung eines späteren Pakets interpretiert und damit das Congestion-Fenster um ein Segment erweitert.

Im Einzelnen wird *Fast-Retransmit* und *Fast-Recovery* wie folgt implementiert. Die folgenden Schritte werden statt der zweiten und dritten Komponente des *Congestion-Avoidance* angewendet, sofern eine dreifache Bestätigung desselben Pakets festgestellt wird. Das nun folgende Protokoll ist unter dem Namen *TCP Reno* bekannt.

1. Wird eine dritte Bestätigung desselben Segments erhalten, setzen wir

$$ssthresh \leftarrow \frac{\min\{cwnd, wnd\}}{2}$$

   und senden das fehlende Segment ein zweites Mal. Außerdem wird die Congestion-Fenstergröße wie folgt gesetzt:

$$cwnd \leftarrow ssthresh + 3\ \text{MSS}$$

2. Trifft danach eine weitere Bestätigung desselben Segments ein, setzt der Sender

$$cwnd \leftarrow cwnd + \text{MSS}$$

   und versucht, falls es die Fenstergrößen erlauben, ein weiteres Segment zu verschicken.
3. Erscheint nach der Folge der Bestätigungen desselben Segments eine Bestätigung eines anderen Segments, bestätigt dies den Erhalt des verlorenen Pakets. Nun wird das Congestion-Fenster wie folgt gesetzt:

$$cwnd \leftarrow ssthresh$$

   Damit befinden wir uns wieder im *Congestion-Avoidance*-Mechanismus, da wir ab jetzt nur die Hälfte der ursprünglichen Datenrate verwenden.

AIMD

In diesem Abschnitt motivieren wir den AIMD-Mechanismus (*Additive Increase/Multiplicative Decrease*) zur Stauvermeidung (*Congestion Avoidance*) gemäß [31]. In Abbildung 2.14 wird gezeigt, wie Datenlast (*load*) mit Antwortzeit und Datendurchsatz typischerweise in einem Netzwerk interagieren. Der Datendurchsatz ist maximal, wenn die angefragte Last die Netzwerkkapazität erreicht. Wird die Last weiter erhöht, nimmt der Datendurchsatz ab, da Datenpuffer überlaufen und Pakete verloren gehen und somit mehrfach versendet werden müssen. Dadurch steigt auch die Antwortzeit extrem an. Diese Situation einer Datenstauung wird *Congestion* genannt.

Statt in einem Netzwerk extreme Lastanforderung zu stellen, empfiehlt es sich, die Last auf das so genannte *Knie* einzustellen. Beim Knie steigt die Antwortzeit erstmals an, während der Datendurchsatz schon nahe der Netzwerkkapazität ist. Eine gute Stauvermeidungsstrategie (*Congestion Avoidance*) versucht, die Datenlast des Netzwerks bei diesem Knie zu halten. Neben dem Datendurchsatz ist es aber auch wichtig, dass die Datenraten aller Teilnehmer gleich sind, also *fair* gewählt werden.

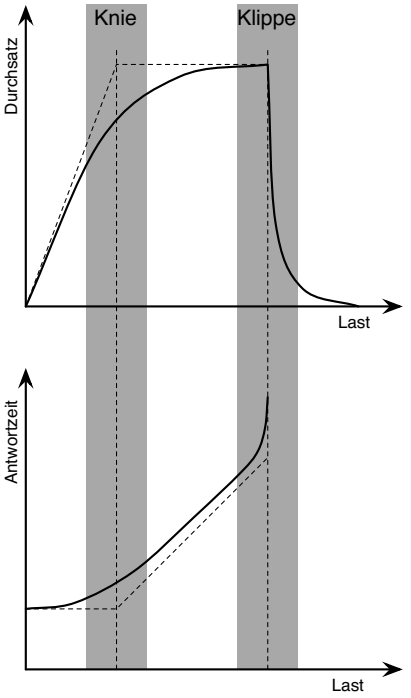


Abb. 2.14. Netzwerkverhalten in Abhängigkeit der Datenlast.

Für ein einfaches Netzwerkmodell kann man das gutartige Verhalten von TCP sogar nachweisen. Hierzu nehmen wir an, dass n Nutzer teilnehmen und B die verfügbare Bandbreite ist. Das System wird in Runden modelliert. Zu Beginn (d.h. in Runde 0) hat Teilnehmer $i \in \{1, \ldots, n\}$ Datenrate $x_i(0) \in [0, B]$. In jeder weiteren Runde t kann der Teilnehmer seine Datenrate auf $x_i(t) \in [0, B]$ setzen. In Runde $t + 1$ steht ihm, neben seiner alten Datenrate, nur die Information zur Verfügung, ob in der Runde t die Summe der Datenraten das Ziel $K \in [0, B]$ überschritten hat, d.h. ob

$$\sum_{i=1}^{n} x_i(t) \;\leq\; K$$

wobei K für die Datenlast an der oben beschriebenen Knieposition steht. Wir bezeichnen mit dem Prädikat $y(t) = 0$, dass diese Ungleichung in Runde t erfüllt wird und mit $y(t) = 1$ das Gegenteil.

Jeder Teilnehmer bestimmt nun in Runde $t + 1$ aufgrund von $y(t)$ und seiner alten Datenratenwahl $x_i(t)$ die Datenrate der nächsten Runde. Diese Entscheidung beschreiben wir durch eine Funktion $f : [0, B] \times \{0, 1\} \to [0, B]$, wobei für alle $i \in \{1, \ldots, n\}$ gilt

$$x_i(t + 1) \;=\; f(x_i(t), y(t)) \; .$$

Das bedeutet insbesondere, dass alle Teilnehmer sich an dieselbe Datenratenanpassungsstrategie f halten. Damit wenden alle Teilnehmer in Runde $t + 1$ entweder die Funktion

$$f_0(x) \;=\; f(x, 0) \qquad \text{oder} \qquad f_1(x) \;=\; f(x, 1)$$

an, und es ergibt sich die Grundanforderung an f_0 und f_1:

- Wenn $y(t) = 0$, also $\sum_{i=1}^{n} x_i(t) \;\leq\; K$, sollen Teilnehmer ihre Datenrate geeignet erhöhen, d.h. $f_0(x) \;>\; x$.
- Wenn $y(t) = 1$, also $\sum_{i=1}^{n} x_i(t) \;>\; K$, sollen Teilnehmer ihre Datenrate geeignet verringern, d.h. $f_1(x) \;<\; x$.

Hierbei ist nicht unmittelbar klar, ob diese Eigenschaft für alle Spieler gelten muss. Wir möchten daher der Wahl der Funktionen f_0 und f_1 eine Funktionenklasse zugrunde legen und diese gemäß einiger Schlüsselkriterien untersuchen.

In diesem Abschnitt beschränken wir uns für die Wahl der Funktionen f_0 und f_1 auf lineare Funktionen:

$$f_0(x) = a_I + b_I x \qquad \text{und} \qquad f_1(x) = a_D + b_D x \; .$$

Folgende Spezialfälle sind besonders interessant. Wir werden sie daher gesondert betrachten:

1. *Multiplicative Increase/Multiplicative Decrease (MIMD)*

   $$f_0(x) = b_I x \qquad \text{und} \qquad f_1(x) = b_D x \; ,$$

   wobei $b_I > 1$ und $b_D < 1$.
2. *Additive Increase/Additive Decrease (AIAD)*

   $$f_0(x) = a_I + x \qquad \text{und} \qquad f_1(x) = a_D + x \; ,$$

   wobei $a_I > 0$ und $a_D < 0$.
3. *Additive Increase/Multiplicative Decrease (AIMD)*

   $$f_0(x) = a_I + x \qquad \text{und} \qquad f_1(x) = b_D x \; ,$$

   wobei $a_I > 0$ und $b_D < 1$.

Die Schlüsselkriterien sind *Effizienz* und *Fairness*. Die Datenraten verschiedener Teilnehmer sind absolut fair, wenn $x_i(t) = x_j(t)$ für alle i, j gilt. Die Effizienz wird durch die Nähe der Gesamtlast

$$X(t) := \sum_{i=1}^{n} x_i(t)$$

an der Knielast K gemessen. Ist $X(t) > K$, verlängert sich die Antwortzeit, und ist sogar $X(t) > B$, gehen Datenpakete verloren und müssen erneut versendet werden. Ist dagegen $X(t) < K$, entstehen Opportunitätskosten dadurch, dass die Netzwerkkapazität nicht ausgenutzt wird.

Für zwei Teilnehmer kann man dieses System sehr gut grafisch untersuchen. In Abbildung 2.15 werden die Datenrate x_1 des ersten Teilnehmers und die Datenrate x_2 eines zweiten Teilnehmers als Punkt (x_1, x_2) dargestellt. Die *Effizienzlinie* bezeichnet alle Punkte, die die Netzwerklast X im optimalen Bereich K halten, d.h. $x_1 + x_2 = K$. Punkte unter dieser Linie stehen für Netzwerkzustände, die das Netzwerk zu wenig belasten, Punkte darüber für eine Überlastung des Netzwerks. Mit der *Fairnesslinie* werden alle Punkte mit $x_1 = x_2$ dargestellt. Je nachdem, ob man über oder unter dieser Linie steht, erhält x_2 oder x_1 eine höhere Datenrate.

Für Datenraten (x_1, x_2) bezeichnen Datenraten $(x_1 + c, x_2 - c)$ Netzwerkzustände gleicher Effizienz, da $X = x_1 + x_2$ konstant bleibt. Multipliziert man dagegen die Datenraten mit einem Faktor $c > 0$, so erhält sich die Fairness, wie man hier sieht:

$$\begin{aligned} F(cx_1, cx_2) &= \frac{(cx_1 + cx_2)^2}{2((cx_1)^2 + (cx_2)^2)} \\ &= \frac{(x_1 + x_2)^2}{2((x_1)^2 + (x_2))^2} \\ &= F(x_1, x_2) \; . \end{aligned}$$

In Abbildung 2.16 sind die Punkte gleicher Effizienz und gleicher Fairness wie (x_1, x_2) dargestellt.

Mit Hilfe dieses Diagramms kann man nun leicht die Schwächen der AIAD- und MIMD-Strategie beschreiben. Wählt man die AIAD-Strategie (*Additive Increase/Additive Decrease*), so bewegen sich die Datenraten auf einer Linie $(x_1 + y, x_2 + y)$. Hiermit ist es zwar möglich, sich der Effizienzlinie bis auf einen additiven Term von $\max\{a_I, -a_D\}$ (entspricht der Oszillationsamplitude) anzunähern, es ist aber nicht möglich die Fairness zu verbessern (Abbildung 2.17).

Bei der MIMD-Strategie (Abbildung 2.18) können nur Zustände erreicht werden, die durch die Gerade (cx_1, cx_2) für ein variables c beschrieben werden. Alle Punkte dieser Gerade stehen für Netzwerkzustände gleicher Fairness. Es ist also unmöglich, mit MIMD die Fairness zu verbessern. Daneben kann der Schnittpunkt mit der Effizienzlinie nur durch den Faktor $\max\{b_I, 1/b_D\}$ approximiert werden.

Abbildung 2.19 zeigt, dass die AIMD-Strategie für zwei Spieler anscheinend den Kriterien von Fairness und Effizienz genügt. In [31] wurde diese Beobachtung für n Spieler verallgmeinert:

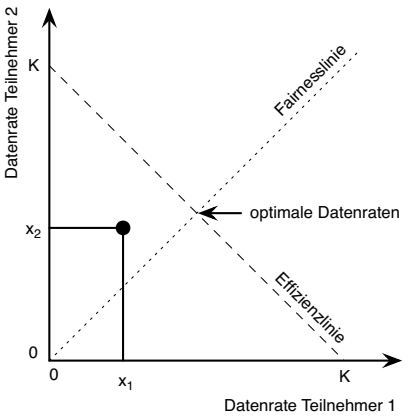


Abb. 2.15. Vektordiagramm für zwei Teilnehmer mit Datenrate x_1, x_2.

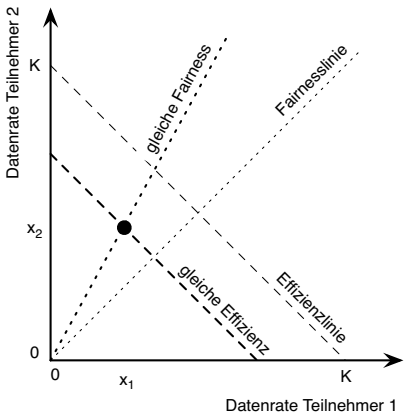


Abb. 2.16. Stellen gleicher Effizienz oder gleicher Fairness wie (x_1, x_2).

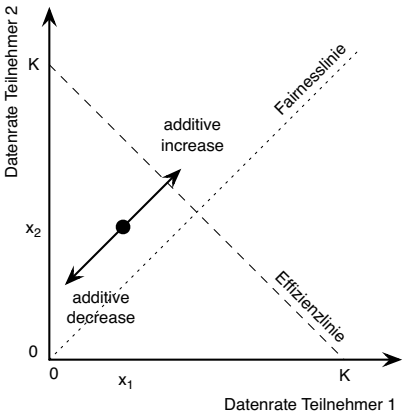


Abb. 2.17. Vektordiagramm der additive increase/additive decrease-Strategie (AIAD).

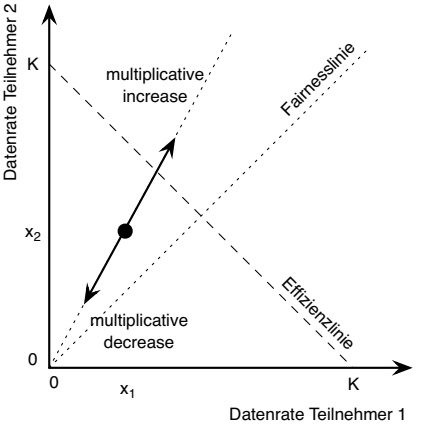


Abb. 2.18. Vektordiagramm der multiplicative increase/multiplicative decrease-Strategie (MIMD).

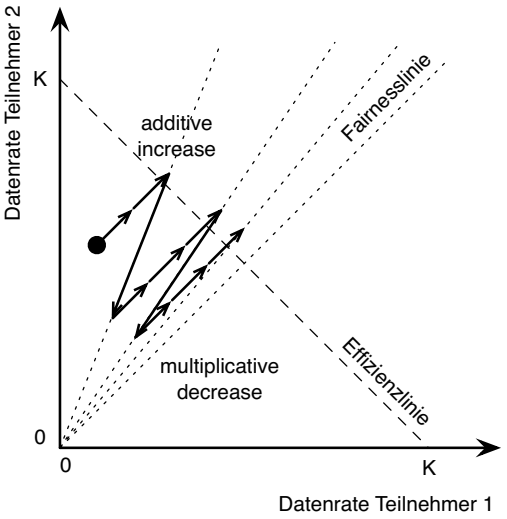


Abb. 2.19. Vektordiagramm des AIMD-Algorithmus.

Theorem 2.2. *Werden für f_0 und f_1 lineare Funktionen gewählt, die Fairness und Effizienz gewährleisten, muss für $x \geq 0$ gelten:*

$$f_0(x) \geq x + a_I \,, \qquad \textit{wobei } a_I > 0 \textit{ und}$$
$$f_1(x) = b_D x \,, \qquad \textit{wobei } 0 < b_D < 1 \,.$$

Ein Beweis findet sich in [31]. Wir erkennen also, dass die Datenratenerhöhung einen additiven Anteil und die Datenratenverkleinerung einen multiplikativen Anteil haben muss, was die Wahl einer AIMD-Strategie rechtfertigt.

Diese Stauvermeidungsstrategie wird als *TCP Reno* bezeichnet. Sie hat sich aus *TCP Tahoe* entwickelt, das noch ohne *Fast Retransmit* arbeitet. Eine neuere Variante ist das so genannte *TCP Vegas*. In dieser Stauvermeidungsstrategie wird neben der Information über den Paketverlust auch die *Umlaufzeit* (RTT) verwendet. Damit soll die Benachteiligung langsamer Verbindungen bei TCP gelöst werden.

Die genannten Stauvermeidungsmechanismen beruhen darauf, dass alle Teilnehmer TCP zur Datenübertragung verwenden. Nun basiert das Internet nicht auf einer genormten Software, sondern vielmehr auf der korrekten Anwendung der Protokolle. Da es sich bei TCP um ein *End-to-End*-Protokoll handelt, kann man nicht überprüfen, ob die Teilnehmer nach diesen Normen handeln. Außerdem kann man auch innerhalb der von den *Request-for-Comments* (*RFC*) vorgegebenen Regeln diese Stauvermeidungsstrategie aushebeln. Beispielsweise lässt sich aus der Anwendung heraus eine Folge von UDP-Paketen verwenden, die dann mit einer eigenen Flusskontrolle (wie zum Beispiel AIAD) arbeiten. Wenn nun eine aggressivere Flusskontrolle auf TCP stößt, dann hat das zur Folge, dass alle TCP-Verbindungen benachteiligt werden und bei Ressourcenknappheit überhaupt nicht mehr zum Zuge kommen.

Andererseits gibt es Anwendungen, die nicht oder nur schlecht mittels TCP erstellt werden können. Das ist zum Beispiel beim Lookup von Indexdateien in Peer-to-Peer-Netzwerken der Fall. Der Entwickler eines Peer-to-Peer-Netzwerks muss sich hier einer besonderen Verantwortung bewusst werden: Da sich die Bandbreitenkonflikte an der schlechtesten Verbindung ausprägen, leiden hierunter zuerst diejenigen Endbenutzer, bei denen die Peer-to-Peer-Netzwerk-Verbindung mit anderen Verbindungen wie etwa FTP oder HTTP konkurriert.

2.4 Das Internet im Jahr 2007

Abschließend werden wir in diesem Kapitel noch einige neuere Dienste und Entwicklungen im Internet vorstellen, die gerade für Peer-to-Peer-Netzwerke von besonderem Interesse sind.

Network und Port Address Translation (NAT und PAT)

Wie bereits erwähnt, gibt es eine allgemeine Knappheit von IPv4-Adressen. Als Beispiel betrachten wir einen privaten Anwender, der von seinem *Internet-Service-Provider* eine IP-Adresse zugewiesen bekommen hat. Nun stellt sich heraus, dass ein zweiter Rechner oder ein Netzwerkdrucker in sein privates Teilnetzwerk aufgenommen werden soll. Jetzt müssten diese entweder eigene IP-Adressen erhalten, oder der private Anwender müsste sich eine eigene Teilnetzadresse besorgen und dann diese Geräte darin zuweisen. Er hat aber andererseits kein Interesse, den Drucker über das Internet für alle erreichbar zu machen.

Network Address Translation (*NAT*) liefert hierfür eine elegante Lösung, indem es davon ausgeht, dass ein Router den Netzwerkverkehr zwischen Teilnetz und Rest-Internet kontrolliert. Nun werden für das lokale Netz IP-Adressen aus einem speziellen Adressraum vergeben (beginnend mit 10 oder 192.168). Kommt jetzt ein Paket von einem dieser Teilnehmer, so wird diese IP-Adresse übersetzt in eine von den wenigen für das Teilnetzwerk vorhandenen globalen Internet-Adressen. Versucht nun ein externer Teilnehmer einen lokalen Rechner zu erreichen, so wird die Adresse am Router rückübersetzt. Auf diese Weise kann man (fast) beliebig viele lokale Adressen vergeben, solange die Anzahl der Clients die Menge der externen IP-Adressen nicht überschreitet.

Nun löst dies aber nicht das eingangs formulierte Problem, falls nur eine Internet-Adresse zur Verfügung steht. Die Lösung liefert die *Port Address Translation (PAT)*, bekannt auch als *Network Address Port Translation (NAPT)*, *Hiding NAT* oder *Masquerading*. Hier wird der Socket, d.h. die Internet-Adresse und die Port-Nummer, übersetzt. Beim Verbindungsaufbau aus dem Teilnetzwerk heraus wird die lokale Netzwerkadresse und die korrekte Port-Nummer in ein neues Socket übersetzt. Dieses besteht aus der verfügbaren Netzwerkadresse und einer neuen Port-Nummer. Bei eingehenden Verbindungen wird das entsprechende Socket-Paar wieder rückübersetzt. Diese Zuordnungen werden in einer lokalen Tabelle im Router auf der Schnittstelle zwischen lokalem Teilnetz und Internet gespeichert und fortwährend aktualisiert.

Problematisch ist in beiden Fällen der Versuch eines auswärtigen Teilnehmers, Rechner in einem Netzwerk hinter einem NAT oder PAT erstmals anzusprechen. Dann liegen nämlich beim Router noch keine Einträge vor und der Router kann nicht wissen, wohin diese Anfrage weiterzuleiten ist. Dies kann man nur dadurch lösen, dass wichtige Port-Nummern standardmäßig an bestimmte Rechner weitergeleitet werden. So kann man auch einen Mail-Server oder einen HTML-Server hinter einem NAT/PAT-Router betreiben.

Die Unfähigkeit, Rechner hinter einem NAT-Router direkt anzusprechen, hat andererseits den Vorteil, dass Virus- oder Denial-of-Service-Attacken auf Rechner in einem lokalen Netzwerk ins Leere laufen: Alle Pakete bleiben an dem NAT/PAT-Router hängen.

Letztlich stellen NAT und PAT deswegen für Peer-to-Peer-Netzwerke ein großes Hindernis dar: Kein Peer kann eine Verbindung für zwei benachbarte Peers hinter einem NAT/PAT vermitteln. Der Grund ist, dass diese beiden Rechner in den jeweiligen Tabellen nicht vorhanden sind und nicht angelegt werden können, weil dies voraussetzt, dass ein erstmaliger Kontakt vorhanden ist. Bei diesem Versuch scheitert der eine Peer an dem anderen Router und umgekehrt. Es ist in der Regel noch nicht einmal möglich, die auswärts verwendeten Port-Adressen zu ermitteln. Daher muss auch für Peer-to-Peer-Netzwerke, wie auch schon für Mail-Server, eine Ausnahmeregelung an dem Router programmiert werden. Hat der Teilnehmer nicht die Befugnisse dies durchzuführen, dann kann er nur neue Peer-to-Peer-Verbindungen zu Rechnern ohne NAT und PAT aufnehmen. Dadurch wird die Effizienz der Peer-to-Peer-Netzwerke stark eingeschränkt, siehe auch Abbildung 2.20.

Dynamic Host Configuration Protocol (DHCP)

Eine weitere Methode, die Problematik der knappen IPv4-Adressen zu lösen, ist das sogannte *Dynamic Host Configuration Protocol (DHCP)*. Die dynamische Zuweisung von IPv4-Adressen taucht zum ersten Mal beim *Boot-Protocol (BOOTP)* auf. Dieses Protokoll wurde für festplattenlose Rechner entworfen, die mittels einer Netzwerkverbindung mit den Daten der Festplatte eines weiteren Rechners starten (booten). Hierzu muss der festplattenlose Rechner zuerst seine IP-Adresse erfahren. Schließlich konnten mehrere Rechner derart gestartet werden.

Beim DHCP entkoppelt man die Adress-Zuweisung vom Boot-Prozess. Der bereits gestartete Rechner kann seine IP-Adresse von einem DHCP-Server erfragen. Nur die Adresse dieses Rechners muss zuvor bekannt sein. Man unterscheidet hier zwischen einer manuellen Zuordnung, einer automatischen, festen Zuordnung und einer dynamischen Zuordnung. Bei der manuellen Zuordnung wird z.B. durch eine Bindung an die MAC-Adresse (*Medium Access Layer*) jeweils eine feste IP-Adresse vergeben. (Die MAC-Adresse ist in den meisten Geräten eindeutig durch den Hersteller festgelegt. Tatsächlich kann auch sie in einigen Geräten manipuliert werden.) Häufiger findet man aber eine automatische Zuordnung. Hier wird einem Rechner eine eigene IP-Adresse zugewiesen, welche dieser auch erhält, falls er für einige Zeit nicht aktiv ist. Diese Konfiguration ist typisch für Firmen-Netzwerke, in denen die Rechneranzahl bekannt ist.

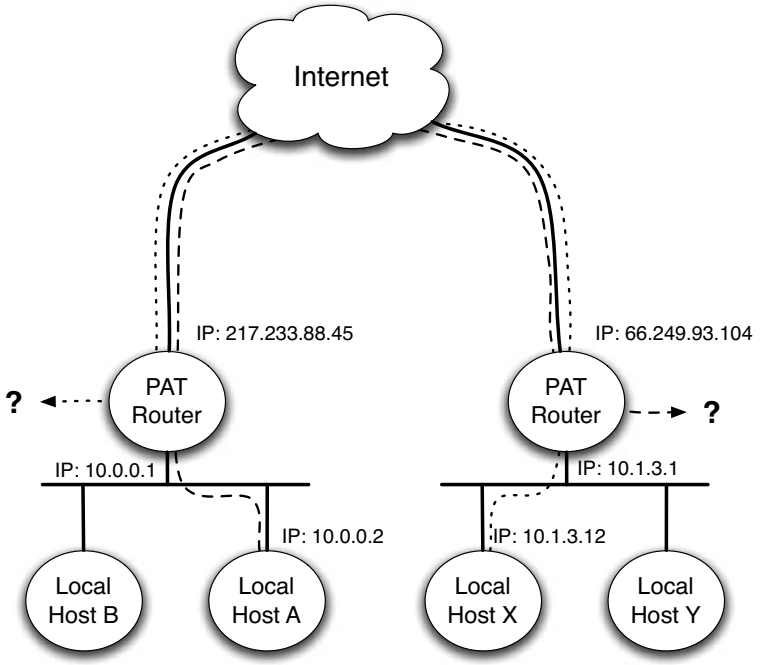


Abb. 2.20. Befinden sich zwei Rechner jeweils hinter dem PAT-Router, ist ein direkter Verbindungsaufbau nicht möglich.

Am Häufigsten findet man die dynamische Zuordnung bei Internet-Service-Providern, die eine große, schwankende Anzahl von Teilnehmern verwalten. Hierbei wird jedem Teilnehmer nach jeder Anmeldung dynamisch eine freie Adresse zugewiesen. Mitunter werden die Teilnehmer auch gezwungen, sich vom Netzwerk zu trennen und wieder anzumelden. Erhält der DHCP-Server einige Zeit keine Rückmeldung von einem Rechner, kann es passieren, dass die IP-Adresse neu vergeben wird. Versucht der Rechner dann diese Adresse erneut zu verwenden, kommt es zu einem Konflikt, der durch die Vergabe einer neuen IP-Adresse gelöst werden muss. Man spricht in diesem Zusammenhang auch vom „Stehlen“ von IP-Adressen.

Ein Vorteil von DHCP ist, dass man die knappen Netzwerkadressen eines Teilnetzwerks effizienter vergeben kann. Ein weiterer Vorteil ist, dass die Teilnehmer sich anonym im Internet aufhalten können. Von diesem Vorteil profitieren die meisten Privatnutzer, da DHCP zum Standard geworden ist, siehe Abbildung 2.21. Andererseits führen die Telekommunikationsunternehmen Log-Dateien, in denen die vergebenen IP-Adressen mit den Anschlussnummern (Telefonnummern) und der Zeit dokumentiert werden. Abhängig von der nationalen Gesetzeslage sind die Unternehmen berechtigt oder verpflichtet, diese zu führen und den Untersuchungsbehörden zu veröffentlichen.

Letztlich stellt DHCP ein Hindernis dar, einen bestimmten Peer wiederzufinden, nachdem er vom Peer-to-Peer-Netzwerk getrennt wurde. Es wird notwendig, eine eigene Identifikation für das Peer-to-Peer-Netzwerk zu führen, die unabhängig von der IP-Adresse ist. Zumeist treten DHCP und NAT/PAT gemeinsam auf, was man bei der Implementierung von Peer-to-Peer-Netzwerken beachten muss.

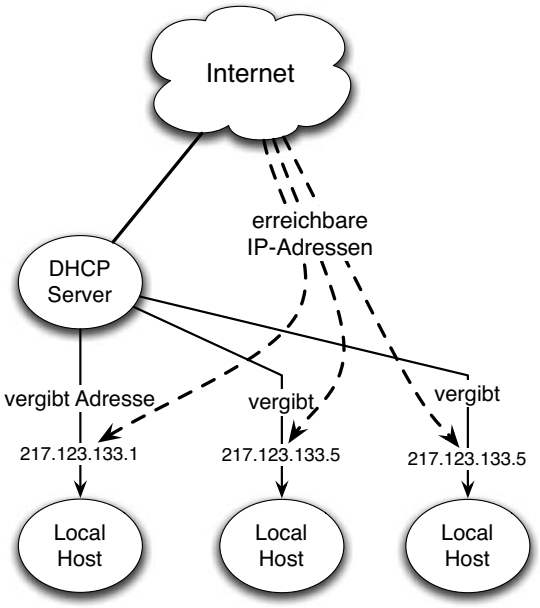


Abb. 2.21. Typische Situation eines DSL-Users: IP-Adressvergabe durch DHCP.

Firewalls

Das englische Wort *Firewall* kann mit Brandschutzwand übersetzt werden. Ziel einer Firewall ist es, Schadprogramme nicht in ein Netzwerk oder einen Rechner zu lassen. Man unterscheidet dahingehend *Netzwerk-* und *Host-Firewalls*.

Eine Netzwerk-Firewall unterscheidet das externe Netz, das interne Netz und eine so genannte *demilitarisierte Zone*. Das interne (Teil-) Netzwerk wird im Allgemeinen als vertrauenswürdig eingestuft. Diese Einstufung kann problematisch sein, insbesondere wenn durch Datenträger (CDs, Disketten, Memory-Stick) oder durch tragbare Rechner Viren eingeschleust werden. Im Allgemeinen wird das Internet als externes, gefährliches Netzwerk betrachtet. Eventuelle Verbindungsversuche aus diesem Netzwerk werden an einer Netzwerk-Firewall aufgehalten. Bestimmte Server im Internet werden als sicher angesehen. Diese werden als so genannte demilitarisierte Zone von der Blockade ausgenommen.

Für Peer-to-Peer-Netzwerke stellt eine Netzwerk-Firewall in der Regel eine unüberwindliche Hürde dar. Viele Netzwerkadministratoren verweigern aus Sicherheitsbedenken auch grundsätzlich die Verwendung von Peer-to-Peer-Netzwerken, selbst wenn diese Telefondiensten wie etwa *Skype* dienen.

Eine *Host-Firewall* kontrolliert den gesamten Datenverkehr eines Rechners und schützt ihn vor Attacken von außerhalb und innerhalb. Von außerhalb gefährden *Viren*, *Makro-Viren* und *Würmer* die Systemsicherheit. Von innen können so genannte *Trojaner* schützenswerte Information nach außen kommunizieren. Die Methoden einer Host-Firewall sind *Paketfilter* und das Sperren von Port-Nummern oder IP-Adressen. Paketfilter durchsuchen Paketinhalte nach *SPAM-Mails* (unverlangt zugesandte Werbe-Mails), *Viren*, *Active-X-* oder *Javascript* in HTML-Seiten. Schädlich eingestufte Pakete werden dann gelöscht.

Eine weitere Methode sind so genannte *Proxies* (Stellvertreter). Hier wird der gesamte Datenverkehr über einen Rechner umgeleitet. Dieser untersucht den Inhalt und kann so verhindern, dass ein Dienstverweigerungssangriff (*DoS: Denial-of-Service*) alle Rechner betrifft. Proxies sind transparente (von außen sichtbare) Hosts, die zu einer Kanalisierung der Kommunikation führen und so die Angriffe auf gesicherte Rechner mit einem zentralen Mechanismus verhindern können.

Ein Standard im Bereich der Firewalls ist das so genannte *Stateful-Inspection*. Hierbei wird eine Verbindung durch Zustände charakterisiert, und es werden je nach Zustand verschiedene Schutzmechanismen angewendet. Eröffnet beispielsweise der Rechner eine Verbindung nach außen, so ist die Antwort des angesprochenen Rechners legitim. Kommt dasselbe Paket aber ohne vorherigen Verbindungsaufbau, so ist es illegitim und als möglicher Angriff zu werten.

Es hat sich herausgestellt, dass Firewalls (absichtlich oder auch unabsichtlich) eine Hürde für Peer-to-Peer-Netzwerke darstellen. Firewalls wurden für den klassischen Fall von Client-Server-Netzwerken entwickelt, und selbst da können sie zu Störungen führen. Dieselben Mechanismen und Ausnahmeregeln, die dort weiterhelfen, sind daher auch für Peer-to-Peer-Netzwerke notwendig. Andererseits ist der Markt für Firewalls heftig umkämpft, so dass die weitgehende Verbreitung der Peer-to-Peer-Netzwerke zu einem Umdenken geführt hat.

IPsec

In den Anfangstagen des Internets waren die Router grundsätzlich unter der Aufsicht von vertrauenswürdigen und zuverlässigen Ingenieuren. Das ist heute nicht mehr der Fall. Daher ist man im Internet den Angriffen verschiedener bösartiger Teilnehmer ausgeliefert. Grund hierfür sind fehlende Sicherheitsmechanismen in IPv4. So gibt es zum Beispiel keine eingebaute Authentifizierung. Wenn in einem Paket ein Absender steht, so ist dies nicht unbedingt der richtige. Daher kann man in der Regel auch keine *SPAM-Mail* zum Absender zurückverfolgen.

Auf der anderen Seite gibt es im Bereich der Kryptographie eine Vielzahl sicherer Methoden. Aus diesem Grund wurde im IP-Protokoll von IPv6 ein Sicherheitsmechanismus eingführt, das so genannte IPSec-Protokoll (Sec steht für *Security*).

Dieses Protokoll kennt zwei Betriebsarten: Transportmodus für direkte Verbindungen und den Tunnelmodus für andere Verbindungen.

Zu Beginn wird ein *Internet-Key-Exchange*-Protokoll (*IKE*) durchgeführt. Mit diesem wird eine so genannte *Security-Association* vereinbart. In dieser identifizieren sich die Teilnehmer und vereinbaren Schlüssel für die nun folgende Kommunikation. Außerdem werden Erneuerungszeiträume für die Authentifizierung und die Schlüssel festgelegt. Nach der Etablierung der Verbindung kann diese Security-Association im Schnellverfahren erzeugt werden.

Ein *IPSec-Paket* besteht aus den Nutzdaten und der *Encapsulating-Security-Payload* (*ESP*). Hierbei wird der IP-Header unverschlüsselt, die Nutzdaten hingegen verschlüsselt und authentifiziert übertragen. Im *Transportmodus* befindet sich der *IPsec-Header* zwischen IP-Header und Nutzdaten, und die Überprüfungen finden in den IP-Routern statt, in denen jeweils IPSec verwendet werden muss.

Im *Tunnelmodus* wird das komplette Paket verschlüsselt und mit dem IP-Sec-Header als neu kodiertes IP-Paket verschickt. Nur an den Endstellen muss dann IP-Sec vorhanden sein. IPSec ist fester Bestandteil der neuen Version von IPv6. Es existieren aber Rückportierungen nach IPv4. Gerade der Tunnelmodus ermöglicht daher den Betrieb über ungesicherte IPv4-Router.

Für Peer-to-Peer-Netzwerke ist IPSec ein Pluspunkt. Dadurch können die Daten sicher und zuverlässig zwischen zwei Teilnehmern versendet werden. Auf diese Weise gelangen sie auch durch Netzwerk-Firewalls oder Host-Firewalls hindurch. Auf der anderen Seite ist der Verbindungsaufbau mit erheblichem Zusatzaufwand verbunden.

IPv6

Mit IPSec und DHCP haben wir schon Bestandteile von IPv6 kennengelernt. Der Hauptgrund für IPv6 war die Adressenknappheit. Zwar gibt es rund vier Milliarden denkbare Kombinationen in IPv4 mit dessen 32-Bit-Adressen. Diese sind aber statisch organisiert in Netzwerkteil und Rechnerteil und selbst bei einer geeigneten Vergabe kann man absehen, dass alleine diese Anzahl für alle Menschen nicht ausreicht (da die Vernetzung global fortschreitet). Wenn dann auch noch Funktelefone, Kühlschränke, Fernseher usw. IP-Adressen besitzen, kann man dieses Problem auch nicht durch eine geschickte dynamische Adressvergabe lösen. Weitere Bestandteile von IPv6 sind DHCP, Mobile-IP, Umnummerierung von IP-Adressen, IPSec, Qualitätssicherung (QoS) und Multicast.

Eines der Hauptmerkmale von IPv6 sind die Vereinfachungen für Router, die aufgrund der Zunahme des Durchsatzes notwendig sind. So werden keine IP-Prüfsummen berechnet und IP-Pakete nicht weiter unterteilt (partitioniert).

In Abbildung 2.22 kann man den vereinfachten Header für IPv6-Pakete sehen. Er besteht aus den Feldern Version, Traffic-Class, Flow Label, Payload Length, Next Header, Hop Limit, Source Address sowie Destination Address. In *Traffic-Class* gibt es die Möglichkeit, für Qualitätssicherung (*QoS: Quality of Service*) eine Prioritätsvergabe zu organisieren. Auch *Flow-Label* dient dem QoS und der

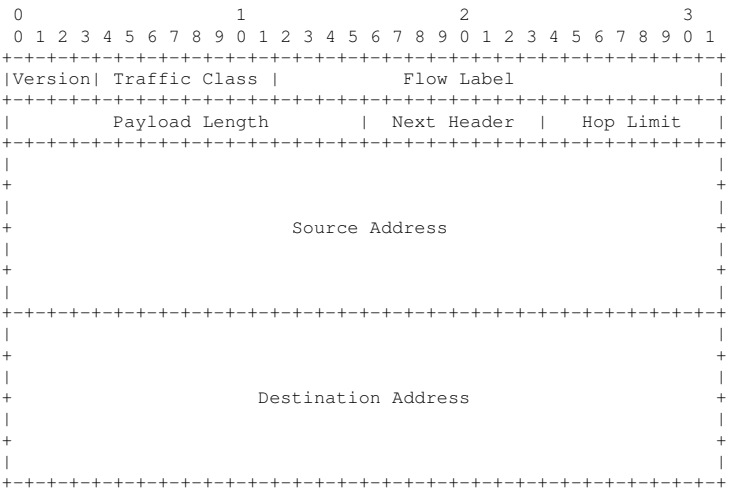


Abb. 2.22. IPv6-Header-Definition [32].

Flusskontrolle. In *Payload-Length* steht die Anzahl der Nutzdaten im IP-Paket (ohne Header). *Next-Header* entspricht dem Protocol-Feld bei IPv4. Hier steht, welcher Nachrichtentyp befördert wird. Gemäß der Datenkapselung steht der Header der nächsthöheren Schicht innerhalb der Nutzdaten, z.B. UDP, TCP, ICMP, IGMP, EGP,...; daher der etwas irreführende Name.

2.5 Zusammenfassung

Alle Peer-to-Peer-Netzwerke verwenden das Internet. Das Internet ist eher ein loser Zusammenschluss verschiedenartiger Rechnernetzwerke. Also genau das Gegenteil einer zentralen Kommunikationsstruktur mit homogener Hard- und Software. Bisher hat es sich auch jeder Vereinnahmung hinsichtlich Hardware-, Software-, und Betriebssystemhersteller erfolgreich widersetzt. Quality of Service, Sicherheit und Anonymität sind in IPv4 nicht vorhanden.

IPv4 ist das vorherrschende Protokoll. Das Problem der Adressknappheit hat sich aber durch Netzwerkmasken, DHCP, NAT und PAT mittlerweile entschärft. Das ist ein Grund dafür, dass IPv6 sich trotz seiner Überlegenheit bisher kaum durchgesetzt hat. Außerdem hat sich für IPv6 noch kein Domain-Name-Service etabliert. Ein weiterer Grund ist, dass viele interessante Eigenschaften wie IPsec und DHCP für IPv4 zurückportiert wurden. In der Praxis zeigt sich, dass kaum ein Teilnehmer auf eine IPv4-Adresse verzichten will. Hat man diese, ist der Hauptgrund für IPv6 (nämlich die Größe des Adressraums) schon entfallen.

Bei Peer-to-Peer-Netzwerken muss berücksichtigt werden, dass NAT, PAT, DHCP und Firewalls direkte Peer-to-Peer-Verbindungen be- oder verhindern. IPSec

ermöglicht den unkontrollierten direkten Download. Peer-to-Peer-Verkehr ist dann nur noch nachweisbar, wenn man auf der Sender- oder Empfängerseite sitzt.

In der Transportschicht etabliert TCP zuverlässige Punkt-zu-Punkt-Verbindungen, die die Bandbreite fair und effizient aufteilen. Es entsteht aber beim Verbindungsaufbau ein gewisser Overhead. UDP kann als Ersatz verwendet werden. Dabei besteht aber die Gefahr, dass eine Reihe von UDP-Verbindungen die TCP-Verbindung kannibalisieren. In guten Peer-to-Peer-Netzwerken sollten diese Probleme berücksichtigt werden.

3

Die ersten Peer-to-Peer-Netzwerke

When there are such lands there should be profitable things without number.
Chistoph Columbus

3.1 Napster

Napster war das erste als solches bezeichnete Peer-to-Peer-Netzwerk. Wir werden es hier hauptsächlich aus diesem Grund vorstellen. Algorithmisch hat es wenig zu bieten und ob es wirklich als echtes Peer-to-Peer-Netzwerk bezeichnet werden kann, ist umstritten.

Geschichte

Shawn „Napster" Fanning (Jahrgang 1980) veröffentlichte im Juni des Jahres 1999 eine Beta-Version der *Client-Software* seines mittlerweile legendären Peer-to-Peer-Netzwerks Napster. Die ursprüngliche Aufgabe dieser Software war die Bereitstellung eines *File-Sharing-Systems*, mit dem man Dateien auf teilnehmenden Rechnern lokalisieren und direkt von diesen herunterladen kann. Erst durch das Engagement eines Freundes wurde aus dem File-Sharing-System ein Portal, das hauptsächlich zur Verbreitung von Musikdateien diente. Die dadurch hervorgerufene Popularität machte Napster schon im Herbst 1999 zum Download des Jahres.

Wegen der Missachtung von Urheberrechten klagten Musiklabels gegen Fanning und drohten das Netzwerk stillzulegen. Tatsächlich geschah einige Zeit nichts, und die durch den Rechtsfall gesteigerte Publizität erhöhte sogar die Anzahl der Teilnehmer. In einer überraschenden Wendung ging Fanning Ende 2000 einen Kooperationsvertrag mit Bertelsmann Ecommerce ein und begann sein bislang kostenloses System in ein kommerzielles File-Sharing-System umzuwandeln. Heutzutage firmiert somit unter dem Namen Napster ein ganz anderes (völlig *Client-Server*-basiertes) Netzwerk zum Musik-Download.

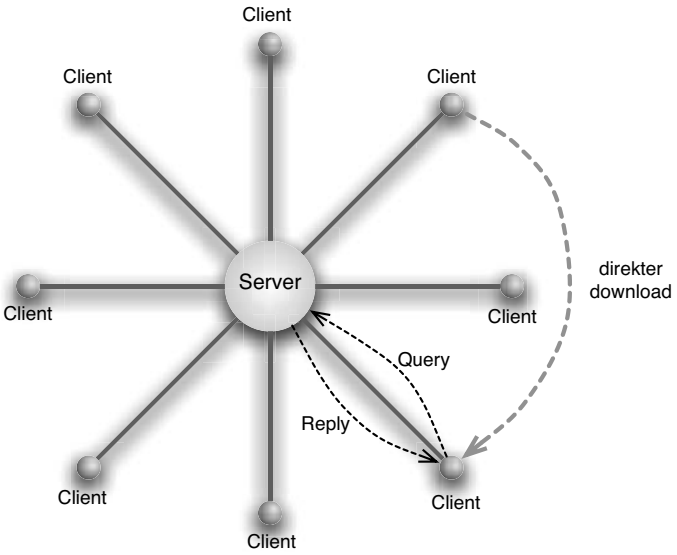


Abb. 3.1. Die Funktionsweise von Napster.

Aufbau

Der Aufbau und die Funktionsweise von Napster sind weitaus weniger aufregend als seine Geschichte. Napster folgt im Wesentlichen einer *Client-Server*-Struktur, siehe Abbildung 3.1. Ein *Server* unterhält einen Index über die von den Netzwerkteilnehmern bereitgestellten Dateien. Ein *Client* kann Dateien zur Verfügung stellen oder abfragen. Der Index besteht dabei aus dem Dateinamen, dem Dateidatum, der IP-Adresse der Peers, die die Datei bereitstellen, und ähnlichen Informationen. Mit dieser Tabelle werden die Anfragen der Clients bedient.

Bei einer Suche (*Query*) kontaktiert ein Client den Server und übermittelt z.B. den Namen der gesuchten Datei. Der Server durchsucht dann seinen Index nach Clients, die die entsprechende Datei bereitstellen. Die Liste dieser Clients wird dann als *Reply*-Nachricht an den Client übermittelt, der die Anfrage gestellt hat. Dieser kann die gesuchte Datei dann direkt von einem dieser Clients herunterladen.

Dieser letzte Schritt des Datei-Austausches ist der einzige Schritt, der unserer eingangs formulierten Definition eines Peer-to-Peer-Netzwerks entspricht. Man kann sich vorstellen, dass diese Lösung genau dann Sinn macht, wenn der Server nicht in der Lage ist, alle Dateien zu speichern. Auch wird so unnötiger Datenverkehr zwischen Server und Clients vermieden. Und genau dieser Aspekt regte die Fantasie vieler Entwickler an, diesen Mechanismus weiter auszubauen.

Diskussion

Die Vorteile von Napster sind dessen Einfachheit und die Möglichkeit, schnell und effizient Dateiverweise zu finden.

Nachteilig ist vor allem die Client-Server-Struktur, die feindliche Eingriffe in das System erleichtert. Dies muss nicht unbedingt eine richterliche Verfügung im Auftrag eines Musikverlegers sein, es kann sich auch um die repressive Verfolgung in einem Unrechtsstaat handeln. Des Weiteren sind auch *Denial-of-Service*-Angriffe eine einfache Methode, solch zentrale Strukturen lahmzulegen. Genauso kann das Versagen eines solchen Systems durch einen Systemausfall der zentralen Komponente geschehen (Programmabsturz, Hardware-Ausfall, etc.). Man darf aber auch nicht übersehen, dass eine Client-Server-Struktur eine Reihe von Vorteilen mit sich bringt. So lässt sich der Zugang zentral koordinieren und für den Fall einer Kommerzialisierung, wie bei Napster geschehen, die Bezahlung der Dienste sehr gut organisieren.

Ein Nachteil einer zentralen Komponente ist die mangelnde Skalierbarkeit. Wenn die Teilnehmerzahl zunimmt, kann der zentrale Server die Anfragen nicht mehr schnell genug bearbeiten. Außerdem kann schon die Speicherung der Meta-Daten einen nicht besonders ausgelegten Server überfordern.

Als Resumee bleibt festzuhalten, dass Napster nur zum Teil ein Peer-to-Peer-Netzwerk darstellt. Bis auf die direkte Verbindung beim Dateiaustausch ist Napster gar kein Peer-to-Peer-Netzwerk im eigentlichen Sinn.

3.2 Gnutella

Das *Gnutella*-Netzwerk [7] wurde im März 2000 von Justin Frankel und Tom Pepper (*Nullsoft*) vorgestellt. Wie auch bei Napster handelt es sich bei Gnutella um ein File-Sharing-Netzwerk. Im Gegensatz zu Napster ist Gnutella jedoch ein Peer-to-Peer-Netzwerk, das ohne zentrale Strukturen auskommt.

Aufbau

Wie kann man erreichen, dass ein Netzwerk ohne die Nutzung zentraler Strukturen entsteht? Das erste Problem tritt bereits auf, wenn sich ein Peer zum ersten Mal beim Netzwerk anmeldet. Ein solcher Peer, der dem Netzwerk beitreten will, muss zwangsläufig einen Peer kontaktieren, der bereits Teil des Netzwerks ist. Da die Peers einander anfangs nicht kennen, müssen Adressen ausgewählter Peers entweder Bestandteil der Software sein, um beim erstmaligen Einsatz dem Netzwerk beitreten zu können. Wie kann man aber verhindern, dass diese nicht als Server dienen?

Gnutella löst diese Problematik, die auch als *Bootstrapping* bezeichnet wird, folgendermaßen: Beim erstmaligen Aufruf eines Gnutella-Clients wird eine mit der Software gelieferte Liste von Peers bzw. IP-Adressen verwendet. Diese werden der Reihe nach durchprobiert, bis ein gerade aktiver Peer gefunden wird. Von diesem aus werden die nächsten Nachbarn und deren Nachbarn bis zum k-nächsten Nachbarn abgefragt. Aus diesen Rückmeldungen aktiver Peers wird eine eigene Liste von

Gnutella Peers aufgebaut. Hat ein Peer erst eine solche Liste gewonnen, so wird diese gespeichert und beim nächsten Start anstelle (oder zusätzlich) zu der in der Software gespeicherten verwendet um sich in das Netzwerk einzubinden.

Gnutella besitzt ein eigenes Protokoll [7], das auf TCP-Verbindungen aufbaut. Es benutzt die fünf Nachrichtentypen *Ping*, *Pong*, *Query*, *QueryHit* und *Push*, die wir im Folgenden kurz erläutern.

Ping wird benutzt, um andere Peers im Netzwerk zu finden und sich ins Netzwerk einzubinden. Der Empfänger einer Ping-Nachricht antwortet mit einer Pong-Nachricht.

Pong ist die Antwort auf eine Ping-Nachricht. Diese Nachricht enthält IP und *Port*-Adresse sowie Anzahl und Größe bereitgestellter Dateien des antwortenden Peers.

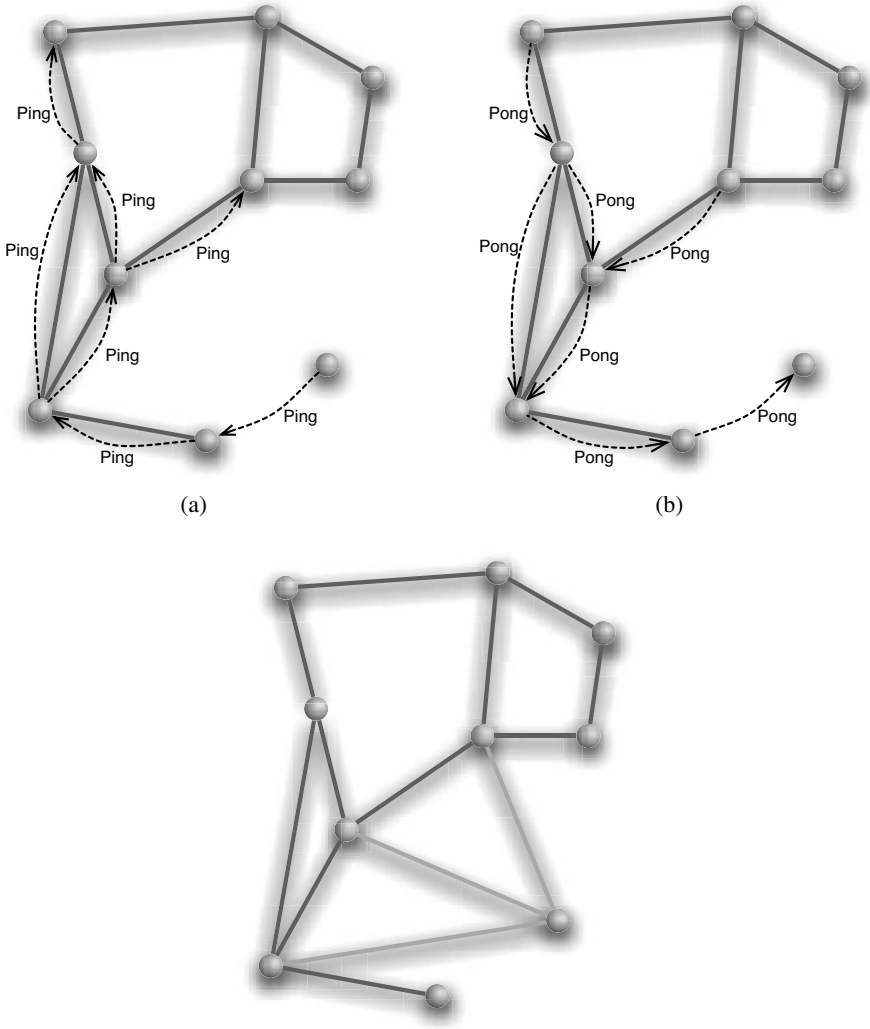
Query ist ein Mechanismus für die Suche nach Dateien im Netzwerk mittels eines Such-Strings. Ein Peer, der eine Query-Nachricht erhält, antwortet mit einer QueryHit-Nachricht, falls eine von ihm bereitgestellte Datei der Anfrage entspricht.

QueryHit ist die Antwort auf eine Query-Nachricht. Diese Nachricht enthält IP und Port-Adresse, Verbindungsgeschwindigkeit und die Beschreibung der Dateien, die der vorausgegangenen Anfrage entsprechen.

Push ist ein Mechanismus um Peers hinter einer Firewall das Bereitstellen von Daten zu ermöglichen (für Details siehe [7]).

Mit dem eingangs beschriebenen Prozess erhält ein Peer p eine Liste möglicherweise aktiver Gnutella Peers, die immer weiter verlängert und gespeichert wird. Dieser Prozess wird auch verwendet um p in das Gnutella Netzwerk einzubinden, weshalb wir diesen Prozess noch einmal etwas genauer beschreiben. Hat Peer p einen aktiven Gnutella Peer p' gefunden, so wird p eine Ping-Nachricht an p' senden. Mit dieser Nachricht verknüpft ist eine Sprungweite TTL (*Time to Live*, nicht zu verwechseln mit dem TTL-Feld von TCP), die bei jedem Schritt der Nachricht im Netzwerk um Eins verringert wird. Jeder Peer, der eine Ping-Nachricht erhält, verringert das TTL-Feld der Nachricht und reicht diese an alle Nachbarn weiter, bis die vom TTL-Wert vorgegebene Anzahl von Schritten erreicht wurde. Peers die eine Ping-Nachricht erhalten haben, antworten auf diese mit einer Pong-Nachricht, die auf dem gleichen Pfad zurückgesendet wird. Auf diese Weise erhält Peer p eine Liste zur Zeit aktiver Gnutella Peers aus denen er zufällig einige als seine Nachbarn auswählt (siehe Abbildung 3.2). Ein typischer Wert für die Größe der Nachbarschaft ist fünf.

Die resultierende Graphstruktur ensteht also durch einen zufälligen Prozess, da sie allein durch die Menge der (zufällig) aktiven Peers und deren gerade aktiven Nachbarn bestimmt wird. Man sieht, dass es keine zentrale Instanz oder Kontrollmechanismen gibt (außer dem TTL-Feld), welche die Struktur des Verbindungsgraphen kontrollieren. Gewisse statistische Gesetzmäßigkeiten lassen sich dennoch entdecken. So unterliegen wichtige Parameter der Graphstruktur der Pareto-Verteilung, wie z.B. die Verteilung des Grades der Knoten [33]. Die Verbindungsstruktur von



(a) (b)

(c)

Abb. 3.2. Anbindung von neuen Peers in Gnutella. Der neue Peer schickt eine Ping-Nachricht mit TTL 3 in das Netzwerk (a). Die erreichten Peers antworten mit Pong-Nachrichten (b). Der neue Peer erzeugt Netzwerkverbindungen zu einem Teil der antwortenden Peers (c).

Gnutella entsteht aus dem völlig unkoordinierten Verhalten der Benutzer. Die hierbei beobachteten Phänomene sind so interessant, dass wir uns im Abschnitt 9.1 auf Seite 174 eingehender damit beschäftigen werden. Ein Schnappschuss des Gnutella Netzwerks aus dem Jahr 2000 [33] wird in Abbildung 3.3 dargestellt.

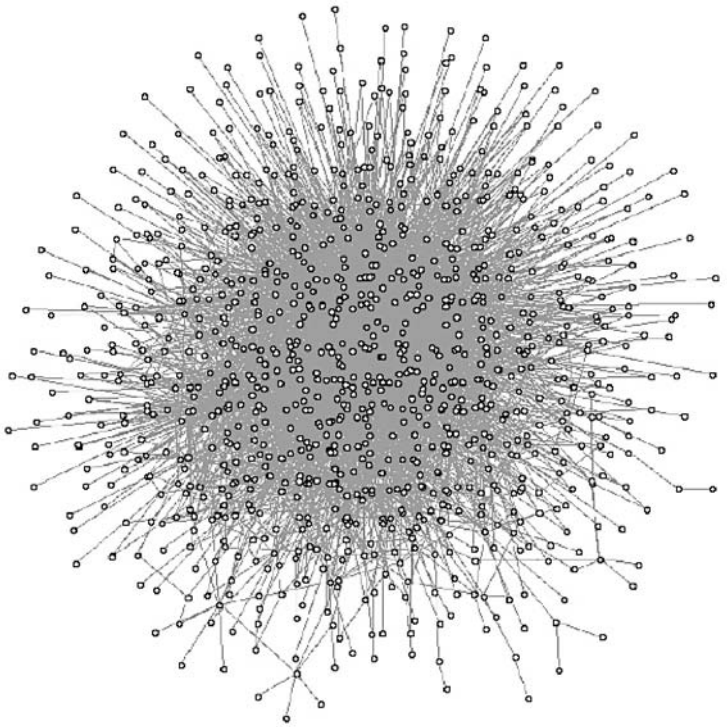


Abb. 3.3. Ein Schnappschuss des Gnutella Netzwerks [33].

Die Suche (*Query*) nach Dateien geschieht völlig analog zum Ping-Pong-Prozess. Eine Query wird durch einen beschränkten Broadcast an alle Nachbarn im Netzwerk bis zur Entfernung TTL weitergeleitet, und in entgegengesetzter Richtung werden die Antworten vom Typ QueryHit auf denselben Pfaden zurückgeleitet. Wenn der Initiator der Suche die Beschreibung der gewünschten Datei in den QueryHit-Nachrichten vorfindet, kann er diese direkt von dem Peer herunterladen, der die entsprechende QueryHit-Nachricht erstellt hat (siehe auch Abbildung 3.4).

Diskussion

Im Gegensatz zu Napster liegt hier zum ersten Mal ein echtes Peer-to-Peer-Netzwerk vor, das vollständig ohne zentrale Kontrollmechanismen auskommt. Die Vorteile von Gnutella beruhen auf der verteilten Netzwerkstruktur. Gnutella ist extrem robust und

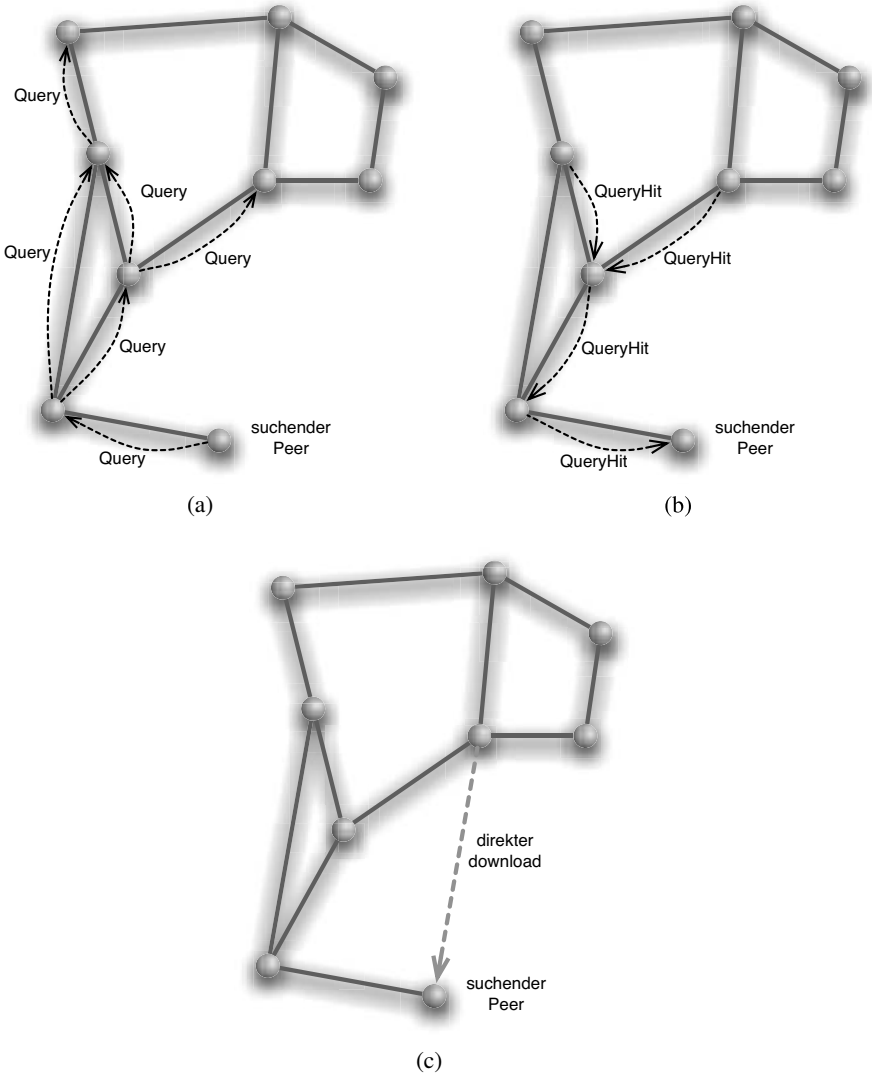


Abb. 3.4. Dateisuche in Gnutella. Ein Peer schickt eine Query-Nachricht mit TTL 3 in das Netzwerk (a). Jeder Peer, der die Query-Nachricht erhält und eine der Anfrage entsprechende Datei bereitstellt, schickt seine Antwort auf demselben Pfad zurück (b). Falls die Suche erfolgreich war, wird die Datei direkt vom gefundenen Peer heruntergeladen (c).

praktisch unangreifbar. Die zufällige Netzwerkstruktur ist auch sehr gut skalierbar, d.h., viele Knoten können ohne Einbußen in der Leistungsfähigkeit aufgenommen werden.

Gnutella hat jedoch auch eine Reihe von Nachteilen. Das Hauptproblem ist, dass durch die tiefenbeschränkte Suche nur in einem Teilnetzwerk nach der Zieldatei gesucht wird. Ist die gesuchte Datei unter den Peers weit verbreitet, dann wird sie schnell und zuverlässig gefunden. Selten vorhandene Dateien werden nur gefunden, wenn sie zufällig von einem Peer im lokalen Umfeld bereitgestellt werden. Dieses Problem könnte man durch Erhöhung des TTL-Eintrags umgehen, wenn dadurch nicht der zweite Nachteil von Gnutella noch verschärft würde. Dies ist das Nachrichtenaufkommen bei einer Suche. Da es völlig unklar ist, wo sich eine gesuchte Datei befindet, muss zwangsweise eine genügend große Teilmenge aller Peers befragt werden, um zumindest einen Großteil der Dateien im Netzwerk auffindbar zu machen.

Es gibt eine Reihe von Verbesserungvorschlägen, um das Nachrichtenaufkommen zu verringern. Eine Möglichkeit besteht zum Beispiel darin, *Random Walks* anstelle von Broadcasts für die Suche zu verwenden [34]. Bei einem Random-Walk wird eine Nachricht nicht an alle Nachbarn, sondern nur an einen zufälligen weitergereicht. Auf diese Weise wird die Suche weiter in das Netzwerk hineingereicht, ohne das Nachrichtenaufkommen stark zu erhöhen.

Ein weiterer Vorschlag ist die passive Replikation [35] von Information entlang eines Pfads. Hierbei werden Ergebnislisten gespeichert, und neue Anfragen nach demselben Dokument können dann sofort ohne weitere Suchnachrichten beantwortet werden.

3.3 Zusammenfassung

Wir haben in diesem Kapitel die Frühphase der Peer-to-Peer-Netzwerke kurz gestreift. Das klassische Peer-to-Peer-Netzwerk ist sicherlich Gnutella, während Napster, obgleich seiner Client-Server-nahen Struktur, durch seine Publizität die Aufmerksamkeit und Fantasie einer größeren Öffentlichkeit erregt hat. Das Gnutella-Netzwerk in seiner ursprünglichen als auch in einer weiterentwickelten Form (Gnutella-2, Seite 244) hat bis heute überlebt. Das kann man als deutlichen Beweis der Tragfähigkeit des Peer-to-Peer-Netzwerk-Ansatzes werten.

4

CAN: Ein Netzwerk mit adressierbaren Inhalten

It's hip to be square.
Huey Lewis And The News.

Im Jahr 2000 veröffentlichten Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp und Scott Shenker ein „Skalierendes Netzwerk mit adressierbaren Inhalten" (*Scalable Content Addressable Network*) [8], abgekürzt als *CAN*.

Im Wesentlichen ist CAN eine verteilte Datenstruktur, durch die Daten bestimmten Peers zugewiesen werden. Die Peers sind dabei durch eine Gitterstruktur verbunden, siehe Abbildung 4.1. Diese Gitterstruktur kann durch einfache lokale Operationen aufgebaut und aufrechterhalten werden. CAN war das erste Peer-to-Peer-Netzwerk, das effiziente Datenstrukturen mit einem verteilten Overlay-Netzwerk kombinierte.

Wir konzentrieren uns im Folgenden bei der Beschreibung von CAN auf die verteilte Datenstruktur und gehen nicht näher auf Details des Verbindungsaufbaus oder des Kommunikationsprotokolls ein.

4.1 Verteilte Hash-Tabellen

CAN versucht die Daten gerecht auf die Peers zu verteilen. Für die Zuordnung wird eine quadratische Fläche $Q = [0, 1) \times [0, 1)$ verwendet. Diese Fläche wird partitioniert in Rechtecke und Quadrate, die jeweils einem Peer zugewiesen werden. Des Weiteren wird jedem Datum ein Punkt des Quadrates Q zugeordnet. Jeder Peer verwaltet dann die Daten, die seinem Quadrat oder Rechteck zugewiesen wurden.

Idealerweise sollten die Daten gleichmäßig über das ganze Quadrat verteilt werden und die Flächen der Peers gleich groß sein, so dass die Daten gleichmäßig auf die Peers verteilt werden. Die Verteilung der Daten geschieht über eine so genannte *Hash-Funktion* $f : \mathcal{K} \to Q$, die jedem Datum eine scheinbar zufällige Position zuweist. Hierbei bezeichnet $\mathcal{K}$ die Menge aller möglichen *Datenschlüssel*. Dies können z.B. Dateinamen sein.

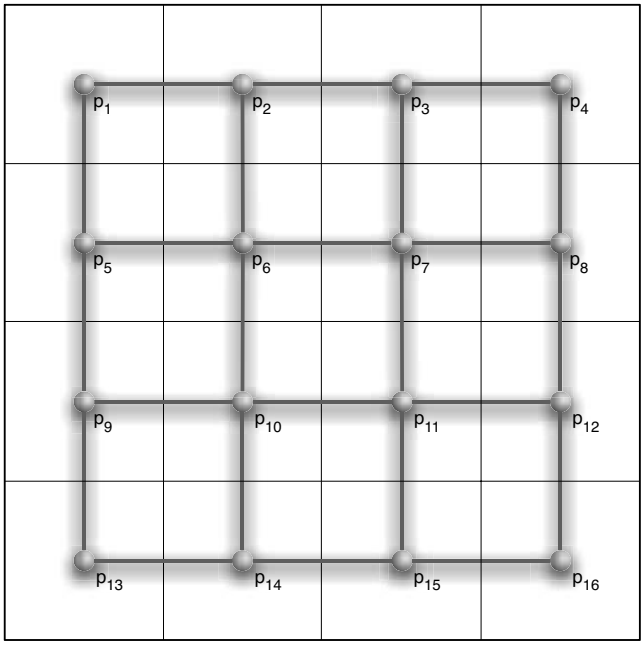


Abb. 4.1. Ein CAN mit idealer Struktur.

Das englische Wort *Hash* bedeutet soviel wie „zerhacken“. Eine Hash-Funktion bildet die Eingabe, hier den Schlüssel, auf eine Ausgabe bestimmter Länge ab, den Hash-Wert . Im Idealfall erhält jede Eingabe einen eindeutigen Hash-Wert. Da der Schlüsselraum jedoch typischerweise wesentlich größer ist als der Bildraum, lässt es sich nicht vermeiden, dass einige Eingaben dem gleichen Hash-Wert zugeordnet werden. Hierbei spricht man von so genannten *Kollisionen*. Tatsächlich treten Kollisionen in der Praxis jedoch kaum auf, wenn der Wertebereich der Hash-Funktion groß genug gewählt wird, d.h. der Wertebereich wesentlich größer als die Anzahl der tatsächlich verwendeten Schlüssel ist. Dies wird z.B. durch 128-Bit-Hash-Werte erreicht, wodurch sich bereits $2^{128} > 3,4 \cdot 10^{38}$ mögliche Hash-Werte ergeben. Gute Hash-Funktionen zu finden, ist eine Wissenschaft für sich. Typischerweise bilden Hash-Funktionen ähnliche Eingaben auf sehr unterschiedliche Hash-Werte ab. Im Idealfall verhält sich eine Hash-Funktion so, als ob jedem Schlüssel ein zufälliger Hash-Wert zugeordnet würde. Für die Analyse von auf Hash-Funktionen beruhenden Datenstrukturen wird genau das angenommen.

Hash-Funktionen spielen in der Kryptographie eine große Rolle bei der Authentifizierung von Daten. Es gibt Hash-Funktionen (z.B. *SHA-2*), bei denen man für einen gegebenen Hash-Wert nicht einen einzigen Schlüssel rekonstruieren kann, wenn man den Hash-Wert nicht selbst erzeugt hat. Hash-Funktionen mit dieser Eigenschaft bezeichnen wir als *kryptographisch sicher*. So können Hash-Funktionen als kompakter Fingerabdruck verwendet werden. Außerdem werden Hash-Funktionen in der Fehlerkorrektur bei der Datenübertragung verwendet oder bei der Suche in Texten

(z.B. Rabin-Karp Text-Suche). Für Peer-to-Peer-Netzwerke wählt man typischerweise kryptographisch sichere Hash-Funktion, wie um Beispiel *MD-5* (*Message-Digest Version 5)*[36], SHA-1 (*Secure Hash Algorithm 1)*[37] oder SHA-2 (Secure Hash Algorithm 2)[38], von denen mittlerweile aber nur noch SHA-2 als sicher erachtet wird.

Um das Funktionsprinzip einer Hash-Funktion zu erläutern, betrachten wir die einfache Modulo-Funktion. Diese ist im Allgemeinen ungeeignet und dient hier nur der Veranschaulichung. Wir wählen einen eindimensionalen Bildbereich der Größe $B = 5$ und möchten diesem Bildbereich Schlüssel in Form von Dateinamen zuweisen. Dafür interpretieren wir den Schlüssel als ganze Zahl. Wenn der Schlüssel beispielsweise `music.mp3` ist, so entspricht dies folgender ASCII-Zeichenkette in Hexadezimal-Darstellung: `6d 75 73 69 63 2e 6d 70 33`. Als Dezimalzahl ist dies $870.920.545.682.538.843.149$. Da unser Bildbereich die Größe 5 hat, wählen wir als Hash-Funktion $h(x) = x \bmod 5$. Damit ergibt sich in unserem Beispiel als Hash-Wert: $h(\texttt{music.mp3}) = 4$.

Man verwendet Hash-Funktionen zumeist zum Speichern und Suchen von Daten. Hierzu platziert man ein Element mit Schlüssel K an der Position $h(K)$, d.h. an der Speicheradresse des Hash-Werts von K in der *Hash-Tabelle*. In unserem Beispiel wird also das Datum in der Hash-Tabelle an der Position 4 abgelegt.

Diese Technik lässt sich für das Speichern von Daten auf Peers nicht direkt anwenden. Das hat folgende Gründe: Die Peers sind nicht durchgehend nummeriert wie die Speicheradressen. Zudem wechselt die Anzahl der Peers in einem Netzwerk ständig: Neue Peers kommen fortwährend hinzu, während andere das Netzwerk verlassen. Daher müsste eine eventuelle Nummerierung ständig angepasst werden. Würde man sich sklavisch an die Datenstruktur der Hash-Tabelle halten, müsste mit jedem neuen (oder verlorenen) Peer eine neue Hash-Funktion gewählt werden, so dass die Größe des Bildbereichs der Anzahl der gerade anwesenden Peers entspricht. Dies hätte zur Folge, dass sich die Zuordnung der Daten und Peers fast vollständig ändert. Solch eine vollständige Neuverteilung der Daten bei einer marginalen Änderung der Netzwerkteilnehmer ist ineffizient und inakzeptabel.

Dieses Problem wurde in [9] elegant gelöst. Ziel dieser Arbeit war die Verteilung von Web-Seiten auf Web-Servern. Die dort entwickelte Technik ist allgemein unter dem Namen *verteilte Hash-Tabelle* (*DHT: Distributed Hash Table*) bekannt und stellt eine Standardmethode im Bereich der Peer-to-Peer-Netzwerke dar.

In CAN werden verteilte Hash-Tabellen wie folgt umgesetzt: Wir betrachten das Quadrat Q als zweidimensionalen Bildbereich $(x, y) \in [0, 1) \times [0, 1)$. Jedem Datenelement wird durch zwei verschiedene Hash-Funktionen eine x- und y-Koordinate zugewiesen. Das Quadrat ist, wie bereits erwähnt, in Rechtecke partitioniert. Jedem Rechteck ist ein Peer zugeordnet. Jeder Peer ist für alle seinem Rechteck zugewiesenen Daten verantwortlich. Damit ist eindeutig bestimmt, welcher Peer welches Datum speichert.

Meldet sich nun ein neuer Peer im CAN an, so wird eines der Rechtecke in der Mitte halbiert, und der bisher Verantwortliche und der neue Peer erhalten je eine Hälfte. Somit bleibt die Zuordnung von Daten anderer Peers von dieser Einfügeope-

ration unberührt. Diese Eigenschaft wird *Konsistenz* genannt. Daher verwendet man für verteilte Hash-Tabellen mitunter auch den Begriff *Consistent Hashing*.

Wenn die Hash-Funktion die Daten gleichmäßig auf das Quadrat verteilt, muss nur noch gewährleistet werden, dass den Peers gleich große Bereiche zugewiesen werden, um die Daten gleichmäßig auf die Peers zu verteilen.

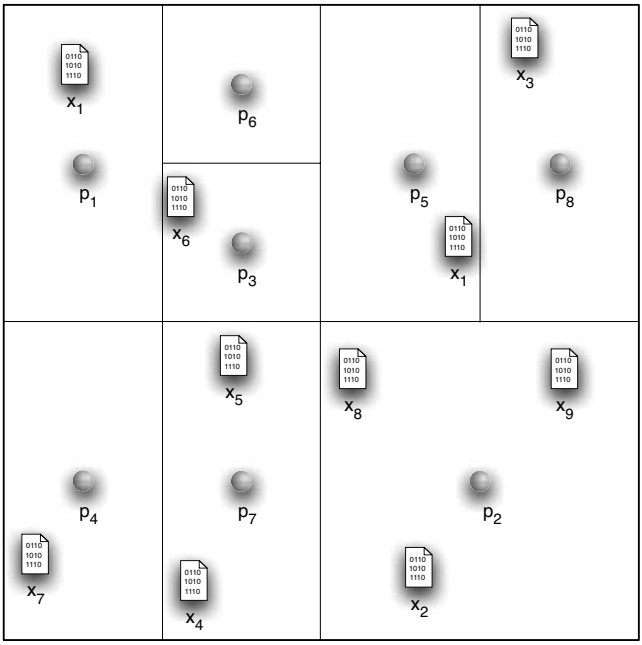


Abb. 4.2. Zuordnung von Daten zu Peers im CAN.

4.2 Einfügen von Peers

Zu Beginn besteht ein CAN aus nur einem Peer, der das gesamte Quadrat Q verwaltet. Wir beschränken uns zunächst auf den Fall, dass nur neue Peers hinzukommen und keine Peers das Netzwerk verlassen.

Jeder neue Peer wählt zuerst einen zufälligen Punkt im Quadrat Q. Dann kontaktiert der neue Peer den für diesen Punkt z zuständigen Peer. Das ist der Besitzer des z umgebenden Rechtecks. Diese Kontaktaufnahme geschieht durch die Suchfunktion, die wir später erläutern. Nun wird dieses Rechteck halbiert. Wenn es sich um ein Quadrat handelt, dann zuerst vertikal entlang der y-Achse und bei einem Rechteck entlang der x-Achse. In Abbildung 4.3 wird dargestellt, wie dieser Prozess das Quadrat immer weiter in kleinere Rechtecke unterteilt.

Nach der Unterteilung wird auch die Netzwerkstruktur entsprechend der neuen Nachbarschaft angepasst. Wir wenden uns zunächst der Frage zu, wie gleichmäßig die Daten durch diese Einfügeoperation auf die Peers verteilt werden.

Der Anteil der Daten eines Peers ist erwartungsgemäß proportional zur Fläche des von ihm verwalteten Rechtecks. Zwar ist jeder Eintrittspunkt gleichwahrscheinlich, so dass große Rechtecke beim Einfügen weiterer Peers mit höherer Wahrscheinlichkeit geteilt werden als kleine, eine vollkommen gleichmäßige Aufteilung ist jedoch sehr unwahrscheinlich.

Wir betrachten einen Peer p mit Rechteck $R(p)$ in CAN. Sei $A(p)$ die Fläche dieses Rechtecks. Das folgende Lemma liefert eine obere Schranke für die Wahrscheinlichkeit, dass das Rechteck $R(p)$ nicht weiter unterteilt wird, wenn n weitere Peers eingefügt werden.

Lemma 4.1. *Sei $P_{R,n}$ die Wahrscheinlichkeit, dass ein Rechteck R mit Fläche $A(R)$ nach dem Einfügen von n Peers ungeteilt bleibt. Dann gilt*

$$P_{R,n} \leq e^{-nA(R)} .$$

Beweis. Wir betrachten ein Rechteck R mit der Fläche $q = A(R)$. Die Wahrscheinlichkeit, dass ein bestimmter Peer diese Fläche im Zuge des Einfügens nicht weiter unterteilt, also zufällig keinen Punkt innerhalb von R wählt, ist $1 - q$. Das geschieht unabhängig vom Verhalten anderer Peers. Somit ist die Wahrscheinlichkeit, dass n Peers keinen Punkt aus R wählen, das Produkt aller n Einzelwahrscheinlichkeiten $1 - q$ und damit $(1 - q)^n$. Nun gilt für alle $m > 0$:

$$\left(1 - \frac{1}{m}\right)^m \leq \frac{1}{e} .$$

Daraus folgt

$$P_{R,n} = (1 - q)^n = \left((1 - q)^{\frac{1}{q}}\right)^{nq} \leq e^{-nq} = e^{-nA(R)}$$

und somit das Lemma. □

So ist es zum Beispiel extrem unwahrscheinlich, dass eine Fläche der Größe $\frac{1}{2}$ nach dem Eintreffen von n weiteren Peers ungeteilt bleibt. Die Wahrscheinlichkeit ist nach Lemma 4.1 höchstens $e^{-n/2}$ (tatsächlich sogar höchstens 2^{-n}).

Mit Hilfe von Lemma 4.1 können wir nun eine Aussage darüber treffen, wie groß die Rechtecke in einem CAN höchstens sein werden.

Theorem 4.2. *In einem CAN wird es nach dem Einfügen von n Peers mit hoher Wahrscheinlichkeit, d.h. mit Wahrscheinlichkeit $\geq 1 - n^{-c}$ für eine Konstante $c > 0$, kein Rechteck R mit Fläche $A(R) \geq 2c\frac{\ln n}{n}$ geben.*

Beweis. Bezeichne R_i ein Rechteck mit Fläche $A(R_i) = 2^{-i}$. Mit Hilfe von Lemma 4.1 können wir die Wahrscheinlichkeit $P_{R_i, c2^i \ln n}$, dass ein Rechteck der Größe R_i von $c2^i \ln n$ Peers nicht geteilt wird, abschätzen:

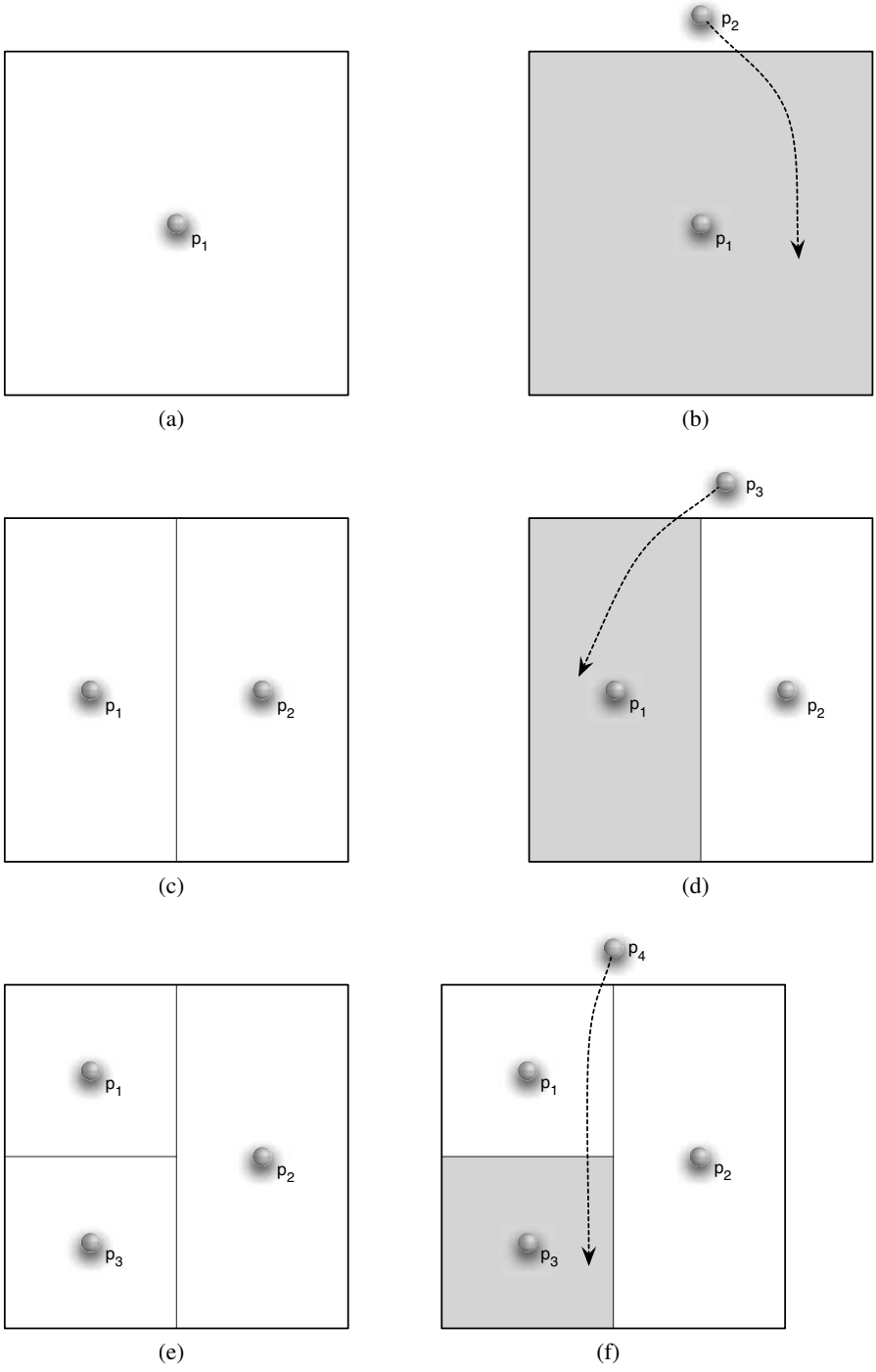


Abb. 4.3. Einfügen von Peers in das CAN.

$$\begin{aligned} P_{R_i, c2^i \ln n} &\leq e^{-A(R_i) c 2^i \ln n} \\ &= e^{-c \ln n} \\ &= n^{-c} \,. \end{aligned}$$

Es genügen also $c\, 2^i \ln n$ Peers, um R_i mit Wahrscheinlichkeit $1 - n^{-c}$ zu teilen. Damit ein Rechteck R_{i+1} überhaupt entsteht, muss das entsprechende Rechteck R_i geteilt werden. Wir betrachten das Einfügen weiterer Peers nun staffelweise für $i = 1, 2, \ldots, \log \frac{n}{2c \ln n}$. In Staffel i werden $c2^i \ln n$ neue Peers eingefügt. Diese teilen das umschließende Rechteck R_i eines Peers mit hoher Wahrscheinlichkeit.

So wird nach $\log \frac{n}{2c \ln n}$ solcher Staffeln mit hoher Wahrscheinlichkeit auch das Rechteck der Größe $2c\frac{\ln n}{n}$ geteilt. Die Summe der Peers in den Staffeln ist nach folgender Rechnung höchstens n:

$$\begin{aligned} \sum_{i=1}^{\log \frac{n}{2c \ln n}} c2^i \ln n &= c(\ln n) \sum_{i=1}^{\log \frac{n}{2c \ln n}} 2^i \\ &\leq c(\ln n) 2 \frac{n}{2c \ln n} \\ &= n \,. \end{aligned}$$

Damit wird ein bestimmtes Rechteck der Größe $2c\frac{\ln n}{n}$ mit Wahrscheinlichkeit $n^{-c} \log n$ von n Peers nicht geteilt. Von diesen Rechtecken gibt es höchstens n Stück. Damit ist die Wahrscheinlichkeit, dass eines dieser Rechtecke ungeteilt bleibt, höchstens $n \cdot n^{-c} \log n \leq n^{-c+2}$. Wählt man also c groß genug, so kann jede *polynomiell kleine Fehlerwahrscheinlichkeit* erreicht werden. □

Wie schon erwähnt, ist die erwartete Anzahl von Daten, die ein Peer verwalten muss, proportional zur Fläche seines Rechtecks. Im Durchschnitt hat diese Fläche die Größe $1/n$. Theorem 4.2 zeigt, dass die Fläche eines Peers mit hoher Wahrscheinlichkeit kleiner als $(2c \ln n)/n$ ist. Anders formuliert bedeutet dies, dass ein Peer mit hoher Wahrscheinlichkeit höchstens $2c(\ln n)$ mal mehr Daten als der Durchschnitt verwalten muss.

4.3 Netzwerkstruktur und Routing

Die Netzwerkstruktur von CAN ergibt sich aus der Lage der Rechtecke der Peers. Ein Peer unterhält Verbindungen zu allen Peers, die für ein orthogonal benachbartes Rechteck verantwortlich sind. Beim Einfügen kann jeder Peer diese Information von dem Peer erfragen, dessen Rechteck hierbei geteilt wird. Natürlich müssen die Nachbar-Peers diese Information dann auch anpassen. Sind alle Rechtecke gleich groß und jeder Peer nur für ein Rechteck verantwortlich, dann ist der Grad des Netzwerk-Graphen konstant groß (siehe Abbildung 4.1). Tatsächlich ist es jedoch unwahrscheinlich, dass alle Rechtecke gleich groß sind. Ein Grund hierfür ist der

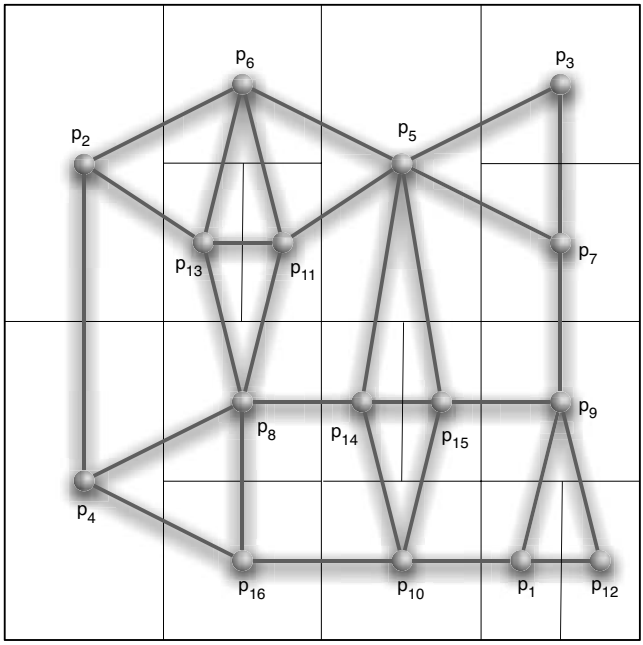


Abb. 4.4. Ein CAN mit typischer Struktur.

eben beschriebene Einfügeprozess. So ist eher eine Netzwerkstruktur wie in Abbildung 4.4 zu erwarten.

Zusätzlich werden so genannte Torus-Kanten zwischen den Peers der Rechtecke am rechten Rand mit den Peers der Rechtecke am linken Rand gleicher Höhe unterhalten. Analog werden die Zeiger zwischen Peers in Rechtecken des unteren und oberen Randes, die genau übereinander liegen, angelegt. In Abbildung 4.1, 4.4 und 4.5 wurde aus Gründen der Übersichtlichkeit auf die Darstellung der Torus-Kanten verzichtet. Die Torus-Kanten verringern die durchschnittliche Entfernung im Netzwerk-Graphen um den Faktor Zwei. Bei einer völlig gleichmäßigen Unterteilung des Quadrats Q ergibt sich ein Netzwerk-Durchmesser von $\mathcal{O}(\sqrt{n})$ für n Peers.

Das Routing bzw. die Suche nach einem Peer, der für einen Punkt in Q verantwortlich ist, verwendet nun die orthogonalen Zeiger der Netzwerkstruktur. Ausgehend von einem Peer im CAN wird die Suchanfrage an einen Nachbar-Peer weitergeleitet, dessen Rechteck bezüglich der x- oder y-Koordinate näher am gesuchten Punkt liegt. Dies wird wiederholt, bis der Peer gefunden worden ist, der das den Punkt umschließende Rechteck besitzt. Diese Suchfunktion wird auch beim Einfügen eines Peers verwendet.

Die Suche nach einem Datum geschieht in zwei Schritten. Zuerst werden aus dem Schlüssel des Datums mit Hilfe der Hash-Funktion die Koordinaten des zugehörigen Punkts berechnet. Dann wird mit dem eben beschriebenen Algorithmus die Anfrage an den Peer weitergeleitet, der den Punkt und damit das Datum verwaltet.

Wir wenden uns nochmals dem Grad des Netzwerk-Graphen zu. Ist jeder Peer nur für ein Rechteck verantwortlich[1], so ist der durchschnittliche Grad konstant. Das ergibt sich aus folgendem Gedankenexperiment: Wir richten jede Kante zwischen zwei Peers so aus, dass sie von dem Peer mit dem kleineren zu dem Peer mit dem größerem Rechteck zeigt. Zwischen Peers mit gleich großen Rechtecken wählen wir eine zufällige Richtung. Nun gehen von jedem Rechteck höchstens vier Kanten aus, da diese immer zum Größeren oder Gleich-Großen zeigen. Insgesamt kann es also höchstens $4n$ Kanten geben. Somit hat jeder Peer durchschnittlich höchstens acht Kanten, da eine Kante zwei Peers verbindet. Jeder Peer muss daher im Durchschnitt höchstens acht Verbindungen verwalten.

Der maximale Grad eines Peers hängt sehr stark von der Anzahl kleiner Rechtecke ab. Der schlimmste Fall ist, wenn um ein großes Rechteck lauter kleine Rechtecke liegen. Wir haben schon gesehen, dass die größten Rechtecke höchstens um einen logarithmischen Faktor größer sind als der Durchschnitt. Es ist jedoch nicht ausgeschlossen, dass ein überdurchschnittlich großes Rechteck von unterdurchschnittlich kleinen Rechtecken umgeben ist. Eine genaue mathematische Analyse dieses Grades wurde bislang noch nicht durchgeführt.

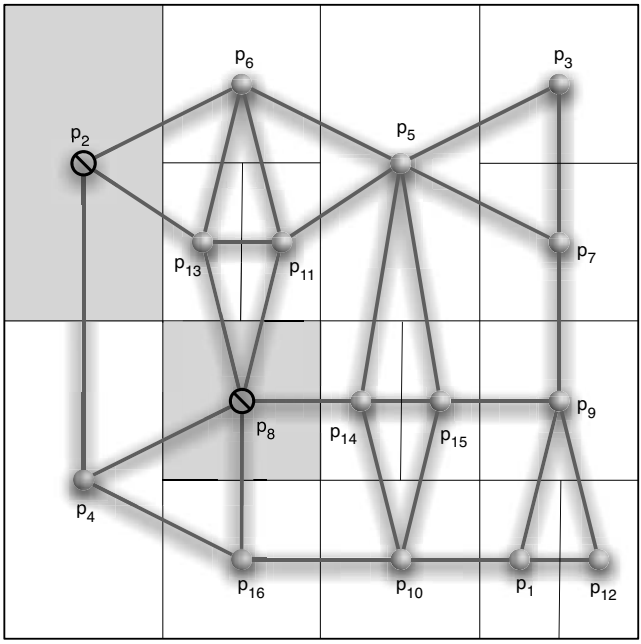
Im CAN wird durch die unterschiedlich großen Rechtecke also nicht vermieden, dass einige Peers viel weniger (oder mehr) Daten speichern als der Durchschnitt. Auch wenn der durchschnittliche Grad des Netzwerks konstant ist, werden einige Peers durch die unterschiedlich großen Rechtecke einen nicht konstanten Grad aufweisen. Dies kann nicht nur bei Peers mit großen Rechtecken auftreten. Hiervon können ebenfalls Peers mit durchschnittlich großen Rechtecken betroffen sein, wenn sie von unterdurchschnittlich kleinen Rechtecken umgeben sind. Es handelt sich hierbei allerdings um Ausnahmen, und die meisten Peers besitzen einen konstanten Grad und verwalten einen (bis auf einen konstanten Faktor) fairen Anteil des Datenvolumens.

4.4 Defragmentierung nach dem Entfernen von Peers

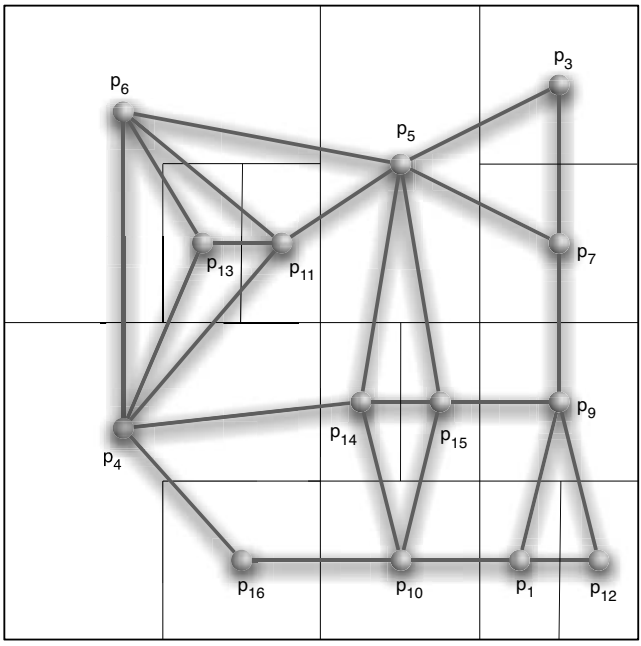
Die CAN-Netzwerkstruktur erlaubt es Peers, für mehr als ein Rechteck verantwortlich zu sein. Das wird notwendig, wenn ein Peer das Netzwerk verlässt. In einem Peer-to-Peer-Netzwerk kann nicht davon ausgegangen werden, dass ein Peer, der das Netzwerk verlässt, seine Nachbarn rechtzeitig darüber informiert. Daher überprüft jeder Peer regelmäßig, ob seine Nachbarn noch erreichbar sind. Meldet sich ein Nachbar nicht zurück, so übernimmt der erste Peer, der dies bemerkt, das Gebiet des abwesenden Peers nach Ablauf eines Wartezeitraums (siehe Abbildung 4.5).

So wird das fortlaufende Einfügen und Löschen von Peers zu einer immer stärkeren Fragmentierung des Netzwerks in kleine Rechtecke führen. Auf Dauer ist es nicht hinnehmbar, dass ein Peer ein Konglomerat aus benachbarten Flächen verwaltet. Daher wird von Zeit zu Zeit eine Defragmentierung im CAN vorgenommen.

[1] Wir werden gleich sehen, dass ein Peer durchaus für mehrere Rechtecke verantwortlich sein kann.



(a)



(b)

Abb. 4.5. Entfernen von Peers im CAN. Verlässt ein Peer das Netzwerk (a), übernimmt ein Nachbar sein Gebiet (b).

Um den Defragmentierungsprozess zu verstehen, ist es hilfreich, das CAN als Binärbaum zu betrachten. Jedes CAN besitzt eine äquivalente Darstellung als Binärbaum. Abbildung 4.6 zeigt zum Beispiel die Binärbaumdarstellung des CANs aus Abbildung 4.5(b). Die Wurzel des Baumes repräsentiert das gesamte Quadrat $[0,1)\times[0,1)$ des Netzwerks. Die beiden Kind-Knoten der Wurzel repräsentieren jeweils die rechte Hälfte $[0,0.5)\times[0,1)$ bzw. linke Hälfte $[0.5,1)\times[0,1)$ dieses Quadrats. Deren Kinder repräsentieren dann jeweils die obere und untere Hälfte ihrer Vater-Knoten. Diese Aufteilung wird fortgesetzt, bis ein Knoten genau den Zuständigkeitsbereich eines Peers beschreibt. So entspricht jedes Blatt der Baumdarstellung genau einem Peer.

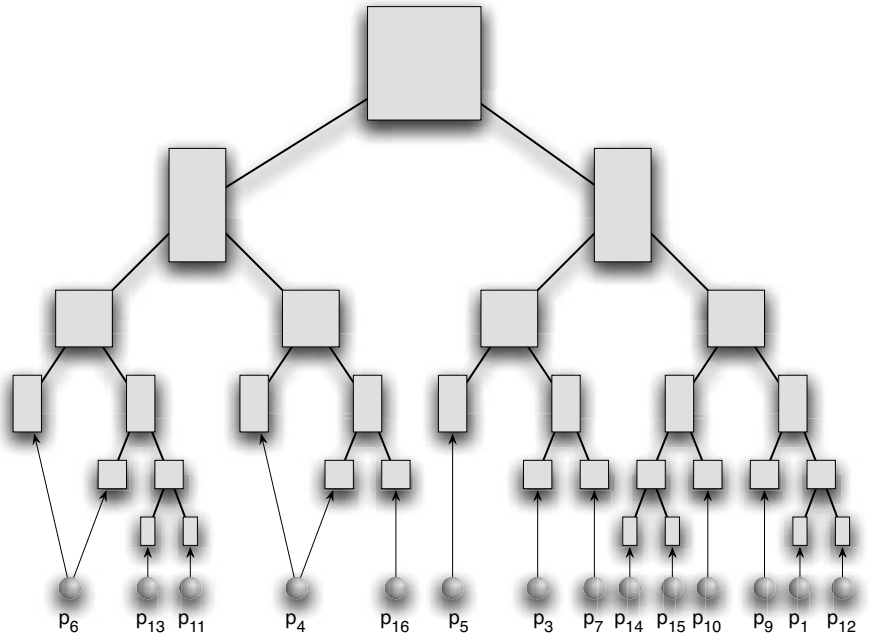


Abb. 4.6. Baumdarstellung des CANs aus Abbildung 4.5(b).

Wir beschreiben nun den Defragmentierungsprozess. Dieser kann von jedem Peer angestoßen werden, der für mehr als ein Rechteck verantwortlich ist. Dabei versucht ein Peer das kleinste von ihm verwaltete Rechteck an einen anderen Peer zu übergeben. Sei A dieses kleinste Rechteck. In der Binärbaumdarstellung entspricht A einem Blatt. Der Peer betrachtet gemäß der oben beschriebenen Binärbaumdarstellung den Bruderbaum von Blatt A. Besteht dieser ebenfalls nur aus einem Blatt B, also aus einer ungeteilten Zone, so kann der Peer das Rechteck A an den Peer übergeben, der B verwaltet. Dieser Peer wird dann die Rechtecke A und B vereinigen, so dass ein einziges Rechteck mit doppelter Fläche entsteht und die Fragmentierung des Netzwerks verringert wird.

Dieser Vorgang ist beispielhaft in Abbildung 4.7 dargestellt. Peer p_4 verwaltet zwei Zonen und startet die Defragmentierung. Der Nachbarbaum des kleinsten von p_4 verwalteten Rechtecks besteht nur aus einem Blatt, für das Peer p_{16} verantwortlich ist. Peer p_{16} übernimmt das kleinste Rechteck von p_4 und vereinigt die beiden nun von ihm verwalteten Rechtecke zu einem einzigen mit doppelter Fläche. Der Baum besitzt nun ein Blatt weniger, und das Netzwerk ist weniger fragmentiert.

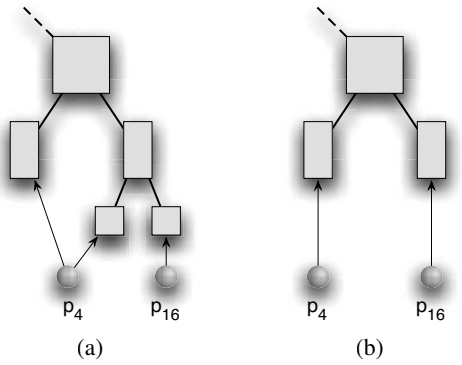


Abb. 4.7. Der einfache Fall der Defragmentierung.

Die Defragmentierung gestaltet sich jedoch nicht immer so einfach. Sei wieder A das kleinste Rechteck eines Peers, der für mehrere Rechtecke verantwortlich ist. Besteht der Nachbarbaum B des Blattes, das A repräsentiert, jetzt nicht nur aus einem Blatt, kann das gerade beschriebene Verfahren nicht angewendet werden. Stattdessen wird der Peer, der die Defragmentierung durchführt, im Nachbarbaum B nach zwei benachbarten Blättern suchen.[2] Diese werden dann zu einem Rechteck doppelter Fläche vereinigt. Der dabei frei gewordene Peer übernimmt nun die Verantwortlichkeit für das Rechteck A.

Abbildung 4.8 veranschaulicht den schwierigeren Fall der Defragmentierung. Peer p_6 verwaltet zwei Rechtecke und versucht das kleinere abzugeben. Hierfür wird eine Tiefensuche im Nachbarbaum durchgeführt, bei der zwei benachbarte Blätter gefunden werden, die von p_{11} und p_{13} verwaltet werden. Diese beiden Blätter werden vereinigt und von p_{13} übernommen. Der nun frei gewordene Peer p_{11} übernimmt das kleinere Rechteck von p_6. Wieder wurde die Anzahl der Blätter des Baumes und somit die Fragmentierung des Netzwerks verringert.

4.5 Weitere Konzepte in CAN

Es existieren mehrere Konzepte, um die Effizienz von CAN zu steigern. Diese werden wir nun kurz vorstellen.

[2] Die Existenz benachbarter Blätter ist durch den Einfügeprozess garantiert.

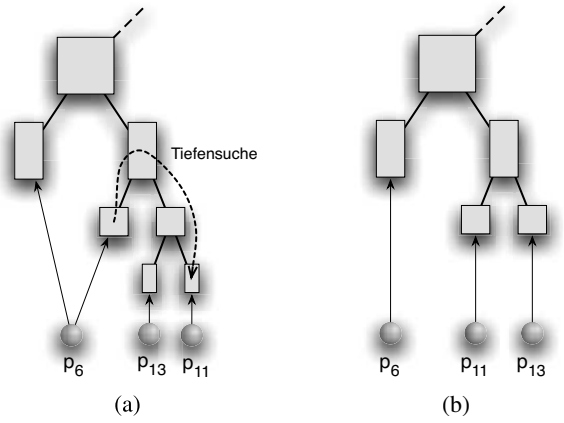


Abb. 4.8. Der schwierige Fall der Defragmentierung.

Dimensionen

Bis jetzt haben wir den Fall eines zweidimensionalen Quadrats vorgestellt. Wenn man zu einem dreidimensionalen Würfel übergeht, erhöht sich der durchschnittliche Grad auf sechs, und der Durchmesser verringert sich auf $\mathcal{O}(n^{1/3})$. Natürlich kann man das auf d Dimensionen verallgemeinern, wodurch man einen Durchmesser von $\mathcal{O}(n^{1/d})$ und einen Grad von $\mathcal{O}(d)$ erhält (vorausgesetzt, der Zufallsprozess zum Einfügen erzeugt ungefähr gleich große Hyper-Rechtecke). Abbildung 4.9 zeigt, wie sich der Durchmesser eines CANs verändert, wenn die Anzahl der Dimensionen erhöht wird.

Realitäten

Ein weiteres Konzept ist die Verwendung von mehreren Netzwerken zugleich, die einzelnen Netzwerke werden dann als *Realitäten* bezeichnet (siehe Abbildung 4.10). Jeder Peer und jedes Datum nimmt gleichzeitig in allen r Realitäten (CAN-Netzwerken) teil. In jedem der r CANs wird eine andere Hash-Funktion gewählt, so dass die Peers und Daten in allen r Realitäten verschieden an- und zugeordnet werden.

Dadurch erhöht sich der Grad um den Faktor r, während die verschiedenen Realitäten sowohl die Zuverlässigkeit erhöhen als auch den Durchmesser des Netzwerks verringern. Die erhöhte Zuverlässigkeit ergibt sich allein aus der Tatsache, dass man $r - 1$ Ersatznetzwerke hat. Der Durchmesser verringert sich erheblich. Man kann zeigen, dass dieser mit hoher Wahrscheinlichkeit nur noch $\mathcal{O}(\log n)$ ist. Das Routing profitiert davon aber nur bis zu einem gewissen Maß, denn den einzelnen Peers ist der kürzeste Weg zu einem Rechteck nicht bekannt.

Eine gewisse Verkürzung des Weges ist dennoch möglich. Hierzu wählt das Routing den jeweils nächsten Peer aus der Realität, in der der Abstand zum Ziel gemäß der jeweiligen Koordinaten am kleinsten ist. Dadurch wird am Anfang häufig zwischen den verschiedenen Realitäten hin und her gesprungen. Ist man in einer Realität

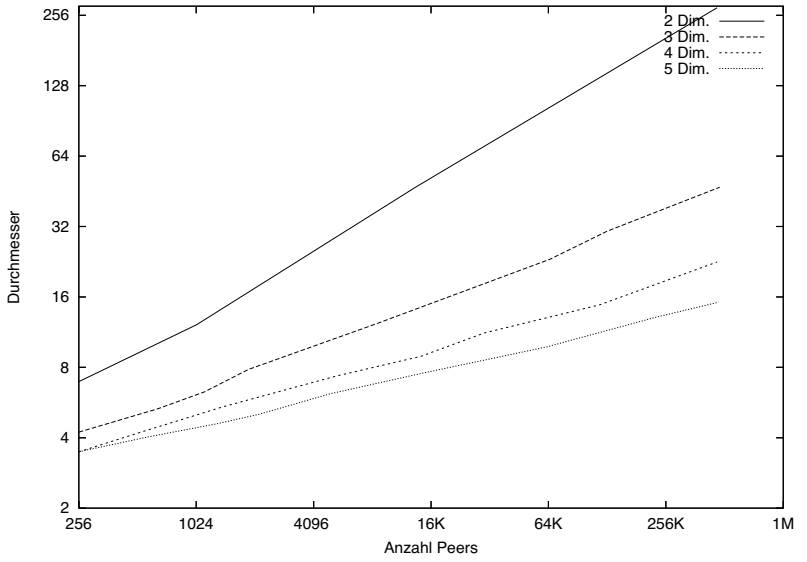


Abb. 4.9. Verringerung des Durchmessers durch Erhöhen der Dimension in CAN [8].

schon ziemlich nah am Ziel, sinkt jedoch die Wahrscheinlichkeit, dass das Ziel in einer anderen Realität näher ist.

Abbildung 4.11 zeigt, wie sich der Durchmesser eines CANs verändert, wenn mehrere Realitäten verwendet werden. Vergleicht man die resultierende Anzahl von Suchschritten mit den Gewinnen aus einer Vergrößerung der Dimension, so schneidet die Dimensionserhöhung besser ab, wenn man den Grad der Knoten jeweils gleich groß wählt.

Überladen von Rechtecken und latenzoptimiertes Routing

Bislang war jeweils ein Peer für ein Rechteck im CAN verantwortlich. Werden jedem Rechteck mehrere Peers zugeordnet, so nennt man es „Überladen“ eines Rechtecks. Hierzu wird eine Obergrenze gewählt, die wir mit `Maxpeers` bezeichnen. Die Netzwerkstruktur wird dahingehend erweitert, dass ein Peer Verbindungen zu allen anderen Peers unterhält, die für das gleiche Rechteck verantwortlich sind. Für die Verbindung zu benachbarten Rechtecken kann ein Peer jeweils einen beliebigen Peer aus den bis zu `Maxpeers` Peers wählen. Dadurch erhöht sich der Grad des Netzwerks nur um den additiven Term $\texttt{Maxpeers} - 1$. Damit die maximal `Maxpeers` Peers eines Rechtecks sich vollständig ersetzen können, unterhalten alle Peers eines

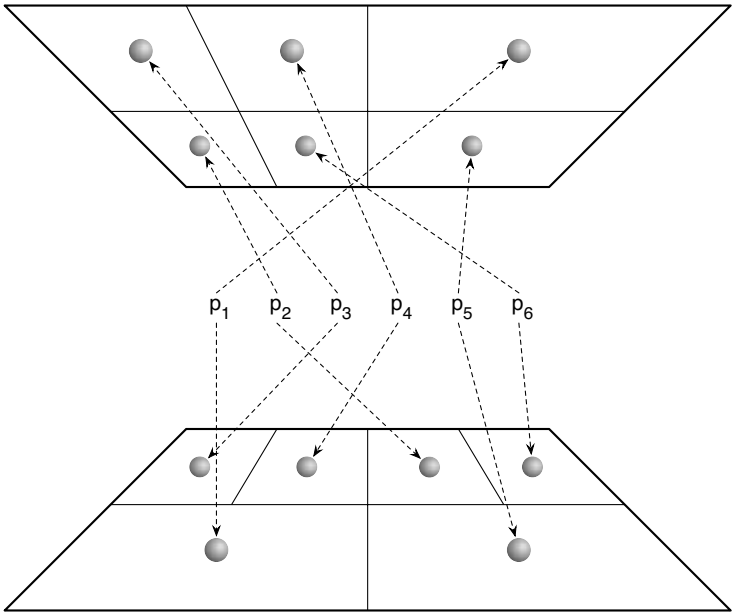


Abb. 4.10. Zwei Realitäten eines (zweidimensionalen) CANs.

Rechtecks Kopien der Daten in diesem Rechteck. Ein solches überladenes CAN ist in Abbildung 4.12 dargestellt.

Ein Vorteil dieser Methode ist, dass ein Peer für seine Netzwerkverbindungen jeweils einen beliebigen der bis zu `Maxpeers` Peers eines Rechtecks wählen kann. So kann zum Beispiel immer ein latenznaher Peer gewählt werden, um das Routing im Netzwerk zu beschleunigen. Man muss sich verdeutlichen, dass die Nachbarschaft in CAN nichts über die Laufzeit der Nachrichten aussagt. So können sich im Netzwerk benachbarte Rechner auf unterschiedlichen Kontinenten oder im selben Raum befinden. Die latenznahen Nachbarn können durch Messung der RTT (*Round Trip Time*) von Testnachrichten gefunden werden. Wenn man unter den Nachbarn den reaktionsschnellsten wählen kann, ergibt sich eine wesentliche Verbesserung der Suchgeschwindigkeit. Wir werden später sehen, dass diese Idee auch von anderen Peer-to-Peer-Netzwerken aufgegriffen worden ist.

Ein weiterer Effekt des Überladens ist die Verringerung des Netzwerkdurchmessers. Der Grund hierfür ist, dass ein unfragmentiertes Netzwerk mit n Peers nun nicht mehr auf n Rechtecke aufgeteilt wird, da für jedes Rechteck mehrere Peers erforderlich sind. Die weitere Unterteilung eines Rechtecks A findet immer nur dann statt, wenn die Anzahl der Peers in A bereits `Maxpeers` beträgt und ein neuer Peer mittels der Hash-Funktion in das Rechteck A platziert wird. So besteht ein CAN mit n Peers möglicherweise nur noch aus $\lceil n/\texttt{Maxpeers} \rceil$ Rechtecken.

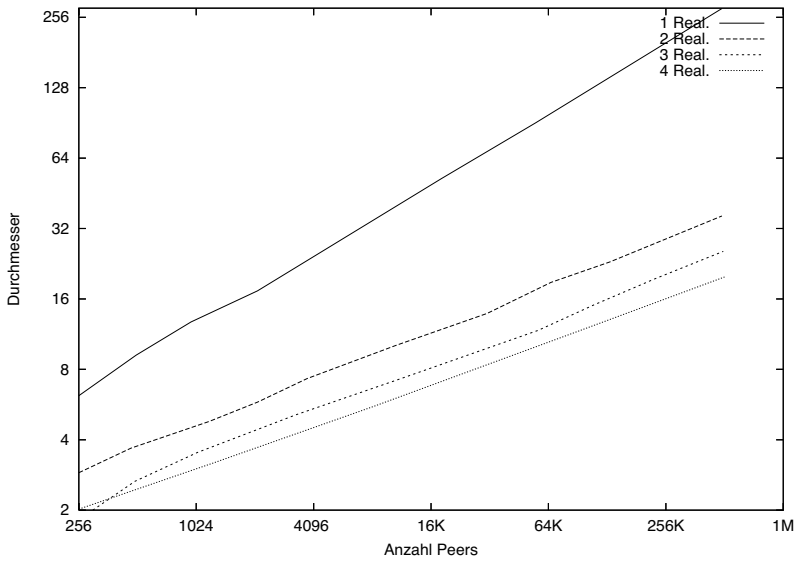


Abb. 4.11. Verringern des Durchmessers eines zweidimensionalen CANs durch Verwendung mehrerer Realitäten [8].

Ein weiterer nicht zu vernachlässigender Effekt ist, dass das Nezwerk durch den höheren Grad und die Redundanz robuster wird. Diese Robustheit kann in einem Peer-to-Peer Netzwerk von großer Bedeutung sein.

Latenzoptimiertes Routing kann natürlich auch ohne das Überladen von Rechtecken verwendet werden. Soll zum Beispiel vom Punkt $(0, 0)$ zum Punkt $(0.5, 0.5)$ geroutet werden und die Netzwerkstruktur besteht aus lauter gleich großen Rechtecken der Größe $2^{-16} \times 2^{-16}$, so müssen acht Schritte in x- und acht in y-Richtung in beliebiger Reihenfolge gegangen werden. Hierbei kann dann mindestens achtmal die Richtung gewählt werden, in der der bezüglich Latenzzeit nähere Peer liegt. Dies ist jedoch nicht so effektiv wie im Fall von überladenen Rechtecken, wo bei jedem der 16 Schritte einer von `Maxpeers` Peers gewählt werden kann. Eine noch bessere Zeitersparnis kann sich durch eine der Topologie angepasste Netzwerkkonstruktion ergeben, die das CAN von Anfang an gemäß der Latenzzeiten aufbaut. Dieser Ansatz setzt aber Struktur- und Ortsinformationen im Internet voraus, die nicht einfach herauszufinden sind.

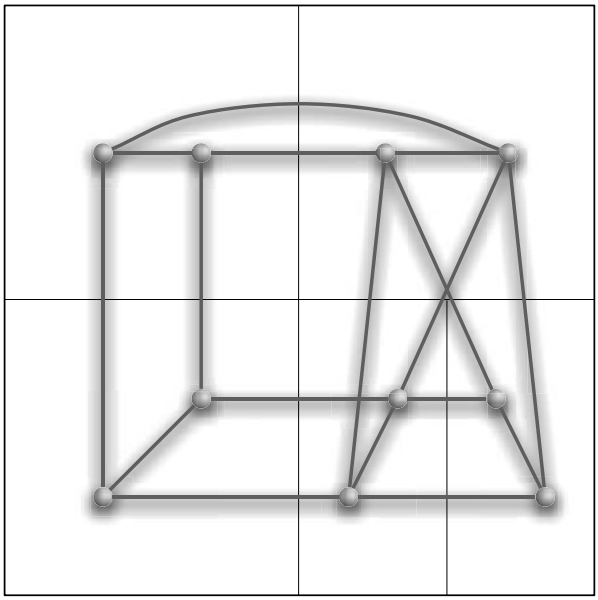


Abb. 4.12. Ein CAN mit überladenen Rechtecken.

Datenkopien

Genauso wie Peers an verschiedenen Stellen des Quadrats helfen können die Struktur zu verbessern, können auch Daten mehrfach platziert werden. Dafür wird einfach der Schlüssel für ein Datum mit einer Zahl k aus $\{1, 2, .., \texttt{Copies}\}$ kombiniert. Das erhöht ebenfalls die Robustheit und verringert die Anzahl der Zwischenstationen für die Suche und damit die Latenzzeit, da für die Suche immer die nächstgelegene Kopie gewählt werden kann. Die Anzahl der Hops zu der nächsten Kopie nimmt hierbei mit der Anzahl der Kopien ab.

4.6 Zusammenfassung

Verglichen mit Gnutella und Napster ist CAN ein großer Fortschritt. Zum ersten Mal kann man garantieren, dass ein Datum gefunden wird, wenn es im Netzwerk vorhanden ist. Hierzu wird die Anfrage nur gemäß einer Richtung zum Ziel weitergeleitet. Nur ein kleiner Bruchteil der Peers ist involviert und der Weg vom anfragenden Peer zu demjenigen Peer, der das Datum bereitstellt, ist relativ kurz. In einem zweidimensionalen CAN sind nur $\mathcal{O}(n^{1/2})$ Peers an einer Suche beteiligt. Dies ist ein erheblicher Fortschritt im Vergleich zu Gnutella, das nicht garantieren kann, dass die Suche überhaupt erfolgreich ist.

CAN ist auch das erste Netzwerk, das das Konzept der verteilten Hash-Tabellen (DHT) verwendet. Das ist mittlerweile eine Standardmethode im Bereich der Peer-

to-Peer-Netzwerke. Trotz allem hat sich CAN nicht durchgesetzt, denn die Peer-to-Peer-Netzwerke, die wir in den folgenden Kapiteln vorstellen, sind hinsichtlich Grad und Routing-Zeit noch effizienter.

5

Chord

A chord is by no means an agglomeration of intervals.
Paul Hindemith

Beim von Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek und Hari Balakrishnan entwickelten *Chord*-Netzwerk [10] handelt es sich um eins der bekanntesten Peer-to-Peer-Netzwerke wissenschaftlicher Herkunft. Dies liegt zum einen sicherlich daran, dass Chord eins der ersten Peer-to-Peer Netzwerke wissenschaftlichen Ursprungs ist. Nicht weniger ausschlaggebend dürfte jedoch die Tatsache sein, dass es mit Chord gelungen ist, eine vergleichsweise einfache Netzwerkstruktur aufzubauen, die zugleich sehr effizient ist.

5.1 Verteilte Hash-Tabellen in Chord

Wie schon bei CAN im vorigen Kapitel, werden in Chord verteilte Hash-Tabellen für die Zuordnung zwischen Daten und Peers eingesetzt. Das Prinzip der verteilten Hash-Tabellen wird in beiden Netzwerken jedoch unterschiedlich umgesetzt. Wir erinnern uns, dass im Fall von CAN ein zweidimensionaler Raum unter den Peers aufgeteilt wurde. Um die Effizienz zu steigern, wurde dann zum Beispiel die Anzahl der Dimensionen weiter erhöht. Chord wählt gerade den anderen Weg und beschränkt sich auf einen eindimensionalen Raum.

Bezeichne im Folgenden $P = \{p_1, \ldots, p_n\}$ die Menge der Peers und $X = \{x_1, \ldots, x_k\}$ die Menge der Daten im Netzwerk. Bei Chord werden Peers sowie Daten mittels kryptographischer Hash-Funktionen $h : P \to \mathbb{Z}_{2^m}$ und $h' : X \to \mathbb{Z}_{2^m}$ auf die Zahlen in $\mathbb{Z}_{2^m} = \{0, \ldots, 2^m - 1\}$ abgebildet. Dass hier auf ganze Zahlen anstelle eines kontinuierlichen Intervalls abgebildet wird, ist ein unwesentliches Detail. Die Zahl m muss genügend groß gewählt werden, um die Wahrscheinlichkeit von Kollisionen beim Hashing vernachlässigbar klein zu halten. Ein typischer Wert für m ist 128.

Für die Zuordnung der Daten zu den Peers wird der Zahlenraum $\mathbb{Z}_{2^m}$ als Ring betrachtet. Deshalb und wegen der Verbindungsstruktur, die wir im nächsten Abschnitt

betrachten werden, wird auch vom so genannten *Chord-Ring* gesprochen. Auf dem Chord-Ring wird jedes Datum demjenigen Peer zugeordnet, der der Position (also dem Hash-Wert) des Datums in aufsteigender Zählrichtung folgt. Somit ist ein Peer für den Bereich hinter seinem Vorgänger bis zu seiner eigenen Position auf dem Chord-Ring zuständig. Abbildung 5.1 veranschaulicht die Zuordnung von Daten zu Peers an einem Chord-Ring der Größe 2^3 mit 3 Peers und 3 Dateien.

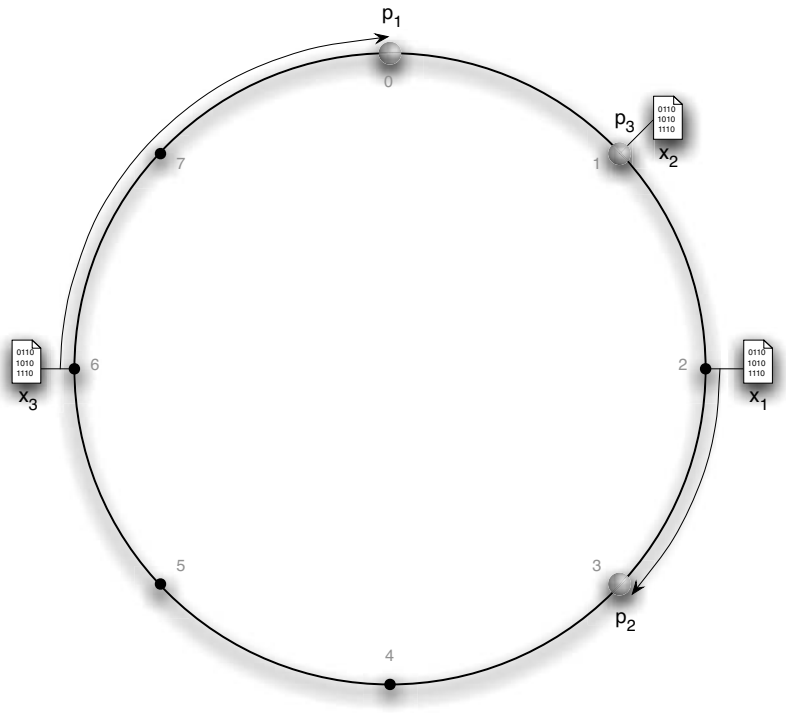


Abb. 5.1. Zuordnung von Daten zu Peers in Chord.

Wenn dem Netzwerk nun ein neuer Peer p_{n+1} beitritt, so wird diesem durch Hashing die Position $h(p_{n+1})$ auf dem Chord-Ring zugewiesen. Sei Peer p_i zuständig für den Bereich $h(p_j) + 1$ bis $h(p_i)$ des Chord-Rings. Liegt $h(p_{n+1})$ nun in diesem Bereich, so wird Peer p_{n+1} Peer p_i kontaktieren, und die Zuständigkeiten werden wie folgt verändert: Der Zuständigkeitsbereich von p_i wird auf den Bereich von $h(p_{n+1}) + 1$ bis $h(p_i)$ verkleinert, und der neue Peer p_{n+1} ist von nun an verantwortlich für den Bereich $h(p_j) + 1$ bis $h(p_{n+1})$ (siehe Abbildung 5.2).

Meldet sich ein Peer p_j vom Netzwerk ab oder fällt dieser Peer aus technischen Gründen aus, so muss der bislang durch p_j verwaltete Bereich des Chord-Rings von einem anderen Peer übernommen werden. Der dafür zuständige Peer ist nach dem oben beschriebenen Schema der Nachfolger p_i von p_j auf dem Chord-Ring. Falls ein

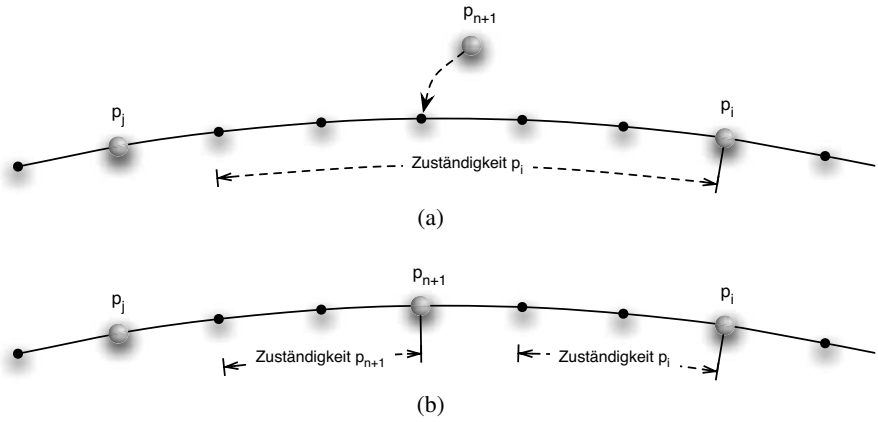


Abb. 5.2. Veränderung der Zuständigkeiten auf dem Chord-Ring beim Einfügen eines Peers.

anderer Peer p_k, $k \neq j$ den Ausfall von p_j als erster bemerkt, so wird p_i von p_k über den Ausfall informiert und p_i anschließend seinen Zuständigkeitsbereich anpassen.

Wie schon im Fall von CAN, ändert sich auch bei Chord beim Einfügen bzw. Entfernen eines Peers lediglich der Zuständigkeitsbereich genau eines anderen Peers. Somit ist auch hier die für dynamische Netzwerke wichtige Konsistenzeigenschaft gegeben.

Wir werden nun untersuchen, wie fair die Daten auf die Peers verteilt werden. Wenn die Hash-Funktion H' die Daten gleichmäßig auf dem Chord-Ring verteilt, ist die Anzahl der Daten, die ein Peer zu verwalten hat, proportional zur Größe des von ihm verwalteten Bereichs des Chord-Rings. Da die Peers durch die Hash-Funktion H ebenfalls zufällig auf dem Chord-Ring verteilt werden, wird jeder der n Peers im Erwartungswert einen Bereich der Größe $\frac{2^m}{n}$ verwalten.

Natürlich wird es Peers geben, die einen Bereich des Chord-Rings verwalten müssen, der mehr oder weniger vom Erwartungswert abweicht. Das folgende Lemma zeigt, welche Schranken hierbei mit hoher Wahrscheinlichkeit gelten.

Lemma 5.1. *In einem Chord-Netzwerk mit n Peers und Ringgröße 2^m ist der Abstand zweier Peers auf dem Chord-Ring (und somit der Zuständigkeitsbereich eines Peers) mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\frac{2^m}{n} \log n)$ und nicht kleiner als $\Omega(\frac{2^m}{n^c})$.*

Beweis. Für die untere Schranke betrachten wir einen Peer p_i, der einen Bereich der Größe $\frac{2^m}{n^c}$ des Chord-Rings verwaltet. Die Wahrscheinlichkeit, dass ein Peer, der dem Netzwerk beitritt, einer Position in diesem Bereich zugewiesen wird und den Zuständigkeitsbereich von p_i weiter verkleinert, ist n^{-c}. Da dies höchstens n mal passiert, ist die Wahrscheinlichkeit, dass der Zuständigkeitsbereich von p_i weiter verkleinert wird, beschränkt durch n^{-c+1}.

Für die obere Schranke untersuchen wir die Wahrscheinlichkeit, dass kein Peer einer Position in einem festen Bereich B der Größe $c\frac{2^m}{n} \log n$ zugeordnet wird und

somit einer der Peers für einen Bereich dieser Größe zuständig sein muss. Da ein Peer mit Wahrscheinlichkeit $\frac{c \log n}{n}$ Bereich B zugeordnet wird, gilt, dass ein Peer mit Wahrscheinlichkeit $1 - \frac{c \log n}{n}$ eine Postition außerhalb von B erhält. Hieraus folgt

$$\begin{aligned} Pr[\text{kein Peer wird } B \text{ zugeordnet}] &= \left(1 - \frac{c \log n}{n}\right)^n \\ &\leq e^{-\frac{c \log n}{n} n} \\ &\leq n^{-c} \,. \end{aligned}$$

Damit wird der Bereich B mit hoher Wahrscheinlichkeit durch einen der n Peers unterteilt, und der Abstand zweier Peers auf dem Chord-Ring ist mit hoher Wahrscheinlichkeit geringer als $\mathcal{O}(\frac{2^m}{n} \log n)$. □

Mit Hilfe von Lemma 5.1 können wir nun genauer abschätzen, für wie viele Daten ein Peer höchstens verantwortlich sein wird.

Theorem 5.2. *In einem Chord-Netzwerk mit n Peers und k Daten verwaltet jeder der Peers mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log n)$ Daten, falls $k \in \mathcal{O}(n)$, und höchstens $\mathcal{O}\left(\frac{k}{n} \log n\right)$ Daten, falls $k \in \Omega(n)$.*

Beweis. Wir betrachten einen Peer p_i. Aus Lemma 5.1 wissen wir, dass für den Zuständigkeitsbereich B_{p_i} von p_i mit hoher Wahrscheinlichkeit $B_{p_i} \in \mathcal{O}(\frac{2^m}{n} \log n)$ und damit $B_{p_i} \leq c\frac{2^m}{n} \log n$ für eine Konstante c gilt. Da die Daten durch eine kryptographische Hash-Funktion gemäß unseres Modells zufällige Positionen auf dem Chord-Ring erhalten, ist die Wahrscheinlichkeit, dass ein Datum p_i zugewiesen wird, proportional zur Größe von B_{p_i}. Wir nehmen im Folgenden an, dass p_i tatsächlich für einen Bereich der Größe $c\frac{2^m}{n} \log n$ verantwortlich ist. So können wir die Zuordnung der k Daten, entweder zu Peer p_i oder einem der anderen Peers, als unabhängige Bernoulli Experimente $X_1, \ldots, X_k$ betrachten.

Die Erfolgswahrscheinlichkeit $Pr[X_j = 1]$, d.h., ein Datum wird p_i zugeordnet, ist damit gegeben durch

$$Pr[X_j = 1] = \frac{c\, 2^m \log n}{n\, 2^m} = \frac{c \log n}{n} \,.$$

Sei nun $X = \sum_{j=1}^{k} X_j$ die Anzahl der erfolgreichen Experimente bzw. die Anzahl der p_i zugewiesenen Daten. Dann ist die erwartete Anzahl der p_i zugewiesenen Daten in einem Netzwerk mit k Daten

$$E[X] = k \frac{c \log n}{n} \,.$$

Wir können jetzt mit Hilfe der Chernoff Schranke (siehe Seite 268) abschätzen, wie viele Daten p_i mit hoher Wahrscheinlichkeit höchstens zugewiesen werden. Die Chernoff-Schranke besagt, dass für jedes $\delta > 0$

$$Pr\left[X \geq (1+\delta)E[X]\right] \leq e^{-\frac{1}{3}\min\{\delta,\delta^2\}E[X]} \tag{5.1}$$

gilt. In der folgenden Analyse werden wir die beiden Fälle $k \in \mathcal{O}(n)$ und $k \in \Omega(n)$ unterscheiden.

Wir zeigen zunächst, dass im Fall $k \in \mathcal{O}(n)$ jeder Peer mit hoher Wahrscheinlichkeit höchstens für $\mathcal{O}(\log n)$ Daten verantwortlich ist. Hierfür wählen wir $\delta = \frac{3c' \ln n}{E[X]}$ für eine beliebige aber feste Konstante c', so dass $\delta \geq 1$ gilt. Dies ist möglich, weil in diesem Fall $E[X] \in \mathcal{O}(\log n)$ gilt. Durch das so gewählte δ ergibt sich auf der linken Seite von Ungleichung 5.1

$$\begin{aligned}(1+\delta)E[X] &= E[X] + 3c' \ln n \\ &= \mathcal{O}(\log n)\ .\end{aligned}$$

Des Weiteren gilt $\delta \leq \delta^2$ und wir erhalten durch einsetzen in Ungleichung 5.1

$$\begin{aligned}Pr\left[X \geq (1+\delta)E[X]\right] &\leq e^{-\frac{1}{3}\min\{\delta,\delta^2\}E[X]} \\ &\leq e^{-\frac{1}{3}\delta E[X]} \\ &= e^{-c' \ln n} \\ &= n^{-c'}\ ,\end{aligned}$$

was die erste Aussage beweist.

Wir betrachten nun den Fall $k \in \Omega(n)$ und zeigen, dass jeder Peer mit hoher Wahrscheinlichkeit höchstens für $\mathcal{O}\left(\frac{k}{n}\log n\right)$ Daten verantwortlich ist. Hierfür wählen wir $\delta \geq 1$ als Konstant und groß genug, so dass $\delta E[X] \geq 3\ c' \ln n$ für eine zuvor gewählte Konstante c' gilt. Dies ist möglich, da jetzt $E[X] \in \Omega(\log n)$ ist. Diese Wahl von δ liefert

$$\begin{aligned}(1+\delta)E[X] &\leq (1+\delta)k\frac{c\log n}{n} \\ &= \mathcal{O}\left(\frac{k}{n}\log n\right)\end{aligned}$$

für die linke Seite von Ungleichung 5.1. Wiederum gilt $\delta \leq \delta^2$ und durch einsetzen erhalten wir

$$\begin{aligned}Pr\left[X \geq (1+\delta)E[X]\right] &\leq e^{-\frac{1}{3}\min\{\delta,\delta^2\}E[X]} \\ &\leq e^{-\frac{1}{3}\delta E[X]} \\ &\leq e^{-c' \ln n} \\ &= n^{-c'}\ .\end{aligned}$$

Dies beweist die zweite Aussage und damit das Theorem. □

5.2 Netzwerkstruktur und Routing

Um die im vorigen Abschnitt beschriebene verteilte Hash-Tabelle in einem Netzwerk zu realisieren, genügt ein Minimum an Routing-Information: Jeder Peer muss

lediglich seinen Vorgänger und Nachfolger auf dem Chord-Ring kennen. Die Suche nach einem Datum kann dann einfach entlang der Ringstruktur geschehen, in dem die Suchanfrage solange weitergeleitet wird, bis der erste Peer erreicht wird, dessen Hash-Wert größer als der des gesuchten Datums ist. Abgesehen davon, dass diese Art der Suche recht viel Zeit, nämlich linear in der Anzahl der Peers, benötigt, ist ein solches Ringnetzwerk auch nicht sehr fehlertolerant. Schon beim Ausfall zweier Peers zerfällt ein Ring in zwei Zusammenhangskomponenten.

Um diese beiden Probleme zu beheben, besitzt jeder Peer des Chord-Netzwerks neben Zeigern auf seinen Vorgänger und Nachfolger auf dem Chord-Ring noch so genannte *Finger-Zeiger*. Sei der Zahlenraum des Chord-Rings wieder $\mathbb{Z}_{2^m}$, dann unterhält jeder Peer p höchstens m solcher Finger-Zeiger in einer so genannten *Finger-Tabelle*. Der i-te Eintrag ($1 \leq i \leq m$) der Finger-Tabelle besteht dabei jeweils aus der Adresse des Peers p_j, der der Position $(h(p) + 2^{i-1}) \mod 2^m$ auf dem Chord-Ring zugeordnet ist. Ist diese Postion des Chord-Rings unbesetzt, d.h., es existiert kein Peer p_j mit $h(p_j) = (h(p) + 2^{i-1}) \mod 2^m$, so wird derjenige Peer gewählt, der dieser Position in aufsteigender Zählrichtung folgt. Der Einfachheit halber werden wir diesen Zeiger eines Peers auch als *i-ten Finger* oder als *Finger*$[i]$ referenzieren. Der i-te Fingerzeiger eines Peers überbrückt also eine Distanz von 2^{i-1} auf dem Chord-Ring. Des Weiteren ist zu beachten, dass die Finger-Zeiger im Fall $i = 1$ gerade den Nachfolgerzeigern des Chord-Rings entsprechen.

Abbildung 5.3 veranschaulicht die Finger-Zeiger. Dargestellt werden die Finger-Zeiger des Peers p_4 mit $h(p_4) = 0$. Die duchgezogenen Pfeile zeigen auf die Zielpositionen $(h(p) + 2^{i-1}) \mod 2^m$ der Finger-Zeiger von p_4 auf dem Chord-Ring. Einige Positionen sind jedoch nicht mit Peers besetzt. Dann unterhält p_4 Finger-Zeiger zu den jeweils nachfolgenden Peers, gekennzeichnet mit gestrichelten Pfeilen. In diesem Fall sind das gerade Peer p_1 an Position 2, p_5 an Position 6, p_3 an Position 12 und p_6 an Position 17.

Die ersten Finger-Zeiger, die nur kleine Distanzen überbrücken, zeigen zumeist nur auf den selben Peer, da die Anzahl n der Peers im Netzwerk typischerweise wesentlich geringer als die Größe 2^m des Chord-Rings ist. So wird die Anzahl unterschiedlicher Finger-Zeiger zumeist deutlich niedriger als das theoretische Maximum m sein. Das folgende Lemma liefert eine genauere Aussage.

Lemma 5.3. *Ein Peer im Chord-Netzwerk besitzt mit hoher Wahrscheinlichkeit lediglich $\mathcal{O}(\log n)$ unterschiedliche Finger-Zeiger.*

Beweis. Diese Aussage folgt aus der unteren Schranke des Zuständigkeitsbereichs eines Peers (siehe Lemma 5.1). Vereinfachend nehmen wir an, dass alle Peers einen Abstand von nur $\frac{2^m}{n^c}$ haben. Rein theoretisch könnten sich maximal n^c Peers mit diesem Abstand im Netzwerk befinden. Somit ergibt sich, dass ein Peer mit hoher Wahrscheinlichkeit lediglich $\log n^c = c \log n$ unterschiedliche Finger-Zeiger besitzt. □

Damit ist die Chord-Datenstruktur sehr kompakt: Jeder Peer besitzt mit hoher Wahrscheinlichkeit lediglich eine Nachbarschaft logarithmischer Größe. Somit ist die im Chord-Netzwerk von einem Peer zu verwaltende Nachbarschaftsinformation auch im Fall von sehr vielen Netzwerkteilnehmern noch überschaubar gering.

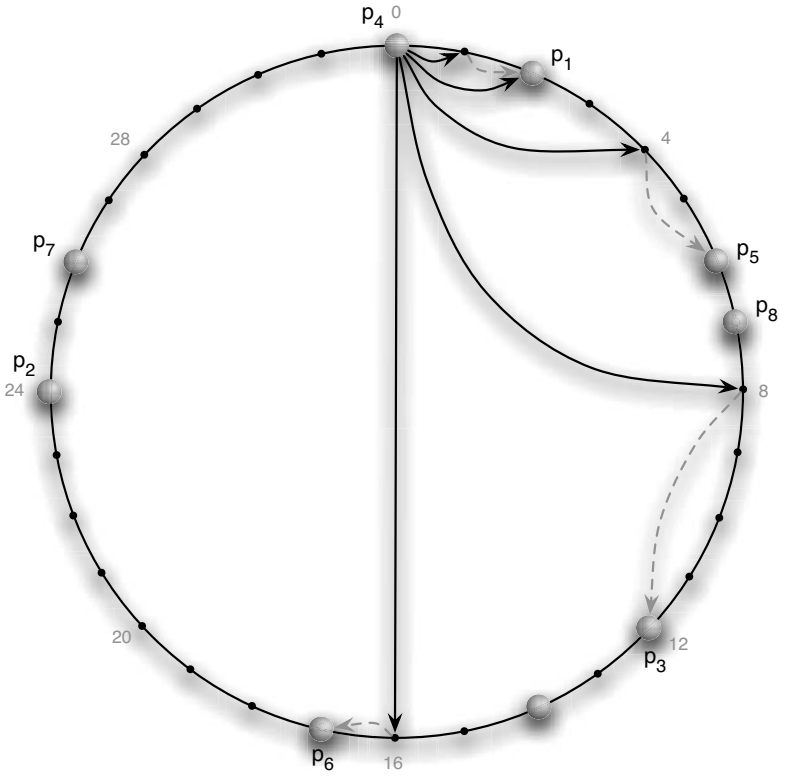


Abb. 5.3. Die Finger-Zeiger eines Peers in Chord.

Routing in Chord

Wir werden nun sehen, wie mit Hilfe der Finger-Zeiger der für ein Datum verantwortliche Peer im Netzwerk gefunden werden kann. Das Vorgehen bei der Suche ist denkbar einfach: Ein Peer p, der nach einem Datum x mit Position $h'(x) = id$ auf dem Chord-Ring sucht, überprüft zunächst, ob er das Datum x selbst verwaltet. Ist dies nicht der Fall, wird die Suchanfrage über einen der Finger-Zeiger an einen anderen Peer weitergeleitet. Hierbei wird immer der Finger-Zeiger gewählt, der am nächsten an die Postion $h'(x) = id$ auf dem Chord-Ring zeigt und gleichzeitig noch vor der Position id liegt. Auf diese Weise wird die Anfrage an den letzten Peer p' vor Position id des Chord-Rings geleitet. Der Nachfolger von p' ist dann der für das Datum x verantwortliche Peer. Der gerade beschriebene Algorithmus findet sich als Pseudo-Code in Abbildung 5.4.

Abbildung 5.5 zeigt den möglichen Verlauf einer Suche im Netzwerk. Wichtig ist wiederum die Frage, wie viele Peers in eine Suchanfrage involviert sind und wieviele Nachrichten benötigt werden. Das folgende Theorem zeigt, dass die Suche in Chord sehr effizient ist.

```
// finde ausgehend von p den für id verantwortlichen Peer
p.suche(id)
    if id ∈ (h(p), p.Nachfolger]
        return p.Nachfolger
    else
        // wähle größten Finger-Zeiger von p, der noch vor id liegt
        for i ← 1 to m
            if p.Finger[i] ∈ [h(p), id)
                p' ← p.Finger[i]
        return p'.suche(id)
```

Abb. 5.4. Pseudo-Code Algorithmus für die Suche in Chord. Variablen wie Finger-Zeigern und Funktionsaufrufen ist jeweils der betreffende Peer vorangestellt. So bezeichnet z.B. $p.Finger[i]$ den Peer, auf den der i-te Finger-Zeiger des Peers p zeigt.

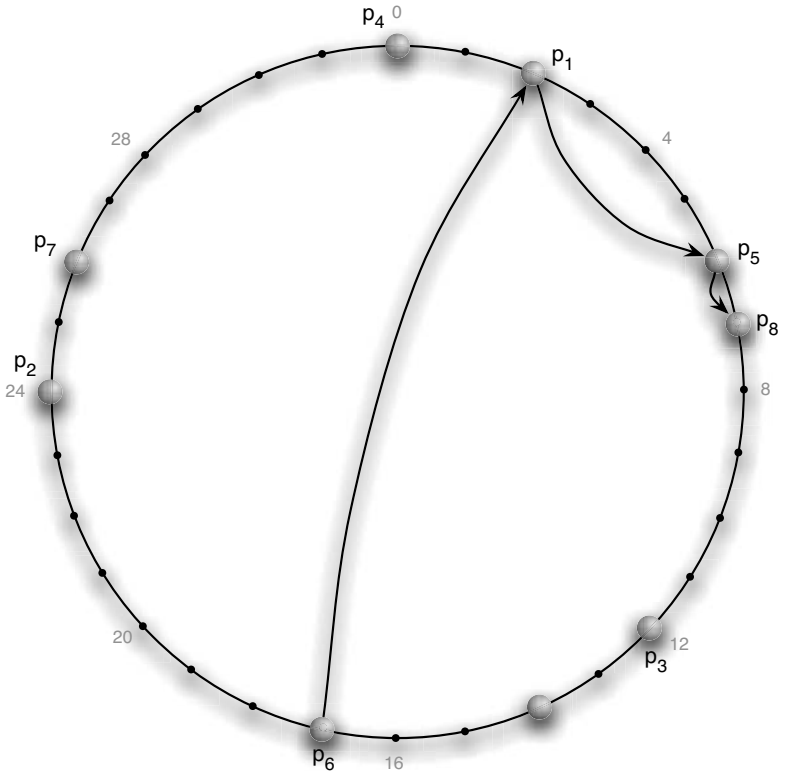


Abb. 5.5. Suche im Chord-Netzwerk mittels des Algorithmus aus Abbildung 5.4. Ausgehend von Peer p_6 mit Position $h(p_6) = 17$ auf dem Chord-Ring wird nach einem Datum mit $id = 8$ gesucht.

Theorem 5.4. *Die Suche im Chord-Netzwerk benötigt höchstens $\mathcal{O}(m)$ und mit hoher Wahrscheinlichkeit nicht mehr als $\mathcal{O}(\log n)$ Nachrichten.*

Beweis. Für die obere Schranke von $\mathcal{O}(m)$ Nachrichten müssen wir zeigen, dass die Distanz zwischen Start und Ziel der Suche in jedem Schritt mindestens halbiert wird. Bei Ring-Größe 2^m ergeben sich dann $\log 2^m = m$ Schritte im obigen Algorithmus und somit $\mathcal{O}(m)$ Nachrichten.

Nehmen wir an, Peer p sucht ein Datum mit Position $h(p) - 1$ auf dem Chord-Ring, also ein Datum, für das sein Vorgänger p' verantwortlich ist, sofern p nicht der einzige Peer im Netzwerk ist. Da bei der Suche nur im Uhrzeigersinn auf dem Ring vorangeschritten wird, ist die Distanz zwischen Start und Ziel der Suche hier $2^m - 1$ und somit maximal.

Peer p wird die Suchanfrage an denjenigen der ihm bekannten Peers weiterleiten, der sich am nächsten an Position $h(p) - 1$ befindet, ohne diese Position zu überschreiten. Dies wäre der im m-ten Finger-Zeiger *Finger*$[m]$ gespeicherte Peer. Wir erinnern uns, dass der i-te Finger-Zeiger eine Distanz von mindestens 2^{i-1} auf dem Cord-Ring überbrückt. Somit wird die Suche an einen Peer weitergeleitet, dessen Distanz zum Ziel höchstens noch $2^m - 2^{m-1}$ beträgt. Das bedeutet, dass die Distanz zum Ziel halbiert wird. Dies gilt auch für alle weiteren Schritte der Suche, so dass die Distanz nach m Schritten Eins ist und wir am Ziel angekommen sind. So sind maximal $\mathcal{O}(m)$ Peers in eine Suche involviert und es werden maximal $\mathcal{O}(m)$ Nachrichten benötigt.

Da die Positionen der Peers auf dem Chord-Ring jedoch zufällig sind und wir annehmen, dass sich wesentlich weniger als 2^m Peers im Netzwerk befinden, benötigt die Suche mit hoher Wahrscheinlichkeit lediglich $\mathcal{O}(\log n)$ Nachrichten. Dies gilt, da die Distanz zwischen aktuellem Peer und Ziel nach $\log n$ Schritten der Suche höchstens $\frac{2^m}{2^{\log n}} = \frac{2^m}{n}$ beträgt. Die erwartete Anzahl an Peers in einem Bereich dieser Größe ist im Erwartungswert eins und nach Lemma 5.1 mit hoher Wahrscheinlichkeit geringer als $\mathcal{O}(\log n)$. Hieraus folgt die Behauptung. □

Nicht weniger wichtig als eine effiziente Suche ist das effiziente Einbinden neuer Peers in das Netzwerk. Das gilt insbesondere im Fall von Peer-to-Peer Netzwerken: Diese unterliegen typischerweise einer andauernden Veränderung der Netzwerkteilnehmer. So werden wir im Folgenden analysieren, wie ein neuer Peer in ein Chord-Netzwerk eingebunden werden kann und wie viele Nachrichten notwendig sind.

In Abschnitt 5.1 haben wir bereits gesehen, dass beim Einfügen eines neuen Peers p_{n+1} der Zuständigkeitsbereich des für die Position $h(p_{n+1})$ des Chord-Rings verantwortlichen Peers p_i zwischen p_i und p_{n+1} aufgeteilt wird (siehe Abbildung 5.2). Um dies zu initiieren, muss Peer p_{n+1} also zunächst Peer p_i kontaktieren. Dies geschieht ausgehend von einem beliebigen, Peer p_{n+1} bereits bekannten, Peer mit Hilfe des Such-Algorithmus und benötigt mit hoher Wahrscheinlichkeit lediglich $\mathcal{O}(\log n)$ Nachrichten.

Wurde der Bereich des Chord-Rings aufgeteilt und p_{n+1} in die Ringstruktur eingebunden, gilt es noch die Finger-Zeiger anzupassen. Hier müssen einerseits die ausgehenden Finger-Zeiger von p_{n+1} aufgebaut werden und andererseits Finger-Zeiger von Peers aktualisiert werden, die in den Bereich zwischen p_{n+1} und seinem

Vorgänger auf dem Chord-Ring zeigen. Betrachten wir zunächst die Finger-Zeiger von p_{n+1}. Wir haben bereits gesehen, dass ein Peer mit hoher Wahrscheinlichkeit nur $\mathcal{O}(\log n)$ verschiedene Finger-Zeiger besitzt, so dass das Anpassen der Finger-Zeiger von p_{n+1} mit hoher Wahrscheinlichkeit mit $\mathcal{O}\left(\log^2 n\right)$ Nachrichten geschehen kann.

Die Frage, wie viele Nachrichten für das Anpassen der auf p_{n+1} zeigenden Peers notwendig sind, ist schwieriger zu beantworten. Wir überlegen uns zunächst, wie viele Peers auf p_{n+1} zeigen werden. Das folgende Lemma liefert eine obere Schranke für diesen Wert.

Lemma 5.5. *Die Anzahl der Peers, die einen Finger-Zeiger auf einen Peer p besitzen, ist im Erwartungswert $\mathcal{O}(\log n)$ und mit hoher Wahrscheinlichkeit beschränkt durch $\mathcal{O}\left(\log^2 n\right)$.*

Beweis. Die Aussage über den Erwartungswert ergibt sich daraus, dass die Anzahl der eingehenden Kanten gleich der Anzahl der ausgehenden Kanten ist. Damit ist für einen Peer der erwartete Eingrad gleich dem erwarteten Ausgrad.

Um zu zeigen, dass der Eingrad mit hoher Wahrscheinlichkeit durch $\mathcal{O}\left(\log^2 n\right)$ beschränkt ist, betrachten wir ein Peer p_k mit Vorgänger p_j auf dem Chord-Ring. Aus Lemma 5.1 folgt, dass es mit hoher Wahrscheinlichkeit nur $\mathcal{O}(\log n)$ Bereiche gibt, aus denen Peers auf p_k zeigen.

Auf p_k zeigen alle diejenigen Peers, die einen Finger-Zeiger in das Intervall $I = (h(p_j), h(p_k)]$ des Chord-Rings besitzen. Dieses Intervall ist wiederum nach Lemma 5.1 mit hoher Wahrscheinlichkeit nicht größer als $c\frac{2^m}{n}\log n$. Wir nehmen im Folgenden eben diesen großen Abstand zwischen p_j und p_k an. Damit haben die $\mathcal{O}(\log n)$ Bereiche, in denen auf p_k zeigende Peers liegen, ebenfalls mit hoher Wahrscheinlichkeit eine Größe von maximal $c\frac{2^m}{n}\log n$.

Wir untersuchen nun, wie viele Peers sich in einem Intervall I der Größe $l = c\frac{2^m}{n}\log n$ befinden. Sei X die Zufallsvariable, die die Anzahl der Peers im Intervall I angibt. Nach Chernoff gilt für $\delta \geq 1$

$$Pr[X \geq (1+\delta)c\log n)] \leq e^{-\frac{1}{3}\delta c \log n} \leq n^{-\frac{1}{3}\delta c}\ ,$$

und somit befinden sich mit hoher Wahrscheinlichkeit nicht mehr als $\mathcal{O}(\log n)$ Peers in einem Intervall der Größe l.

Fassen wir unsere Überlegungen zusammen, ergibt sich, dass Peers aus $\mathcal{O}(\log n)$ Bereichen auf den Peer p_k zeigen und sich in jedem dieser Bereiche mit hoher Wahrscheinlichkeit nicht mehr als $\mathcal{O}(\log n)$ Peers befinden. Somit ist der Eingrad von p_k mit hoher Wahrscheinlichkeit durch $\mathcal{O}\left(\log^2 n\right)$ beschränkt. □

Wenden wir uns wieder der Anzahl benötigter Nachrichten zum Einfügen eines neuen Peers p_{n+1} zu. Wir wissen jetzt, dass mit hoher Wahrscheinlichkeit nicht mehr als $\mathcal{O}\left(\log^2 n\right)$ Peers auf p_{n+1} zeigen werden. Diese Peers gilt es nun effizient über den neuen Netzwerkteilnehmer p_{n+1} zu benachrichtigen.

Wir nutzen aus, dass die $\mathcal{O}\left(\log^2 n\right)$ Peers in $\mathcal{O}(\log n)$ zusammenhängenden Bereichen liegen und betrachten zunächst nur einen dieser Bereiche. Um einen Peer

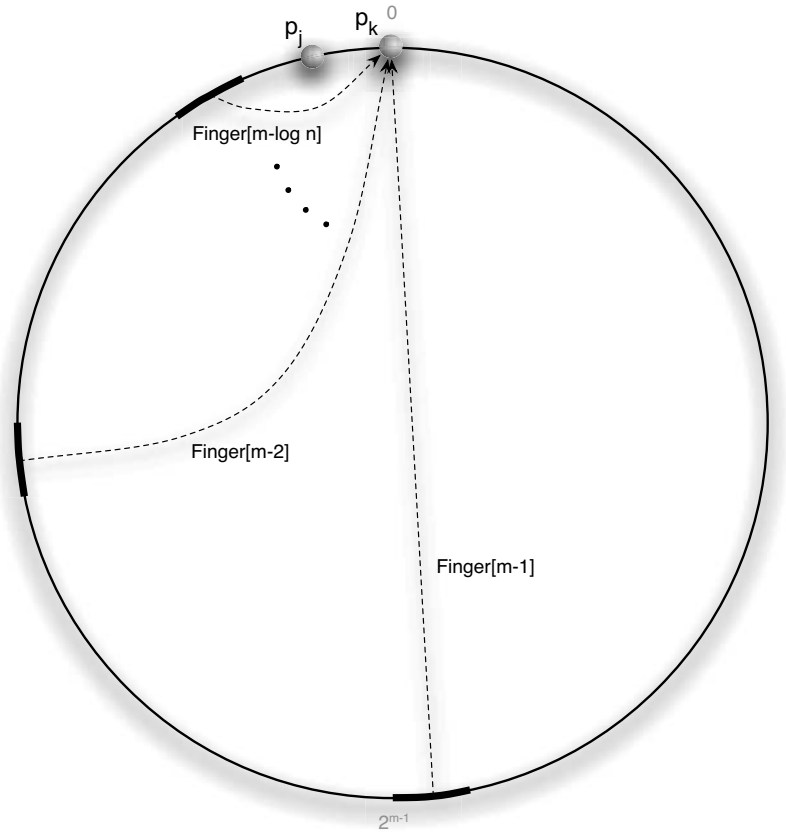


Abb. 5.6. Eingrad eines Peers p_k in Chord. Es zeigen Peers aus $\log n$ Bereichen des Chord-Rings auf einen Peer p_k.

in diesem Bereich zu benachrichtigen, können wir den Suchalgorithmus verwenden. Um dann die weiteren $\mathcal{O}(\log n)$ Peers dieses Bereichs zu informieren, sind nur $\mathcal{O}(\log n)$ weitere Nachrichten erforderlich, wenn die Nachricht entlang des Chord-Rings weitergeleitet wird. Des Weiteren ist der Aufwand zur Anpassung des jeweiligen Finger-Zeigers konstant. Somit benötigen wir $\mathcal{O}(\log n)$ Nachrichten für die Anpassung der Finger-Zeiger aller Peers eines dieser Bereiche.

Da es insgesamt $\mathcal{O}(\log n)$ dieser Bereiche gibt, werden mit hoher Wahrscheinlichkeit $\mathcal{O}\left(\log^2 n\right)$ benötigt, um alle auf p_{n+1} zeigenden Finger-Zeiger anzupassen. Durch unsere Überlegungen ergibt sich folgendes Theorem:

Theorem 5.6. *Um einen neuen Peer in Chord einzufügen, genügen mit polynomiell hoher Wahrscheinlichkeit* $\mathcal{O}\left(\log^2 n\right)$ *Nachrichten.*

5.3 Latenzoptimiertes Routing

Wir haben bereits im Fall von CAN gesehen, dass es Möglichkeiten gibt, die Latenzzeiten beim Routing zu verringern oder die Zuverlässigkeit eines Netzwerks zu erhöhen. Auf den ersten Blick ist dies für Chord nicht ohne Weiteres möglich. Tatsächlich kann man aber mit Hilfe neuer Techniken auch in Chord die Latenzzeiten verringern. Wir werden hier die von Dabek, Li, Robertson und Kaashoek in [39] vorgestellten Ergebnisse erläutern.

Wir gehen dabei von der im letzten Abschnitt beschriebenen Chord-Struktur mit Chord-Ring und Finger-Zeigern aus. Unser Ziel ist es, Routing-Zeiten im Netzwerk zu reduzieren. Um dies zu erreichen, werden wir zum einen den Such-Algorithmus etwas abwandeln und zum anderen versuchen, die Chord-Verbindungsstruktur dahingehend zu optimieren, dass die Finger-Zeiger der Peers zu möglichst latenznahen Peers führen — also Peers, die im Underlay-Netzwerk, dem Internet, nicht weit entfernt sind.

Der im letzten Abschnitt vorgestellte Such-Algorithmus (siehe Abbildung 5.4) benötigt doppelt so viele Nachrichten, wie Peers besucht werden. Mit jedem weiteren Schritt der Suche werden zwei weitere Nachrichten benötigt: eine zum nächsten Peer p der Suche und die entsprechende Antwortnachricht des Peers p. Verläuft die Suche also über Peers $p_1, \ldots, p_j$, werden zunächst Nachrichten von p_1 nach p_2 usw. bis p_j gesendet und anschließend in umgekehrter Reihenfolge zurückgesendet.

Hier können $j-1$ Nachrichten und somit ein nicht unerheblicher Anteil der Suchzeit eingespart werden, wenn mit der Suche gesendet wird, welcher Peer die Suche initiiert hat. Anstatt die Antwort dann vom Ziel-Peer über alle anderen $j-1$ Peers zurückzuleiten, kann die Antwort dann direkt an den Start-Peer gesendet werden. Als Nachteil dieser modifizierten Suche ist die eingeschränkte Fehlerkontrolle anzusehen. Geht eine der Nachrichten verloren, z.B. falls ein kontaktierter Peer das kurz zuvor Netzwerk verlassen hat, so erhält der suchende Peer keinerlei Rückmeldung an welcher Stelle und weshalb die Suche gescheitert ist. In diesem eher seltenen Fall kann jedoch jedoch auf den Such-Algorithmus aus Abbildung 5.4 zurückgegriffen werden, der eine solche Rückmeldung ermöglicht. Wir werden bei unseren weiteren Überlegungen von dem Suchalgorithmus mit reduzierter Nachrichtenanzahl ausgehen.

Eine grundsätzliche Technik für die Minimierung von Latenzzeiten beim Routing wird von Gummadi et al. in [40] vorgestellt: *Proximity Neighbor Selection (PNS)*. Bei PNS werden beim Aufbau der Routing-Tabellen latenznahe Peers bevorzugt. Wir widmen uns nun der Frage, wie sich dies in Chord realisieren lässt.

Im ursprünglichen Chord-Netzwerk zeigt der i-te Finger-Zeiger eines Peers p auf einen Peer im Bereich $(h(p)+2^i) \mod 2^m$ bis $(h(p)+2^{i+1}) \mod 2^m$ des Chord-Rings, wenn wir annehmen dass der i-te Finger-Zeiger ungleich dem $(i+1)$-ten und $(i-1)$-ten Finger-Zeiger ist. Der PNS-Algorithmus betrachtet anstelle des ersten Peers, der der Position $(h(p)+2^i) \mod 2^m$ auf dem Chord-Ring folgt, die ersten k Peers, die dieser Position folgen (möglicherweise auch weniger als k, falls sich nicht genügend Peers in diesem Intervall befinden). Für den Finger-Zeiger wird derjenige dieser k Peers gewählt, dessen Latenz zu Peer p am geringsten ist.

Das Routing in Chord bleibt bei Verwendung von Proximity-Neighbor-Selection unverändert: In jedem Schritt wird derjenige Peer ausgewählt, der dem Ziel nächsten liegt, ohne dieses zu überschreiten. Experimente in [39] zeigen, dass es in der Praxis genügt, jeweils den latenznahesten aus $k = 16$ Peers zu wählen, anstatt alle in Frage kommenden Peers zu betrachten.

Interessanterweise nimmt die Latenzzeit einer Suche mit der Netzwerkgröße kaum zu. Wie lässt sich dies erklären? Eine Intuition liefert die folgende Überlegung: Wir nehmen vereinfachend an, dass die Latenzzeiten gleichmäßig (uniform) verteilt sind. Sei des Weiteren δ die durchschnittliche Latenzzeit. Wir betrachten nun eine Suche im Netzwerk. Diese besteht aus $\mathcal{O}(\log n)$ Hops vom Start zum Ziel und einem letzten Hop vom Ziel zurück zum Startpunkt der Suche. Für den letzten Hop der Suche gibt es keine Wahlmöglichkeiten zwischen verschiedenen Peers, denn er muss vom vorletzten zum Ziel-Peer führen und benötigt somit Latenzzeit δ. Beim vorletzten Schritt kann aus zwei möglichen Peers der nähere gewählt werden, so dass sich hier eine erwartete Latenzzeit von $\delta/2$ ergibt. In jedem weiteren Schritt davor verdoppelt sich jeweils die Anzahl möglicher Peers. Summieren wir diese Latenzzeiten auf und fügen noch einmal Latenzzeit δ für den Hop vom Ziel zurück zum Start hinzu, ergibt sich eine gesamte Latenzzeit von

$$\delta + \left(\delta + \frac{\delta}{2} + \frac{\delta}{4} + \frac{\delta}{8} + \cdots\right) = \delta + 2\delta = 3\delta.$$

Dieses Verhalten wird durch Simulationen bestätigt. Die Latenzzeit beim Routing mit PNS kann also durch eine unendliche geometrische Reihe approximiert werden, die schnell konvergiert. Dies ist der Grund dafür, dass die Gesamtlatenz einer Suche, fast unabhängig von der Anzahl der Netzwerkteilnehmer n, immer nahe dem Wert 3δ liegt, auch wenn die Anzahl der Hops für die Suche mit $\mathcal{O}(\log n)$ deutlich schwächer wächst als n.

5.4 Zusammenfassung und Vergleich

Vergleicht man das im vorigen Kapitel vorgestellte CAN mit Chord, so bieten beide Netzwerke die gleiche Funktionalität: Sowohl CAN als auch Chord stellen Suchfunktionen bereit, die ein Datum garantiert finden, falls es sich im Netzwerk befindet. Unterschiede bestehen in der Effizienz. Während die grundlegende CAN-Struktur $\mathcal{O}(n^{1/2})$ Schritte für die Suche benötigt, ist ein Datum in Chord bereits nach $\mathcal{O}(\log n)$ Schritten lokalisiert. Diese nicht unwesentliche Verbesserung der Routing-Zeit wird erkauft durch die größeren Routing-Tabellen in Chord, wo jeder Peer mit $\mathcal{O}(\log n)$ anderen Peers verbunden ist. Zudem erlauben sowohl CAN als auch Chord, das Routing an das Underlay-Netzwerk (das Internet) anzupassen, um Latenzzeiten zu reduzieren.

6

Pastry und Tapestry

You like potato and I like potahto
You like tomato and I like tomahto
Potato, potahto, Tomato, tomahto.
Let's call the whole thing off.
Ira Gershwin

Pastry und *Tapestry* sind zwei unabhängig voneinander entstandene Peer-to-Peer-Netzwerke, die auf derselben Idee aufbauen, nämlich der Verwendung eines Routing-Verfahrens von Plaxton, Rajaraman und Richa [13]. Hierbei handelt es sich um ein effizientes verteiltes Verfahren zum Zugriff auf Datenkopien in einem verteilten statischen Netzwerk.

Auf dieser Basis wurden Pastry und Tapestry entwickelt, die das Verfahren unter anderem um Mechanismen zum Einfügen und Löschen von Peers und Daten erweitern. Wegen der gemeinsamen Vorarbeit unterscheiden sich Pastry und Tapestry daher nicht sehr und werden hier wegen ihrer Ähnlichkeit in einem Kapitel beschrieben. Wie wir sehen werden, existieren jedoch auch Unterschiede, sowohl in der Philosophie als auch in der Umsetzung der Netzwerke.

6.1 Verfahren von Plaxton, Rajamaran und Richa

Das Verfahren von Plaxton, Rajamaran und Richa [13] wurde schon 1997 konzipiert. Ziel war nicht ein Peer-to-Peer-Netzwerk oder ein geeignetes Routing dafür zu finden, schließlich waren Peer-to-Peer-Netzwerke 1997 noch unbekannt. Vielmehr ging es darum, in einem *verteilten Netzwerk* Kopien der bereitgestellten Daten (hier als Objekte bezeichnet) so zu platzieren, dass die Teilnehmer möglichst schnell darauf zugreifen können. Dieses klassische Problem [41] im Bereich der verteilten Netzwerke wurde zuvor bereits von anderen Autoren untersucht [42, 9, 43]. In der Arbeit von Plaxton, Rajamaran und Richa wurde jedoch erstmals der Aufwand für den Zugriff und Speicherbedarf unter einem allgemeineren Kostenmodell untersucht. Das

Verfahren selbst ist komplex und insbesondere komplexer als die darauf aufbauenden Peer-to-Peer-Netzwerke Pastry und Tapestry. Daher werden wir hier nur eine vereinfachte Beschreibung geben und uns auf die für uns wesentlichen Aspekte konzentrieren.

6.1.1 Das Modell

Das *verteilte Netzwerk* wird durch einen vollständigen Graph $G = (V, V^2)$ modelliert, d.h. ein Knoten $v \in V$ des Netzwerks hat die Möglichkeit, beliebige Knoten $u \in V$ in seine Routing-Tabelle aufzunehmen. Die Kommunikationskosten zwischen zwei Knoten des Netzwerks werden durch eine Funktion $c : V^2 \to \mathbb{R}$ beschrieben, die zu einer *Metrik* gehört. Also gelten für beliebige $u, v, w \in V$ die folgenden Eigenschaften einer Metrik:

- *Identität*: $c(v, w) = 0$ genau dann, wenn $v = w$,
- *Symmetrie*: $c(v, w) = c(w, v)$,
- *Dreiecksungleichung*: $c(u, w) \leq c(u, v) + c(v, w)$.

Die Kosten zur Übertragung von n Bits von u nach v werden durch den Term $c(u, v) \cdot f(n)$ modelliert, wobei $f : \mathbb{N} \to \mathbb{R}^+$ eine beliebige, monoton steigende Funktion mit $f(1) = 1$ darstellt.

Es wird angenommen, dass die Knoten des Netzwerks einigermaßen gleichmäßig in der Metrik c verteilt sind. Um dies zu formalisieren, definieren wir für einen Knoten $u \in V$ und ein $r \in \mathbb{R}$ die Menge

$$M(u, r) := \{v \in V : c(u, v) \leq r).$$

$M(u, r)$ enthält also alle Knoten im Umkreis r von u. Für die Verteilung der Knoten im Raum nehmen wir an, dass reelle Konstanten $\delta > 8$ und Δ existieren, so dass für alle $r \geq 1$ gilt:

$$\min\{\Delta|M(u, r)|, n\} \leq |M(u, 2r)| \leq \delta|M(u, r)|.$$

Nur unter dieser Annahme ist das Verfahren nachweisbar effizient. Hierbei ist anzumerken, dass nicht klar ist, ob das Internet als verteiltes Netzwerk dieser Annahme tatsächlich genügt.

Objekte als Kopien

Jedes der m Objekte hat eine eindeutige Identifikation aus $\log m$ Bits. Die Datenmenge in Bits eines Objekts A wird mit $\ell(A)$ beschrieben. Genauso sei für jeden der $|V| = n$ Knoten eine eindeutige Adresse als Bitstring der Länge $\log n$ vorhanden.

Ziel ist es, Anfragen nach Objekten so schnell wie möglich zu beantworten. Um die Zugriffszeit zu minimieren, werden Kopien der Objekte auf geeigneten Knoten gespeichert. Zugleich soll der Unterhalt der Datenstruktur einfach sein, d.h., es sollen nur wenige Knoten aktualisiert werden müssen, wenn sich die Netzwerkteilnehmer ändern. Für diese Eigenschaft wird in [13] der Begriff der Anpassungsfähigkeit (*Adaptability*) eingeführt.

Die Güte des Verfahrens wird anhand der folgenden Maße bewertet:

- *Zugriffszeit auf ein Objekt:* Diese ergibt sich aus der Summe der nachfolgenden Kopiervorgänge des Objekts A gemäß der Metrik c und der Objektgröße $\ell(A)$, d.h. der Summe von $c(u,v) \cdot f(\ell(A))$ für alle Kopiervorgänge von einem Knoten u zu einem Knoten v, die durch den Zugriff notwendig werden.
- *Lokaler Speicherbedarf*: Die Menge der lokal zu speichernden Kopien, wobei für alle Knoten die gleiche Speichermenge q angestrebt wird.
- *Anzahl der zu aktualisierenden Knoten*: Die Anzahl der Knoten, deren Informationen aktualisiert werden müssen, wenn ein Knoten dem System beitritt oder es verlässt.
- *Aktualisierungszeit für Datenänderung*: Die erwarteten (Zeit-)Kosten für das Aufnehmen oder Ändern eines Objekts.

6.1.2 Netzwerkstruktur

Knoten und Objekte erhalten numerische IDs zur Basis $B = 2^b$, $b \in \mathbb{N}$. Für die Zuweisung von Objekten zu Knoten gilt, dass ein Objekt mit ID A auf demjenigen Knoten gespeichert wird, dessen ID das längste gemeinsame Präfix hat. Gibt es mehrere solcher Knoten, so wird derjenige Knoten gewählt, dessen Suffix die größte bitweise Übereinstimmung mit A aufweist. Dieser eindeutige Knoten wird dann *Wurzelknoten* (*Root*) von A genannt. Ziel dieser Zuordnung ist eine gleichmäßige Verteilung der Daten auf die Knoten.

Für das Routing unterhält jeder Knoten $v \in V$ zwei Tabellen: Eine *Nachbarschaftsliste* und eine *Zeigerliste*. Die Einträge der Zeigerliste eines Knotens u sind Tripel (A, v, k). Ein solches Tripel gibt an, dass Objekt A sich auf Knoten $v \in V$ befindet und dass $c(u,v) \leq k$ gilt. Die Zeigerliste enthält eine Auswahl aller Objekte im Netzwerk und wird durch alle drei Operationen aktualisiert.

In der nun folgenden Beschreibung der Nachbarschaftsliste bezieht sich der Parameter i auf die Länge eines Präfixes einer Knoten- bzw. Objekt-ID, und der Parameter j bezeichnet ein Zeichen aus der Basis $\{0, 1, \ldots, B-1\}$. Die Nachbarschaftsliste eines Knotens v enthält Einträge der folgende Mengen von Einträgen:

Primäre (i,j)-Nachbarn. Hier wird der Knoten $u \in V$ gewählt, dessen ID mit v den Präfix der Länge $i-1$ teilt und dessen i-tes Zeichen der ID j entspricht. Falls nun mehrere Knoten existieren, die diesen Eigenschaften genügen, so wird derjenige gewählt, der $c(u,v)$ minimiert. Existiert hingegen kein Knoten mit diesen Eigenschaften, so wird der mit dem längsten gemeinsamen Präfix zu v gewählt. Trifft dies wiederum auf mehrere Knoten zu, so wird aus diesen derjenige gewählt, der das längste Suffix mit v teilt. Ist diese Wahl immer noch nicht eindeutig, wird aus dieser Menge der Knoten mit der größten ID gewählt.

Sekundäre (i,j)-Nachbarn. Sei u der primäre (i,j)-Nachbar von v und sei das i-te Zeichen der ID von u gerade j. Als sekundäre (i,j)-Nachbarn von v wird dann die konstante Anzahl d von Knoten v' gewählt, deren Präfix bis zur $(i-1)$-ten Stelle mit der ID von v übereinstimmt, als nächstes Zeichen den Buchstaben j enthalten und zugleich die Abstandsbedingung $c(v,v') \leq d \cdot c(v,u)$ erfüllen. Besteht diese Menge aus weniger als d Elementen, so wird die Menge mit Knoten v'' erweitert, für die der Wert $c(v,v'')$ möglichst gering ist.

Primäre (i, j)-Rückwärtsnachbarn. Ein Knoten u ist ein primärer (i, j)-Rückwärtsnachbar von v, wenn u ein primärer (i, j)-Nachbar von v ist.

Man beachte, dass die Anzahl der sekundären Nachbarn sehr groß werden kann, wenn nichts über die Struktur der Kostenmetrik c bekannt ist. Deshalb ist die oben formalisierte gleichmäßige Verteilung der Knoten wichtig. Die komplexe Struktur der primären Adresse dient hauptsächlich zur Vermeidung von hohen Eingraden, die bei der Definition der Rückwärtsnachbarn zu hohen Ausgraden führen würde.

6.1.3 Operationen

Das Verfahren stellt drei einfache Operationen zur Verfügung: *Read*, *Insert* und *Delete*. Diese werden wir im Folgenden kurz erläuten.

Read

Die Read-Operation eines Objektes besteht zum einen natürlich aus dem Routing vom anfragenden Knoten zu dem Knoten, der das Objekt bereitstellt, beschränkt sich jedoch nicht hierauf. Die Read-Operation beinhaltet auch die automatische Erstellung von Kopien des Objektes, um die Kommunikationskosten für zukünftige Anfragen zu senken.

Das Routing selbst geschieht durch das einfache Abfolgen der primären Nachbarschaftszeiger, indem der Präfix nach und nach ergänzt wird mit der Adressinformation des gesuchten Objektes A. Auf dem Weg dieser Suchanfrage muss also ein guter Weg gefunden werden, um das Objekt zum Anfrager zu schicken. Dabei ist nicht klar, dass ein direkter Kopiervorgang dies leistet. Es könnte sein, dass ein anderer Weg als der über die primären Nachbarn Ersparnisse bringt. Deshalb werden die sekundären Nachbarn beauftragt zu überprüfen, ob es nicht wesentlich bessere Wege gibt. Diese speichern die Zwischenergebnisse in ihrer Zeigerliste. Finden sich über diese keine kürzeren Wege, so wird der primäre Nachbarschaftszeiger verwendet und die Kopie direkt versendet. Ansonsten wird eine Kopie an dem sekundären Zeiger abgespeichert, der einen kürzeren Weg anbieten konnte.

Dieser Prozess erzeugt unter Umständen eine große Anzahl von Nachrichten im verteilten Netzwerk durch die Abfrage bei den sekundären Zeigern. Der Aufwand dieser Nachrichten wird bei der Analyse vernachlässigt. Die Autoren beschreiben nur den Aufwand für die eigentlichen Kopieraktionen des Objekts.

Insert

Beim Einfügen eines Objekts wird über die primären Rückwärtsnachbarn die Zeigerliste aller Knoten aktualisiert, die auf dem Weg vom bereitstellenden Knoten eines Objekts zum Adressknoten des Objekts liegen. Hierfür werden die aufsummierten Kosten gespeichert.

Delete

Das Löschen eines Objekts A macht es notwendig, dass in allen Zeigerlisten der zu A vorhandene Eintrag gelöscht werden muss. Wir haben gesehen, dass diese Einträge bei Insert- und Read-Operationen ergänzt werden. Um also einen Eintrag zu löschen, muss im ganzen Netzwerk nach solchen Einträgen gesucht werden. Diese Suche lässt sich dadurch beschleunigen, dass die Einträge durch die primären Rückwärtsnachbarn gefunden werden können.

6.1.4 Ergebnisse

Unter den getroffenen Annahmen erweist sich das Zugriffsverfahren als äußerst effizient. Es gelten folgende Aussagen, auf deren Beweis wir hier verzichten.

Theorem 6.1. *[13]*

1. *Die erwarteten Kosten für Read-Operationen sind asymptotisch optimal, d.h. beschränkt durch $\mathcal{O}(f(\ell(A))c(v,u))$, falls das Datum A von v nachgefragt wird und u der nächste Knoten ist, der das Objekt A speichert.*
2. *Die erwarteten Kosten für eine Insert-Operation sind $\mathcal{O}(C)$; für eine Delete-Operation $\mathcal{O}(C \log n)$, wobei $C = \max\{c(u,v) : u, v \in V\}$.*
3. *Falls q die gewünschte Speichergröße in jedem Knoten ist, so wird mit hoher Wahrscheinlichkeit in jedem Knoten höchstens $\mathcal{O}(q \log^2 n)$ Speicher verwendet.*
4. *Die Anzahl der Knoten, die beim Einfügen oder Löschen eines Objekts eine Aktualisierung vornehmen müssen, ist erwartungsgemäß $\mathcal{O}(\log n)$ und mit hoher Wahrscheinlichkeit $\mathcal{O}(\log^2 n)$.*

Für den sehr umfangreichen Beweis und die genaue Beschreibung der Operationen sei auf den Originalartikel [13] verwiesen.

Dieser Artikel war offensichtlich seiner Zeit voraus. Jedoch beruht er auf ein paar Annahmen, die für Peer-to-Peer-Netzwerke im Internet wohl nicht zutreffen. Zum einen ist die angenommene Metrik wahrscheinlich nicht angemessen für Peers im Internet. Ist diese Metrik jedoch nicht vorhanden, so nimmt die Anzahl der sekundären Nachbarn überhand. Außerdem können weder die Knoten noch die Daten per Bijektion auf einen kleinen Raum aus $\log n$ oder $\log m$ Bits abgebildet werden. Ferner ist man in Peer-to-Peer-Netzwerken oft nicht bereit, für andere Peers Kopien von Daten anzulegen, nur um deren Zugriffszeit zu verkürzen.

Diese Probleme lassen sich jedoch durch einfache Modifikationen beheben. So kann die Bijektion durch eine Hash-Funktion auf etwas längere Bitstrings ersetzt werden. Auf die Erstellung von Kopien (die ursprüngliche Motivation des Verfahrens) kann verzichtet werden, und das Problem der Metrik wird verdrängt. Daher verzichtet man auf die sekundären Nachbarn und setzt eine stärker eingeschränkte Metrik voraus.

Die folgenden Kapitel 6.2 und 6.3 beschreiben die ein paar Jahre später erschienenen und auf dem Plaxton-Routing basierenden Peer-to-Peer-Netzwerke *Pastry* und *Tapestry*.

6.2 Pastry

Peter Druschel und Antony Rowstron entwickelten in Cambridge bei Microsoft Research das Peer-to-Peer-Netzwerk *Pastry* [11]. Zielsetzung von Pastry ist ein vollständig dezentralisiertes, fehlertolerantes, skalierbares Peer-to-Peer-Netzwerk, das zur Optimierung der Latenzzeiten an das Underlay-Netzwerk angepasst ist.

Überblick

Genau wie Chord basiert Pastry auf verteilten Hash-Tabellen über einem eindimensionalen Raum der Größe 2^{128}. D.h. jedem Peer und jedem Datum im Pastry-Netzwerk wird eine eindeutige 128-Bit ID durch Hashing zugewiesen. Im Falle der Daten geschieht dies wiederum durch eine kryptographische Hash-Funktion, die Peer-IDs können wahlweise auch zufällig gewählt werden. Diese IDs werden in Pastry als Zahlen zur Basis 2^b, $b \in \mathbb{N}$ interpretiert. In der Praxis wird typischerweise der Parameter $b = 4$ und somit die Basis 16 verwendet. Bezeichne fortan $B = \{0, 1, \ldots, 2^b - 1\}$ die Menge der möglichen Ziffern für Peer- bzw. Daten-IDs.

In Pastry wird ein Datum jeweils von demjenigen Peer verwaltet, dessen ID der ID des Datums numerisch am nächsten ist. Beim näheren Hinsehen unterscheidet sich dieses Verfahren zur Zuweisung von Daten zu Peers nicht sehr von demjenigen im Chord-Netzwerk, schließlich wird in beiden Netzwerken ein eindimensionaler Raum für diese Zuweisung verwendet.

6.2.1 Netzwerkstruktur

Die Netzwerkstruktur von Pastry ist eng angelehnt an die primären Nachbarschaftslisten des Plaxton-Routings [13]. Die den primären Nachbarschaftslisten entsprechenden Verbindungen eines Peers werden in Pastry als *Routing-Tabelle* R bezeichnet. Zusätzlich zur Routing-Tabelle unterhält jeder Peer noch zwei weitere Mengen von Netzwerkverbindungen, welche als *Leaf-Set* L und *Nachbarschaftsmenge* M bezeichnet werden. Wir werden den Aufbau dieser drei Mengen von Netzwerkverbindungen im Folgenden näher betrachten.

Die Routing-Tabelle R

Wie bereits erwähnt, wird die ID eines Peers als Zahl zur Basis $|B| = 2^b$ interpretiert. Ein Peer p kennt für jedes Präfix z der eigenen ID und für jede Ziffer $j \in B$ einen anderen Peer, dessen ID mit $z \circ j$ beginnt (siehe Abbildung 6.1), es sei denn $z \circ j$ ist ein Präfix der ID von p. Hier stellt $\circ$ die Konkatenation zweier Ziffernfolgen dar. Ein Beispiel für eine Routing-Tabelle für den Fall $b = 2$ und 16-Bit IDs ist für einen Peer p mit ID 2310 in Abbildung 6.1 dargestellt. Die oberste Zeile der Routing-Tabelle enthält Peers, deren ID kein gemeinsames Präfix mit p teilen, die nächste Zeile enthält Peers, deren ID, ebenso wie diejenige von p, mit 2 beginnt, usw. Grau hinterlegt sind die Einträge der Routing-Tabelle, die einem Präfix der ID 2310

entsprechen und somit leer sind. Wir bezeichnen die Einträge der Routing-Tabelle im Folgenden auch als Level i-Nachbarn, wobei i die Länge des gemeinsamen Präfixes bezeichnet, also die Zeile in der Routing-Tabelle.

Peer-ID: **2310**

Routing-Tabelle R

0331	1300	2	3101
2012	2130	2203	3
2303	1	2323	2330
0	-	2312	-

Abb. 6.1. Routing-Tabelle R eines Peers in Pastry.

Existiert im Netzwerk für ein Präfix z und ein $j \in B$ kein Peer, dessen ID mit $z \circ j$ beginnt, so bleibt der entsprechende Eintrag der Routing-Tabelle leer. Existieren hingegen mehrere Peers, deren IDs dieser Bedingung genügen, wird aus diesen der nächste Peer bezüglich der Latenzzeit (RTT — *Round Trip Time*) im Underlay-Netzwerk gewählt.

Die Routing-Tabelle eines Peers ist also nicht notwendigerweise vollständig besetzt. Durch die zufällige Wahl der Peer-IDs lassen sich folgende Aussagen über die Anzahl der Einträge in der Routing-Tabelle treffen.

Lemma 6.2. *In einem Pastry Netzwerk mit n Peers hat ein Peer mit hoher Wahrscheinlichkeit nicht mehr als $\mathcal{O}\left(\frac{\log n}{b} 2^b\right)$ Einträge in seiner Routing-Tabelle.*

Beweis. Die Wahrscheinlichkeit, dass ein Peer ein bestimmtes m-stelliges Präfix erhält, ist 2^{-bm}. Damit ist die Wahrscheinlichkeit, dass ein m-stelliges Präfix im Netzwerk an einen Peer vergeben wird, höchstens

$$n2^{-bm} .$$

Setzen wir nun $m = (c+2)\frac{\log n}{b}$ für eine positive Konstante c in diese Abschätzung ein, erhalten wir

$$\begin{aligned} n2^{-bm} &= n2^{-(c+2)\log n} \\ &= n \cdot n^{-c-2} \\ &= n^{-c-1} . \end{aligned}$$

Es gilt also mit hoher Wahrscheinlichkeit, dass kein Peer mit dem gleichen Präfix der Länge $(c+2)\frac{\log n}{b}$ oder größer vorhanden sein wird. Damit besteht die Routing-Tabelle eines Peers mit hoher Wahrscheinlichkeit aus höchstens $(c+2)\frac{\log n}{b}$ Zeilen mit jeweils $2^b - 1$ Einträgen. Daraus folgt das Lemma. □

Das Leaf-Set L

Das Leaf-Set L, $|L| = \ell$, eines Peers p besteht aus den $\ell/2$ Peers mit der nächst höheren und $\ell/2$ Peers mit der nächst niedrigeren ID zu p. Als triviale Beobachtung halten wir fest, dass durch das Leaf-Set unter anderem eine Ringstruktur entsteht, durch die bereits ein grundlegendes Routing realisiert werden kann. Im Fall $\ell = 2$ entspricht diese gerade dem Chord-Ring (siehe auch Kapitel 5). In Abbildung 6.2 wird beispielhaft für den Fall $\ell = 4$ das Leaf-Set eines Peers mit ID 2310 dargestellt.

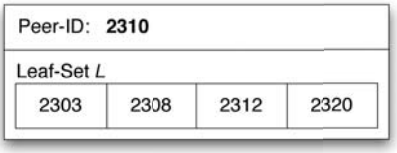


Abb. 6.2. Leaf-Set L eines Peers in Pastry.

Die Nachbarschaftsmenge M

Die Nachbarschaftsmenge M besteht typischerweise aus $|M| = 2^b$ Verbindungen zu den Peers des Netzwerks, die gemäß der Latenzzeit am nächsten sind. Die IDs der Peers in M ist hierbei beliebig. Hierbei ist anzumerken, dass die Nachbarschaftsmenge M normalerweise nicht für das Routing verwendet wird, sondern z.B. falls beim Routing Probleme durch ausgefallene Peers oder ähnlich auftreten.

Insgesamt verfügt ein Peer in Pastry also über die drei in Abbildung 6.3 dargestellten Klassen von Nachbarn. Abbildung 6.4 veranschaulicht die Nachbarschaft eines Peers am Netzwerkgraph. (Aus Gründen der Übersichtlichkeit wurde dabei auf die Darstellung der latenznahen Peers der Menge M verzichtet.)

6.2.2 Routing

Nachdem wir im letzten Abschnitt die Netzwerkstruktur von Pastry vorgestellt haben, werden wir nun untersuchen wie in dieser Struktur effizientes Routing von Nachrichten geschehen kann. Wie eingangs erwähnt wird in Pastry ein Datum jeweils von demjenigen Peer verwaltet, dessen ID der ID des Datums numerisch am nächsten ist. Abbildung 6.5 zeigt Pastrys Routing-Algorithmus als Pseudo-Code. Wird dieser Algorithmus beispielsweise für die Suche nach einem Datum im Netzwerk verwendet, wird zunächst die ID des gesuchten Datums (im Folgenden auch als Ziel-ID bezeichnet) mit Hilfe der Hash-Funktion berechnet und dann der Algorithmus mit eben dieser aufgerufen.

Wird der Algorithmus von einem Peer p aufgerufen, wird zunächst überprüft, ob die Ziel-ID innerhalb der Grenzen des eigenen Leaf-Sets liegt. Ist dies der Fall,

Peer-ID: **2310**

Leaf-Set *L*

2303	2308	2312	2320

Routing-Tabelle *R*

0331	1300	2	3101
2012	**21**30	**22**03	3
2303	1	**23**23	**23**30
0	-	**231**2	-

Nachbarschaftsmenge *M*

1300	1002	1133	1330

Abb. 6.3. Die drei Nachbarschaftsklassen eines Pastry Peers.

wird die Anfrage an denjenigen Peer des Leaf-Sets weitergeleitet, dessen ID numerisch am nächsten an der Ziel-ID liegt. Dieser Peer wird dann in der Lage sein, die Anfrage bzw. Nachricht zu beantworten, da er für die Ziel-ID verantwortlich ist. War die Suche im Leaf-Set hingegen erfolglos, wird in der Routing-Tabelle ein Peer p' gewählt, dessen gemeinsames Präfix mit der Ziel-ID um mindestens eine Ziffer länger ist als das gemeinsame Präfix von p und der Ziel-ID. Dieser Peer p' wird das Routing dann mit demselben Algorithmus fortsetzen.

Es kann, wie bereits erwähnt, vorkommen, dass die Routing-Tabelle fehlerhafte oder fehlende Einträge besitzt. In diesem Fall wird die Suche an einen Peer p' aus allen benachbarten Peers weitergeleitet, dessen gemeinsames Präfix mit der Ziel-ID mindestens so lang ist wie dasjenige des aktuellen Peers p und dessen ID numerisch näher an der Ziel-ID liegt als diejenige von p. Ein solcher Peer p' muss zumindest im Leaf-Set existieren, es sei denn die Nachricht ist bereits beim Empfänger, also dem für die Ziel-ID verantwortlichen Peer, angekommen. Ferner muss einer dieser Peers im Leaf-Set aktiv und erreichbar sein, wenn nicht gerade der unwahrscheinliche Fall des simultanen Ausfalls von $|L|/2$ auf der Ring-Struktur benachbarten Peers eingetreten ist.

Es ist offensichtlich, dass dieser einfache Routing-Algorithmus sein Ziel immer erreichen wird: In jedem Schritt wird die Nachricht entweder an einen Peer weitergeleitet, dessen gemeinsamer Präfix mit der Ziel-ID mindestens eine Ziffer länger ist, oder an einen Peer, dessen ID zumindest numerisch näher an der Ziel-ID liegt.

Betrachten wir nun die Anzahl der benötigten Nachrichten bzw. involvierten Peers unter der Annahme korrekt geführter Routing-Tabellen und Leaf-Sets.

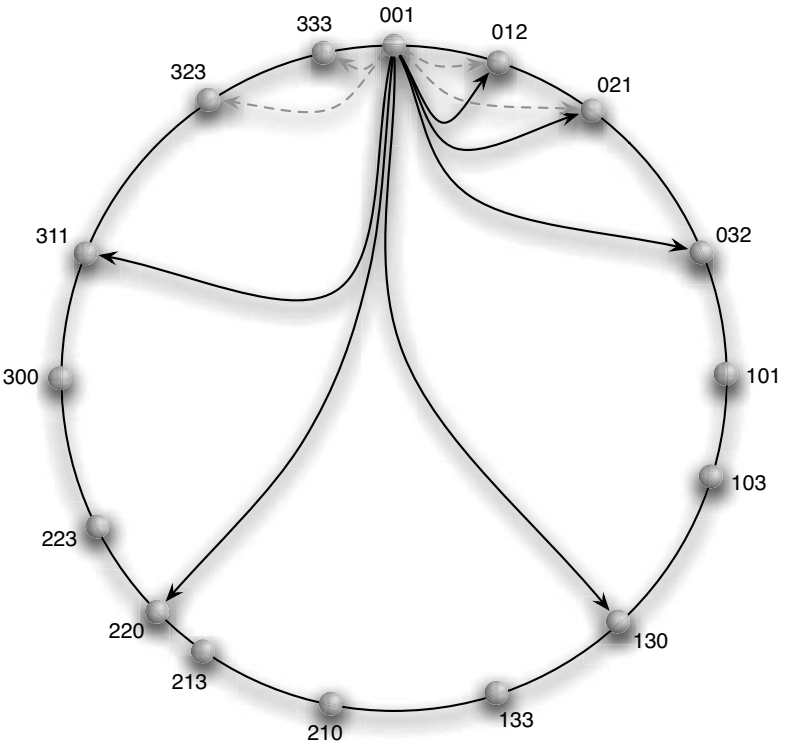


Abb. 6.4. Die Nachbarschaft eines Pastry Peers mit ID 001 als Netzwerkgraph. Schwarze Pfeile entsprechen den Einträgen der Routing-Tabelle R und grau gestrichelte Pfeile den Einträgen des Leaf-Sets L.

Theorem 6.3. *Unter der Voraussetzung korrekter Routing-Tabellen und Leaf-Sets benötigt Routing im Pastry-Netzwerk höchstens $\mathcal{O}(n/\ell)$ Schritte und im Erwartungswert $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ Nachrichten.*

Beweis. Für den Beweis der Sprunganzahl $\mathcal{O}(n/\ell)$ nehmen wir an, dass lediglich die Leaf-Sets für das Weiterreichen einer Nachricht verwendet werden. In diesem Fall würde sich die Distanz zur Ziel-ID mit jedem Schritt um $|L| = \ell$ Peers verringern, so dass das Ziel nach spätestens nach n/ℓ Schritten erreicht wird.

Auf einen formalen Beweis für die erwartete Anzahl von $\mathcal{O}(\log_{2^b} n) = \mathcal{O}(\frac{\log n}{b})$ Nachrichten verzichten wir an dieser Stelle und beschreiben lediglich die dahinterstehende Idee. Wird in jedem Schritt des Routings ein Peer aus der Routing-Tabelle ausgewählt, bedeutet das, dass die Länge des gemeinsamen Präfix zwischen aktuellem Peer und Ziel-ID mit jedem Schritt erhöht wird. In Lemma 6.2 haben wir gesehen, dass Peers mit hoher Wahrscheinlichkeit nur Einträge in den $\mathcal{O}(\frac{\log n}{b})$ ersten Zeilen ihrer jeweiligen Routing-Tabelle besitzen. Da mit jedem Schritt des Routings die verwendete Zeile der Routing-Tabelle erhöht wird, bedeutet das, dass nach $\mathcal{O}(\frac{\log n}{b})$ Schritten mit hoher Wahrscheinlichkeit kein Peer mit längerem gemeinsamen Präfix

Suche(id)
  if $(L_{-\ell/2} \leq id \leq L_{-\ell/2})$
    // Verwende das Leaf-Set
    Leite weiter an Peer $p' \in L$ für den $|id - p'|$ minimal ist
  else
    // Verwende die Routing-Tabelle
    $r \leftarrow$ *pfxl*(p, id)
    if $(R_r^{id_r} \neq null)$
      Leite weiter an Peer $R_r^{id_r}$
    else
      // seltener Fall
      Leite weiter an $p' \in L \cup R \cup M$ mit
      pfxl$(p', id) \geq r \wedge |p' - id| < |p - id|$

Abb. 6.5. Routing-Algorithmus für Pastry. R_r^i bezeichnet den Eintrag der Routing-Tabelle R in Spalte i und Zeile r, $L_{-\ell/2}$ bzw. $L_{-\ell/2}$ bezeichnen die IDs der beiden äußeren Peers des lokalen Leaf-Sets L, id_r bezeichnet die r-te Ziffer von id und die Funktion *pfxl*(p, id) liefert die Länge des gemeinsamen Präfix von p und id.

im Netzwerk existiert und das Ziel erreicht oder mit einem Schritt über das Leaf-Set erreicht werden kann.

Es kann jedoch vorkommen, dass kein passender Eintrag für den nächsten Schritt des Routings in einer Routing-Tabelle vorhanden ist und die Ziel-ID noch nicht innerhalb des Leaf-Sets liegt. Unter der Annahme korrekter Routing-Tabellen bedeutet dies, dass sich kein Peer mit passendem Präfix im Netzwerk befindet. Dieser Fall ist jedoch bei adäquater Wahl von $|L|$, z.B. 2^b, sehr unwahrscheinlich. Des Weiteren würde dadurch mit hoher Wahrscheinlichkeit lediglich ein zusätzlicher Schritt im Netzwerk benötigt [11]. □

Die in Theorem 6.3 angegebenen Schranken gelten lediglich unter der Voraussetzung korrekter Routing-Tabellen. Durch die in einem Peer-to-Peer-Netzwerk ständig wechselnden Teilnehmer wird es jedoch zumeist auch Peers mit fehlerhaften Routing-Tabellen Einträgen geben. So kann Routing aus theoretischer Sicht in extremen Situationen natürlich auch n Schritte benötigen oder gar scheitern, falls das Netzwerk den Zusammenhang verloren hat. Tatsächlich wird das Routing jedoch selbst bei durchgehend stark beschädigten Routing-Tabellen so gut wie niemals n/ℓ oder gar n Schritte benötigen, da bereits wenige korrekte Routing-Tabelleneinträge eine erhebliche Reduzierung der Hop-Anzahl ermöglichen. Wir werden später in Abschnitt 6.2.5 experimentelle Resultate betrachten, die zeigen wie sich ausfallende Peers auf den Zustand der Routing-Tabellen auswirken.

6.2.3 Einfügen von Peers

Das Einfügen eines neuen Peers in Pastry unterscheidet sich vom Einfügen neuer Peers in CAN und Chord, da bei Pastry unter anderem in der Routing-Tabelle jeweils latenznahe Peers aus vielen möglichen Kandidaten ausgesucht werden sollen.

Will ein neuer Peer p in das Pastry-Netzwerk aufgenommen werden, so muss er natürlich zunächst einen Peer p_a, der bereits Teil des Netzwerks ist, kontaktieren. Bei Pastry wird jedoch abweichend von den bisher vorgestellten Peer-to-Peer-Netzwerken angenommen, dass der Peer p_a im Underlay-Netzwerk einen geringen Abstand zu p hat, d.h. latenznah ist. Ausgehend von p_a wird dann mit einer speziellen *join*-Nachricht nach dem Peer p_z mit dem längsten gemeinsamen Präfix zu p gesucht, indem p_a im Netzwerk nach der ID von p sucht.

Die Peers p_a und p_z sowie alle weiteren Knoten auf dem Weg der join-Nachricht von p_a nach p_z senden als Antwort auf diese jeweils ihre drei Nachbarschaftsklassen an den Peer p. Peer p initialisiert seine Routing-Tabelle, das Leaf-Set und die Nachbarschaftsmenge M anhand dieser Nachrichten auf die im nächsten Absatz beschriebene Art und Weise. Zugleich informiert p diejenigen der ihm nun bekannten Peers, die p in ihre Routing-Tabelle, ihr Leaf-Set oder ihre Menge M latenznaher Peers aufnehmen müssen.

Wir widmen uns nun der Frage, wie Peer p aus den erhaltenen Nachbarschaftsinformationen seine eigene Nachbarschaft unter Berücksichtigung der Netzwerklokalität aufbauen kann. Da Peer p_a latenznah zu p ist, kann p die Menge M latenznaher Peers von p_a übernehmen. Des Weiteren hat Peer p_z das längste gemeinsame Präfix zu p, so dass p sein Leaf-Set leicht aus demjenigen von p_z aufbauen kann.

Es bleibt zu zeigen, wie Peer p seine Routing-Tabelle aufbauen kann. Wir gehen dabei vom allgemeinen Fall aus, dass die IDs von p und p_a keinen gemeinsamen Präfix besitzen. Die Level 0-Einträge der Routing-Tabelle sind jedoch unabhängig von der ID. Somit kann Peer p diese Einträge vom Peer p_a übernehmen. Die anderen Level der Routing-Tabelle von p_a sind für p uninteressant, da die IDs von p und p_a keinen gemeinsamen Präfix besitzen. Die Level 1-Einträge der Routing-Tabelle können jedoch von dem zweiten Peer auf dem Weg der join-Nachricht von p_a nach p_z übernommen werden. Dies ist möglich, da in jedem Schritt die Länge des gemeinsamen Präfix um mindestens eine Ziffer anwächst. Auf die gleiche Art und Weise wird für die weiteren Level der Routing-Tabelle bzw. Peers auf dem Weg von p_a nach p_z verfahren. Der beschriebene Vorgang wird in Abbildung 6.6) veranschaulicht. Dass hierbei auch tatsächlich latenznahe Nachbarn ausgewählt werden, werden wir später zeigen.

Der Aufwand dieser Einfügeoperation besteht aus ℓ Nachrichten an die Nachbarn der Nachbarschaftsmengen und erwartet $2^b/b \log n$ Nachrichten an Knoten mit gemeinsamem Präfix sowie natürlich $\mathcal{O}(\frac{\log n}{b})$ Nachrichten für die Suche nach dem Peer p_z mit längstem gemeinsamen Präfix.

Reparaturmechanismus

Die Einfügeoperation von Pastry ist sehr einfach, wenn man bedenkt, dass beim Aufbau der Nachbarschaft Netzwerklokalität berücksichtigt wird. Allerdings stellt sie

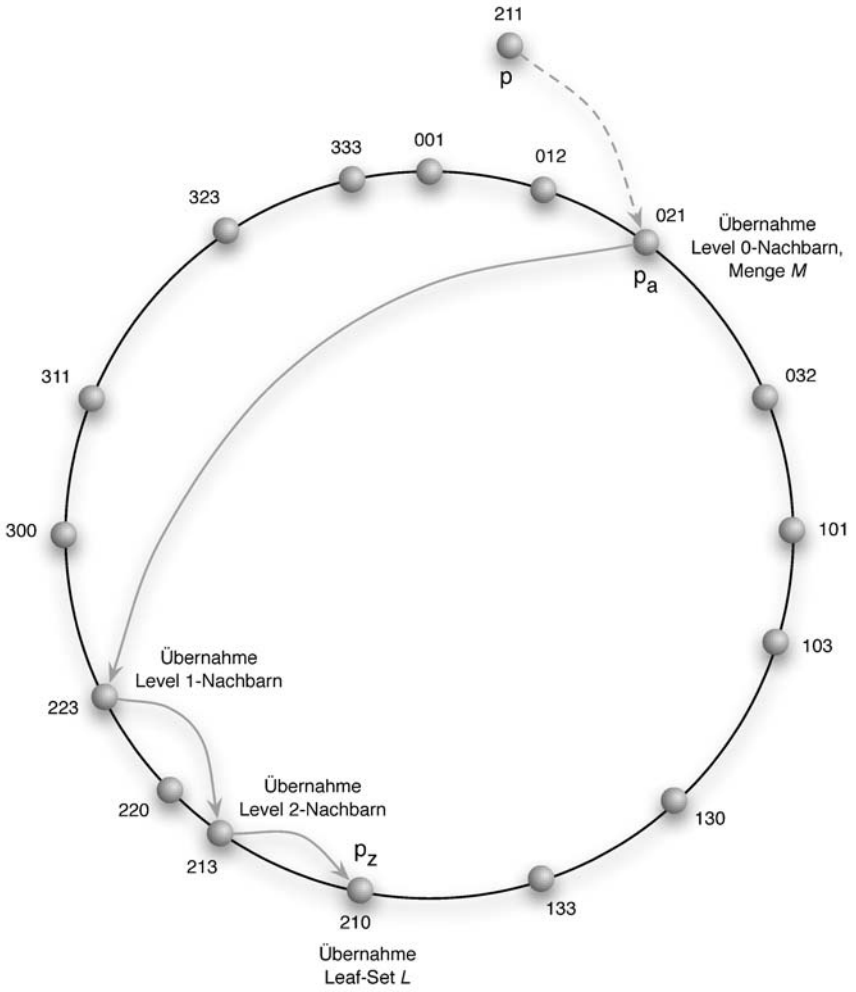


Abb. 6.6. Einfügen eines Peers in das Pastry Netzwerk.

nicht sicher, dass dabei stets korrekte Routing-Tabellen entstehen. Es kann durchaus vorkommen, dass notwendige Nachbarschaftseinträge nicht aktualisiert werden. Ein einfaches Beispiel hierfür ist der folgende Fall: Ist der neu eingefügte Peer p zum Beispiel der erste Peer im Netzwerk, dessen ID mit der Ziffer $1 \in B$ beginnt, so reicht es nicht aus, alle Peers aus den teilweise übernommenen Nachbarschaftslisten der Peers p_a bis p_z zu informieren. Tatsächlich müssten nämlich alle Peers des Netzwerks Peer p in ihre Routing-Tabelle aufnehmen.

Nun ist aber das korrekte Routing bzw. die Suche im Netzwerk selbst dann sichergestellt, wenn Einträge in der Routing-Tabelle fehlen, solange die Leaf-Sets der Peers korrekt sind. Der für ein Datum zuständige Peer kann also in jedem Fall gefun-

den werden. Jedoch werden dann unter Umständen erheblich mehr Schritte für die Suche im Netzwerk benötigt.

Um dem Problem der fehlenden Routing-Tabelleneinträge entgegenzuwirken, gibt es einen Reparaturmechanismus in Pastry. Angenommen, der Eintrag für Level j und Ziffer $i \in B$ fehlt einem Peer p in seiner Routing-Tabelle. Peer p bemerkt diesen fehlenden Eintrag zum Beispiel dadurch, dass eine Nachricht an oder von einem Peer mit diesem Präfix durch p befördert wird. Es kann aber auch vorkommen, dass ein Peer das Netzwerk verlässt und dadurch alte Einträge in der Routing-Tabelle falsch oder überflüssig werden.

Bemerkt ein Peer p, dass er einen fehlenden Eintrag in seiner Routing-Tabelle hat, wird folgender Reparaturmechanismus verwendet: Zuerst werden andere, p bekannte Nachbarn aus demselben Level j der Routing-Tabelle kontaktiert. Fehlt also beispielsweise dem Peer 021 ein Link zum Präfix $01*$, dann fragt er seine Nachbarn mit Präfix $00*$, $02*$ und $03*$ nach einem Peer für den fehlenden Eintrag. Falls einer dieser Peers einen passenden Eintrag für dieses Präfix hat, wird er von p übernommen. Scheitert dies, wird der gleiche Vorgang für die Level $j+1, j+2, \ldots$ der Routing-Tabelle wiederholt.

6.2.4 Lokalität

Eine wichtige Eigenschaft von Pastry ist, dass beim Aufbau der Nachbarschaft eines Peers und beim Routing versucht wird, Routingzeiten durch Nutzung latenznaher Verbindungen zu verringern. Wir bezeichnen diese Eigenschaft von Pastry auch als *Netzwerklokalität*.

Bevor wir auf die Details bezüglich Pastry und Lokalität näher eingehen, halten wir ein paar grundlegende Dinge diesbezüglich fest. Eine wichtige Beobachtung zur Latenzzeit ist, dass die Nähe eines Peers bezüglich der Ringmetrik (Raum über den durch Hashing zugewiesenen IDs $\in \{0,1\}^{128}$)[1] und der Latenzmetrik (gemessen in der Zeit, die eine Nachricht durchschnittlich zwischen zwei Peers benötigt) völlig unabhängig voneinander sind. Das heißt insbesondere, dass direkte Nachbarn im Peer-to-Peer-Netzwerk typischerweise nicht nah bezüglich der Latenzmetrik sind.

Wir haben in Abschnitt 2.3.2 (Seite 35) gesehen, dass das TCP-Protokoll des Internets sowieso Latenzzeiten durch die Round Trip Time RTT misst. In Pastry wird davon ausgegangen, dass die so gewonnenen Latenzzeiten eine euklidische Metrik definieren — die *Latenzmetrik*. Diese Latenzmetrik ist die Grundlage für die Suche eines nahen Peers für die Einträge in der Routing-Tabelle.

Die Verfahren in Pastry beruhen auf heuristischen Überlegungen, und ein mathematischer Nachweis der Güte der Verfahren ist bisher nicht erbracht worden. Insbesondere gilt dies auch für die Annahme, dass die Latenzmetrik euklidisch ist (d.h., es existiert eine Einbettung der Punkte in einem Raum, so dass die Latenzmetrik durch den euklidischen Abstand dieser Punkte entsteht).

[1] Wir erinnern uns, dass die IDs $\in \{0,1\}^{128}$ in Pastry als Zahlen zur Basis 2^b interpretiert werden, also eine 0-1-Folge der Länge b jeweils eine Ziffer beschreibt.

Lokalität in der Routing-Tabelle

Wir werden nun untersuchen, ob ein dem Netzwerk beitretender Peer p durch die Einfüge Operation tatsächlich eine an Latenzmetrik angepasste Routing-Tabelle erhält. Hierfür halten wir uns zunächst noch einmal das Verfahren zum Aufbau der Routing-Tabelle vor Augen: Der neue Peer p kontaktiert einen latenznahen Peer p_a, von dem aus zum Peer p_z mit längstem gemeinsamen Präfix zu p geroutet wird. Peer p übernimmt von p_a die Level 0-Einträge der Routing-Tabelle, vom zweiten Peer auf dem Weg zu p_z die Level 1-Einträge usw.

Wir nehmen im Folgenden an, dass alle anderen Peers im Netzwerk ihre Routing-Tabelle bereits bezüglich der Latenzmetrik optimiert haben. Da die Level 0-Einträge der Routing-Tabelle von p_a nur Peers enthalten, die nah zu p_a sind, p_a nah zu p ist und weil die Dreiecksungleichung gilt, folgt, dass die Einträge ebenfalls relativ nah zu p sind.

Betrachten wir nun die Level 1-Einträge der Routing-Tabelle des Peers p. Diese werden vom ersten Peer p_b auf dem Weg von p_a nach p_z übernommen. Hier befinden sich die eingetragenen Peers wiederum in der Nähe von p_b, jedoch nicht mehr notwendigerweise in der Nähe von p. Tatsächlich sind diese Einträge jedoch noch in einer so geringen Entfernung zu p, dass die Übernahme der Einträge in die Routing-Tabelle sinnvoll ist. Um dies zu verdeutlichen, erinnern wir uns, dass die Einträge der Routing-Tabellen mit jedem weiteren Level aus einer exponentiell schrumpfenden Menge gewählt werden. Dies gilt, da mit jedem weiterem Level eine weitere Ziffer des Präfixes festgelegt wird. Daher ist der erwartete Abstand zwischen p_b und den Peers in den Level 1-Einträgen von p_b's Routing-Tabelle bereits weitaus größer als der erwartete Abstand zwischen p_a und p_b. So macht es durchaus noch Sinn, die Level 1-Einträge von p_b's Routing-Tabelle für p zu übernehmen. Die gleiche Argumentation gilt für die weiteren Zeilen der Routing-Tabelle.

Nachdem p seine Routing-Tabelle auf diese Weise initialisiert hat, approximiert diese also bereits die gewünschte Lokalitätseigenschaft. Jedoch werden die Einträge weiter optimiert, indem p jeden Peer aus seiner Routing-Tabelle und der Menge M latenznaher Peers kontaktiert und deren Routing-Tabellen anfordert. Peer p vergleicht dann seine eigenen Einträge mit denen aus den erhaltenen Routing-Tabellen und ersetzt ggf. Einträge in der eigenen Routing-Tabelle.

Abbildung 6.7 veranschaulicht, warum die Level i-Einträge des i-ten Peers auf dem Weg von p_a nach p_z auch für den neuen Peer p gute Lokalitätseigenschaften aufweisen. Die Kreise zeigen die durchschnittliche Distanz in der Latenzmetrik der Level 0- bis Level 2-Einträge der jeweiligen Peers. Man beachte, dass Peer p zwar innerhalb jedes der Kreise, jedoch jeweils außerhalb des jeweiligen Zentrums liegt. So wird p durch Übernahme der Level 1-Einträge von p_b möglicherweise noch bessere Level 0-Einträge als die von p_a finden.

Lokalität im Routing

Wir betrachten nun, wie sich die Lokalitätseigenschaft der Routing-Tabellen auf das Routing auswirkt. Bei jedem Schritt des Routings wird jeweils ein zum aktuellen

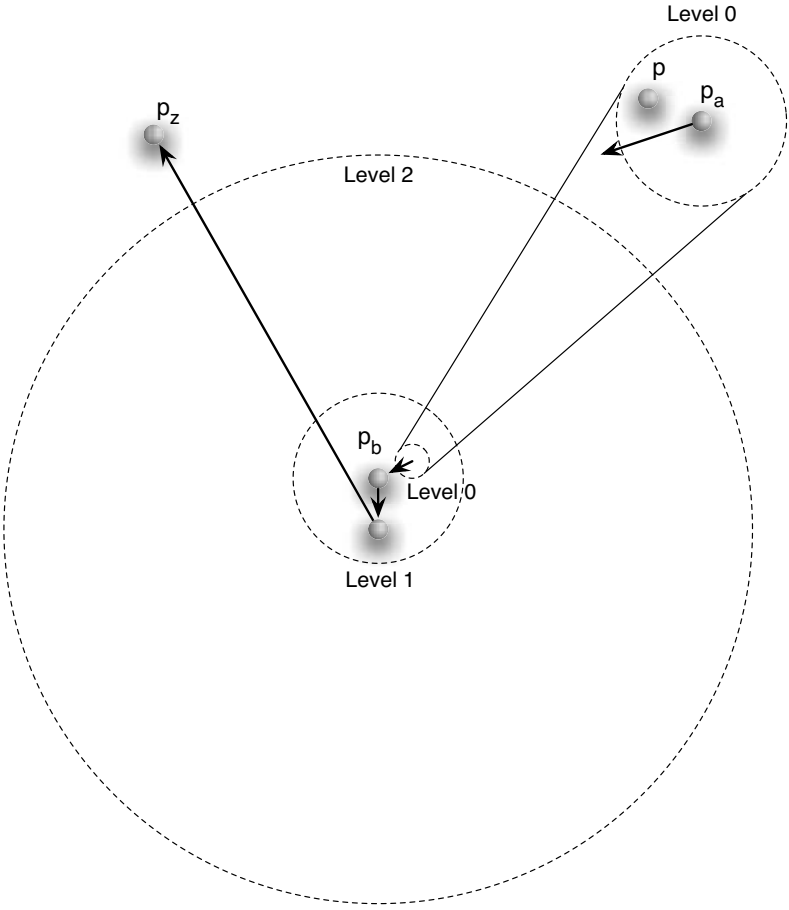


Abb. 6.7. Beim Routing zurückgelegte Distanzen in der Latenzmetrik.

Peer latenznaher Peer ausgewählt, dessen ID dem gewünschten Präfix entspricht. Da kein Peer eine globale Sicht des Netzwerks hat, muss der Routing-Algorithmus den nächsten Schritt jeweils anhand lokaler Informationen treffen. Das bedeutet, dass beim Routing zwar die Distanz der einzelnen Schritte minimiert wird, allerdings ohne Berücksichtigung der (globalen) Richtung des endgültigen Ziels. Dies zeigt, dass beim Routing nicht garantiert der bezüglich der Latenzmetrik kürzeste Weg zwischen Start und Ziel gefunden wird. Jedoch wird das Routing relativ gute Wege wählen. Dies machen wir uns anhand der beiden folgenden Beobachtungen klar:

1. Macht eine Nachricht innerhalb des Routings einen Schritt von einem Peer p_a zu einem Peer p_b mit Distanz d bezüglich der Latenzmetrik, so wird der nächste vom Routing gewählte Peer p_c eine Distanz größer als d zu p_a haben. Dies folgt direkt aus dem Routing-Algorithmus und der Annahme korrekter Routing-

Tabellen Einträgen. Wäre die Distanz zwischen p_a und p_c nämlich geringer als d, hätte p_a die Nachricht nicht an p_b sondern stattdessen an p_c gesendet.

2. Mit jedem Schritt, also mit jedem Level der Routing-Tabelle, steigt die Länge der Schritte bezüglich der Latenzmetrik exponentiell an. Der Grund dafür ist, dass ein Eintrag für das Level i der Routing-Tabelle aus einer Menge von $n/2^{bi}$ Peers gewählt werden kann. Durch die zufällige und uniforme Verteilung der Peer-IDs und unter der Annahme, dass die Peers im euklidischen Raum der Latenzmetrik ebenfalls gleichverteilt sind, bedeutet das, dass die erwartete Distanz des nächsten Peers für die Einträge der Routing-Tabellen ebenfalls exponentiell ansteigt.

Kombiniert implizieren diese beiden Beobachtungen, dass die Distanz einer Nachricht zu ihrem Ursprung mit jedem Schritt monoton wächst und die Länge der Schritte jeweils exponentiell zunimmt. Teuer bezüglich der Latenzzeiten sind also insbesondere die letzten Schritte des Routings. Ist eine Nachricht bereits nah an der Ziel-ID, kann jedoch oftmals ein direkter Sprung mit Hilfe der Leaf-Set Einträge gemacht werden und somit weitere, sehr teure Schritte eingespart werden. Im folgenden Abschnitt werden wir unter anderem experimentelle Resultate über das Lokalitätsverhalten des Routings betrachten.

6.2.5 Experimentelle Resultate

Die Autoren des Pastry-Netzwerks haben zahlreiche experimentelle Untersuchungen durchgeführt um das Netzwerk zu evaluieren [11]. Einige der Resultate betrachten wir in diesem Abschnitt.

Skalierbarkeit

Für den Nachweis der Skalierbarkeit des Netzwerks wurde für Parameter $b = 4$, $\ell = 16$ und $|M| = 32$ die durchschnittlich beim Routing auftretende Hop-Distanz gemessen. Abbildung 6.8 zeigt die gemessenen Werte. Es ist zu erkennen, dass die durchschnittliche Hop-Distanz logarithmisch mit der Knotenanzahl steigt. Die erwartete Anzahl an Hops ist nach Theorem 6.3 $\mathcal{O}(\frac{\log n}{b})$, was durch die experimentelle Analyse bestätigt wird.

Verteilung der Hop-Distanz beim Routing

In Abbildung 6.9 ist die Auswertung der Experimente hinsichtlich der Verteilung der Hop-Distanz dargestellt. Gemessen wurde die Verteilung der Hop-Distanz in einem Netzwerk mit $n = 10^6$ Peers. Die Abweichung von der erwarteten Hop-Distanz $(\log_{2^b} 10^6)$ ist extrem gering. Tatsächlich sagt auch die Analyse eine Abweichung mit polynomiell kleiner Wahrscheinlichkeit voraus, so dass auch hier eine gute Übereinstimmung von formaler und experimenteller Analyse gegeben ist.

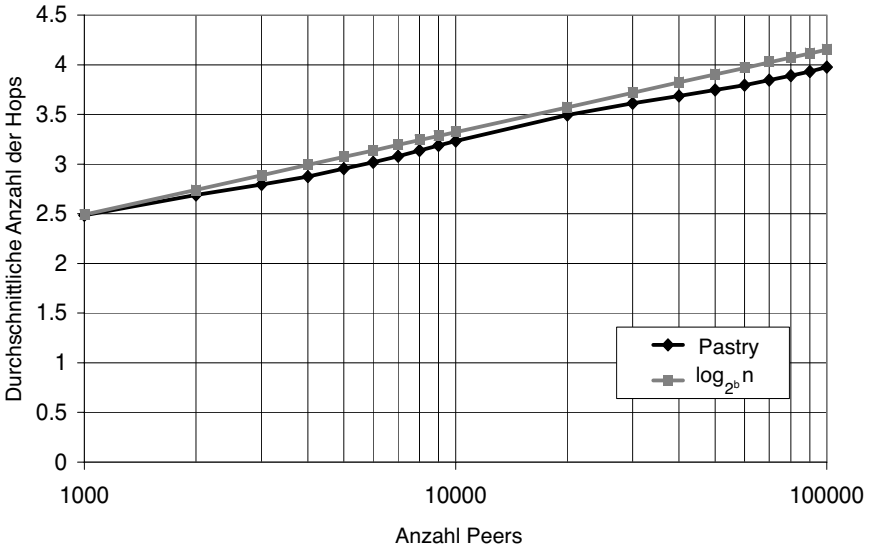


Abb. 6.8. Durchschnittliche Anzahl von Hops beim Routing bei steigender Anzahl der Peers ($b = 4$, $\ell = 16$ und $|M| = 32$) [11].

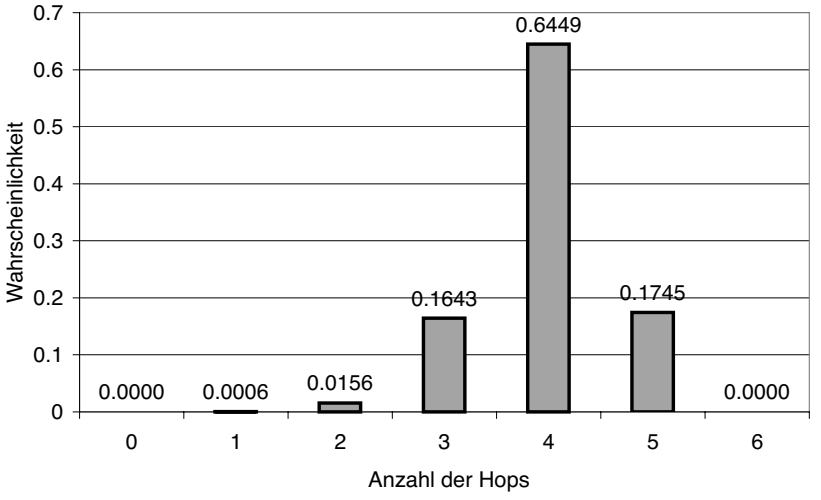


Abb. 6.9. Verteilung der Hop-Distanzen beim Routing in einem Pastry-Netzwerk ($b = 4$, $\ell = 16$, $|M| = 32$ und $n = 10^6$) [11].

Latenzzeit

Weiterhin wurden die Latenzzeiten beim Routing in Pastry mit den Latenzzeiten beim Routing in einem fiktiven Netzwerk mit vollständigen Routing-Tabellen verglichen — also direkten Verbindungen zwischen allen Peers (siehe Abbildung 6.10). Die Ergebnisse wurden anhand der Routing-Zeiten im fiktiven Netzwerk normiert.

Gemessen wurde in Netzwerken mit $n = 1000$ bis $n = 100.000$ Peers. Der Unterschied zwischen den Routing-Zeiten in Pastry und dem vollständigen Netzwerk ist erstaunlich gering: Das Routing in Pastry benötigt durchweg höchstens 30%-40% mehr Zeit als das Routing im vollständigen Netzwerk, also als die bestmögliche Verbindung. Des Weiteren scheint die Latenzzeit beim Routing mit steigender Knotenzahl nicht zuzunehmen.

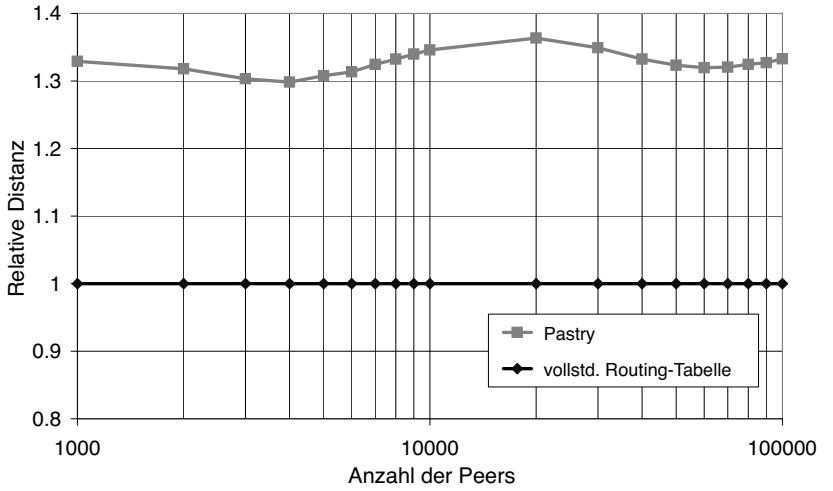


Abb. 6.10. Latenzzeiten beim Routing in Pastry verglichen mit den Latenzzeiten beim Routing in einem vollständigem Netzwerk ($b = 4$, $\ell = 16$, $|M| = 32$) [11].

Mit der Parameterwahl $b = 4$, $\ell = 16$ und $|M| = 32$ wird die Anzahl der Verbindungen je Peer relativ groß. Damit kann der Faktor $2^b/b = 4$ das Ergebnis erheblich beeinflussen. Wenn sich zum Beispiel $n = 10^6$ Peers im Netzwerk befinden, dann unterhält jeder Peer bereits $(2^b - 1)\log_{2^b} n > 60$ Verbindungen in seiner Routing-Tabelle. Hinzu kommen noch 16 Verbindungen im Leaf-Set L und 32 in der Menge M latenznaher Peers. Dadurch ist der Grad des Netzwerks verglichen mit dem vollständigen Netzwerk zwar noch gering, im Verhältnis zu anderen Peer-to-Peer-Netzwerken wie Chord allerdings sehr groß.

Knotenausfälle

Zuletzt seien hier noch die experimentellen Ergebnisse für die Zuverlässigkeit des Reparaturmechanismus aufgeführt (Abbildung 6.11). Ausgehend von einem Netzwerk mit $n = 5000$ Peers, $b = 4$, $\ell = 16$ und $|M| = 32$ werden 10% der Peers aus dem Netzwerk entfernt. Nach diesen simulierten Peer-Ausfällen werden eine ID und zwei Peers zufällig gewählt. Dann wird von diesen beiden Peers eine Suche nach der zufälligen ID durchgeführt. Dieser Vorgang wurde 100.000 mal wiederholt. Abbildung 6.11 zeigt den Zustand der Routing-Tabellen vor und nach diesem Experiment, sowie mit und ohne Reparaturmechanismus.

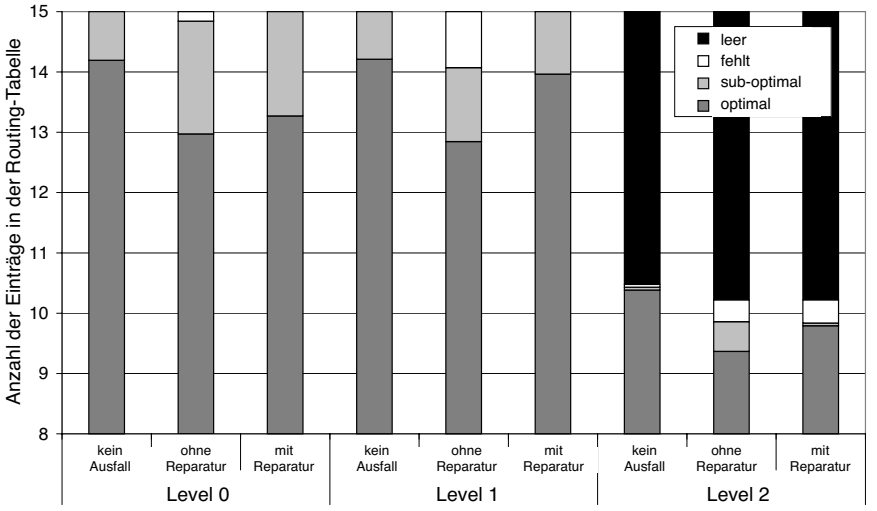


Abb. 6.11. Güte der Routing-Tabelle von Pastry vor und nach dem Ausfall von 500 von 5.000 Peers und nach Abschluss der Reparatur-Routine [11].

6.3 Tapestry

Das Peer-to-Peer-Netzwerk *Tapestry* [12] wurde in seiner ursprünglichen Version von Ben Y. Zhao, John D. Kubiatowicz und Anthony Joseph an der University of California, Berkeley, entwickelt. Wie auch bei Pastry, ist die Netzwerkstruktur an diejenige des Plaxton-Routings angelehnt. Als Besonderheit ist im Vergleich zu einigen anderen Peer-to-Peer-Netzwerken hervorzuheben, dass Tapestry wie auch Pastry beim Aufbau der Netzwerkstruktur die Netzwerklokalität berücksichtigt, um die beim Routing entstehenden Latenzzeiten gering zu halten. Wir werden zunächst einige grundlegende Begriffe kennenlernen und uns dann mit der Netzwerkstruktur, dem Routing, der Verwaltung von Daten und dem Einfügen neuer Peers beschäftigen.

Grundlegende Terminologie

Wie auch in den Netzwerken Chord und Pastry beruht die Zuordnung von Daten zu Peers in Tapestry auf verteilten Hash-Tabellen über einem eindimensionalen Raum. Das bedeutet, dass Daten bzw. Peers im Tapestry-Netzwerk durch eindeutige *Daten-IDs* bzw. *Peer-IDs* identifiziert werden. Peer-IDs können wie zuvor zufällig vergeben oder durch Hash-Funktionen bestimmt werden. Daten-IDs werden, wie schon bei CAN, Chord und Pastry, durch kryptographische Hash-Funktionen berechnet.

Wie beim Verfahren von Plaxton et al. und Pastry werden die IDs als Zahlen zu einer Basis $2^b, b \in \mathbb{N}$ interpretiert. Sei von nun an $B = \{0, \ldots, 2^b - 1\}$ die Menge der möglichen Ziffern für Peer- sowie Daten-IDs und bezeichne $|\alpha|$ für eine Zahl α zur Basis 2^b die Länge dieser Zahl in Ziffern aus B. Des Weiteren stellt die Schreibweise $\alpha \circ \beta$ für zwei Zahlen α und β fortan die Konkatenation eben dieser dar.

6.3.1 Netzwerkstruktur

Die Nachbarschaft eines Peers in Tapestry ist eng angelehnt an die Verbindungsstruktur beim Verfahren von Plaxton et al. und ähnelt somit auch der Routing-Tabelle eines Pastry-Peers (siehe Seite 100). Im Gegensatz zu Pastry beschränken sich die Nachbarschaftsinformationen eines Peers in Tapestry jedoch auf eine einzige Menge, die wir fortan als Routing-Tabelle bezeichnen. Wir werden im Folgenden den Aufbau der Routing-Tabelle beschreiben. Die Beschreibung ist dabei etwas formaler als zuvor im Fall von Pastry, da wir die formale Notation für die späteren Analysen benötigen.

In seiner Routing-Tabelle unterhält ein Peer p Verbindungen zu ausgewählten anderen Peers, die ein Präfix z seiner ID mit ihm teilen. Wieder ist die Routing-Tabelle in so genannte *Level* unterteilt, wobei der Level die Länge des gemeinsamen Präfix zwischen Peer-ID und den Einträgen der Routing-Tabelle angibt.

Grundsätzlich kann ein Peer p mit einem Peer p' im Level ℓ benachbart sein, falls es eine Zahl z der Länge $|z| = \ell$ gibt, so dass $p = z \circ \delta$ und $p' = z \circ \delta'$ für zwei geeignete Zahlen δ, δ', die sich in ihrer ersten Ziffer unterscheiden. Also bezeichnet $\ell = |z|$ den Level der Nachbarschaft. Abbildung 6.12 zeigt beispielhaft

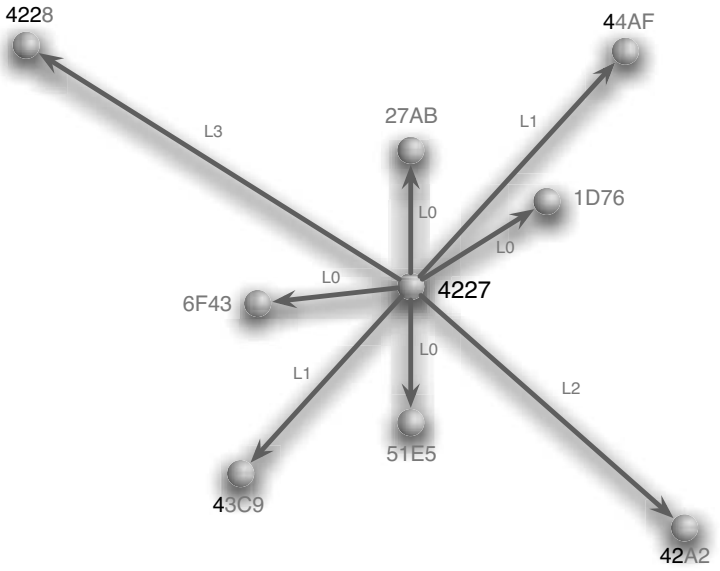


Abb. 6.12. Ausschnitt der Nachbarschaft eines Peers mit ID 4227.

einen Ausschnitt der Nachbarschaft eines Peers mit ID 4227 in einem Netzwerk mit 32-Bit IDs und $2^b = |B| = 16$.

Die Nachbarn eines Peers p werden in die Nachbarschaftsmengen $N^p_{z,j}$ eingeteilt. Für jeden Präfix z und jede Ziffer $j \in B$ enthält die Nachbarschaftsmenge $N^p_{z,j}$ Peers, deren IDs das Präfix $z \circ j$ mit p teilen. Wir werden diese im Folgenden auch als (z, j)-*Nachbarn* bezeichnen. Für jedes Präfix z und jedes $j \in B$ wird der gemäß einer festgelegten Metrik nächste Peer zu p in $N^p_{z,j}$ als *primärer* (z, j)-*Nachbar* bezeichnet, alle weiteren als *sekundäre* (z, j)-*Nachbarn*. Die Kardinalität der Nachbarschaftsmengen ist durch eine Konstante begrenzt, denn sonst würden die Level 0-Einträge der Routing-Tabelle eines Peers alle anderen Peers des Netzwerks enthalten.

Diese Darstellung entspricht im Wesentlichen den primären und sekundären Nachbarn im Plaxton-Routing. Die Vereinigung $\bigcup_{j \in B} N^p_{z,j}$ der $|B| = 2^b$ Nachbarschaftsmengen eines Peers p für einen festen Präfix z mit Länge $|z| = \ell$ entspricht gerade den Level ℓ-Einträgen der Routing-Tabelle des Peers p.

Wir werden im Folgenden einige wichtige Eigenschaften der Nachbarschaftsmengen in Tapestry erläutern. Eine wesentliche Eigenschaft ist die folgende Konsistenzeigenschaft.

Eigenschaft 1: Konsistenz. *Falls $N^p_{z,j} = \emptyset$ für ein Peer p gilt, dann existieren keine (z, j)-Nachbarn im gesamten Netzwerk. Wir bezeichnen dies als ein* Loch *in der Routing-Tabelle von p im Level $|z|$ mit Buchstaben j.*

Aus der Konsistenzeigenschaft lässt sich direkt folgern, dass das Netzwerk zusammenhängend ist. Somit kann man einen einfachen Routing-Algorithmus für das

Netzwerk angeben. Angenommen, wir möchten von einem Peer p' mit ID A' zu einem Peer p mit ID $A = a_1 \circ a_2 \circ \cdots \circ a_n$ mit $a_1, \ldots, a_n \in B$ routen. Dies gelingt, indem wir im ersten Schritt einen Peer $p_1 \in N^p_{\emptyset,a_1}$ wählen. Von Peer p_1 aus wählen wir schrittweise Peers $p_2 \in N^{p_1}_{a_1,a_2}$, $p_3 \in N^{p_2}_{a_1 \circ a_2,a_3}$ usw. Auf diese Weise wird in jedem Routing-Schritt eine weitere Ziffer der ID A' von p' an die ID A von p angepasst und nach $\log_{2^b} |A'|$ Schritten sind wir am gewünschten Ziel angelangt. Wir vernachlässigen zunächst, dass wir beim Routing auf Löcher in den Routing-Tabellen stoßen können. Eine Lösung für diesen Fall werden wir in Kürze kennenlernen.

Tapestry übernimmt vom Plaxton-Routing den Gedanken zur Verringerung der Latenzzeiten beim Routing durch das Bevorzugen bezüglich der Latenzmetrik naher Peers. So ergibt sich die folgende Lokalitätseigenschaft.

Eigenschaft 2: Lokalität. *Jede Nachbarschaftsmenge $N^p_{z,j}$ enthält die bezüglich der Latenzmetrik nächsten (z, j)-Nachbarn von Peer A. Der nächste Peer mit Präfix $z \circ j$ ist der primäre Nachbar, alle anderen sind sekundäre Nachbarn.*

Diese Eigenschaft gewährleistet, dass die beim Routing entstehenden Latenzzeiten möglichst gering gehalten werden. Des Weiteren kann durch diese Eigenschaft jeder Peer p seinen (gemäß der Latenzmetrik) nächsten Nachbarn in der Menge $\bigcup_{j \in B} N^p_{\emptyset,j}$ finden.

6.3.2 Daten und Routing

Peers, die Daten bereitstellen, werden in Tapestry *Storage-Server* genannt. Da ein Datum von mehreren Storage-Servern bereitgestellt werden kann, gibt es zusätzlich zu den Storage-Servern so genannte *Root-Peers*, die für bestimmte Objekte verantwortlich sind.

Für die Verwaltung eines Datums mit ID ψ wird eine Menge R_ψ von Root-Peers eingesetzt, die durch eine Funktion MapRoots(ψ) (dazu gleich mehr) bestimmt werden kann. Die Menge R_ψ bezeichnen wir im Weiteren als *Root-Set*. Aufgabe der Root-Peers ist es, Links auf diejenigen Peers zu verwalten, die das entsprechende Datum bereitstellen. Es ist zu beachten, dass jeder Peer sowohl Storage-Server als auch Root-Peer sein kann. Um ein Datum jederzeit lokalisieren zu können, muss Folgendes gewährleistet sein:

Eigenschaft 3: Eindeutiges Root-Set. *Das Root-Set R_ψ für jedes Objekt ψ muss eindeutig sein, und insbesondere muss die Funktion MapRoots(ψ) immer das gleiche Root-Set R_ψ generieren, egal wo im Netzwerk die Funktion aufgerufen wird.*

Eine einfache Lösung, die garantiert, dass MapRoots(ψ) immer die gleichen Server zurückliefert, egal wo im Netzwerk die Funktion aufgerufen wird, ist die Verwendung einer Hash-Funktion. Das bedeutet, MapRoots(ψ) würden z.B. einfach die ID des Datums ψ zurückliefern (da diese ja auch durch Hashing entsteht). Nun sollte die Ausgabe von MapRoots(ψ) jedoch die ID eines oder mehrerer Root-Peers sein. Mit hoher Wahrscheinlichkeit wird jedoch kein Peer im Netzwerk existieren, der zufällig

genau die gleiche ID wie das Datum besitzt. Dieses Problem könnte gelöst werden, indem der dem berechneten Hash-Wert numerisch am nächsten liegende Peer verwendet wird. Hierfür ist jedoch eine globale Ordnung aller Peer-IDs notwendig, die in einem Peer-to-Peer-Netzwerk schwer aufrecht zu erhalten ist. Wie sich dieses Problem ohne globale Ordnung lösen lässt, werden wir in Abschnitt 6.3.3 sehen.

Storage-Server müssen den zuständigen Root-Servern mitteilen, dass sie das entsprechende Datum ψ bereitstellen, um es im Netzwerk verfügbar zu machen. Dies geschieht, indem sie eine entsprechende Nachricht entlang primärer Nachbarn zu jedem Peer $p \in R_\psi$ des Root-Sets schicken. Jeder der Peer, der diese Nachricht weiterleitet, speichert dabei einen Datenlink auf den Storage-Server, der die Nachricht abgeschickt hat.

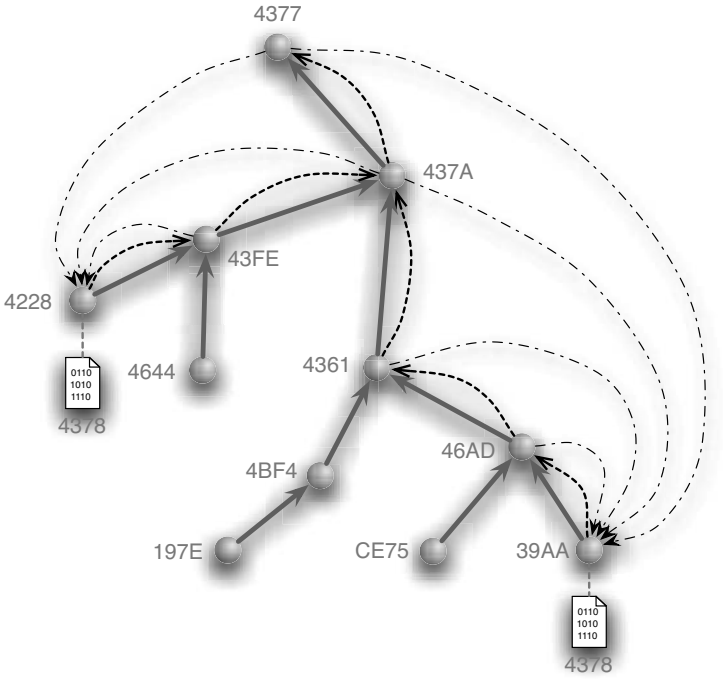


Abb. 6.13. Publikation von Objekten im Tapestry-Netzwerk.

Abbildung 6.13 veranschaulicht diesen Vorgang. Die Peers mit ID 39AA und 4228 sind Storage-Server des Datums mit ID 4378. Beide Peers publizieren ihr Datum entlang primärer Nachbarn beim zuständigen Root-Peer mit ID 4377. Jeder Peer, der die Publikationsnachricht weitergeleitet hat, speichert lokal einen Zeiger auf denjenigen Peer, der das publizierte Datum bereitstellt. So speichert der Peer mit ID 4361 beispielsweise einen Link für das Datum 4378 auf Peer 39AA.

Um Fehlertoleranz zu gewährleisten, werden alle beim Publikationsprozess entstandenen Zeiger nach einer festgelegten Zeit verworfen. Ein Storage-Server muss

die von ihm bereitgestellten Daten also in festgelegten Intervallen erneut publizieren. Dies hat den Vorteil, dass Daten von Peers, die unerwartet ausgefallen sind, automatisch aus dem System entfernt werden. Des Weiteren werden Peers, die sich erst nach dem ersten Publikationsprozess dem Netzwerk angeschlossen haben und nun auf dem Publikationspfad liegen, nach kurzer Zeit automatisch mit den entsprechenden Links auf die Daten versorgt.

Falls von einem Peer aus nach einem Datum ψ gesucht wird, so wird von diesem Peer das entsprechende Root-Set R_ψ berechnet und anschließend zu einem beliebigen Root-Server aus R_ψ geroutet. Wird auf dem Weg zu dem ausgewählten Root-Server ein Link für das gesuchte Datum getroffen, so wird direkt zu dem im Link gespeicherten Storage-Server gesprungen. Auf diese Weise werden Peers, die nahe (bezüglich ihrer ID) zu dem gesuchten Storage-Server liegen, bei der Suche schnell auf Abkürzungen durch Links treffen und müssen nicht bis zum Storage-Server routen.

In Abbildung 6.14 wird der gerade beschriebene Routing-Vorgang dargestellt. Der Peer mit ID 197E sucht nach dem Datum mit ID 4378 und schickt die Anfrage an den zuständigen Root-Server 4377. Auf dem Weg zum Root-Server wird beim Peer mit ID 4361 ein Link der vorangegangenen Publikation für das gesuchte Datum gefunden. Das Routing wird daraufhin abgebrochen und es wird direkt zu demjenigen verlinkten Peer mit ID 39AA gesprungen, der das gesuchte Datum bereitstellt, und somit werden zwei Hops bis zum Root-Server eingespart.

6.3.3 Surrogate-Routing

Ein bislang vernachlässigtes Problem ist, dass die von MapRoots(ψ) gelieferten Peer IDs unter Umständen gar nicht im Netzwerk existieren. Dies ist sogar wahrscheinlich, da der Namensraum nur sehr dünn besetzt ist, um Kollisionen zu vermeiden. Ein anderes Problem sind die dadurch entstehenden Löcher in den Routing-Tabellen.

Die Lösung für dieses Problem wurde von den Tapestry-Autoren *Surrogate-Routing* genannt (übersetzt etwa: Ersatz-Wegewahl) und funktioniert wie folgt: Wie zuvor wird schrittweise versucht, den Root-Server zu erreichen. Stößt man dabei auf ein Loch an der Stelle (z, j) der Routing-Tabelle eines Peers, so wird stattdessen nach $(z, j + 1)$ weitergeleitet. Handelt es sich auch dabei um ein Loch in der Routing-Tabelle, wird jeweils das nächst höhere $j \in B$ gewählt und nach dem Erreichen der größten Ziffer aus B wird mit der Ziffer 0 fortgefahren. Dieser Vorgang wird so lange wiederholt, bis der gesuchte Peer erreicht wurde oder der aktuelle Peer keine Einträge in höheren Levels und nur den einen, über den er erreicht wurde, im aktuellen Level seiner Routing-Tabelle hat. Es gilt nun folgendes Theorem über das Verhalten des Surrogate-Routing:

Theorem 6.4. *Wenn Eigenschaft* 1 *(*Konsistenz*) gilt, dann wird durch* Surrogate-Routing *ein eindeutiger Peer erreicht, egal von wo im Netzwerk das Routing startet.*

Beweis. Wir werden dieses Theorem durch einen Widerspruch beweisen. Wir nehmen an, eine Anfrage nach einem Datum ψ endet bei zwei verschiedenen Peers p_a

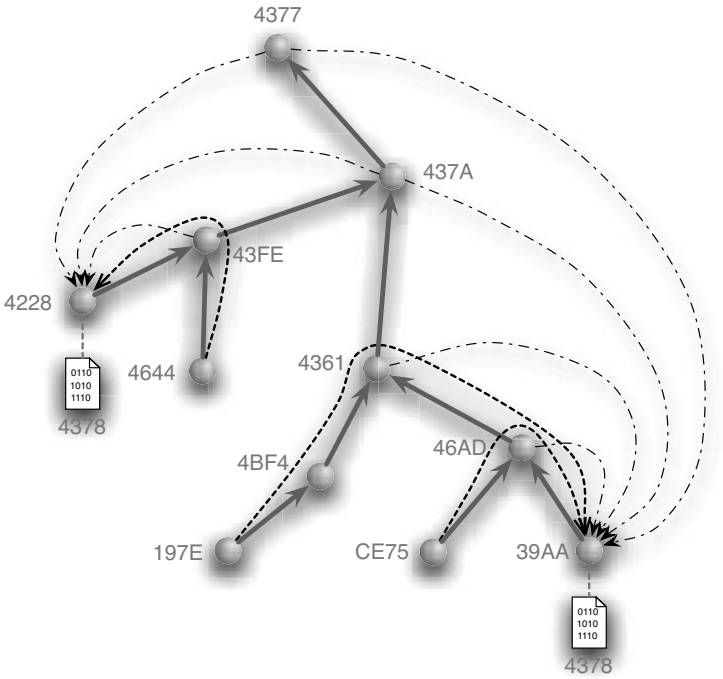


Abb. 6.14. Routing im Tapestry-Netzwerk.

und p_c. Sei z das längste gemeinsame Präfix von p_a und p_c mit $|z| = i$. Seien p'_a und p'_c die Peers, die den $i+1$-ten Routing-Schritt durchführen. Man beachte, dass nach diesem Schritt die ersten $i+1$ Ziffern der ID stets gleich bleiben. p'_a und p'_b sind also diejenigen Peers, die unterschiedliche Wege wählen. Wegen der Konsistenzeigenschaft gilt jedoch, dass von den Mengen $N^{p'_a}_{z,*}$ und $N^{p'_b}_{z,*}$ die jeweils gleichen Mengen leere Mengen sein müssen. Deshalb werden p'_a und p'_b die Nachricht an einen Peer mit derselben Ziffer an Stelle $i+1$ schicken, es sei denn, die Konsistenz ist gestört. Dies ist ein Widerspruch. □

Wir halten an dieser Stelle fest, dass Surrogate-Routing für insgesamt n Peers höchstens $\mathcal{O}(\log n)$ Hops benötigt, um das gewünschte Ziel zu erreichen, da mit jedem Schritt eine Ziffer an die gesuchte ID angepasst wird und mit hoher Wahrscheinlichkeit lediglich die ersten $\mathcal{O}(\log n)$ Level der Routing-Tabellen mit Einträgen besetzt sind. Letzteres folgt aus dem Beweis des Lemma 6.2.

6.3.4 Einfügen neuer Peers

Wird ein neuer Peer in ein bestehendes Netzwerk eingefügt, so soll das resultierende Netzwerk demjenigen gleichen, das entstanden wäre, wenn das Netzwerk von Grund auf mit diesem Peer entstanden wäre. Um dies zu gewährleisten, muss folgende Eigenschaft gelten, welche schon im Plaxton-Routing sichergestellt wurde.

Eigenschaft 4: Objekt-Links auf dem Publikations Pfad. *Liegt ein Peer p auf dem Pfad zwischen dem publizierenden Peer von Objekt ψ und dessen Root-Server, dann besitzt p einen Link auf den Peer, der ψ bereitstellt.*

Daten bleiben jedoch auch erreichbar, wenn diese Eigenschaft nicht erfüllt ist. Im schlimmsten Fall benötigt die Suche etwas länger (jedoch immer noch höchstens $\mathcal{O}(\log n)$ Schritte). Deshalb verzichten wir an dieser Stelle darauf zu zeigen, wie diese Eigenschaft sichergestellt werden kann. Wir halten hier lediglich fest, dass ein einfacher Algorithmus hierfür existiert (siehe [12]).

Kommen wir nun zum *Einfüge-Algorithmus* für neue Peers. Dieser besteht aus drei wesentlichen Schritten, die wir im Folgenden näher erläutern werden:

1. Suchen nach der eigenen ID im Netzwerk und Kopieren der Routing-Tabelle des so gefundenen Surrogate-Peers.
2. Kontaktieren derjenigen Peers, die Löcher in ihren Routing-Tabellen haben, die durch den neuen Peer gefüllt werden können.
3. Aufbauen und Optimieren der eigenen Routing-Tabelle.

Schritt 1 lässt sich einfach mit dem zuvor beschriebenen Surrogate-Routing umsetzen und benötigt lediglich $\mathcal{O}(\log n)$ Hops, so dass wir hier nicht auf weitere Details dieses Schritts eingehen.

Schritt 2 ist erforderlich, um Eigenschaft 2 (Konsistenz) gewährleisten zu können. Um dies zu erreichen, wird der so genannte *Acknowledged-Multicast-Algorithmus* verwendet. Hierfür wird zunächst das längste gemeinsame Präfix z von dem neuen Peer und des beim Einfügen gefundenen Surrogate-Peers bestimmt. Daraufhin geht man wie folgt vor:

- Der neue Peer sendet eine Multicast-Nachricht an seinen Surrogate-Peer. Diese besteht aus dem gemeinsamen Präfix und einer Funktion. (Dazu gleich mehr.)
- Empfängt ein Peer p eine Multicast-Nachricht, so sendet er sie an jeweils einen seiner Nachbarn $N^p_{z,j}$, $j \in B$, mit Präfix $z \circ j$ weiter.
- Empfängt ein Peer eine solche Nachricht, die er nicht mehr weiterleiten kann, führt er die in der Nachricht enthaltene Funktion aus.
- Die mitgesendete Funktion transferiert Links auf Daten zum neuen Peer, falls dieser einen alten Surrogate-Peer ersetzt, und entfernt nicht mehr benötigte Links. Auf diese Weise wird verhindert, dass Daten unerreichbar werden.
- Jeder Peer, der Multicast-Nachrichten verschickt hat, erwartet eine Rückmeldung (*Acknowledgement*) von allen Empfängern. Sind alle Rückmeldungen eingetroffen, so benachrichtigt er den im Aufrufbaum „über“ ihm liegenden Peer.

Wir werden nun zeigen, dass der Acknowledged-Multicast-Algorithmus tatsächlich alle Peers mit Präfix z im Netzwerk erreicht.

Theorem 6.5. *Wenn der Empfänger einer Multicast-Nachricht mit Präfix z sein Acknowledgement versendet, dann haben zuvor alle Peers mit Präfix z eine Multicast-Nachricht erhalten.*

Beweis. Wir zeigen die Korrektheit des Theorems durch Induktion über die Länge des Präfix z.

Induktionsanfang: Wir nehmen an, dass Peer p eine Multicast-Nachricht mit Präfix z erhält und p der einzige Peer mit Präfix z ist. In diesem Fall ist die Behauptung offensichtlich korrekt.

Induktionsschritt: ($|z| = i \rightarrow |z| = i - 1$): Angenommen, die Behauptung gilt für ein Präfix z mit $|z| = i$. Des Weiteren nehmen wir an, dass Peer p eine Multicast-Nachricht mit Präfix z erhält. Dann sendet p Multicast-Nachrichten an jeweils einen Peer mit den möglichen Erweiterungen von z (also $z \circ j$, für alle $j \in B$). Sobald p von jedem dieser Peers Acknowledgment-Nachrichten erhalten hat, wurden alle Peers mit Präfix z erreicht. Da p auf diese Acknowledge-Nachrichten wartet, bevor die eigene Acknowledge-Nachricht gesendet wird, haben bereits alle Peers mit Präfix z eine Multicast-Nachricht erhalten, wenn p sein Acknowledgement sendet.

Dies beweist das Theorem. □

Wir kommen nun zu Schritt 3 des Einfüge-Algorithmus für neue Peers. In diesem müssen die Nachbarschaftsmengen des neuen Peers aufgebaut und optimiert werden, um die Eigenschaften 1 (Konsistenz) und insbesondere Eigenschaft 2 (Lokalität) zu erfüllen. Dies kommt dem Lösen des Nächste-Nachbarn-Problems für viele verschiedene Präfixe gleich.

Der Aufbau der Nachbarschaftsmengen geschieht Level für Level. Aus dem vorhergehenden Schritt (Multicast) kennen wir bereits alle Peers, die das längste gemeinsame Präfix z, $|z| = i$, mit dem neuen Peer p teilen. Somit sind alle potenziellen Level i-Nachbarn bekannt, und es müssen nur noch für jede der Mengen $N^p_{z,j}$, $j \in B$, die jeweils k nächsten von diesen ausgewählt werden.

Anschließend werden sukzessive die Level $(i-1)$-Nachbarn mit Hilfe der zuvor berechneten Level i-Nachbarn berechnet, bis Level 0 erreicht wurde. Dies geschieht mit Hilfe des folgenden Algorithmus:

1. Fordere von allen Level i-Nachbarn Listen ihrer Level $(i-1)$-Nachbarn an.
2. Messe die Entfernung zu jedem der in Schritt 1 erhaltenen Peers gemäß einer gewählten Metrik (z.B. durch die RTT — *Round Trip Time*).
3. Wähle die k nächsten Elemente für jedes j aus und speichere diese in der entsprechenden Nachbarschaftsmenge.

Schritt 2 des Algorithmus ist von elementarer Bedeutung für die Lokalitätseigenschaft des Netzwerks. Damit der oben stehende Algorithmus tatsächlich eine Routing-Tabelle generiert, die der Lokalitätseigenschaft genügt, muss gelten, dass mit Hilfe der k nächsten Level i-Nachbarn mit hoher Wahrscheinlichkeit die k nächsten Level $(i-1)$-Nachbarn gefunden werden können. Dies ist tatsächlich mit hoher Wahrscheinlichkeit möglich, wenn zum einen $k \in \mathcal{O}(\log n)$ gewählt wird, d.h., jede der Nachbarschaftsmengen enthält $\mathcal{O}(\log n)$ Elemente, und für den durch die Latenzzeiten definierten metrischen Raum die folgende Restriktion gilt:

Wachstumsrestriktion: Bezeichne $B_p(r)$ die Menge aller Peers, die sich im Ball mit Radius r um Peer p befinden. Dann muss für eine Konstante c

$$|B_A(2r)| \leq c|B_A(r)|$$

und $c^2 < 2^b$ gelten. Die Konstante c wird auch als *Expansionskonstante* des Netzwerks bezeichnet.

Diese Restriktion ähnelt derjenigen im Plaxton-Routing. Man kann sich dies etwa als eine gleichmäßige Verteilung der Peers im Raum vorstellen, die gelten muss. Gilt diese Annahme nicht, so ist nicht mehr garantiert, dass der Algorithmus mit hoher Wahrscheinlichkeit die Lokalitätseigenschaft der Nachbarschaftsmengen aufrechterhalten kann. Wir werden auf den Beweis der Korrektheit des Algorithmus an dieser Stelle verzichten. Dieser kann in [12] nachgelesen werden.

Betrachten wir abschließend den Aufwand des Einfüge-Algorithmus. Hierbei konzentrieren wir uns wieder auf die benötigten Schritte im Netzwerk und vernachlässigen lokale Berechnungen. In Schritt 1 kann der Surrogate-Peer mit $\mathcal{O}(\log n)$ Hops gefunden werden. Der Acknowledged-Multicast-Algorithmus (Schritt 2) benötigt $\mathcal{O}(m)$ Hops, wobei m die Zahl der erreichten Peers ist. Hierbei ist m klein im Vergleich zu n und konstant. Der Aufbau der Nachbarschaftsmengen kann mit $\mathcal{O}(\log^2 n)$ Hops geschehen. Da jeder Peer eine konstante Anzahl Nachbarn je Level hat, beträgt die Laufzeit des Algorithmus zum Aufbau der Nachbarschaftsmengen $\mathcal{O}(k) = O(\log n)$ pro Level und somit insgesamt $\mathcal{O}(\log^2 n)$ für alle Level. Insgesamt ergibt sich somit:

Theorem 6.6. *Das Einfügen eines neuen Peers in Tapestry kann mit $\mathcal{O}(\log^2 n)$ Hops geschehen.*

6.4 Zusammenfassung

Pastry und Tapestry werden gerne in einem Atemzug genannt. Tatsächlich beruhen beide Peer-to-Peer-Netzwerke auf demselben Routing-Prinzip von Plaxton, Rajaraman und Richa [13]. Dieses stellt eine Verallgemeinerung von Routing auf dem Hypercube dar, wobei die entstehende Kommunikationslatenz nur konstant größer ist als die auf dem schnellsten Pfad.

Tapestry ist nicht vollständig selbstorganisierend, achtet aber stark auf die Konsistenz der Routing-Tabelle. Des Weiteren ist Tapestry sehr nahe am Originalverfahren von Plaxton et al. gehalten und daher analytisch gut verstanden: Tapestry hat nachweisbare Performanzeigenschaften, wenn die Annahmen zutreffen.

In Pastry werden statt analytischer Methodik viele heuristische Methoden bei der Umsetzung vom Plaxton-Routing eingesetzt. Die eigentliche algorithmische Beschreibung ist ungenau (zum Beispiel die Bestimmung der Menge M latenznaher Peers), und experimentelle Verifikation ersetzt hier analytische Untersuchungen. Dagegen lässt sich Pastry praktisch besser umsetzen und ist durch die Einführung der Leaf-Sets sehr robust.

7

Gradminimierte Netzwerke

In Paderborn entspringt die nur 4 km lange Pader — der kürzeste Fluss Deutschlands!
`www.paderborn-ueberzeugt.de`

Es gab einige Zeit lang einen Wettlauf, den Grad und Durchmesser von Peer-to-Peer-Netzwerken zu minimieren. Der Grad eines Netzwerks ist die maximale Anzahl von Nachbarn eines Peers und der Durchmesser ist die Länge des längsten aller kürzesten Pfade zwischen zwei Peers im zugehörigen Verbindungsgraphen des Netzwerks.

Bezeichne im Folgenden d den Grad eines Netzwerks. Somit besitzt ein Peer höchstens d direkte Nachbarn und höchstens d^i Nachbarn in Distanz kleiner gleich i. Damit ein Netzwerk den Durchmesser h besitzt, muss

$$d^h \geq n$$

gelten, woraus $h \geq \frac{\log n}{\log d}$ folgt. Es gibt also einen *Trade-Off* zwischen Grad und Durchmesser eines Netzwerks. Um zum Beispiel einen konstanten Durchmesser c zu erreichen, muss ein Netzwerk mindestens den Grad $n^{1/c}$ besitzen. Auf der anderen Seite impliziert ein konstanter Grad, dass das Netzwerk mindestens einen Durchmesser von $\Omega(\log n)$ haben wird.

Aus dieser Sicht ist das in in Kapitel 4 vorgestellte CAN mit einem polynomiellen Durchmesser und einem konstanten Grad nicht sonderlich effizient. Auch Chord, Pastry und Tapestry sind mit logarithmischem Grad und Durchmesser nicht optimal. Wir stellen in diesem Kapitel drei gradminimierte Netzwerke vor, die es schaffen, ein effizientes Peer-to-Peer-Netzwerk mit konstantem Grad und logarithmischem Durchmesser zu organisieren. *Viceroy* war das erste Peer-to-Peer-Netzwerk, das mit konstantem Ein- und Ausgrad einen logarithmischen Durchmesser erreicht hat. Kurze Zeit später folgten dann das *Distance-Halving* Netzwerk und *Koorde*. Beide reduzieren nochmals den Grad und verwenden eine einfachere Netzwerkstruktur als Viceroy.

Die drei in diesem Kapitel vorgestellten Netzwerke verwenden alle verteilte Hash-Tabellen für die Zuordnung von Daten zu Peers. Die Umsetzung der verteil-

ten Hash-Tabellen geschieht dabei weitgehend wie im Chord-Netzwerk, d.h., Peers und Daten werden eindeutige Positionen in einem eindimensionalem Raum zugewiesen, welcher als Ring betrachtet wird. Ein Datum wird jeweils von demjenigen Peer verwaltet, das der Position des Datums in aufsteigender Zählrichtung auf dem Ring folgt. Wir verweisen an dieser Stelle für weitere Details lediglich auf Kapitel 5.1 und werden bei den drei in diesem Kapitel vorgestellten Netzwerken nicht nochmals auf die Umsetzung der verteilten Hash-Tabellen eingehen, es sei denn es existieren Unterschiede zu Chord.

7.1 Viceroy

Viceroy ist ein von Dahlia Malkhi, Moni Naor und David Ratajczak entwickeltes Peer-to-Peer Netzwerk und wurde im Jahr 2001 vorgestellt. Viceroy ist das englische Wort für Vizekönig und bezeichnet zugleich eine Schmetterlingsart. Letzteres weist auf die Grundstruktur des Netzwerks hin: Viceroy stellt eine skalierbare und dynamische Emulation des *Butterfly-Netzwerks* in einem Peer-to-Peer-Netzwerk dar. Aus diesem Grund werden wir im folgenden Abschnitt zunächst das Butterfly-Netzwerk vorstellen und dann sehen wie dieses in Viceroy eingesetzt wird.

7.1.1 Das Butterfly-Netzwerk

Das Butterfly-Netzwerk ist ein sehr bekanntes Netzwerk mit hervorragenden Kommunikationseigenschaften, das schon lange vor der Zeit der Peer-to-Peer-Netzwerke eingesetzt wurde. Im Gegensatz zu Peer-to-Peer-Netzwerken, welche wie wir inzwischen wissen sehr dynamisch sind, handelt es sich bei dem Butterfly-Netzwerk um ein statisches Netzwerk, d.h., es ist nicht so konzipiert, dass während des Betriebs weitere Netzwerkknoten eingefügt werden können. Ein typisches Anwendungsbeispiel für das Butterfly-Netzwerk ist zum Beispiel die Verbindung der Prozessoren eines Parallel-Rechners.

Wir kommen nun zur formalen Definition des Butterfly-Netzwerks. Bezeichne von nun an $u(i)$ für ein Binärwort $u \in \{0,1\}^*$ das Binärwort, das sich lediglich im i-ten Bit von u unterscheidet. Dann ist das Butterfly-Netzwerk wie folgt definiert.

Definition 7.1 (Butterfly-Netzwerk). *Das Butterfly-Netzwerk der Dimension k wird als $BF(k) = (V_k, E_k)$ bezeichnet. Knotenmenge V_k und Kantenmenge E_k sind wie folgt definiert:*

$$V_k = \left\{(i,u); 0 \leq i < k, u \in \{0,1\}^k\right\}$$

$$E_k = \left\{\{(i,u),((i+1) \bmod k,u)\}; 0 \leq i < k, u \in \{0,1\}^k\right\}$$
$$\cup \left\{\{(i,u),((i+1) \bmod k,u(i))\}; 0 \leq i < k, u \in \{0,1\}^k\right\}$$

Die Kanten der ersten Mengen bezeichnen wir dabei als Kreiskanten *und die Kanten der zweiten Menge als* Kreuzkanten.

Die Knoten (i, u) des Butterfly-Netzwerks sind in *Ebenen* eingeteilt, wobei i jeweils die Ebene angibt. Abbildung 7.1 zeigt das Butterfly-Netzwerk BF(3). Man beachte, dass dieses lediglich aus drei Ebenen besteht, denn die Knoten der Ebene 0 werden in Abbildung 7.1 zweimal dargestellt, um das Einzeichnen der Kreuzkanten zwischen den Ebenen 0 und $k - 1$ zu vereinfachen.

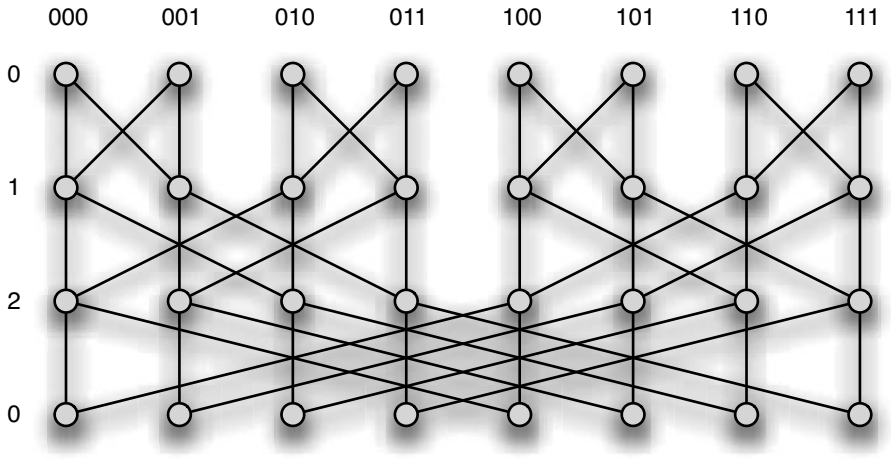


Abb. 7.1. Das Butterfly-Netzwerk BF(3).

Das folgende Theorem fasst einige grundlegenden Eigenschaften des Butterfly-Netzwerks zusammen.

Theorem 7.2. *Das Butterfly-Netzwerk BF(k) der Dimension k besitzt die folgenden Eigenschaften:*

1. *BF(k) besitzt $k2^k$ Knoten und $4k2^{k-1} = k2^{k+1}$ Kanten.*
2. *BF(k) ist 4 regulär.*
3. *BF(k) hat Durchmesser $k + \lfloor \frac{k}{2} \rfloor = \lfloor \frac{3k}{2} \rfloor$.*

Beweis. Die Anzahl der Knoten folgt direkt aus der Definition des Netzwerks. Der reguläre Grad 4 entsteht, da jeder Knoten jeweils eine Kreiskante und eine Kreuzkante zum nächst höheren und niedrigeren Level besitzt. Aus diesen beiden Eigenschaften lässt sich nun leicht die Anzahl der Kanten bestimmen, indem die Anzahl der Knoten mit dem Grad multipliziert und durch 2 geteilt wird, da jede Kante zu zwei Knoten adjazent ist.

Der Durchmesser lässt sich konstruktiv durch den im Folgenden angegebenen Routing-Algorithmus beweisen. Um von einem Knoten (i, u) zu einem beliebigen Knoten (j, v) zu gelangen, muss zum einen von Ebene i nach Ebene j gewechselt

und zum anderen das Binärwort u in v geändert werden. Betrachten wir die Kanten eines Knoten (i, u) der Ebene i genauer, so erhöhen (oder verringern) diese die Ebene jeweils um Eins. Wird eine Kreiskante verwendet, so bleibt das Binärwort u unverändert. Bei Verwendung einer Kreuzkante wird hingegen das i-te Zeichen des Binärworts u geändert und Knoten $((i+1) \mod k, u(i))$ erreicht. Aus dieser Beobachtung ergibt sich ein einfacher Routing-Algorithmus: Es werden k-mal Kreis- oder Kreuzkanten in jeweils aufsteigenden Ebenen verwendet, um das Binärwort u in v zu überführen, was k Schritte benötigt. Anschließend muss noch die Ebene von i nach j gewechselt werden, wofür maximal $\lfloor \frac{k}{2} \rfloor$ Kreiskanten benötigt werden.

Somit benötigt das Routing $k+\lfloor \frac{k}{2} \rfloor = \lfloor \frac{3k}{2} \rfloor$ Schritte. Man kann sich klar machen, dass auch tatsächlich Knotenpaare $(i, u), (j, v)$ existieren, für die kein kürzerer Weg existiert. Deshalb entspricht die angegebene Laufzeit des beschriebenen Algorithmus zugleich dem Durchmesser des Netzwerks. □

Der Durchmesser des Butterfly-Netzwerks ist sehr gering, wenn man den nur konstanten Grad bedenkt. Ist $n = k2^k$ die Anzahl der Knoten von BF(k), so gilt wegen $k \leq \log n$ für den Durchmesser $\lfloor \frac{3k}{2} \rfloor \leq \frac{3}{2} \log n$. Mit anderen Worten ist dieser also bis auf einen kleinen konstanten Faktor optimal.

Darüber hinaus besitzt das Butterfly-Netzwerk eine ganze Reihe weiterer positiver Eigenschaften, auf die wir hier allerdings nicht näher eingehen möchten. Beispielhaft sei die hohe Fehlertoleranz erwähnt, d.h., ein Butterfly-Netzwerk bleibt auch beim Ausfall vieler Knoten zusammenhängend. Wir halten also fest, dass viele Gründe dafür sprechen ein Butterfly-Netzwerk als Verbindungsstruktur zu wählen — auch für Peer-to-Peer-Netzwerke.

7.1.2 Übersicht

Die Zielsetzungen von Viceroy waren neben der Skalierbarkeit die Bewältigung von dynamischen Lastanforderungen und die gleichmäßige Verteilung des Nachrichtenaufkommens (*Traffic*). Diese gleichmäßige Verteilung misst man mit dem Begriff der *Congestion*. Die Congestion für einen bestimmten Routing-Algorithmus und ein bestimmtes Routing-Problem (in unserem Fall eine bestimmte Menge von Routing-Operationen) ist definiert als die maximale Anzahl von Paketen, die ein Peer transportieren muss. Es ist offensichtlich, dass die Congestion eine untere Schranke für die Zeit, die zur Beförderung aller Nachrichten benötigt wird, darstellt.

Neben der Congestion sollen natürlich sowohl die Kosten für das Einfügen und das Entfernen von Peers gering gehalten werden als auch die Länge der Suchpfade. All diese Eigenschaften erfüllt Viceroy. Zugleich war Viceroy das erste Peer-to-Peer-Netzwerk mit optimalem Verhältnis zwischen Grad und Durchmesser.

7.1.3 Netzwerkstruktur von Viceroy

Wie bereits erwähnt, basiert Viceroy auf verteilten Hash-Tabellen nach Chord-Vorbild. Der einzige Unterschied zur Umsetzung in Chord ist, dass der Raum für

das Hashing hier nicht aus den ganzen Zahlen $0, \ldots, 2^m - 1$, sondern aus dem reellen $[0, 1)$ Intervall besteht.

Der Aufbau der Netzwerkstruktur von Viceroy ist etwas komplexer als der vergleichbarer Netzwerke. Vom Prinzip her wird in Viceroy versucht die Peers auf die Knoten eines Butterfly-Netzwerks zu setzen. So wählt jeder Peer eine zufällige ID u aus dem Intervall $[0, 1)$ sowie eine zufällige Ebene i, $1 \leq i \leq \lfloor \log n \rfloor$. Im Gegensatz zur Ebene, wird die ID eines Peers während seiner Anwesenheit im Netzwerk niemals geändert. Wie wir in Abschnitt 7.1.4 sehen werden, wird die Ebene unter Umständen neu gewählt, wenn sich die Anzahl n der Peers im Netzwerk verändert.

Bei der Umsetzung des Butterfly-Netzwerks entstehen jedoch unter anderem die folgenden Probleme. Zunächst ist es notwendig, den Logarithmus der Anzahl der Peers im Netzwerk (also $\log n$) zu kennen um die Anzahl der Ebenen festzulegen. Des Weiteren entstehen offensichtlich Probleme durch die zufällige Wahl der IDs sowie Ebenen und insbesondere dann, wenn die Anzahl der Peers nicht von der Form $n = k \cdot 2^k$ mit $k \in \mathbb{N}$ ist, da das statische Butterfly-Netzwerk nur für ebensolche n definiert ist. Aus diesen Gründen stellt Viceroy keine Eins-zu-eins-Umsetzung des Butterfly-Netzwerks, sondern eher eine Approximation eben dieses dar.

Für die dynamische Approximation des Butterfly-Netzwerk werden in Viceroy die drei im Folgenden beschriebenen Klassen von Kanten verwendet.

Ring. Dieser Ring verbindet ähnlich wie der Chord-Ring alle Peers. So ist z.B. ein Peer mit ID u mit den Peers mit nächst größerer sowie kleinerer ID zu u verbunden. Der Ring sorgt für einen elementaren Zusammenhalt, erlaubt Routing im Fall fehlerhafter Kanten in den beiden anderen Mengen von Kanten und bestimmt letztendlich für welchen Teil des Hash-Raumes, also für welchen Bereich der Daten-IDs, ein Peer verantwortlich ist.

Ebenen-Ringe. Diese Ringe verbinden jeweils die Peers derselben Ebenen. So ist ein Peer mit Ebene i und ID u mit den Peers mit nächst größerer sowie kleinerer ID zu u derselben Ebene verbunden.

Butterfly-Kanten. Schließlich kommen noch die eigentlichen Butterfly-Kanten hinzu. Hat ein Peer Ebene i und ID u gewählt, so unterhält er die folgenden drei Verbindungen zu anderen Peers:

Linke Abwärtskante: Eine Verbindung zu dem Peer auf Ebene $i + 1$, dessen ID der Position u in aufsteigender Zählrichtung folgt.

Rechte Abwärtskante: Eine Verbindung zu dem Peer auf Ebene $i + 1$, dessen ID der Position $u + 1/2^i$ in aufsteigender Zählrichtung folgt.

Aufwärtskante: Eine Verbindung zu dem Peer auf Ebene $i - 1$, dessen ID der Position u in aufsteigender Zählrichtung folgt.

Ausnahmen bilden hier die Peers der ersten Ebene, die keine Aufwärtskanten besitzen, und die Peers der letzten Ebene, die keine Abwärtskanten besitzen.

Der Zusammenhang zwischen Butterfly-Kanten und IDs der Peers wird in Abbildung 7.2 dargestellt. Wir halten fest, dass der Ausgrad jedes Peers im Vieceroy-Netzwerk gerade 7 ist und sich aus zwei Zeigern für den Ring, zwei Zeigern für den Ebenen-Ring und drei Zeigern für die Butterfly-Kanten zusammensetzt. Gleiches gilt

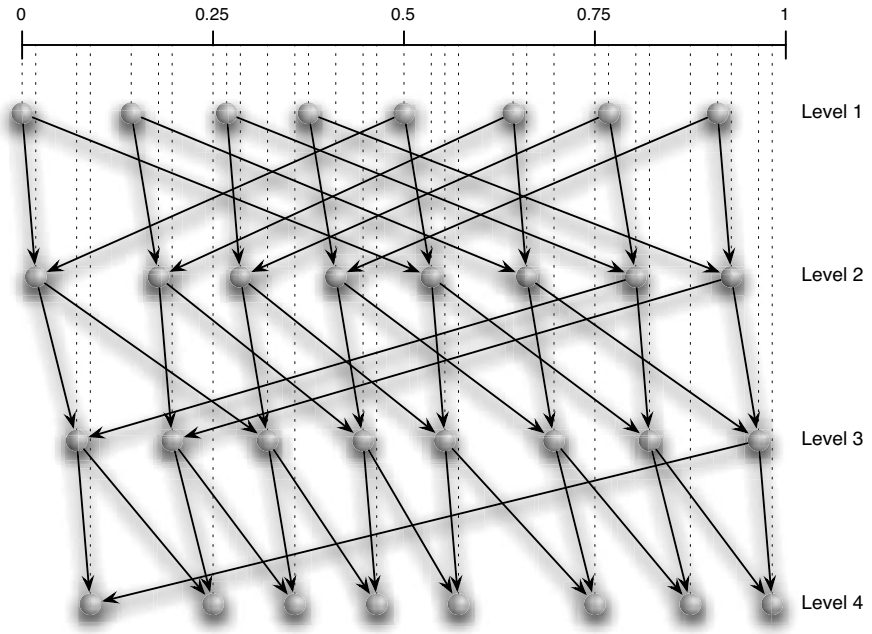


Abb. 7.2. Die Butterfly-Kanten von Viceroy [44]. Auf die Darstellung der Aufwärtskanten wurde aus Gründen der Übersichtlichkeit verzichtet.

für den erwarteten Eingrad. Allerdings ist der Eingrad nur dann konstant, wenn die Peers auf den Ebenen des Netzwerks perfekt verteilt werden, d.h., der Abstand zum Nachbarn entspricht dem erwarteten Abstand bis auf einen konstanten Faktor. Da die Positionen auf dem Ring jedoch zufällig gewählt werden, kommt mit konstanter Wahrscheinlichkeit auch ein um den Faktor $\log n$ erhöhter Abstand vor. Daraus ergibt sich, dass mit konstanter Wahrscheinlichkeit auch Peers mit logarithmischem Eingrad existieren. Dieses Problem, dass ein Peer mit großem Abstand auf seinem Ebenen-Ring viele eingehende Kanten anzieht und die Nachbarn in der vorigen Ebene mit kleineren Abständen den Eingrad erhöhen, kann man nur lösen, indem man die Peers nicht rein zufällig einordnet. Um mit hoher Wahrscheinlichkeit auch einen konstanten Eingrad für Peers zu erreichen schlagen die Autoren von Viceroy einen so genannten *Bucket-Mechanismus* vor, dessen Details wir hier jedoch nicht beschreiben werden (siehe [44]). Viel eleganter erhält man aber eine gleichmäßige Verteilung mit dem Prinzip der vielfachen Auswahl [45], das wir auf Seite 138 vorstellen werden.

Es bleibt zu zeigen, wie ein Peer eine ganzzahlige Ebene aus $1, \ldots, \lfloor \log n \rfloor$ wählen kann, oder anders formuliert: wie ein Peer die Anzahl n der sich im Netzwerk befindlichen Peers abschätzen kann.

7.1.4 Bestimmung der erforderlichen Ebenen

Da in Viceroy mit steigender Anzahl n an Peers auch die Anzahl $\lfloor \log n \rfloor$ der erforderlichen Ebenen steigt, ist es notwendig, die Größe von n oder zumindest einen Näherungswert n' von n zu kennen. In einem Peer-to-Peer-Netzwerk handelt es sich hierbei keineswegs um eine triviale Aufgabe. Schließlich verlassen und betreten ständig Peers das Netzwerk, ohne dass dieses lokal wahrgenommen wird. So ist präzises Zählen in einem großen Peer-to-Peer-Netzwerk nicht möglich, da dadurch bei jeder Änderung der Teilnehmerzahl Kommunikation zwischen allen n Peers erforderlich wird.

Die Autoren von Viceroy schlagen die Berechnung des Näherungswertes n' anhand des erwarteten Abstands zweier auf dem Ring benachbarter Peers im $[0, 1)$ Intervall vor. Da wir annehmen, dass die Peer-IDs uniform im $[0, 1)$ Intervall verteilt sind, ist der erwartete Abstand zwischen zwei benachbarten Peers auf dem Ring gerade $1/n$. Bezeichnet nun $dist(p_a, p_b)$ den Abstand in positiver Zählrichtung zwischen zwei benachbarten Peers p_a und p_b im $[0, 1)$ Intervall, dann kann der Näherungswert n' durch

$$n' = \frac{1}{dist(p, p.\text{Nachfolger})}$$

bestimmt werden. Der so bestimmte Näherungswert n' ist eine relativ grobe Schätzung, wie das folgende Lemma zeigt.

Lemma 7.3. *In einem Netzwerk mit n Peers wird durch das oben beschriebenen Verfahren jeder Peer die geschätzte Anzahl der erforderlichen Ebenen $\lfloor \log n' \rfloor$ so bestimmen, dass Konstanten c und c' existieren, so dass*

$$\log \frac{n}{c \log n} \leq \lfloor \log n' \rfloor \leq c' \log n$$

mit hoher Wahrscheinlichkeit gilt.

Beweis. Durch Lemma 5.1 (siehe Seite 83) wissen wir, dass der Abstand zweier auf dem Ring benachbarter Peers mit hoher Wahrscheinlichkeit höchstens um einen logarithmischen Faktor vom Erwartungswert nach oben und höchstens um einen polynomiellen Faktor nach unten abweicht. Konkret bedeutet dies, dass der Abstand mit hoher Wahrscheinlichkeit geringer als $\frac{c \log n}{n}$ und größer als $\frac{1}{n^{c'}}$ für Konstanten c und c' ist. Daraus folgt das Lemma. □

Ein positiver Effekt dieses Verfahrens zur Schätzung ist, dass die Schätzung n' eines Peers p lediglich von seinem Nachfolger auf dem Ring abhängt und sich die Schätzung somit auch lediglich ändert, wenn sich der Nachfolger auf dem Ring ändert. Falls sich durch den neuen Schätzwert n' auch der Wert $\lfloor \log n' \rfloor$ ändert, würde Peer p eine neue Ebene wählen. Tatsächlich wird p die Ebene jedoch nur wechseln, falls die zuvor gewählte Ebene nicht länger existiert oder die neu gewählte Ebene zuvor nicht existiert hat (andernfalls ist die zuvor getroffene zufällige Wahl ausreichend).

Wird eine genauere Schätzung von n benötigt, so kann man anstelle des Abstands zum nächsten Peer auf dem Ring, den Abstand d zum k-nächsten Peer des Rings im $[0, 1)$ Intervall für eine ausreichend groß gewählte Konstante k betrachten. Die Anzahl n der Peers kann dann durch

$$n' = \frac{k}{d}$$

abgeschätzt werden. Wir werden dieses Verfahren zur Schätung von nun an als *erweitertes Schätzverfahren* bezeichnen.

Für den Erwartungswert $E[d]$ des Abstands gilt $E[d] = k/n$. Auch hier kann gezeigt werden, dass die Abweichung von diesem Erwartungswert mit hoher Wahrscheinlichkeit durch einen logarithmischen Faktor nach oben und durch einen polynomiellen Faktor nach unten beschränkt ist. Für die Schätzung der erfoderlichen Anzahl an Ebenen gelten beim erweiterten Schätzverfahren insbesondere die im folgenden Lemma angegebenen Schranken, welche die Grundlage für unsere weitere Analyse des Viceroy-Netzwerks darstellen.

Lemma 7.4. *Für jedes $c' > 1$ und $c > 0$ kann in einem Viceroy-Netzwerk mit n Peers durch das erweiterte Schätzverfahren mit geeignet gewählten Parameter $k \in \mathcal{O}(1)$ jeder Peer die geschätzte Anzahl der erforderlichen Ebenen $\lfloor \log n' \rfloor$ so bestimmen, dass*

$$\log \frac{n}{c \log n} \leq \lfloor \log n' \rfloor \leq c' \log n$$

mit hoher Wahrscheinlichkeit gilt.

Der Beweis ergibt sich durch die Betrachtung des Nachbarintervalls eines messenden Peers und Anwendung der Chernoff-Schranken. Wir werden wie im Originalartikel im Folgenden annehmen, dass eine Abschätzung von

$$\log \frac{n}{2 \log n} \leq \lfloor \log n' \rfloor \leq 3 \log n$$

mit hoher Wahrscheinlichkeit vorliegt (obgleich eine genauere Schätzung möglich ist).

7.1.5 Routing

Wie schon die Netzwerkstruktur, ist auch Viceroys Routing-Algorithmus komplizierter als diejenigen vergleichbarer Netzwerke. Das Routing geschieht in drei Phasen, für die wir hier eine textuelle Beschreibung anstelle einer Pseudo-Code Beschreibung geben werden, um das Verständnis nicht unnötig zu erschweren. Im Folgenden beschreibt p jeweils den gerade aktiven Peer des Routings sowie dessen ID im $[0, 1)$ Intervall. Mit $p.ebene$ bezeichnen wir die Ebene des Peers p und mit $id \in [0, 1)$ das Ziel des Routings. Die drei Phasen gestalten sich dann wie folgt.

Phase 1: *Routing zu Ebene* 1. Route entlang der Aufwärtskanten bis zu einem Peer der Ebene 1. Sobald ein Peer der Ebene 1 erreicht wurde, fahre mit Phase 2 fort. Falls eine Aufwärtskante nicht existiert, übergebe die Anfrage an den Nachfolger auf dem Ring und fahre mit Phase 1 fort.

Phase 2: *Baum traversieren.* Falls $dist(p, id) < 1/2^{p.ebene}$ route entlang der linken Abwärtskante und fahre mit Phase 2 fort. Falls $dist(p, id) \geq 1/2^{p.ebene}$ route entlang der rechten Abwärtskante und fahre mit Phase 2 fort. Sollte die gewählte Abwärtskante nicht existieren oder auf ein Peer p' mit $p' > id$ zeigen, fahre mit Phase 3 fort.

Phase 3: *Ring traversieren.* Falls p der Nachfolger der Position id auf dem Ring ist, wurde das Ziel erreicht. Andernfalls route in Abhängigkeit des geringeren Abstands zur Ziel-ID zum Nachfolger oder Vorgänger von p auf dem Ring und fahre mit Phase 3 fort.

In den Phasen 1 und 2 besteht die Möglichkeit, dass entsprechende Aufwärts- bzw. Abwärts-Kanten nicht existieren und dann entlang des Rings geroutet werden muss. Der Grund hierfür ist, dass die Ebenen zufällig gewählt werden und somit nicht auszuschließen ist, dass bestimmte Ebenen von keinem der n Peers gewählt wurden. Dies wird wegen Lemma 7.3 eher auf große Ebenen zutreffen. Aufgrund der zufälligen Wahl der IDs, des im vorigen Abschnitt beschriebenen Verfahrens zur Bestimmung der erforderlichen Ebenen sowie des Verfahrens zur Auswahl der Ebene erfüllt das Viceroy-Netzwerk jedoch die folgenden Eigenschaften:

1. In einem Bereich der Größe $\frac{\log n}{n}$ des Rings befinden sich mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log n)$ Peers.
2. Ausgehend von einem beliebigem Peer einer beliebigen Ebene $i \leq \log \frac{n}{2 \log n}$ werden im Erwartungswert $\mathcal{O}(\log n)$ und mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log^2 n)$ Schritte entlang des Rings benötigt um einen Peer einer *bestimmten* Ebene $j \leq \log \frac{n}{2 \log n}$ zu erreichen.
3. Ausgehend von einem beliebigem Peer genügen mit hoher Wahrscheinlichkeit $\mathcal{O}(\log n)$ Schritte entlang des Rings um einen Peer einer *beliebigen* Ebene $i \leq \log \frac{n}{2 \log n}$ zu erreichen.

Wir werden diese Eigenschaften hier nicht beweisen, sie jedoch in den folgenden Lemmata und Theoremen verwenden. Wir werden nun den Routing-Algorithmus für Viceroy analysieren. Das folgende Lemma trifft zunächst eine Aussage über die ersten beiden Phasen des Algorithmus.

Lemma 7.5. *In einem Viceroy Netzwerk mit n Peers benötigen die Phasen 1 und 2 des Routing-Algorithmus mit hoher Wahrscheinlichkeit $\mathcal{O}(\log n)$ Schritte.*

Beweis. Nach Lemma 7.3 ist die Anzahl der Ebenen mit hoher Wahrscheinlichkeit geringer als $\mathcal{O}(\log n)$, wodurch in Phase 1 maximal $\mathcal{O}(\log n)$ Aufwärtskanten verwendet werden. Wie bereits angemerkt, kann es jedoch vorkommen, dass eine gewünschte Aufwärtskante nicht existiert. Falls dies in einer Ebene größer als $\log \frac{n}{2 \log n}$ geschieht, ist jedoch durch Eigenschaft 3 sichergestellt, dass nach maximal $\mathcal{O}(\log n)$ Schritten ein Peer p einer Ebene kleiner als $\log \frac{n}{2 \log n}$ erreicht wird. Ausgehend von Peer p existieren dann nach Eigenschaft 2 mit hoher Wahrscheinlichkeit Aufwärtskanten, so dass mit $\mathcal{O}(\log n)$ weiteren Schritten ein Peer der Ebene 1 erreicht werden kann. Somit werden innerhalb der ersten Phase mit hoher Wahrscheinlichkeit maximal $\mathcal{O}(\log n)$ Schritte benötigt.

Phase 2 benötigt ebenfalls mit hoher Wahrscheinlichkeit maximal $\mathcal{O}(\log n)$ Schritte, da die Anzahl der Ebenen nach Lemma 7.3 durch $\mathcal{O}(\log n)$ beschränkt ist und in jedem Schritt in die nächst höhere Ebene gewechselt wird. □

Die dritte Phase des Algorithmus benötigt unter Umständen mehr als logarithmisch viele Schritte, wie das folgende Lemma zeigt.

Lemma 7.6. *In einem Viceroy Netzwerk mit n Peers benötigt die Phase 3 des Routing-Algorithmus im Erwartungswert $\mathcal{O}(\log n)$ und mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log^2 n)$ Schritte.*

Beweis. In Phase 2 (Baum traversieren) wird mit jedem Schritt die maximal mögliche Distanz zum Ziel halbiert. Somit ist die Distanz zum Ziel in Ebene k maximal 2^{1-k}. Da durch Eigenschaft 2 mit hoher Wahrscheinlichkeit Abwärtskanten für alle Ebenen $\leq \log \frac{n}{2 \log n}$ existieren, ist beim Erreichen eines Peers der Ebene $\log \frac{n}{2 \log n}$ die maximale Distanz zum Ziel gerade $4 \frac{\log n}{n}$. Durch Eingenschaft 1 genügen dann mit hoher Wahrscheinlichkeit $\mathcal{O}(\log n)$ Schritte entlang des Rings um das Ziel zu erreichen.

Es ist jedoch auch möglich, dass $dist(p, id) \geq 1/2^{p.ebene}$ gilt, die rechte Abwärtskante jedoch hinter das Ziel zeigt und somit in Phase 3 übergegangen wird. In diesem Fall wird das Ziel im Erwartungswert um $\mathcal{O}(\log n)$ und mit hoher Wahrscheinlichkeit höchstens um $\mathcal{O}(\log^2 n)$ Peers auf dem Ring überschritten (dies folgt aus Eigenschaft 2).

Somit benötigt Phase 3 im Erwartungswert $\mathcal{O}(\log n)$ und mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log^2 n)$ Schritte auf dem Ring. □

Da sich die Gesamtzahl der benötigten Schritte des Routing-Algorithmus aus der Summe der Schritte der drei Phasen ergibt, erhalten wir aus Lemma 7.5 und Lemma 7.6 folgendes Korollar:

Korollar 7.7. *In einem Viceroy Netzwerk mit n Peers benötigt der Routing-Algorithmus im Erwartungswert $\mathcal{O}(\log n)$ und mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log^2 n)$ Schritte.*

Der beschriebene Routing-Algorithmus benötigt also mehr Schritte als das Routing in den zuvor beschriebenen Netzwerken Chord, Pastry und Tapestry. Allerdings haben wir bislang auch nicht die Ebenen-Ringe für das Routing verwendet. Tatsächlich lässt sich mit ihrer Hilfe die Anzahl der Routing-Schritte mit hoher Wahrscheinlichkeit auf $\mathcal{O}(\log n)$ beschränken, was das Routing ähnlich effizient wie in den zuvor genannten Netzwerken macht.

Die Laufzeit des Routing-Algorithmus wird bislang von der Laufzeit der dritten Phase dominiert, die mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log^2 n)$ Schritte benötigt. Um die Gesamtlaufzeit zu verringern, müssen wir also einen Weg finden um in der dritten Phase Schritte einzusparen. Dies ist möglich, wenn wir Phase 3 wie folgt abändern. In der Beschreibung bezeichnen wir mit $p.ebene\_nach$ bzw. $p.ebene\_vor$ den Nachfolger bzw. Vorgänger eines Peers p auf dem von p gewähltem Ebenen-Ring.

Phase 3: *Ring traversieren.* Falls p der Nachfolger der Position id auf dem Ring ist, wurde das Ziel erreicht. Andernfalls route zum Peer $p.ebene\_nach$ falls gilt $p.ebene\_nach \in [p, id]$ oder zum Peer $p.ebene\_vor$ falls gilt $p.ebene\_vor \in [p, id]$ und fahre mit Phase 3 fort. Zeigt keine der Ebenen-Kanten in das Intervall $[p, id]$, dann route in Abhängigkeit des geringeren Abstands zur Ziel-ID zum Nachfolger oder Vorgänger von p auf dem Ring und fahre mit Phase 3 fort.

Die Phasen 1 und 2 bleiben von den Änderungen unberührt, und wir werden den Algorithmus mit modifizierter dritten Phase fortan als *erweiterten* Routing-Algorithmus bezeichnen.

Das folgende Lemma zeigt, dass die modifizierte dritte Phase die Anzahl der benötigten Schritte tatsächlich um einen logarithmischen Faktor reduziert.

Lemma 7.8. *Die dritte Phase des erweiterten Routing-Algorithmus benötgt mit hoher Wahrscheinlichkeit $\mathcal{O}(\log n)$ Schritte.*

Beweis. Im Beweis zu Lemma 7.6 haben wir bereits festgestellt, dass wir zu Beginn der dritten Phase mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log^2 n)$ Schritte entlang des Rings vom Ziel entfernt sind.

Wir unterscheiden den Wechsel von Ebenen und die Schritte, die wir in einer Ebene durchführen können. Die Anzahl der $\mathcal{O}(\log n)$ Ebenen beschränkt die Anzahl der Ebenenwechsel, da jede Ebene so lange verwendet wird, bis die nächste Kante des Ebenen-Rings über das Ziel zeigt. Die erwartete Sprungweite bei einem Ebenenwechsel ist $\frac{1}{n}$. Die Sprünge entlang der Ebenen-Kanten haben eine erwartete Sprungweite von $\mathcal{O}(\frac{\log n}{n})$.

Wenn nun $c \log n$ Sprünge entlang der Ebenen-Kanten durchgeführt werden müssen, dann kann man mit Hilfe der Chernoff-Schranke nachweisen, dass mit hoher Wahrscheinlichkeit eine Distanz von mehr als $\frac{\log^2 n}{n}$ überwunden wird. Da das Ziel mit diesem Algorithmus nicht übersprungen werden kann, wird die Suche also schon vorher erfolgreich beendet. Das heißt nach höchstens $\mathcal{O}(\log n)$ Ebenen-Wechseln und insgesamt $\mathcal{O}(\log n)$ Sprüngen in einer Ebene. □

7.1.6 Einfügen eines Peers

Betritt ein neuer Peer p mittels eines Peers p_a das Viceroy-Netzwerk, so wählt er zunächst zufällig seine ID $u \in [0, 1)$ und erhält von p_a eine Schätzung n' der Peer-Anzahl n, mittels welcher er ebenfalls zufällig eine Ebene aus $i \in \{0, \ldots, \lfloor \log n' \rfloor\}$ wählt. Peer p wird dann über Peer p_a eine Suche nach dem für seine eigene ID u verantwortlichen Peer p_z durchführen. Zur Einbindung in das Netzwerk wird p dann zunächst in die Ringstruktur integriert und einen Teil der von p_z verwalteten Daten übernehmen. Darauf wird p in den Ring der Ebene i eingebunden und seine den Butterfly-Kanten entsprechenden Nachbarn wählen. Ebenso werden die entsprechenden Peers der benachbarten Ebenen informiert, ihre Zeiger neu zu justieren.

Die Anzahl der Schritte und Nachrichten für das Einfügen eines Peers besteht damit im Wesentlichen aus der Zeit für die Suche nach der eigenen ID mit $\mathcal{O}(\log n)$ und das Aufbauen der Routing-Tabelle mit Zeit $\mathcal{O}(\log n)$.

7.1.7 Diskussion

Viceroy war das erste Peer-to-Peer-Netzwerk mit konstantem Ausgrad. Mittels weiterer (hier nicht vorgestellter) Techniken, kann auch der Eingrad mit hoher Wahrscheinlichkeit konstant gehalten werden. Diese Techniken verursachen jedoch zusätzlichen Aufwand. Des Weiteren ist die Netzwerktopologie durch die mehrfache Ringstruktur vergleichsweise kompliziert und erschwert die praktische Umsetzung des Netzwerks.

Letzteres war wohl einer der Gründe, warum einer der drei Entwickler des Viceroy-Netzwerks selbst ein anderes Peer-to-Peer-Netzwerk als Nachfolger von Viceroy vorschlug, das er Distance-Halving-Netzwerk nannte.

7.2 Distance-Halving

Im Jahre 2003 stellten Moni Naor und Udi Wieder das *Distance-Halving*-Netzwerk vor [45]. Ziel der Autoren war nicht lediglich ein weiteres Peer-to-Peer-Netzwerk zu entwerfen, sondern es wurde besonderer Wert auf das im Folgenden vorgestellte Prinzip der kontinuierlichen Graphen gelegt. Dieses liegt eigentlich schon den Netzwerken CAN und Chord zu Grunde, von Naor und Wieder wurde das Prinzip jedoch erstmals formalisiert.

7.2.1 Kontinuierliche Graphen

Genauso wie man diskrete Graphen als Kantenmenge über einer endlichen Knotenmenge definiert, kann man auch kontinuierliche Graphen über einer unendlichen Menge von Knoten definieren. Als Beispiel eines kontinuierlichen Graphen betrachten wir direkt den im Distance-Halving-Netzwerk verwendeten. Die Knotenmenge V ist das kontinuierliche Intervall der reellen Zahlen $[0, 1)$. Die Kantenmenge $E \subseteq V \times V$ bildet eine unendlich große Paarmenge zwischen diesen Knoten. Der Graph des Distance-Halving-Netzwerks besteht aus vier Grundtypen von Kanten (siehe Abbildung 7.3):

- *Linkskanten*: $\left(x, \frac{x}{2}\right)$,
- *Rechtskanten*: $\left(x, \frac{1}{2} + \frac{x}{2}\right)$,
- *Rückwärts-Linkskanten* $\left(\frac{x}{2}, x\right)$ und
- *Rückwärts-Rechtskanten* $\left(\frac{1}{2} + \frac{x}{2}, x\right)$

für $x \in [0, 1)$. Betrachtet man zwei Punkte x und y im kontinuierlichen Graphen, so zeigen die Linkskanten auf zwei Punkte $x/2$ und $y/2$, deren Abstand zueinander nur noch halb so groß ist wie der zwischen x und y. Der Abstand reduziert sich nämlich von $|x - y|$ auf $\left|\frac{x}{2} - \frac{y}{2}\right| = \frac{|x-y|}{2}$. Ebenso verringert sich der Abstand der Punkte, auf den die Rechtskanten von x und y zeigen: $\left|\frac{1}{2} + \frac{x}{2} - \frac{1}{2} - \frac{y}{2}\right| = \frac{|x-y|}{2}$. Wegen dieser Halbierung der Distanz erhielt das Peer-to-Peer-Netzwerk seinen Namen Distance-Halving-Netzwerk. Umgekehrt führt die Verwendung der entsprechenden Rückwärtskanten zu einer Verdoppelung der Distanz.

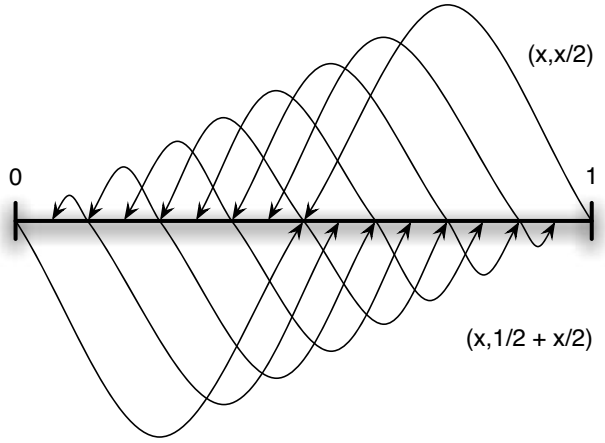


Abb. 7.3. Links- und Rechtskanten des kontinuierlichen Graphen des Distance-Halving-Peer-to-Peer-Netzwerks.

Der Übergang vom kontinuierlichen zum diskreten Graphen

Kontinuierliche Graphen lassen sich wegen der unendlichen Knotenanzahl nicht direkt als Netzwerkstruktur verwenden. Für den Übergang zum diskreten Graphen müssen wir die unendliche Knotenmenge V in endlich viele Teilintervalle partitionieren, welche im diskreten Graph als Knoten dienen und von uns fortan als *Segmente* bezeichnet werden. In unserem Fall entsprechen die Knoten bzw. Segmente des diskreten Graphen gerade den Peers des Netzwerks. Die Peers würden im einfachsten Fall wie schon in den zuvor vorgestellten Netzwerken eine zufällige Position im $[0, 1)$ Intervall wählen und wären dann für die Daten ab ihrer Position bis zur Position des Nachfolgers im $[0, 1)$ Intervall verantwortlich. Tatsächlich wird im Distance-Halving-Netzwerk eine modifizierte Methode zur Platzierung der Peers verwendet, wie wir in Abschnitt 7.2.2 sehen werden. Um den Übergang vom kontinuierlichen zum diskreten Graphen zu beschreiben, gehen wir jedoch zunächst von der zufälligen Wahl der Positionen im $[0, 1)$ Intervall aus und bezeichnen im Folgenden mit $x_1, x_2, \ldots, x_n$ die von den n Peers gewählten Positionen in aufsteigender Sortierung, d.h., $x_i < x_j$ für alle $i < j$. Dem Peer an Position x_i, $1 \leq i \leq n$, wird das Segment $s(x_i) = [x_i, x_{i+1})$ zugeordnet.

Nachdem wir nun die Knoten des diskreten Graphen definiert haben, definieren wir die Menge der Kanten nun wie folgt: Eine Kante zwischen zwei Segmenten $s(x_i)$ und $s(x_j)$ existiert genau dann, falls es Punkte $u \in s(x_i)$ und $v \in s(x_j)$ gibt, so dass (u, v) eine Kante des kontinuierlichen Graphen ist. Zusätzlich existieren noch Kanten zwischen benachbarten Segmenten, also eine Ringstruktur. Auf diese Weise kann man jeden Pfad im kontinuierlichen Graphen auf einen gleich langen Pfad im diskreten Graph abbilden.

Durch die gerade beschriebene Diskretisierung des oben beschriebenen kontinuierlichen Graphen entsteht das so genannte Distance-Halving-Netzwerk. Abbildung 7.4 veranschaulicht den Vorgang der Diskretisierung an einem kleinen Beispiel-Netzwerk mit sieben Peers. Es sei an dieser Stelle erwähnt, dass man das CAN-Netzwerk auf die gleiche Art und Weise als die Diskretisierung des zweidimensionalen Quadrats $[0,1)^2$ betrachten kann.

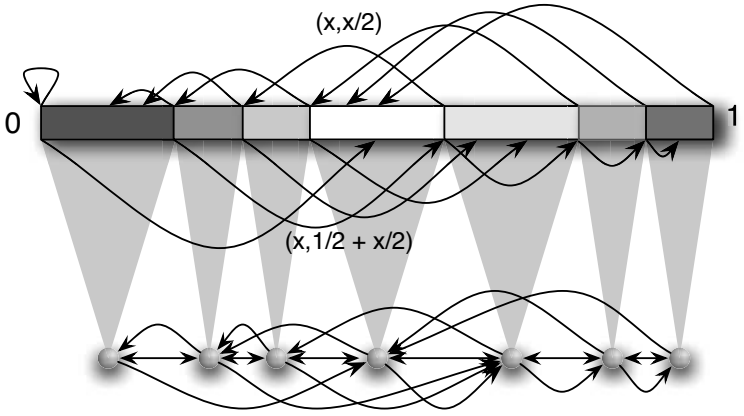


Abb. 7.4. Durch die Diskretisierung eines kontinuierlichen Graphen entsteht das Distance-Halving-Netzwerk.

Aus der Distanzeigenschaft folgt sofort, dass der Grad des Distance-Halving-Netzwerks konstant ist, falls das Verhältnis aus größtem und kleinstem Intervall konstant ist. Die Kanten eines Segments zeigen auf ein Intervall I, das für jeden Kantentyp höchstens doppelt so groß wie das Segment selbst ist. Sei

$$\rho = \max_{1 \leq i,j \leq n} \frac{|s(x_i)|}{|s(x_j)|}$$

das Verhältnis aus der maximalen Segmentgröße und der minimalen Segmentgröße. Dann kann sich das Intervall I nur mit $2\rho + 1$ vielen Segmenten überschneiden.

Ein konstantes Verhältnis $\rho = 4$ kann beim Einfügen durch das im Folgenden vorgestellte Prinzip der vielfachen Auswahl (*principle of multiple choice*) erreicht werden. Damit ergibt sich durch die Diskretisierung eine Graderhöhung um den Faktor neun und somit ein konstanter Grad für das Distance-Halving-Netzwerk.

7.2.2 Einfügen von Peers und das Prinzip der vielfachen Auswahl

Statt beim Einfügen eine zufällige Position im $[0,1)$ Ring zu wählen, betrachtet jeder Peer zunächst $k = c \log n$ zufällige Positionen $y_1, y_2, \ldots, y_k \in [0,1)$, wobei c

eine geeignet gewählte Konstante ist. Für jede dieser Positionen y_i wird die Größe $a(y_i)$ des den Punkt y_i umgebenden Segmentes $s(x_*)$ gemessen, also der Abstand zwischen den potenziellen linken und rechten Nachbarn im $[0, 1)$ Intervall. Nun wird das größte der gefunden Segmente ausgewählt und der neue Peer in der Mitte dieses Segmentes platziert.

Auf diese Weise wird immer ein relativ großes Segment gewählt, was letztendlich zur Folge hat, dass die Abstände zwischen den Peers relativ gleichmäßig sind (siehe auch Abbildung 7.5). Das folgende Lemma trifft eine genauere Aussage über die zu erwartenden minimalen und maximalen Segmentgrößen.

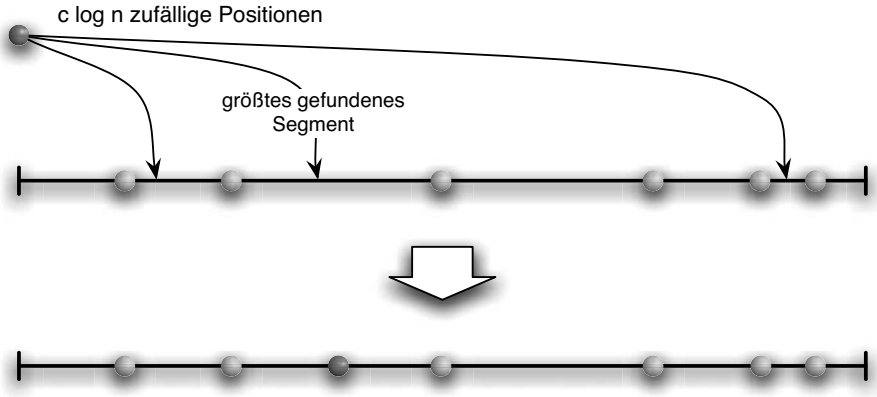


Abb. 7.5. Das Prinzip der vielfachen Auswahl beim Einfügen eines Peers.

Lemma 7.9. *Werden $n = 2^k$, $k \in \mathbb{N}$, Peers mittels des Prinzips der vielfachen Auswahl in den $[0, 1)$ Ring eingefügt, so bleiben mit hoher Wahrscheinlichkeit lediglich Segmente der Größe $\frac{1}{2n}$, $\frac{1}{n}$ und $\frac{2}{n}$ übrig.*

Beweis. Da die Segmente immer genau in der Mitte geteilt werden, entstehen als Segmentgrößen immer nur Zweierpotenzen. Es genügt also zu zeigen, dass zum einen keine Segmente kleiner als $\frac{1}{2n}$ und zum anderen keine Segmente größer als $\frac{2}{n}$ entstehen.

Wir zeigen zunächst, dass mit hoher Wahrscheinlichkeit keine Segmente größer als $\frac{2}{n}$ entstehen. Hierfür beweisen wir zunächst folgendes Lemma:

Lemma 7.10. *Habe das größte Segment die Größe g/n (g kann von n abhängig sein). Dann sind nach dem Einfügen von $2n/g$ Peers alle Segmente kleiner als $g/(2n)$.*

Beweis. Wir betrachten ein Segment der Größe g/n. Werden beim Einfügen jedes Peers $c \log n$ mögliche Positionen betrachtet und $2n/g$ Peers eingefügt, so ist die erwartete Anzahl von Treffern X in ein solches Intervall gerade

$$E[X] = \frac{g}{n} \cdot \frac{2n}{g} \cdot c \log n = 2c \log n \, .$$

Mit Hilfe der Chernoff-Schranke (siehe auch Anhang, Seite 268) erhalten wir dann für $0 \leq \delta \leq 1$:

$$Pr[X \leq (1-\delta)E[X]] \; \leq \; n^{-\delta^2 c} \, .$$

Ist nun $\delta^2 c \geq 2$, werden all diese Intervalle mindestens $2(1-\delta)c \log n$-mal getroffen. Hier muss jedoch berücksichtigt werden, dass jedes Mal, wenn ein Intervall von einem Peer geteilt wird, die $(c \log n) - 1$ weiteren Treffer dieses Peers (in möglicherweise andere große Intervalle) keine Teilung mehr bewirken können.

Für $2(1-\delta) \geq 1$ gilt, dass jedes der Intervalle der Mindestlänge g/n mit hoher Wahrscheinlichkeit geteilt wird. □ (Lemma 7.10)

Wendet man Lemma 7.10 hintereinander für $g = n/2, n/4, ..., 4$ an, dann kann mit hoher Wahrscheinlichkeit nach jeder Runde ausgeschlossen werden, dass ein Intervall der Größe g/n existiert. Die Anzahl der eingesetzten Peers ist $4+8+\ldots+n/4+n/2 \leq n$. Nach der letzten Runde sind keine Segmente größer als $\frac{2}{n}$. Da hier nur $\mathcal{O}(\log n)$ Ereignisse mit hoher Wahrscheinlichkeit eintreffen müssen, gilt die Aussage immer noch mit hoher Wahrscheinlichkeit.

Es bleibt zu zeigen, dass keine Segmente kleiner als $1/(2n)$ entstehen. Die Gesamtlänge aller Segmente der Größe $1/(2n)$ ist vor jedem Einfügen höchstens $n/2$. Damit ist die Wahrscheinlichkeit, dass bei $c \log n$ Versuchen nur solche Segmente gewählt werden, höchstens $2^{-c \log n} = n^{-c}$. Somit wird für $c > 1$ ein Segment der Größe $1/(2n)$ nur mit polynomiell kleiner Wahrscheinlichkeit weiter unterteilt. Damit ist der Beweis von Lemma 7.9 abgeschlossen. □

Wir haben bislang vernachlässigt, dass ein Näherungswert der Anzahl n der Peers im Netzwerk benötigt wird um beim Einfügen eines Peers $c \log n$ Positionen auf dem Ring überprüfen zu können. Im Kapitel über das Viceroy-Netzwerk haben wir gesehen wie diese Schätzung anhand des Abstandes zu den Nachbarn auf der Ringstrukutur geschehen kann. Durch die Verwendung des Prinzips der vielfachen Auswahl und insbesondere Lemma 7.9 ist diese Schätzung im Distance-Halving-Netzwerk sogar bis auf den Faktor 4 genau, schließlich hat das größte Segment mit hoher Wahrscheinlichkeit die Größe $\frac{2}{n}$ und das kleinste die Größe $\frac{1}{2n}$

Beim Einfügen werden die $c \log n$ zu überprüfenden Segmente jeweils durch eine Suche lokalisiert. Hierfür werden, wie wir gleich sehen werden, jeweils $\mathcal{O}(\log n)$ Schritte benötigt. Nachdem das größte Segment ausgewählt wurde, wird der einzufügende Peer zunächst in die Ringstruktur eingebunden und baut dann seine weiteren Verbindungen zu anderen Peers mit Hilfe der auf dem Ring benachbarten Peers auf. Entsprechend aktualisieren auch diese ihre Nachbarschaft im Netzwerk.

7.2.3 Routing im Distance-Halving-Netzwerk

Wir werden in diesem Abschnitt einen Routing-Algorithmus für das Distance-Halving-Netzwerk vorstellen, der obgleich des konstanten Grades lediglich $\mathcal{O}(\log n)$

Schritte benötigt und zugleich die beim Routing entstehende Last gleichmäßig auf die Peers verteilt. Um zunächst die grundlegende Idee des Routings in diesem Netzwerk zu vermitteln, betrachten wir zunächst eine vereinfachte Variante, die die Last nicht so gleichmäßig verteilt, jedoch auch nur eine logarithmische Anzahl an Schritten benötigt. Dieser einfache Routing-Algorithmus arbeitet rekursiv und ist in Abbildung 7.6 dargestellt.

```
Left-Routing(Start, Ziel)
    if Start und Ziel benachbart then
        Leite Nachricht von Start nach Ziel
    else
        Neuer-Start ← Links-Zeiger(Start)
        Neues-Ziel ← Links-Zeiger(Ziel)
        Sende Nachricht von Start nach Neuer-Start
        Left-Routing(Neuer-Start, Neues-Ziel)
        Sende Nachricht von Neues-Ziel nach Ziel
```

Abb. 7.6. Routing mittels Linkskanten und Rückwärts-Linkskanten im Distance-Halving-Netzwerk.

```
Right-Routing(Start, Ziel)
    if Start und Ziel benachbart then
        Leite Nachricht von Start nach Ziel
    else
        Neuer-Start ← Rechts-Zeiger(Start)
        Neues-Ziel ← Rechts-Zeiger(Ziel)
        Sende Nachricht von Start nach Neuer-Start
        Right-Routing(Neuer-Start, Neues-Ziel)
        Sende Nachricht von Neues-Ziel nach Ziel
```

Abb. 7.7. Ein alternativer Routing-Algorithmus mit Rechtskanten und Rückwärts-Rechtskanten für das Distance-Halving-Netzwerk.

Dieser Algorithmus verwendet nur die Linkskanten. Hierzu berechnet der Start-Peer zwei Zwischenstationen und reduziert das Routing-Problem auf eines mit der halben Entfernung. Dies geschieht solange, bis Start- und Zielknoten benachbart sind. Man könnte den Eindruck gewinnen, dass der Zielknoten sich an der Suche beteiligt, was aber nicht korrekt ist. Schließlich „weiß“ dieser nicht, dass er gesucht wird. Die Berechnung der ersten Zwischenstationen wird vom Startknoten durchgeführt. Diesen Zwischenstationen muss nun noch mitgeteilt werden, auf welchem

weiteren Weg die Nachricht zu befördern ist. In Abbildung 7.8 ist ein typischer Routing durch Verwendung von Linkskanten dargestellt. Natürlich funktioniert das Routing auch durch die Verwendung von Rechtskanten, siehe Abbildung 7.7.

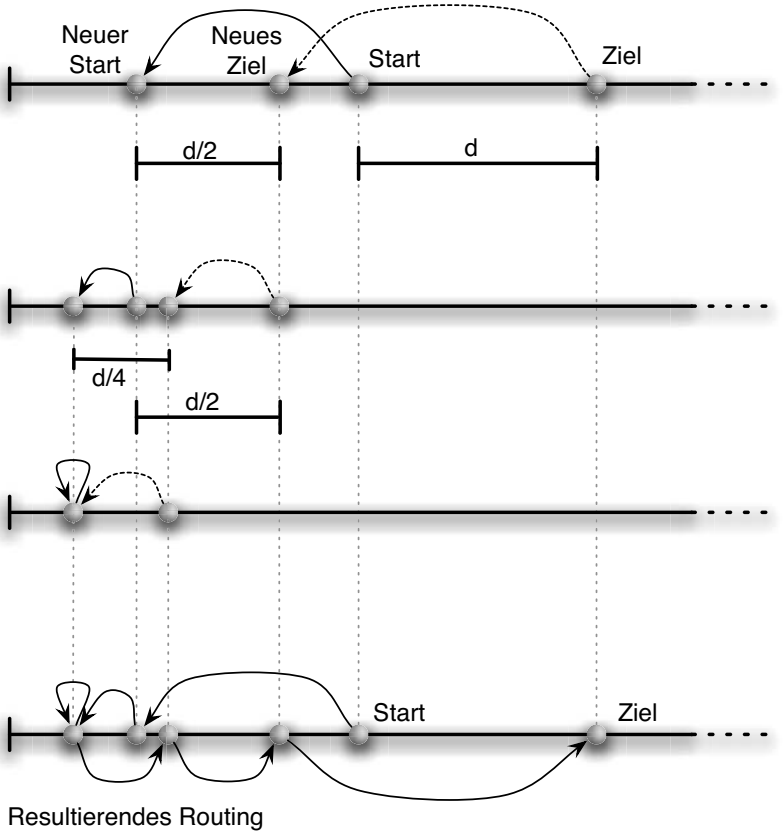


Abb. 7.8. Routing im Distance-Halving-Netzwerk mit Linkskanten.

Für beide Algorithmen wird die Distanz zwischen Start und Ziel mit jedem Rekursionsschritt halbiert und jeder Rekursionsschritt benötigt zwei Schritte. Da sich alle Intervallgrößen nur um den Faktor $\rho = 4$ unterscheiden, benötigt der Routing-Algorithmus höchstens $1 + \log n$ Rekursionen um eine Nachricht auszuliefern. Es ergibt sich somit als Routing-Aufwand $2 \log n + 3$:

Lemma 7.11. *Mit hoher Wahrscheinlichkeit benötigt das Routing im Distance-Halving-Netzwerk höchstens* $2 \log n + 3$ *Nachrichten und Schritte.*

Da die Links- und Rechtskanten in diesen Algorithmen beliebig ausgetauscht werden können, ergibt sich auch die Möglichkeit, die Orientierung (paarweise) durch

```
Random-Routing(Start, Ziel)
    if Start und Ziel benachbart then
        Leite Nachricht von Start nach Ziel
    else
        if Münze fällt auf Zahl then
            Neuer-Start ← Links-Zeiger(Start)
            Neues-Ziel ← Links-Zeiger(Ziel)
        else
            Neuer-Start ← Rechts-Zeiger(Start)
            Neues-Ziel ← Rechts-Zeiger(Ziel)
        Sende Nachricht von Start nach Neuer-Start
        Random-Routing(Neuer-Start, Neues-Ziel)
        Sende Nachricht von Neues-Ziel nach Ziel
```

Abb. 7.9. Congestion-optimierter Suchalgorithmus für das *Distance-Halving-Netzwerk*.

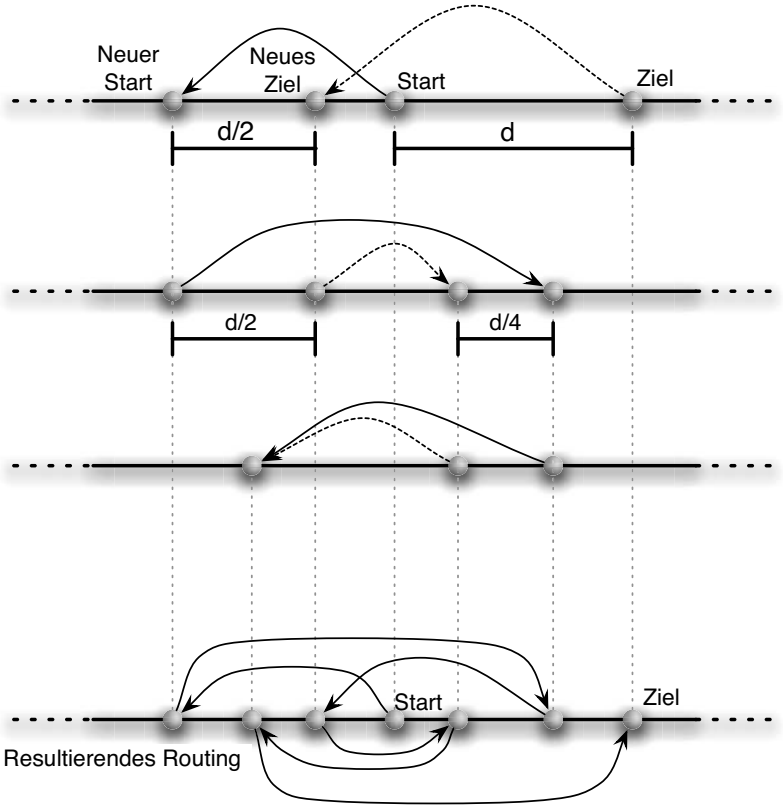


Abb. 7.10. Routing im Distance-Halving-Netzwerk mit Links- und Rechtskanten.

Münzwurf zu bestimmen. So erhält man den in Abbildung 7.9 dargestellten Algorithmus, der ebenfalls höchstens $2\log n + 3$ Schritte benötigt.

Während die ersten beiden Algorithmen dazu tendieren, den Verkehr in die äußerste linke oder rechte Ecke des Intervalls zu senden, sorgt dieser Algorithmus für eine sehr gute Verteilung der Datenlast. Man kann hier zeigen, dass die Belastung (*Congestion*) sehr gering ausfällt.

Es stellt sich heraus, dass das Distance-Halving-Netzwerk eine elegante und einfache Alternative zum komplizierten Butterfly-Graph-basierten Viceroy ist. Wir werden jetzt noch eine weitere einfache Alternative diskutieren.

7.3 Koorde

Das Peer-to-Peer-Netzwerk *Koorde* von David Karger und M. Frans Kaashoek [46] ist eine Verbesserung des Ansatzes von Chord hinsichtlich des Grades der Knoten und dem Aufwand für die Einfügeoperationen. Wir haben bei Chord einen Grad von $\mathcal{O}(\log n)$ (mit hoher Wahrscheinlichkeit) und einen Durchmesser von $\mathcal{O}(\log n)$ beobachtet. Einen wesentlich kleineren Grad kann man bei diesem Durchmesser nicht erwarten, denn wir haben bereits zu Beginn dieses Kapitels festgestellt, dass bei Grad d in h Schritten höchstens d^h Knoten erreicht werden können.

Akzeptiert man $d = \log n$ als sinnvollen Grad für ein Peer-to-Peer-Netzwerk, dann kann man bestenfalls nach $h = \frac{\log n}{\log\log n}$ Schritten jeden beliebigen anderen Peer erreichen. Das ist zwar asymptotisch kleiner als $\log n$, jedoch nur um den Term $\log\log n$, der für alle sinnvollen Netzwerkgrößen höchstens sechs wird. Tatsächlich gibt es Netzwerke mit solch guten Eigenschaften, z.B. den $\log n$-dimensionalen Hyperwürfel. Karger und Kaashoek wollten wie in den eben beschriebenen Netzwerken Viceroy und Distance-Halving — nur vermutlich unabhängig von diesen — den Grad von Chord unter Beibehaltung des Durchmessers reduzieren. Das hierbei entstandene Netzwerk Koorde [46] erreicht dies durch eine elegante Kombination eines Rings mit einem De-Bruijn-Netzwerk. Deshalb betrachten wir zunächst das De-Bruijn-Netzwerk.

7.3.1 Das De-Bruijn-Netzwerk

Das De-Bruijn-Netzwerk ist benannt nach seinem Entdecker Nicolaas Govert de Bruijn und kommt zum Beispiel in Parallelrechnern zum Einsatz. Das Netzwerk besitzt zudem die besondere Eigenschaft so genannte *De-Bruijn-Sequenzen* durch den Ablauf eines *Hamiltonschen Kreises*[1] im Netzwerk erzeugen zu können. Eine De-Bruijn-Sequenz $B(k, n)$ ist eine zyklische Folge von Zeichen über einem Alphabet A der Größe k, in der jede mögliche Teilfolge der Länge n aus A exakt einmal vorkommt. Diese Eigenschaft sei hier jedoch nur am Rande erwähnt.

[1] Ein Hamiltonscher Kreis ist ein geschlossener Pfad, der jeden Knoten genau einmal besucht und keine Kante mehrfach benutzt.

Bevor wir zur eigentlichen Definition des De-Bruijn-Netzwerks kommen, definieren wir zunächst drei einfache Operationen auf Binärwörtern. Bezeichne im Folgenden $S = (s_1, s_2, \ldots, s_m)$ mit $s_i \in \{0, 1\}$ ein Binärwort der Länge m.

Definition 7.12 (Shuffle-Operation). *Die Shuffle-Operation shuffle(S) rotiert die Bits eines Binärworts S um eine Stelle nach links, d.h.*

$$\textit{shuffle}(s_1, s_2, \ldots, s_m) := (s_2, s_3, \ldots, s_m, s_1) \ .$$

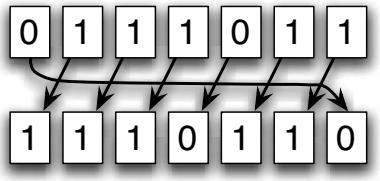


Abb. 7.11. Die Shuffle-Operation.

Definition 7.13 (Exchange-Operation). *Bezeichne $\neg s$ die Negation eines Bits s. Die Exchange-Operation exchange(S) negiert das am weitesten rechts stehende Bit eines Binärworts S, d.h.*

$$\textit{exchange}(s_1, s_2, \ldots, s_m) := (s_1, s_2, \ldots, \neg s_m) \ .$$

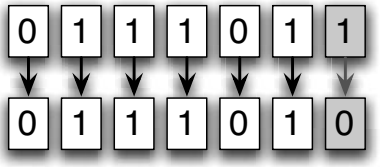


Abb. 7.12. Die Exchange-Operation.

Definition 7.14 (Shuffle-Exchange-Operation). *Bei einer Shuffle-Exchange-Operation (auch als SE bezeichnet) wird zuerst eine Shuffle- und dann eine Exchange-Operation auf S angewendet, d.h.*

$$\textit{shuffle-exchange}(S) := \textit{exchange}(\textit{shuffle}(S)) \ .$$

In der Bit-Darstellung von S ergibt sich:

$$\textit{shuffle-exchange}(s_1, \ldots, s_m) = (s_2, s_3, \ldots, s_m, \neg s_1) \ .$$

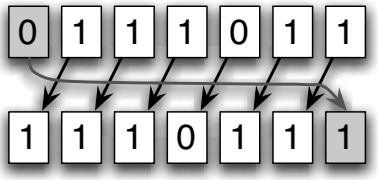


Abb. 7.13. Die Shuffle-Exchange-Operation.

Die drei Operation werden in den Abbildungen 7.11, 7.12 und 7.13 veranschaulicht. Wir halten zunächst folgende Beobachtung fest.

Lemma 7.15. *Ein Binärwort A der Länge m lässt sich durch m-faches Anwenden von Shuffle- und Shuffle-Exchange-Operationen in jedes beliebige Binärwort B gleicher Länge transformieren.*

In Abbildung 7.14 wird eine solche Transformation beispielhaft für die Binärwörter $A = 0111011$ und $B = 1001111$ durchgeführt. Man sieht hier, dass jedes der m Bits sukzessive auf den korrekten Wert gesetzt wird. Die notwendige Folge der Operationen erhält man, indem man A und B bitweise durch *XOR* (Exklusives Oder) verknüpft. Ein 1-Bit in der XOR Verknüpfung zeigt dann an, dass eine Shuffle-Exchange-Operation anzuwenden ist und ein 0-Bit zeigt an, dass eine Shuffle-Operation anzuwenden ist.

Das De-Bruijn-Netzwerk lässt sich jetzt leicht mit Hilfe von Shuffle- und Shuffle-Exchange-Operationen definieren. Das Netzwerk besteht aus $n = 2^k$ Knoten, die jeweils eine k-stellige Binärzahl darstellen. Jeder Knoten v besitzt zwei ausgehende Kanten, wobei die erste Kante von v auf den Knoten shuffle(v) und die zweite Kante auf den Knoten shuffle-exchange(v) zeigt.

Definition 7.16 (De-Bruijn-Netzwerk). *Das DeBruijn-Netzwerk der Dimension k wird als DB$(k) = (V_k, E_k)$ bezeichnet. Knotenmenge V_k und Kantenmenge E_k sind wie folgt definiert:*

$$\begin{aligned} V_k &= \{0,1\}^k \\ E_k &= \left\{ (v, \textit{shuffle}(v)) \,;\, v \in \{0,1\}^k \right\} \\ &\quad \cup \left\{ (v, \textit{shuffle-exchange}(v)) \,;\, v \in \{0,1\}^k \right\} \end{aligned}$$

Die Kanten der ersten Mengen sind die Shuffle-Kanten *und die Kanten der zweiten Menge sind die* Shuffle-Exchange-Kanten.

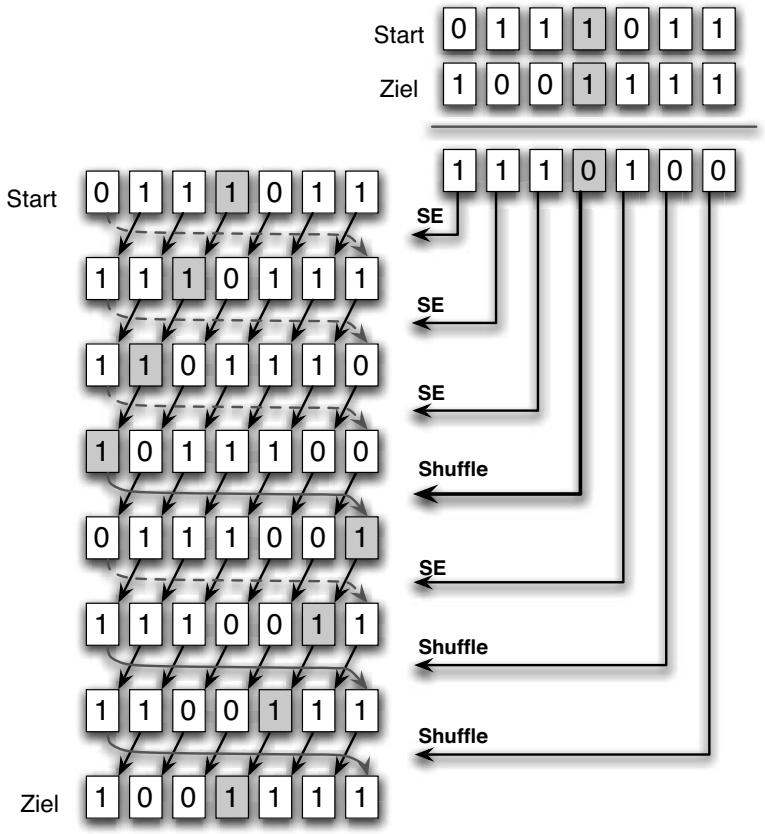


Abb. 7.14. Anwendung der Shuffle- und Shuffle-Exchange-Operation. Jedes Binärwort der Länge m lässt sich durch m Shuffle- oder Shuffle-Exchange-Operationen in jedes andere Binärwort gleicher Länge überführen.

Zur Veranschulichung wird in Abbildung 7.15 das De-Bruijn-Netzwerk DB(4) mit 16 Knoten dargestellt. Es ist gut zu erkennen, dass die gestrichelten Shuffle-Kanten Kreise im Netzwerk bilden. Das folgende Lemma fasst einige Eigenschaften des De-Bruijn-Netzwerks zusammen.

Lemma 7.17. *Das De-Bruijn-Netzwerk DB(k) der Dimension k besitzt die folgenden Eigenschaften:*

1. *DB(k) besteht aus 2^k Knoten und 2^{k+1} Kanten.*
2. *Jeder Knoten des DB(k) hat Eingrad und Ausgrad 2.*
3. *DB(k) hat Durchmesser k.*

Beweis. Die Aussagen über die Anzahl der Knoten, die Anzahl der Kanten sowie den Ausgrad der Knoten folgen direkt aus der Definition von DB(k). Die Aussage

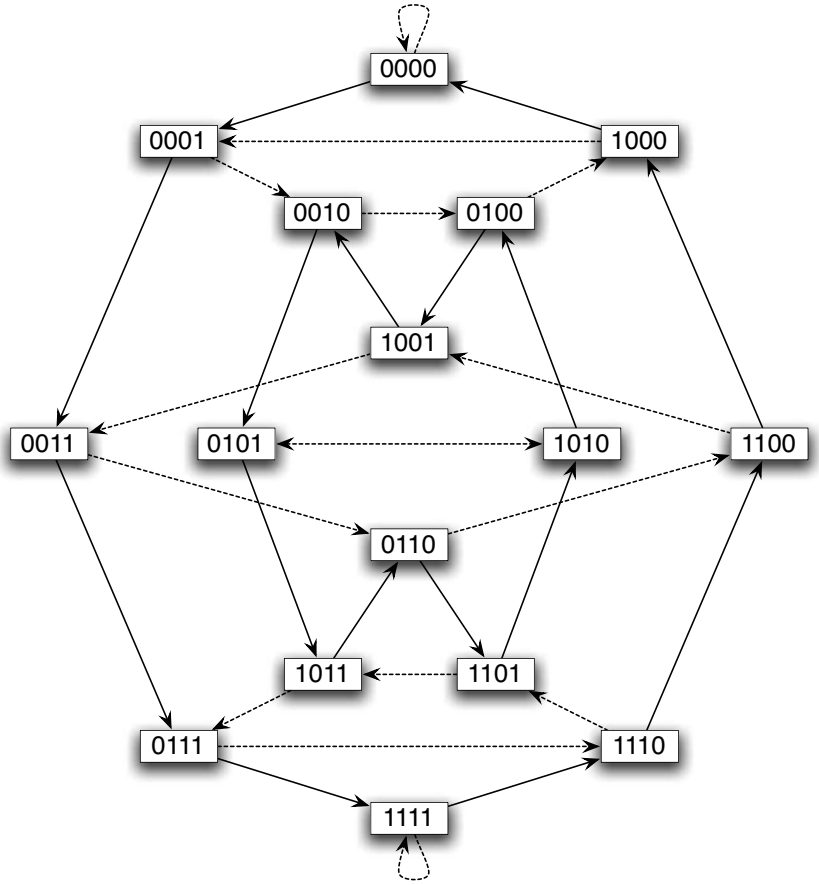


Abb. 7.15. Das De-Bruijn-Netzwerk DB(4). Die Shuffle-Kanten werden gestrichelt dargestellt.

über den Eingrad 2 der Knoten gilt, da sowohl die Shuffle- als auch die Shuffle-Exchange-Operationen Permutationen über den Binärzahlen sind.

Dass der Durchmesser höchstens $k = \log n$ ist, folgt direkt aus Lemma 7.15. Am Beispiel der Knoten $0 \ldots 0$ und $1 \ldots 1$ lässt sich leicht einsehen, dass auch tatsächlich Knoten mit dieser Entfernung existieren. □

Das De-Bruijn-Netzwerk besitzt also ähnlich gute Eigenschaften wie das von Viceroy verwendete Butterfly-Netzwerk. Die Umsetzung des Butterfly-Netzwerks in ein Peer-to-Peer Netzwerk war im Fall von Viceroy jedoch vergleichsweise kompliziert. Wie wir jetzt sehen werden, lässt sich die Struktur des De-Bruijn-Netzwerks eleganter in ein Peer-to-Peer-Netzwerk umsetzen.

7.3.2 Netzwerkstruktur und Routing

Wie erwähnt, kombiniert das Koorde-Netzwerk das De-Bruijn-Netzwerk mit einer Ringstruktur. Die Ringstruktur wird wiederum für das verteilte Hashing verwendet und hat die Größe 2^m. Somit werden auch die Peer-IDs zufällig aus den Zahlen $0, \ldots, 2^m - 1$ gewählt. Würde man $m = 4$ wählen und wäre tatsächlich die somit maximale Anzahl von 16 Peers im Netzwerk vorhanden, so würde sich die in Abbildung 7.16 dargestellte Netzwerkstruktur ergeben. Wie in der Abbildung zu sehen ist, besitzt jeder Peer zwei Ringkanten und zwei De-Bruijn-Kanten.

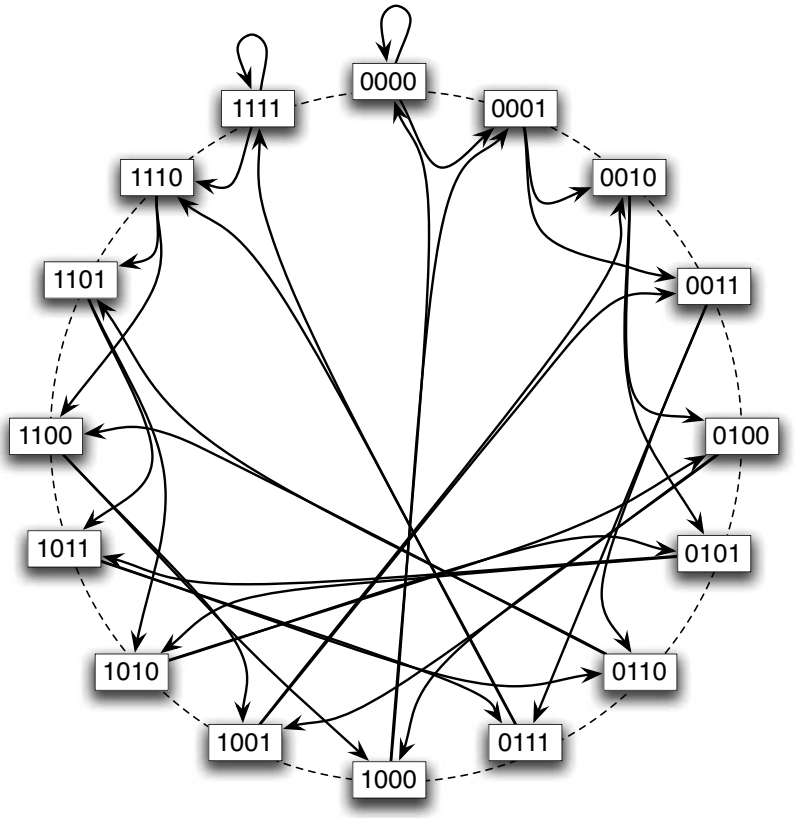


Abb. 7.16. Das De-Bruijn-Netzwerk für 16 Knoten. Die Knoten sind sortiert auf dem Ring aufgetragen.

Um diese scheinbar verwirrende Struktur besser zu verstehen, betrachten wir noch einmal die Shuffle- und Shuffle-Exchange-Operationen und interpretieren die Peer-IDs als ganze Zahlen über dem Ring $\{0, \ldots, 2^m - 1\}$. Für die Operationen gilt:

$$\text{shuffle}(x) = \left\lfloor \frac{2x}{2^m} \right\rfloor + (2x) \mod 2^m ,$$

$$\text{shuffle-exchange}(x) = 1 - \left\lfloor \frac{2x}{2^m} \right\rfloor + (2x) \mod 2^m .$$

Da $\lfloor \frac{2x}{2^m} \rfloor$ entweder den Wert 0 oder 1 annimmt, zeigen die De-Bruijn-Kanten von x auf $2x \bmod 2^m$ und $(2x+1) \mod 2^m$ (siehe auch Abbildung 7.17).

Tatsächlich werden in einem Koorde-Netzwerk jedoch niemals alle Peer-IDs vergeben sein, da der Parameter m gross genug gewählt wird um Kollisionen bei der Zuweisung von Peer-IDs unwahrscheinlich zu machen (typischerweise wird $m = 128$ oder $m = 160$ gewählt). Des Weiteren wird die Anzahl der Peers auch nur selten eine zweier Potenz sein (und nur für diese ist das De-Bruijn-Netzwerk definiert).

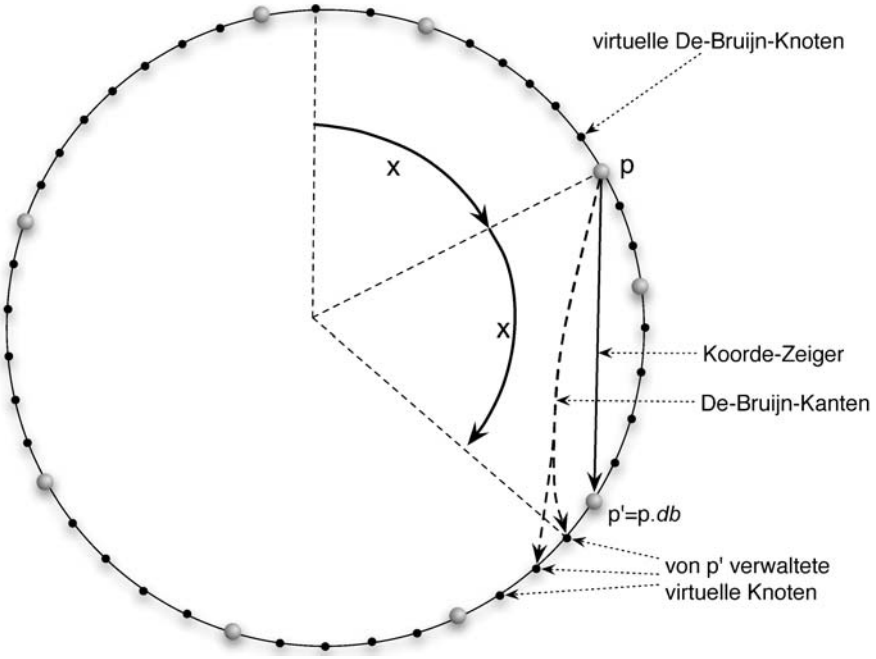


Abb. 7.17. Transformation des De-Bruijn-Graphen in ein Koorde-Peer-to-Peer-Netzwerk.

In Koorde wird dieses Problem folgendermaßen gelöst. In einem Netzwerk mit n Peers sind natürlich auch n (zufällige) Positionen des Rings durch Peers besetzt. Die nicht besetzten $2^m - n$ Positionen bezeichnen wir fortan auch als *virtuelle* De-Bruijn-Knoten. So werden die De-Bruijn-Kanten der Peers mit hoher Wahrscheinlichkeit auf virtuelle De-Bruijn-Knoten zeigen. Ist dies der Fall, wird für diese Kante

der dem virtuellen De-Bruijn-Knoten auf dem Ring vorangehende Peer p' gewählt (dieser wird auch als für den virtuellen Knoten *verantwortlicher* Peer bezeichnet). Da die De-Bruijn-Kanten auf benachbarte Positionen auf dem Ring zeigen, wird ein Peer p zudem nur die $2p \mod 2^m$ entsprechende Verbindung unterhalten. Dies ist sinnvoll, da die Shuffle- und die Shuffle-Exchange-Kante mit hoher Wahrscheinlichkeit ohnehin auf denselben Peer zeigen. So besitzt ein Peer also drei Nachbarn im Netzwerk: zwei Ring-Nachbarn und einen Nachbarn für die De-Bruijn-Kante (siehe auch Abbildung 7.17). Der Eingrad eines Peers kann hingegen höher sein. Um auch einen konstanten Eingrad zu erreichen, könnte z.B. das Prinzip der vielfachen Auswahl bei der Wahl der Peer-IDs bzw. dem Einfügen verwendet werden.

Um im Folgenden das Routing im Koorde-Netzwerk zu beschreiben, bezeichnen wir fortan für einen Peer p seinen Nachfolger auf dem Ring mit $p.Nachfolger$ und seinen entsprechend der De-Bruijn-Kante gewählten Nachbarn mit $p.db$. Des Weiteren führen wir folgende Notation ein: Für ein Binärwort x ist $x \circ 0 := 2x \mod 2^m$ und $x \circ 1 := (2x + 1) \mod 2^m$. Die entstehenden Werte entsprechen also den Shuffle bzw. Shuffle-Exchange-Kanten. Zudem liefere topBit(S) das am weitesten links stehende Bit eines m-stelligen Binärwortes S.

Beim Routing bzw. der Suche nach einem Datum mit der ID id im Koorde-Netzwerk muss zu dem Peer mit nächsthöherer ID als id geroutet werden. Da die meisten Knoten in Koorde virtuelle De-Bruijn-Knoten sind, simuliert Koorde den Weg in einem vollständigen De-Bruijn-Netzwerk,[2] indem für jeden virtuellen Knoten i jeweils durch den Peer geroutet wird, der für diesen verantwortlich ist (also dem letzten Peer vor der Position i des Rings).

Der Pseudo-Code des Routing-Algorithmus wird in Abbildung 7.18 dargestellt. Der momentane virtuelle Knoten wird im Algorithmus als Parameter i übergeben. In einem Routing-Schritt simuliert Koorde den Sprung vom virtuellen Knoten i zum Knoten $i \circ$ topBit(id) und überführt i so schrittweise in id. Dies geschieht durch einen Sprung von p, dem aktuellen Peer, nach $p.db$. Durch diesen Sprung wird ein Peer p' mit ID nahe $2p$ erreicht. Ist p' Vorgänger der Position $i \circ$ topBit(id), kann direkt die nächste De-Bruijn-Kante verwendet werden.

Wenn in jedem Routing-Schritt De-Bruijn-Kanten verwendet werden können, d.h. p' ist jeweils Vorgänger der Position $i \circ$ topBit(id), so werden insgesamt m Schritte benötigt, da mit jedem Schritt ein Bit an die Ziel-ID angepasst wird.

Dies anzunehmen wäre jedoch sehr optimistisch. Auch wenn $p' = p.db$ per Definition der der Position $2p$ vorangehende Peer des Rings ist, muss dies nicht zugleich der Vorgänger der Position $i \circ$ topBit(id) sein. Der Grund hierfür ist die zufällige Wahl der Peer-IDs. Deshalb überprüft jeder Peer, ob er tatsächlich der Vorgänger der Position $i \circ$ topBit(id) ist und routet, falls dies der Fall ist, entlang der De-Bruijn-Kante, und andernfalls an seinen Nachfolger auf dem Ring.

Das folgende Lemma trifft eine Aussage über die erforderlichen Schritte beim Routing in Koorde.

[2] Für ein Beispiel der Wegewahl siehe Abbildung 7.14.

```
// finde ausgehend von p den für id verantwortlichen Peer
p.suche(id, idshift, i)
    if id ∈ (p, p.Nachfolger]
        return p.Nachfolger
    else if i ∈ (p, p.Nachfolger]
        return p.db.suche(id, shuffle(idshift), i ∘ topBit(idshift))
    else
        return p.Nachfolger.suche(id, idshift, i)
```

Abb. 7.18. Routing-Algorithmus für Koorde. Variablen wie De-Bruijn-Kanten und Nachfolgern ist jeweils der betreffende Peer vorangestellt. So bezeichnet z.B. $p.db$ den De-Bruijn-Nachbarn des Peers p.

Lemma 7.18. *Sei 2^m die Ring-Größe eines Koorde-Netzwerks. Dann benötigt das Routing mittels des Algorithmus aus Abbildung 7.18 mit hoher Wahrscheinlichkeit $\mathcal{O}(m)$ Schritte.*

Beweis. Wir haben bereits gesehen, dass m Schritte genügen, falls nur De-Bruijn-Kanten verwendet werden. Es verbleibt also zu zeigen, dass in den Fällen wo $p' = p.db$ nicht zugleich der Vorgänger der Position $i \circ \text{topBit}(id)$ ist, jeweils nur konstant viele und insgesamt maximal $\mathcal{O}(m)$ Schritte entlang des Rings gemacht werden.

Um von einem virtuellen Knoten i zum Vorgänger der Position $i \circ \text{topBit}(id)$ zu gelangen (was gerade der Simulation eines Schrittes im statischem De-Bruijn-Netzwerk entspricht) routen wir zunächst von Peer p (dem Vorgänger von i auf dem Ring) zum Knoten $p' = p.db$ und dann entlang des Rings von p' bis zum Vorgänger der Position $i \circ \text{topBit}(id)$. Die Peers, die wir dabei benutzen, sind diejenigen mit IDs zwischen $2p$ und $2i$. Die erwartete Anzahl u von Peers in diesem Bereich des Rings ist gerade

$$E[u] = \frac{n(2i - 2p)}{2^m}$$

und somit neben n auch von p und i abhängig. Um die Abhängigkeit von p und i zu entfernen betrachten wir den Wert $i - p$, also den Abstand auf dem Ring. Dieser ist durch die zufällige Platzierung der Peers gerade $2^m/n$. Dadurch ergibt sich

$$E[u] = \frac{n\left(2\frac{2^m}{n}\right)}{2^m} = 2\ .$$

Anders formuliert bedeutet das, dass im Erwartungswert nur jeweils 2 Schritte entlang des Rings gemacht werden, was eine erwartete Schrittanzahl von $3m$ für das Routing ergibt. (Verwendet werden m De-Bruijn-Kanten und $2m$ Ring-Kanten.) Um eine Schranke mit hoher Wahrscheinlichkeit zu erhalten kann gezeigt werden, dass die Anzahl Schritte entlang des Rings jeweils mit hoher Wahrscheinlichkeit konstant ist, wodurch sich dann immer noch $\mathcal{O}(m)$ Schritte ergeben. □

Wir sind nun in der Lage mit $\mathcal{O}(m)$ Schritten in Koorde zu routen. Dies sind jedoch noch vergleichsweise viele Schritte, wenn wir bedenken, dass in den zuvor

vorgestellten Netzwerken $\mathcal{O}(\log n)$ Schritte genügten. Jedoch lässt sich auch in Koorde durch die Verwendung eines einfachen Tricks mit $\mathcal{O}(\log n)$ Schritten routen.

Die Idee ist die Folgende: Da der Startknoten p für alle virtuellen De-Bruijn-Knoten bis zu seinem Nachfolger auf dem Ring verantwortlich ist, können wir den virtuellen Startknoten i des Routings frei aus diesem Bereich wählen. Auf diese Weise können bereits einige Bits an die Ziel-ID des Routings angepasst werden. Genauer gesagt werden so viele der unteren Bits des Startknotens wie möglich (ohne den Zuständigkeitsbereich von p zu verlassen) auf den Wert der oberen Bits der Ziel-ID gesetzt und erst dann mit dem Routing begonnen. Wenn wir auf diese Weise k Bits anpassen können, benötigt das Routing nur noch $\mathcal{O}(m-k)$ Schritte.

Wie viele Bits wir anpassen können, hängt von der Größe des Zuständigkeitsbereichs des Start-Peers p ab. Dieser ist mit hoher Wahrscheinlichkeit größer als $2^m/n^c$ für eine Konstante c. Das bedeutet wiederum, dass wir mit hoher Wahrscheinlichkeit

$$k = \log(2^m/n^c) = m - c\log n$$

Bits wie zuvor beschrieben wählen können. Auf diese Weise müssen nur noch $c\log n$ Bits an die Ziel-ID angepasst werden, was lediglich $\mathcal{O}(\log n)$ Schritte im Netzwerk benötigt. Wir halten dieses Ergebnis im folgenden Theorem fest.

Theorem 7.19. *Routing in einem Koorde-Netzwerk mit n Peers kann mit hoher Wahrscheinlichkeit mit $\mathcal{O}(\log n)$ Schritten geschehen.*

Damit liegt uns mit Koorde eine einfache Netzwerkstruktur vor, die mit konstantem Ausgrad äußerst effiziente Operationen zur Verfügung stellt. Schließlich folgt aus dem logarithmischen Aufwand zum Suchen einer Position (Routing), dass das Einfügen oder Löschen eines Peers ebenfalls in logarithmischer Zeit vorgenommen werden kann.

Somit ist Koorde ebenso einfach wie Chord, nur effizienter. Negativ ist, dass für Koorde, im Gegensatz zu Chord, noch keine Stabilisierungsstrategie bekannt ist. Der kleine Grad bringt außerdem nicht nur Vorteile. So ist der Zusammenhang des Netzwerks entsprechend geringer. Damit nimmt die Gefahr zu, dass das Netzwerk bei Ausfällen von Peers auseinanderfällt.

Will man dieses Problem lösen, dann kann man den Grad des Graphen erhöhen, indem man so genannte *Grad k-De-Bruijn-Graphen* verwendet. Hierfür betrachten wir ein Alphabet aus k (statt zwei) Buchstaben, z.B. für $k=3$, $m=2$. Jeder k-De-Bruijn-Knoten x hat nun die Nachfolger $(kx \mod k^m), (kx+1 \mod k^m), (kx+2 \mod k^m), \ldots, (kx+k-1 \mod k^m)$. Dadurch verkürzt sich der Durchmesser auf $(\log m)/\log k$, während sich der Grad auf $\mathcal{O}(k)$ erhöht. Diese Konstruktion stellt eine natürliche Verallgemeinerung von Koorde dar, und die Ergebnisse lassen sich hier weitestgehend übertragen. Außerdem bewegt sich diese Lösung für alle k entlang des eingangs beschriebenen Trade-Offs zwischen Grad und Durchmesser.

7.4 Zusammenfassung

Mit Koorde und dem Distance-Halving-Netzwerk haben wir zum Abschluss einer Reihe auf DHT basierender Peer-to-Peer-Netzwerke zwei einfache, effiziente Peer-to-Peer-Netzwerke kennengelernt. Kombiniert man Koorde mit dem Prinzip der vielfachen Auswahl, so hat es wie das Distance-Halving-Netzwerk und Viceroy mit hoher Wahrscheinlichkeit konstanten Ein- und Ausgrad und logarithmischen Durchmesser. Alle drei Netzwerke erlauben Methoden zur Lastbalancierung und logarithmischen Routing. Die Einfügeoperation von Peers benötigt dann jeweils $\mathcal{O}(\log^2 n)$ Schritte und Nachrichten.

Ferner wurde hier erstmals das Prinzip des Übergangs von kontinuierlichen zu diskreten Graphen vorgestellt. Dieses Prinzip wird implizit in Chord, Koorde, Viceroy und CAN verwendet.

Das Prinzip der vielfachen Auswahl kann auch auf andere Peer-to-Peer-Netzwerke angewendet werden. So kann man damit bei Chord den Ein- und Ausgrad auf $\mathcal{O}(\log n)$ optimieren. Trotzdem wird die Einfügeoperationen weiterhin Aufwand $\mathcal{O}(\log^2 n)$ benötigen. Wie bereits gesagt, kann dieses Prinzip auch bei Koorde angewendet werden. Dann ist dort der Ein- und Ausgrad konstant und der für die Einfüge-Operationen benötigte Aufwand $\mathcal{O}(\log^2 n)$. Bei näherer Betrachtung verschwinden zunehmend die Unterschiede zwischen Koorde und dem Distance-Halving-Netzwerk. Die Komplexität von Viceroy dagegen zeigt, dass das Butterfly-Netzwerk als Konstruktionsprinzip für Peer-to-Peer-Netzwerke kaum geeignet sind.

8

Geordnete Indizierung

Ordnung ist das halbe Leben.
... aber die andere Hälfte ist schöner.
Volksmund

Bis jetzt haben wir hauptsächlich Peer-to-Peer-Netzwerke kennengelernt, deren Indizierung durch verteilte Hash-Tabellen vorgenommen wird. Damit werden die Peers gleichmäßig belastet und kleine Änderungen im Datenbestand oder in der Menge der anwesenden Peers haben nur lokale Auswirkungen. Leider verliert man Beziehungen zwischen den Daten. So werden zum Beispiel die Indizes von „Peer-to-Peer-Netzwerke“ und „Peer-to-Peer Netzwerke“ durch das Hashing wahrscheinlich an völlig verschiedenen Orten abgelegt, obgleich sich die beiden Zeichenketten nur wenig unterscheiden. Eine Suche nach irgend einem Stichwort, das mit „Peer“ anfängt, ist also in verteilten Hash-Tabellen nicht effizient möglich.

In diesem Kapitel stellen wir Methoden vor, die es Peer-to-Peer-Netzwerken ermöglichen, die Beziehungen der Daten im Netzwerk zu erhalten. In den folgenden Netzwerken werden ähnliche Daten auf denselben oder wenigstens leicht erreichbaren Peers abgelegt.

8.1 P-Grid

Das *P-Grid*-Netzwerk [47, 48] wurde 2001 an der EPFL (École Polytechnique Fédérale de Lausanne) von Karl Aberer entwickelt. Es beruht auf der Verwendung eines *binären Suchbaums* und der geeigneten Verteilung der Peers in dieser Datenstruktur.

Ein Suchbaum ist eine klassische Datenstruktur zur effizienten Speicherung und Suche von Daten. Wir beschreiben den Suchbaum als gerichteten Graphen mit einem Wurzelknoten und Kanten, die in Richtung der Blätter des Baums zeigen. P-Grid betrachtet den Spezialfall eines Binärbaums. Alle Kanten sind mit 0 oder 1 nummeriert. Es geht also von jedem inneren Knoten höchstens eine Null- oder Einskante aus.

Mit Hilfe dieser Kanten erhält man zu jedem Knoten des Baums einen eindeutigen Pfad. Die Konkatenation (Aneinanderreihung) der Ziffern an den zugehörigen Kanten weist jedem Knoten eine eindeutige Bitfolge zu. Die Wurzel erhält die leere Zeichenkette ϵ. Umgekehrt kann man so auch jeder Zeichenkette einen Pfad in dem Binärbaum zuordnen, indem man geeignete Knoten und Kanten einfügt, wobei es vorkommen kann, dass man in einem Blatt endet, bevor man die Bitfolge abgearbeitet hat, oder dass die Bitfolge zu kurz ist und man in einem inneren Knoten endet. Solch einen Binärbaum nennt man einen *Trie*.

Der Begriff *Trie* (manchmal ausgesprochen wie das englische Wort *„tree“*, manchmal wie das englische Wort *„try“*) entstand aus den Begriffen *reTRIEval TREE*. Diese Bäume werden unter anderem zur effizienten Suche in Texten und zur Textkompression verwendet. Normalerweise stehen in einem Trie nicht binäre Ziffern, sondern die Buchstaben des lateinischen Alphabets auf den Kanten, siehe Abbildung 8.1. Jeder Knoten bezeichnet dann ein Wort eines Wörterbuchs. Die Wörter eines Textes werden auf den internen Knoten und den Blättern des Baums (Trie) gespeichert. Manch einer kennt vielleicht die Darstellung des *Morse-Codes* als Trie, welche hier zur Veranschaulichung des Prinzips des Tries angeführt wird, siehe Abbildung 8.2. Statt 0 und 1 stehen hier — und • auf den Kanten und die Buchstaben auf den entsprechenden Knoten.

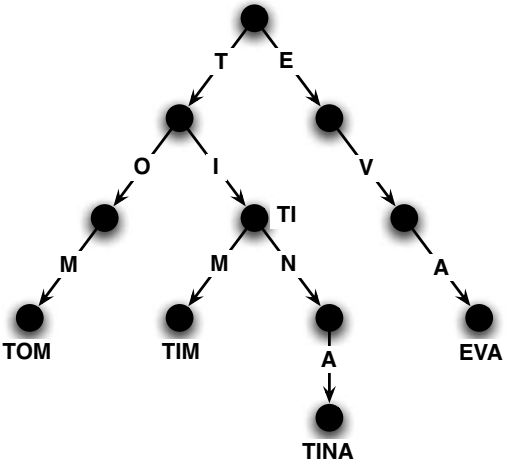


Abb. 8.1. Ein Beispiel eines Tries.

Da P-Grid auf solchen Binär-Tries aufbaut, muss für die Beschreibung von Schlüsselwörtern der Daten eine Umkodierung auf das Binäralphabet stattfinden. Denkbar wäre die Standardkodierung mit ASCII oder Unicode. Weitaus besser ist aber eine *Huffman-Kodierung*, da dadurch die Baumstruktur balanciert ist: Häufig vorkommende Zeichen werden auf kurze Binärstrings abgebildet.

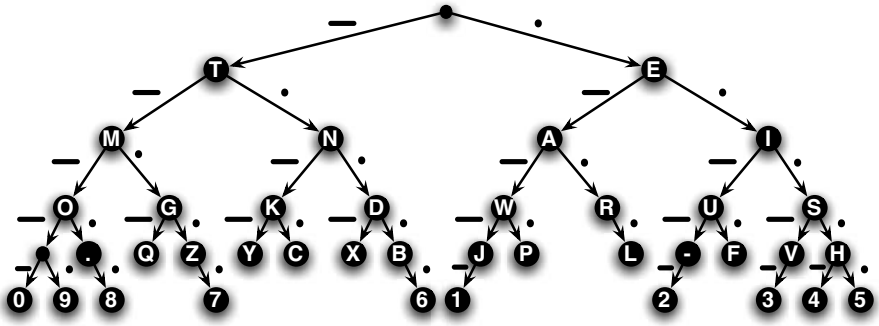


Abb. 8.2. Der Morse-Code als Beispiel für einen Binär-Trie.

Es ist nicht empfehlenswert, einen Baum direkt in ein Peer-to-Peer-Netzwerk umzusetzen, indem man auf jeden Knoten einen Peer setzt. Als erstes wäre es ein ungewöhnlicher Zufall, wenn die Anzahl der Peers der Knotenanzahl entspräche. Zweitens ist die resultierende Netzwerkstruktur nur einfach zusammenhängend. Das heißt, geht ein interner Knoten oder irgendeine Kante verloren, so verliert der Graph und damit das Netzwerk seinen Zusammenhang. Ein solches Verhalten ist nicht akzeptabel für ein Peer-to-Peer-Netzwerk, das andauernd den plötzlichen Ausfall von Peers verkraften muss.

8.1.1 Netzwerkstruktur von P-Grid

Daher wird in P-Grid eine andere Zuordnung zwischen Peers und Knoten des Baums durchgeführt. Peers sind den internen Knoten des Baumes so zugeordnet, dass kein Peer auf dem Pfad eines anderen Peers liegt. Ferner gehen wir davon aus, dass die Daten (oder die Zeiger auf die Daten) so tief im Trie abgespeichert werden, dass niemals ein Peer unterhalb der Daten liegt. Das kann zum Beispiel dadurch erreicht werden, dass man sehr lange Schlüsselwörter durch Anhängen von Kontrollinformationen, wie Erzeugungsdatum, Hersteller etc. erstellt. Zuletzt wird sichergestellt, dass jeder Pfad im Baum zwischen Wurzel und den Blättern wenigstens einen Peer besitzt, siehe Abbildung 8.3.

So gesehen formen die Peers die Blätter eines Baumes, der oberhalb der Peers mit dem Suchbaum übereinstimmt. Die internen Knoten dieses Baums werden durch die Peers des darunter liegenden Teilbaums mit übernommen. Ein Peer hat in seiner Routing-Tabelle für jeden, der zu seinem eigenen Pfad benachbarten Teilbäume, einen Eintrag, der auf einen Peer des Teilbaums verweist. Außerdem wird vorausgesetzt, dass diese Nachbarn gleichwahrscheinlich aus den jeweiligen benachbarten Teilbäumen gewählt werden.

Natürlich besteht nun wiederum die Gefahr, dass ganze Teilbäume wegfallen, wenn sich nur ein Peer aus dem Netzwerk entfernt. Daher geht man davon aus, dass sich statt eines Peers eine bestimmte Mindestanzahl von k Peers die Aufgaben teilen, die mit der Übernahme eines Teilbaums entstehen.

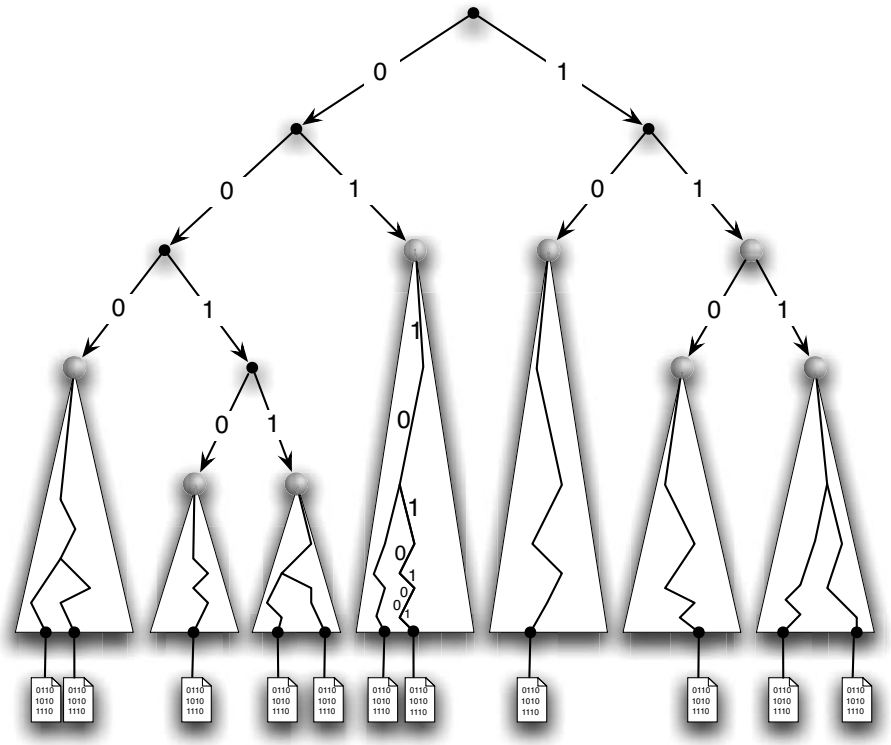


Abb. 8.3. Beispiel eines P-Grid-Netzwerks.

Der Hauptgrund, warum in anderen Peer-to-Peer-Netzwerken Hash-Funktionen verwendet werden, ist die Balancierung der Daten. Die Verteilung der Daten ist nämlich typischerweise nicht gleichmäßig. So sind beispielsweise in der Deutschen Sprache die Buchstaben `N` und `E` viel häufiger als die Buchstaben `X` und `Y`. Auch für Kombinationen von Buchstaben gibt es große Abweichungen. So erscheint z.B. nach einem `Q` fast immer ein `U`. Kodiert man `Q` binär, dann wird der Pfad, der zu `U` führt, zu fast alle Daten führen. Würde man die Peers also wahllos auf die Teilbäume verteilen, so würden die Peers, die in diesem Teilbaum sind, ein deutlich höhere Last haben.

Daher strebt P-Grid an, die Anzahl der Peers in einem Teilbaum proportional zu den dort gespeicherten Daten zu verteilen. Wenn also das Verhältnis $r = m_0/m_1$ der Datenmenge zweier benachbarter Teilbäume bekannt ist, dann kann man Peers gemäß der Wahrscheinlichkeit p, die sich aus der Gleichung $\frac{p}{1-p} = r$ ergibt, auf den linken Teilbaum verteilen und mit der Gegenwahrscheinlichkeit auf den rechten. Beim Einfügen muss nur noch beachtet werden, dass der andere Teilbaum nicht leer ist. Hierzu muss jeder Peer mindestens einen Peer aus dem anderen Teilbaum fin-

den. Dieser Prozess wird durch das so genannnte *Boot-Strapping* gelöst, welches wir gleich besprechen.

Da Peers nur in den Blättern des Netzwerkbaums liegen, sind die Pfadlängen des Baumes dynamisch. Im Idealfall ist die Anzahl der Peers proportional zu der Anzahl der Dateneinträge. Das wird durch P-Grid nicht garantiert. Außerdem müssen sämtliche Peers fortwährend überprüfen, inwieweit sich die Baumstruktur geändert hat. Um die Balancierung zu sichern, wird außerdem eine kontinuierliche Rebalancierung durchgeführt. Hierzu hat jeder Peer eine Minimal- und eine Maximalauslastung. Peers, die überlastet sind, versuchen die Daten auf den Nachbarbaum abzuwälzen. Wie bei der Defragmentierung von CAN wird dadurch der Baum an dieser Stelle kürzer.

Die freien oder freigewordenen Peers suchen fortwährend nach überlasteten Peers. Ist ein solcher gefunden worden, so teilen sie bei diesem Peer den Baum weiter auf.

8.1.2 Einfügen von Peers

Beim Einfügen von Peers entscheidet jeder Peer selbstständig, in welchen der Teilbäume er sich einordnet. Hierzu initiiert er eine Interaktion mit einem zufällig ausgewählten Peer (den er aus der Routing-Tabelle des in der Baumstruktur gelegenen nächsthöheren Peers erhalten hat). Mit Hilfe dieses Peers erhält er evtl. weitere Zeiger auf Peers in diesem Teilbaum. Nun muss der Peer entscheiden, in welchem Teilbaum er eingesetzt wird. Hierzu kann er die Position der Teilbäume, die auf den Peers gespeicherte Datenmenge und deren Zustand verwenden, der anzeigt, ob sie möglicherweise auch gerade einen Teilbaum suchen.

Eine optimale Entscheidung kann nicht garantiert werden. Die Autoren von P-Grid untersuchten für diese Entscheidung verschiedene Heuristiken. Ist die Entscheidung für einen Teilbaum gefallen, so kennt der Peer mindestens einen Knoten aus diesem Teilbaum aus der vorangegangenen Routing-Tabelle (wenn nicht, muss er annehmen, dass er der erste Knoten in diesem Teilbaum ist).

Die Entwickler haben sich ausführlich mit dem Fall befasst, wenn viele Peers das Netzwerk simultan betreten. Dann entsteht das Problem, dass sich sehr viele Peers eines Teilbaums gleichzeitig entscheiden müssen. Dadurch kann es sein, dass ein Peer nur andere Peers kennt, die sich nicht entschieden haben. Wartet nun jeder Peer, bis sich der andere entschieden hat, kann sich der Netzwerkaufbau stark verzögern. Daher haben die Autoren für diesen Fall verschiedene heuristische Strategien untersucht und experimentell verglichen.

8.1.3 Suche

In der Routing-Tabelle eines Peers steht für jeden benachbarten Teilbaum die Adresse eines geeigneten Peers. Betrachtet man den identifizierenden Bitstring, so steht jeder Teilbaum für einen möglichen Präfix. Damit gleicht die Routing-Tabelle der von Pastry und Tapestry. Der entscheidende Unterschied ist, dass Pastry und Tapestry auf verteilten Hash-Tabellen beruhen. Damit ist die Anzahl der Daten in jedem

Präfix durch das Vermischen der Hash-Funktion einigermaßen gleichmäßig balanciert.

Bei P-Grid ist genau das nicht der Fall, da die Daten nicht balanciert werden. Nur die Knoten werden gemäß der Datenanzahl auf die Teilbäume verteilt. Dadurch können Teilbäume beliebig tief werden. Hierbei wäre nun zu befürchten, dass die Suche bei solch ungleichmäßig balancierten Suchbäumen ineffizient werden könnte. Tatsächlich hilft die Balancierung der Peers, die Suche zu beschleunigen.

Die Suche nach einem Datum läuft wie folgt ab: Passend zu dem Datum wählt man das Präfix aus der Routing-Tabelle. Die Suchanfrage wird an den passenden Peer weitergeleitet. So führt die Suche in dem Suchbaum immer weiter herab, bis das Datum gefunden wurde.

Theorem 8.1. *Die erwartete Anzahl an Hops bei der Suche in P-Grid ist $\mathcal{O}(\log n)$, wenn n die Anzahl der Peers ist und die Anzahl der Peers in jedem Teilbaum proportional zu der Anzahl der Daten ist.*

Beweis. Der aktuelle Teilbaum der Suche ist der kleinste Teilbaum, der den aktuellen Peer und das gesuchte Datum enthält. Wir betrachten die Suche nach einem Datum in Phasen. Ein Phasenwechsel tritt dann auf, wenn die Anzahl der Daten im aktuellen Teilbaum halbiert wurde. Ferner gehen wir davon aus, dass in der Routing-Tabelle aller Peers jeder Knoten des entsprechenden Teilbaumes gleichwahrscheinlich vorkommt.

Zwar können die Bäume eine Tiefe von bis zu $n-1$ erreichen, aber jeder Teilbaum (mit dem Datum), der mehr als die Hälfte der Peers besitzt, kann mit nur zwei Schritten erreicht werden. Aus der Markovschen Ungleichung (siehe Anhang Seite 268) folgt, dass die Wahrscheinlichkeit, in einem Schritt in diesem Baum zu gelangen, mindestens $\frac{1}{2}$ ist. Scheitert dieser Schritt, dann erreicht man diesen Teilbaum im nächsten Schritt wieder mit der Wahrscheinlichkeit $\frac{1}{2}$ und so weiter. (Das gilt nur unter der Annahme, dass jeder Knoten eine unabhängige gleichwahrscheinliche Wahl des Peers aus den Nachbarbäumen trifft.) Die erwartete Anzahl von Schritten ist daher höchstens

$$\sum_{k=1}^{n} k \left(\frac{1}{2}\right)^k \leq 2 .$$

Wir betrachten also den kleinsten Teilbaum, der den Suchbaum enthält und nicht die Größe halbiert. Diesen erreichen wir in einer erwarteten Schrittzahl von zwei. Der nächste Teilbaum in der Suche ist damit kleiner als die Hälfte der ursprünglichen Datenzahl. Somit tritt erwartungsgemäß nach spätestens drei Schritten ein Phasenwechsel ein.

Wenn m Daten im Baum gespeichert sind, dann endet die Suche, wenn ein Baum der Größe m/n erreicht wurde unter der Annahme, dass die Heuristiken das Datenvolumen gleichmäßig verteilen. Daher ergibt sich eine erwartete Suchzahl von $3 \log \frac{m}{m/n} = 3 \log n$. □

8.1.4 Weitere Eigenschaften

Neben der geordneten Sortierung und der effizienten Suche hat P-Grid noch folgende Eigenschaften.

- *Lokale Lastbalancierung*
  In P-Grid werden die Peers fortwährend umbalanciert. Wird ein Peer unterfordert, dann übergibt er die Last einem benachbarten Peer und versucht einen Peer im Baum zu finden, der überlastet ist. Die Baumstruktur passt sich so ständig der Datenlast an. Als Information stehen den Peers hierfür nur die Kontaktpartner während der Suche im Baum zur Verfügung.
- *Dynamische Adressen*
  Durch die Baumstruktur erhält jeder Peer dynamische Adressen, die auch außerhalb der Suche für andere Zwecke verwendet werden können.
- *Dezentrales Trust-Management*
  Ein weiterer Bestandteil von P-Grid ist ein Vertrauensmanagementsystem (*Trust-Management*). Dieses wird notwendig, wenn Angreifer versuchen, die Struktur von Peer-to-Peer-Netzwerken zu zerstören. Die Autoren von P-Grid haben sich schon sehr früh mit diesem Problem beschäftigt, das, wie wir noch in Kapitel 10 sehen werden, sehr schwierig ist.
- *Epidemische Informationsverbreitung*
  Bestimmte Informationen, wie die lokale Baumstruktur oder das Fehlen von Teilbäumen, müssen in P-Grid an alle Knoten verteilt werden. Hierfür wird *epidemische Informationsverbreitung* [49] genutzt, auch bekannt unter dem Namen *Rumor Spreading*. Dabei werden die Daten von informierten (infizierten) Knoten für eine bestimmte Zeit an Kontaktpartner weitergegeben. Ähnlich wie eine ansteckende Krankheit breitet sich so die Information im gesamten betroffenen Teilbaum aus. Der Vorteil solcher Verfahren besteht darin, dass keine eigenen Datenstrukturen für diese Informationsverbreitung aufgebaut werden müssen und dass dennoch alle Knoten von der Information angesteckt werden können. So wird eine konsistente Sicht auf den Baum erzeugt.

8.1.5 Zusammenfassung

P-Grid ist ein Peer-to-Peer-Netzwerk, das die Autoren mehrfach in Hinsicht auf seine praktische Einsatzfähigkeit optimiert haben. Es ist das erste Peer-to-Peer-Netzwerk, das Daten geordnet speichert und zugleich effizient arbeitet. Mit P-Grid kann man effizient Bereichsanfragen durchführen. Hierzu sucht man den ersten Peer und kann dann durch eine Baumtraversierung die Daten aus dem Baum auslesen.

P-Grid bietet die Möglichkeit der dynamischen Lastanpassung und verteilt die Daten aktiv auf die Peers. In dieser Hinsicht ist P-Grid sicher *Skip-Net* überlegen, welches wir gleich besprechen werden.

Auf der negativen Seite hat P-Grid Schwierigkeiten, den Verlust von Peers und damit von Informationen in Teilbäumen zu kompensieren. Dieses Risiko wurde insofern abgemildert, da P-Grid es erlaubt, mehrere Peers auf Teilbäume zuzuweisen. Die

Mechanismen zur (Wieder-)Vereinigung von Teilbäumen sind aber (noch) nicht spezifiziert. In P-Grid wurde sehr viel Energie auf die Frage verwendet, wie man einen Massenansturm auf das Peer-to-Peer-Netzwerk handhaben kann. Ob dies wirklich ein nennenswertes Problem ist, erscheint fraglich. Schließlich ist ein solcher Massenansturm nur mit einer zentralen Kontrolle möglich. Es muss aber erwähnt werden, dass P-Grid ohne die erwähnten Heuristiken mit diesem Ansturm nicht fertig werden würde. Somit ist es durchaus legitim, P-Grid gegen diesen (wenn auch unrealistischen) Fall zu wappnen.

8.2 Skip-Net

Von James Aspnes und Gauri Shah wurde im Jahr 2003 *Skip-Graph* [50] als effizientes Peer-to-Peer-Netzwerk eingeführt. Gleichzeitig und unabhängig haben Harvey, Jones, Saroiu, Theimer und Wolman das weitgehend äquivalente Peer-to-Peer-Netzwerk *Skip-Net* [51] entwickelt. Die Begriffe *Skip-Net* und *Skip-Graph* werden zumeist synonym verwendet. Hier werden diese Begriffe verwendet, um das Peer-to-Peer-Netzwerk (Skip-Net) von der zugrunde liegenden Verbindungsstruktur (Skip-Graph) zu unterscheiden. Da von Anfang an die Rechner-Netzwerk-Struktur im Vordergrund stand, verwenden wir hier den Begriff Skip-Net für das Peer-to-Peer-Netzwerk und den Begriff Skip-Graph, wenn wir uns auf die Graphstruktur des Netzwerks beziehen.

Ziel von Skip-Net ist die sortierte Verwaltung von Daten und Peers, so dass neben der einfachen Suche auch kompliziertere Anfragen wie zum Beispiel Bereichsanfragen möglich werden. Der Name (oder der Schlüssel) eines Peers legt fest, welche Daten dieser verwalten wird. Wurde der für ein Datum zuständige Peer gefunden, so wird der Rest der Anfrage durch eine konventionelle Datenstruktur (wie zum Beispiel durch balancierte Suchbäume) lokal auf dem Peer gelöst.

8.2.1 Skip-Listen

Um die Struktur eines Skip-Graphen zu verstehen, hilft es *Skip-Listen* zu kennen. Diese stellen eine Möglichkeit dar, Suchbäume effizient zu implementieren. Zur Veranschaulichung startet man mit einer sortierten und einfach verketteten linearen Liste, siehe Abbildung 8.4. In solch einer Liste ist die Suchzeit im schlimmsten Fall $n - 1$, wenn n Daten vorhanden sind.

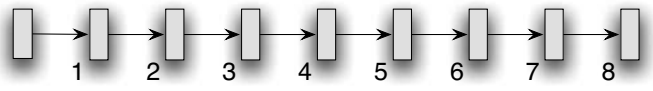


Abb. 8.4. Eine einfach verkettete Liste.

Nun werden abkürzende Zeiger hinzugefügt, indem jeder Knoten per Münzwurf ausgewählt wird. Die ausgewählten Knoten werden wieder als lineare Liste verkettet. Dadurch hat sich der Suchweg im Erwartungswert halbiert, wobei die erwartete Anzahl übersprungener Knoten Eins ist. Unter den ausgewählten Knoten wird dieses Verfahren jetzt wiederholt, bis nur noch ein Knoten übrig bleibt, siehe Abbildung 8.5.

Zur Analyse werden die Knoten der ersten Liste als Ebene 0 bezeichnet. Knoten, die einmal ausgewählt worden sind, befinden sich in Ebene 1. So befindet sich jeder Knoten, der k-mal ausgewählt wurde, in der Liste der Ebene k. Der resultierende Graph ist die Skip-Liste. Man kann sich leicht überlegen, dass mit hoher Wahrscheinlichkeit, d.h. $1 - n^{-c}$, die Höhe der obersten Ebene höchstens $\mathcal{O}(\log n)$ beträgt.

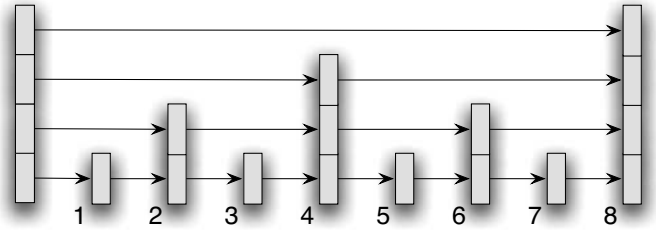


Abb. 8.5. Die Skip-Liste.

Wenn man in einer Skip-Liste sucht, dann fängt man beim vordersten Knoten in der obersten Ebene an und überprüft, ob der angezeigte Knoten hinter dem gesuchten Datum liegt. Ist das nicht der Fall, dann folgt man den Zeigern der Ebene solange man das Ziel nicht überschreitet. Ist das der Fall, verringert man die Höhe und wiederholt das Verfahren in der nächsttieferen Ebene. Da man das Datum nie überspringt, wird man spätestens in der Ebene 0 erfolgreich sein.

Dieser Suchalgorithmus ist sehr effizient: Man kann zeigen, dass in jeder Ebene erwartungsgemäß höchstens zwei Schritte durchgeführt werden. Somit kann in erwarteter Suchzeit von $\mathcal{O}(\log n)$ jedes Datum gefunden werden. Außerdem ist der erwartete zusätzliche Platzverbrauch für diese Datenstruktur $\mathcal{O}(n)$. Ferner ist sowohl der Eingrad als auch der Ausgrad des Graphen mit hoher Wahrscheinlichkeit $\mathcal{O}(\log n)$.

Statt die Höhe der Elemente zufällig zu wählen, kann man auch einen deterministischen Algorithmus verwenden. Dieser weist jedem Element eine Höhe zu, die der Anzahl der Nullen am Ende der Binärdarstellung der Ordnungszahl entspricht. Im Gegensatz zu der randomisierten Wahl kann man diese Darstellung beim Einfügen neuer Datenelemente jedoch nicht aufrecht erhalten. Bei der randomisierten Wahl werden für das neue Element einfach Münzwürfe wiederholt, bis die Höhe bestimmt worden ist, und dann wird das Element entsprechend eingefügt.

8.2.2 Skip-Graph

Eine Skip-Liste eignet sich nicht als Verbindungsstruktur für ein Peer-to-Peer-Netzwerk, da der Verlust derjenigen Peers, die Knoten großer Höhe entsprechen, unweigerlich zur Partitionierung des Netzwerks führen würde. Es wird also eine Datenstruktur angestrebt, die, ähnlich wie Skip-Listen, eine effiziente Suche ermöglicht, aber keine strukturellen Schwachpunkte erzeugt.

Hierzu stellt man sich vor, dass die Peers die Rolle der Daten aus der einfach verketteten Liste wahrnehmen. Diese ist hier doppelt verkettet, die Ordnung bleibt aber erhalten. Wie bei der Skip-Liste wirft man jetzt für jeden Peer eine Münze mit Ergebnis 0 oder 1. Die Gewinner bilden wieder eine neue sortierte Liste, siehe Abbildung 8.6. Nun erlaubt man aber auch den Verlierern, eine eigene sortierte Liste zu bilden. In den beiden disjunkten Listen dieser Ebene wird dieses Verfahren jetzt iteriert, bis nur noch einzelne Peers übrigbleiben, siehe Abbildung 8.7.

Im ursprünglichen Artikel [50] waren die einzelnen Listen, ähnlich wie die Skip-Liste, offen. Seit [51] werden das erste und das letzte Element der einzelnen Listen verbunden, da dadurch die Robustheit der Liste ohne Zusatzaufwand erhöht wird. Es entsteht die baumförmige Struktur aus der Abbildung 8.8.

In Skip-Graphen gibt es nun zwei Methoden der Adressierung: Den *Name-ID* genannten Schlüssel eines Peers, welcher die Sortierung in der einfachen Liste definiert, und die Folge der Zufallsbits *Num-ID* (*Numeric Identification*), welche die Zugehörigkeit in den einzelnen Teilmengen angeben. Man kann sich vorstellen, dass diese Folge länger als die Höhe des entsprechenden Teilzweigs ist.

Die Höhe des Skip-Graphen ist meist größer als die der entsprechenden Skip-Liste. Das asymptotische Wachstum ist dennoch gleich: Die Höhe ist mit hoher Wahrscheinlichkeit $\mathcal{O}(\log n)$. Hierzu überlegt man sich, dass zwei Peers in Höhe h nur dann gleichzeitig vorhanden sein können, wenn deren Num-IDs einen gemeinsamen Präfix der Länge h haben. Dies geschieht mit Wahrscheinlichkeit 2^{-h}. Legt man nun eine Präfixlänge von $h = (c+2)\log n$ zugrunde, so ist die Wahrscheinlichkeit, dass zwei Peers im gemeinsamen Teilbaum der Tiefe h landen, höchstens $\frac{1}{n^2} \cdot \frac{1}{n^c}$. Da maximal $\binom{n}{2} \leq n^2$ dieser Kombinationen zu betrachten sind, kommt dieser Fall mit einer Wahrscheinlichkeit von höchstens $\frac{1}{n^c}$ vor.

Daraus folgt, dass der Eingrad und der Ausgrad eines Skip-Graphen mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log n)$ ist. Damit ist auch der Durchmesser maximal $\mathcal{O}(\log n)$. Skip-Graphen sind außerdem sehr robust gegen Knotenausfälle (*highly resilient*). Das folgt insbesondere aus der Expandereigenschaft von Skip-Graphen [52].

Skip-Net

Wie bereits erwähnt, unterscheiden wir hier Skip-Graphen und Skip-Nets nur darin, ob wir den Graphen oder das zugehörige Peer-to-Peer-Netzwerk betrachten. Die Knoten des Skip-Nets sind die Peers. Kanten stehen in der Routing-Tabelle der Peers. Jeder Peer erhält so zwei Adressen: Die *Name-ID* und die *Num-ID*.

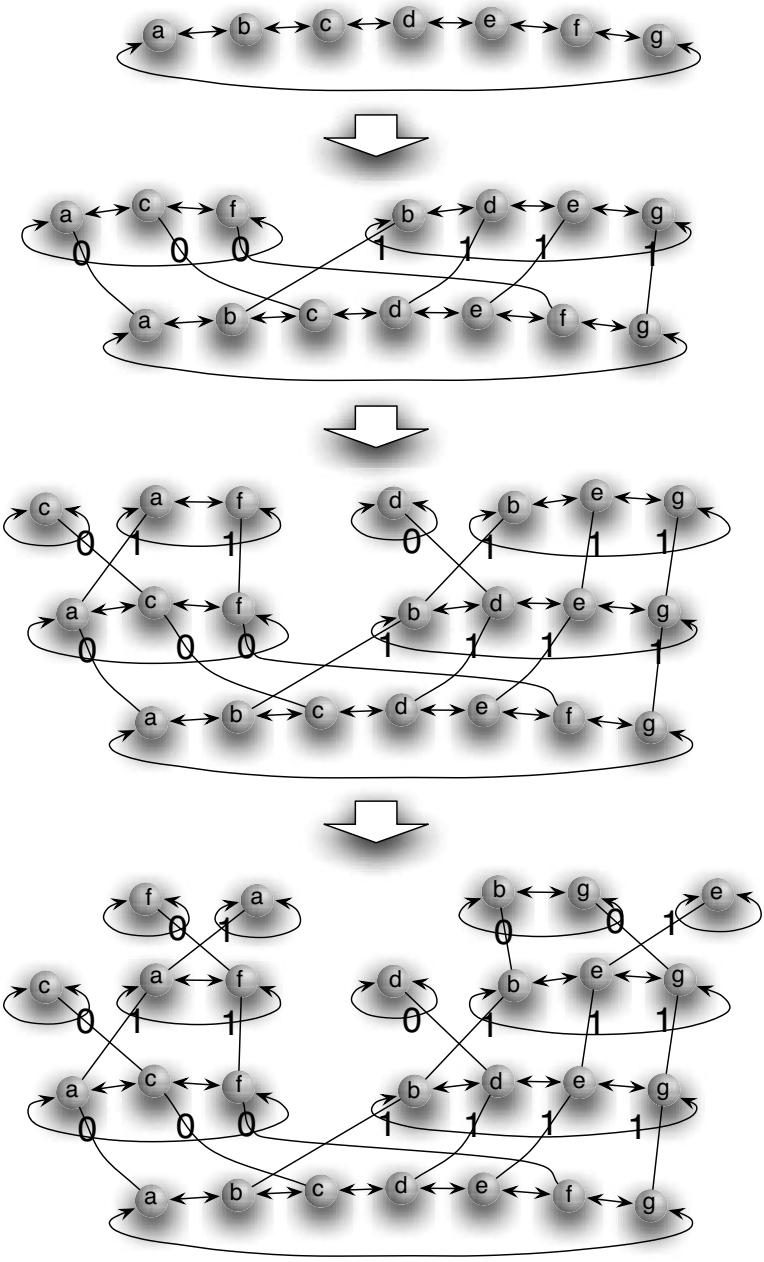


Abb. 8.6. Aufbau eines Skip-Graphen.

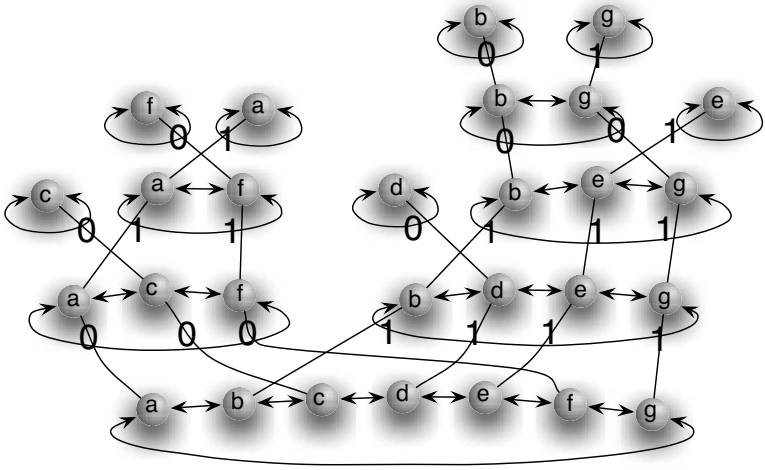


Abb. 8.7. Ein vollständiger Skip-Graph.

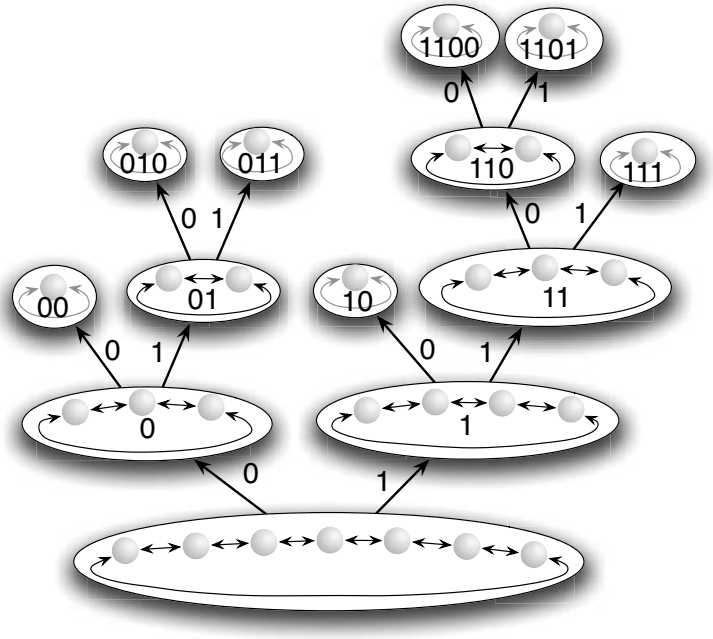


Abb. 8.8. Baumförmige Struktur eines Skip-Graphen.

Die Name-ID ist eine eindeutige Identifikation des Peers, wobei hier zum Beispiel eine inhaltliche Klassifikation vorgenommen werden kann. Zum Beispiel könnte das

```
com.microsoft.research.john
```

sein, in Anlehnung an den Hostnamen

```
john.research.microsoft.com.
```

Dieser Name stünde dann zwischen

`com.microsoft.research.jack` und `com.microsoft.research.june`.

Durch diese Art der Darstellung werden Peers, die sich in gleichen lokalen Domains oder Netzwerken befinden, nahe zueinander angeordnet. Durch die Art der Namenswahl kann man sehr einfach die Lokalität des Netzwerks steuern. Diese Steuerung ist aber nicht Bestandteil von Skip-Graph und muss von den Betreibern des Netzwerks beigesteuert werden.

Die Num-ID wird durch den Zufallsprozess des Skip-Graphen erzeugt. Wir erinnern uns, dass für jeden Knoten eine Folge von Zufallsbits erzeugt wird, bis diese sich unterscheiden. Diese Num-ID dient jetzt als alternative Adresse (zu der IP-Adresse und der Name-ID). Im Gegensatz zur Name-ID sind die Peers hier nämlich nämlich einigermaßen gleichverteilt (im Rahmen des Zufallsprozesses).

8.2.3 Suche im Skip-Graphen

Skip-Graphen erlauben sowohl die effiziente Suche nach den Name-IDs als auch den Num-IDs.

Suche nach Peer-Namen (Name-ID). Diese Suche läuft analog zur Suche in einer Skip-Liste ab. Wir starten von einem Peer des Netzwerks und wählen die höchste Ebene, in der ein Zeiger nicht das Ziel überspringt. Die Suche wird dann in dem verlinkten Peer fortgesetzt. Wieder wird die höchste Ebene gewählt, in welcher der gesuchte Peer nicht übersprungen wird. Das folgende Theorem liefert eine Aussage über die Anzahl der erforderlichen Schritte.

Theorem 8.2. *Für die Suche nach einer Name-ID werden mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log n)$ Schritte benötigt.*

Beweis. Der Beweis dieses Theorems folgt unmittelbar aus der Tatsache, dass in Skip-Listen in logarithmischer Zeit mit hoher Wahrscheinlichkeit gesucht werden kann. □

Suche nach (numerischer) Knoten-ID (Num-ID). Jedem Peer wird eine eindeutige Bitfolge, die Num-ID, zugeordnet. Diese besteht aus einer zufälligen Folge von Bits. Nach dieser numerischen ID kann ebenfalls effizient gesucht werden. Mit Hilfe dieser Suche kann man in Skip-Net auch verteilte Hash-Tabellen (DHT) implementieren.

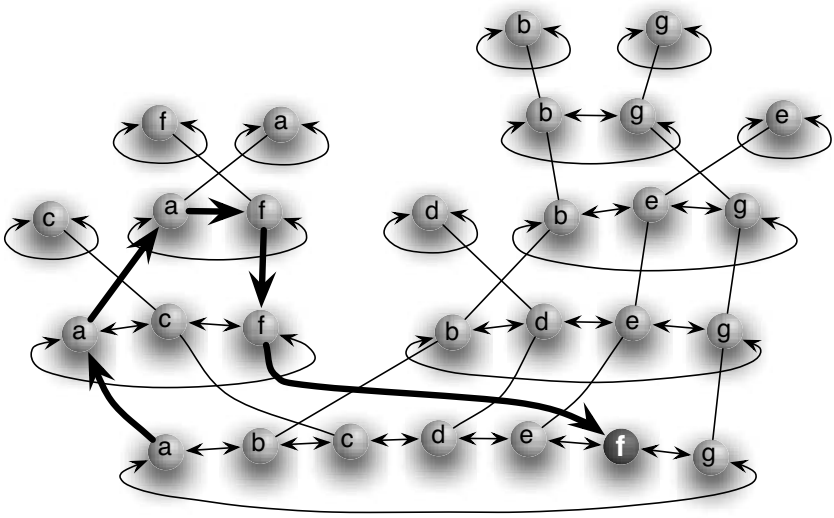


Abb. 8.9. Suche nach dem Schlüsselwort **f** (Name-ID) in Skip-Net

Hierzu startet man die Suche auf dem untersten Ring. Ein Peer sucht in der Ebene i nach einem Peer, der an der $(i+1)$-ten Stelle dasselbe Bit wie die gesuchte ID besitzt. Dies geschieht durch lineares Ablaufen des Rings. Ist das gewünschte Bit gefunden, wechselt man zum nächsthöheren Ring. Dort wird die Suche mit einem um Eins erhöhten i fortgesetzt. Wurde ein Ring erfolglos umlaufen, endet die Suche und zu mindestens wurden alle Peers mit dem längsten gemeinsamen Präfix von Num-ID gefunden.

Theorem 8.3. *Für die Suche nach einer Num-ID werden mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log n)$ Schritte benötigt.*

Beweis. Als erstes beobachten wir, dass es mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log n)$ Ebenen gibt. In jeder Ebene ist die Wahrscheinlichkeit, dass die Suche schon nach dem nächsten Schritt endet $\frac{1}{2}$. Damit beobachten wir $\mathcal{O}(\log n)$ Bernoulli-Experimente. Aus der Chernoff-Schranke, siehe Seite 268, folgt das Theorem. □

Diese Analysen beruhen auf der zufälligen Wahl der Num-IDs. Wir sehen also, dass die Sortierung der Peers keine Einschränkung bei der Suche ergibt.

8.2.4 Einfügen von Peers

Folgender Algorithmus fügt Peers in die Datenstruktur ein. Zuerst wird der korrekte Ort gemäß der Name-ID des Peers gesucht. Dann wird der Peer in den Ring eingefügt. Nun würfelt der Peer die Num-ID und fügt sich rekursiv in die passenden höheren Ebenen ein. Um die nächsthöhere Ebene zu finden, müssen von dem Peer

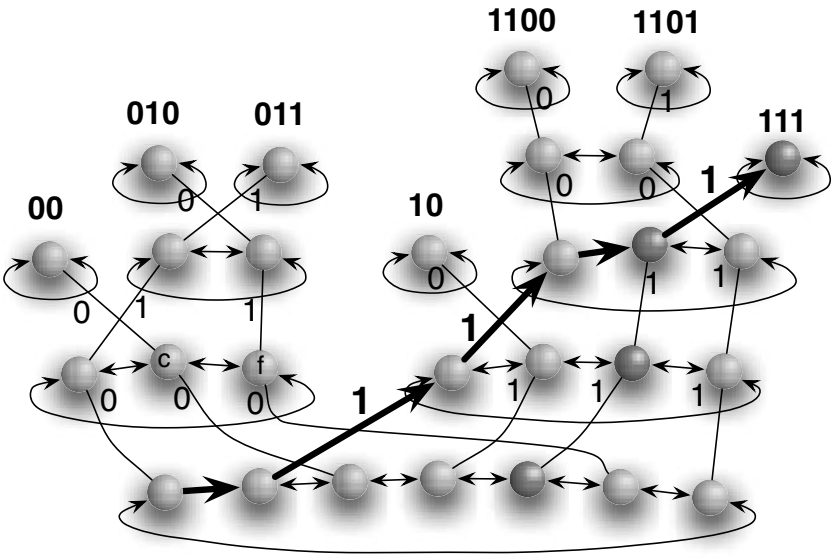


Abb. 8.10. Suche nach der Num-ID `111` in Skip-Net.

aus erwartungsgemäß zwei Nachbarpeers abgefragt werden. Wie wir bereits gesehen haben, gibt es mit hoher Wahrscheinlichkeit höchstens $\mathcal{O}(\log n)$ Ebenen. Daher kann man durch Anwenden der Chernoff-Schranke wieder folgern, dass das Einfügen eines Peers höchstens $\mathcal{O}(\log n)$ Operationen benötigt.

Theorem 8.4. *Das Einfügen eines Peers in Skip-Net benötigt mit hoher Wahrscheinlichkeit $\mathcal{O}(\log n)$ Nachrichten.*

Der Beweis folgt unmittelbar aus den vorangestellten Überlegungen.

Bereichsanfragen

Bereichsanfragen können in Skip-Net bezüglich Num-IDs und Name-IDs durchgeführt werden. Für eine Bereichsanfrage sucht man nach allen Peers, die sich in einem gegebenen Intervall befinden.

Sucht man einen bestimmten Bereich in den Num-IDs, so findet man die Lösung in den Blättern der Struktur. Hierzu sucht man nach entsprechenden Peers am Intervallanfang und Intervallende. Dann kann man sich in der Ringstruktur von dort durch das Intervall bewegen. Die Ziel-Peers befinden sich in einer zusammenhängenden Menge von Teilbäumen.

Man kann aber auch eine Bereichsanfrage bezüglich der Name-IDs stellen. Genau wie zuvor sucht man zuerst den Intervallanfang und das Intervallende. Die gesuchte Menge von Peers befindet sich nun auf einem Kreisausschnitt auf dem

untersten Ring (*Root Ring*). Auch hier kann man mit Hilfe der oberen Ringe im gewünschten Bereich effizient suchen.

Beide Bereichsanfragen können die Intervallgrenzen mit hoher Wahrscheinlichkeit in Zeit $\mathcal{O}(\log n)$ bestimmen. Das sukzessive Durchwandern der Peers dauert bei der Num-ID-Bereichssuche pro Num-ID höchstens $\mathcal{O}(\log n)$ Schritte, im Erwartungswert aber nur konstant viele Schritte. Da bei der Name-ID-Bereichssuche ein Abschnitt auf dem untersten Ring alle gesuchten Peers beinhaltet, ist dort das Durchwandern aller Peers in jeweils einem Schritt möglich.

Lokalität in Skip-Net

Wählt man die Name-IDs so, dass sie die Lokalität der Peers widerspiegeln, so kann man das effizient ausnutzen. Beipielsweise kann die Name-ID die DNS-Adresse eines Rechners, z.B. `john.microsoft.com`, `jack.microsoft.com`, in eine Form umsetzen, welche Rechner in einem lokalen Netzwerk benachbart belässt, also z.B. `com.microsoft.john` und `com.microsoft.jack`. Damit verbleibt die Suche nach einem Schlüsselwort (Name-ID) im lokalen Netzwerk, da dieses lokale Netzwerk einen Bereich im untersten Ring und entsprechende Teilbereiche in den oberen Ringen einnimmt.

Üblicherweise werden von den Enwicklern von Peer-to-Peer-Netzwerken nur singuläre Ausfälle betrachtet. Es kommt aber durchaus vor, dass ganze Teilnetzwerke ausfallen, weil entscheidende Netzwerkverbindungen verloren gehen. Verwendet man verteilte Hash-Tabellen, so sind die Ausfälle gleichmäßig über das ganze Netzwerk verteilt und können verkraftet werden, solange das ausfallende Teilnetzwerk nicht zu groß ist.

In Skip-Net kann man das Netzwerk sehr schnell nach einem solchen Ausfall reparieren. Hierzu müssen die Kanten benachbarter Peers in den Ringen jeweils mit gegenläufigen Zeigern belegt werden. Um den verlorenen Ringausschnitt zu reparieren, werden die Abkürzungen in den höheren Schichten betrachtet. Von diesen ausgehend, kann man die Zeiger in den unteren Schichten rekonstruieren. Dieser Reparaturvorgang dauert nur $\mathcal{O}(\log n)$ Schritte.

8.2.5 Deterministisches Skip-Net

Die zufällige Wahl der Num-ID kann lokal zu Effizienzverlusten führen. So kommt es immer wieder vor, dass in einem Ring benachbarte Peers die gleichen Münzwürfe erhalten und so in denselben Unterring kommen. Dies sorgt speziell bei der Suche zu Effizienzverlusten, da man nun im Ring länger suchen muss. Man kann sich überlegen, dass im unteren Ring mit Wahrscheinlichkeit von $1 - o(1)$ solche Ketten bis zur Länge $(1 - o(1)) \log n$ vorkommen können.

In [53] wird ein deterministischer Rebalancierungsmechanismus vorgestellt, der diesen Fall löst. Hierzu wird in einer zu langen Kette von fünf benachbarten Peers, die alle in den gleichen Ring fallen, der mittlere herausgegriffen und in den anderen Ring umbalanciert. Diese Operation nennt man *Rotation*. Sie ist in Abbildung 8.11

dargestellt. Dort wird der Peer dann rekursiv weiter einsortiert. Für solch ein deterministisches Skip-Net gelten die hier gezeigten Schranken nicht nur mit hoher Wahrscheinlichkeit, sondern absolut.

Hierzu überlegt man sich, dass der linke höchstens viermal so groß ist wie der rechte Teilbaum (und umgekehrt). Damit ist die Tiefe des deterministischen Skip-Graphen höchstens $\log_{\frac{4}{3}} n$. Verwendet man diese Operation auch zum Netzwerkaufbau, so erhält man ein Skip-Net ganz ohne Zufall und ohne entsprechende randomisierte Analysen.

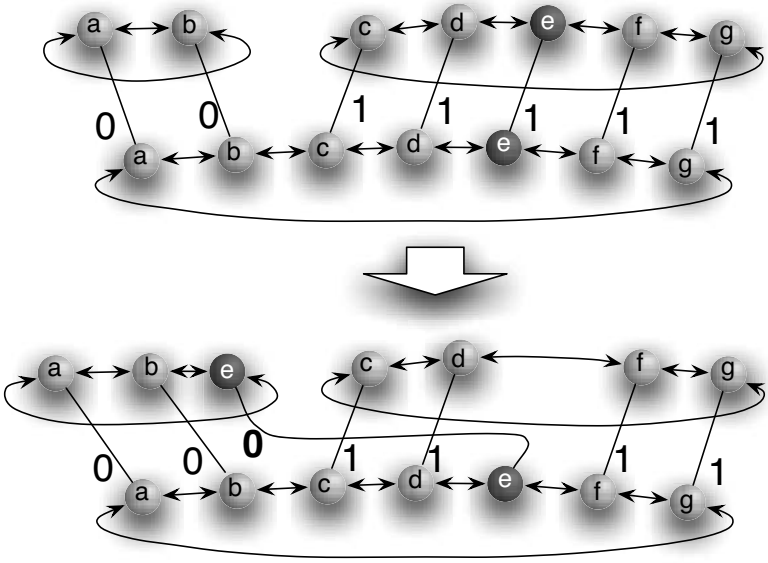


Abb. 8.11. Rotationsoperation für eine deterministische Version von Skip-Net.

8.2.6 Zusammenfassung

Skip-Net stellt eine effiziente und einfache Datenstruktur zur sortierten Speicherung von Daten oder Indizes zur Verfügung. Die Voraussetzung hierbei ist, dass jeder Peer einen zusammenhängenden disjunkten Datenbereich verwaltet. Die Zuordnung dieser Daten und eine mögliche Rebalancierung sind nicht Gegenstand der Betrachtung innerhalb von Skip-Net.

Skip-Net ist skalierbar und besitzt logarithmischen Grad, Durchmesser und eine effiziente Suchzeit für Bereichsanfragen. Ausfälle von Teilnetzwerken können in logarithmischer Zeit repariert werden. Außerdem werden in [50] sehr ausführlich Reparaturmechanismen diskutiert, die in einem so genannten Reißverschlussverfahren (*Zipper*) das Netzwerk reparieren, wenn hier Störungen vorliegen.

Das Routing kann die Lokalität, die durch den Basisring (*Root Ring*) vorgegeben wird, vollständig ausnutzen. Außerdem werden interne IDs vergeben (Num-ID), welche die Verwendung von verteilten Hash-Tabellen (DHT) ermöglichen. Solch eine zusätzliche Indizierung ist aber nicht notwendig.

9

Selbstorganisation

Mein Pferd und ich wären hoffnungslos versunken, wenn ich es nicht geschafft hätte, mich an meinem eigenen Haarschopf aus dem Sumpf zu ziehen.
Karl Friedrich Hieronymus Freiherr von Münchhausen
nacherzählt von Martina Weier

Den Begriff der *Selbstorganisation* verwendet man hauptsächlich in der *Systemtheorie*. *Selbstorganisierende Systeme* haben dort nach allgemeiner Ansicht die folgenden Merkmale: *Komplexität*, *Selbstreferenz*, *Redundanz* und *Autonomie*. Im Gegensatz zur üblichen Verwendung des Komplexitätbegriffs in der Informatik (vgl. Komplexitätsklassen oder Beschreibungs-/Kolmogoroff-Komplexität) versteht man in der Systemtheorie folgendes unter komplexen Systemen:

> Systeme sind *komplex*, wenn ihre Teileinheiten dynamisch vernetzt sind. Sogar die Teile selbst können sich jederzeit verändern. Solche komplexen Systeme sind im Allgemeinen schwer beschreibbar. *Selbstreferenz* bezeichnet die Rückwirkung des Systems auf sich selbst. Das Verhalten des Systems verändert das System. Unter *Redundanz* versteht man die Einheit aus organisierender, gestaltender und lenkender Funktionalität. Jeder Teil ist potenziell auch Gestalter. *Autonomie* bezeichnet die Selbstbestimmtheit des Systems. Ohne fremde Einflüsse erweist sich das System als handlungs- und erhaltungsfähig.

Mit diesen Begriffen fällt es sehr schwer, den Bereich der Peer-to-Peer-Netzwerke zu fassen. So kann ein Peer-to-Peer-Netzwerk nur dann als komplex erachtet werden, wenn mindestens einige hundert Teilnehmer vorhanden sind. Damit wäre dies ein Begriff, der nur real existierende Peer-to-Peer-Netzwerke beschreibt. Man würde aber schwerlich ein Peer-to-Peer-Netzwerk als nicht selbstorganisierend bezeichnen, nur weil es sich in einem frühen Zustand befindet. Der Begriff der Selbstreferenz ist im Allgemeinen schwierig anzuwenden. Sicher verweisen Peers auf andere Peers und diese wieder zurück auf die ursprünglichen Peers. Handelt es sich hierbei aber schon um Selbstreferenz? Genauso verhält es sich mit dem Begriff der

Redundanz, der nicht mit dem gleich lautenden Begriff in der Informatik verwechselt werden darf. Sicher hat jeder Peer in einem wahren Peer-to-Peer-Netzwerk die gleichen Aufgaben. Die Organisation (Netzwerkaufbau), Gestaltung (Suche, Speicherung von Indexdateien) und Lenkung (Umbalancierung von Daten) sind aber in verschiedenen Protokollen mitunter deterministisch und sehr kontrolliert umgesetzt. Zuletzt wirft auch der Begriff der Autonomie Fragen auf. In der Regel werden Peer-to-Peer-Netzwerke weder ohne die Infrastruktur des Internets noch ohne die Aufgabenstellungen aus der Anwendungsschicht existieren oder fortbestehen können.

Wir sehen an dieser Diskussion, dass der biologisch geprägte Begriff der Selbstorganisation nur schwerlich auf den Bereich der Peer-to-Peer-Netzwerke angewendet werden kann. Andererseits entwickeln Peer-to-Peer-Netzwerke ein gewisses Eigenleben und es entstehen in Peer-to-Peer-Netzwerken Strukturen, die am besten durch das Prinzip der Selbstorganisation beschrieben werden können. In diesem Kapitel wollen wir den Begriff der Selbstorganisation an drei Beispielen kennenlernen, die zeigen, wie eine bestimmte Netzwerkstruktur aus kleinen lokalen Operationen entsteht. Man spricht in diesem Zusammenhang auch von *Emergenz*.

Die drei ausgewählten Beispiele sind die so genannten *Pareto-verteilten Netzwerke*, *selbstorganisierende Zufallsnetzwerke* und *Topologiemanagement*.

9.1 Die Verbindungsstruktur von Gnutella

Wir haben Gnutella als das erste echte Peer-to-Peer-Netzwerk kennen gelernt. Durch das Gnutella-Protokoll wird die Verbindungsstruktur nicht vorgegeben. Die Anzahl der Nachbarn eines Peers ergibt sich daher aus der verwendeten Software und den speziellen Wünschen des Benutzers. Außerdem hängt sie davon ab, wieviele Peers das Netzwerk bereits verlassen haben. Denn der Verbindungsaufbau findet nur zu Beginn statt, wobei aber auch nicht explizit verhindert wird, dass ein Gnutella-Peer zwischendurch seine Nachbar-Liste erweitert.

So gesehen handelt es sich in der Häufigskeitsverteilung der Ausgrade (Anzahl der Nachbarn) nicht um ein Resultat eines Algorithmus, sondern um ein soziales Phänomen. In [33, 54, 55] wurde genau diese Häufigkeitsverteilung untersucht. Das Ergebnis ist in Abbildung 9.1 dargestellt. Auf den Achsen ist jeweils die Häufigkeit und der Grad logarithmisch abgetragen. Es zeigt sich ein linearer Zusammenhang zwischen den Größen, d.h., für Konstanten c und k gilt:

$$\log(\text{Anzahl Peers mit Grad } g) = c - k \log d \,.$$

Löst man diese Gleichung nach der Anzahl der Peers auf, so erhält man für $C = 2^c$:

$$\text{Anzahl Peers mit Grad } g \;= \frac{C}{d^k} \,.$$

Pareto-Verteilungen

Im Englischen nennt man einen solchen Zusammenhang ein *Power Law*. Derartige Relationen findet man auch andernorts. Die zugehörigen Wahrscheinlichkeitsverteilungen werden nach ihrem Entdecker Wilfried Fritz Pareto als *Pareto-Verteilungen*

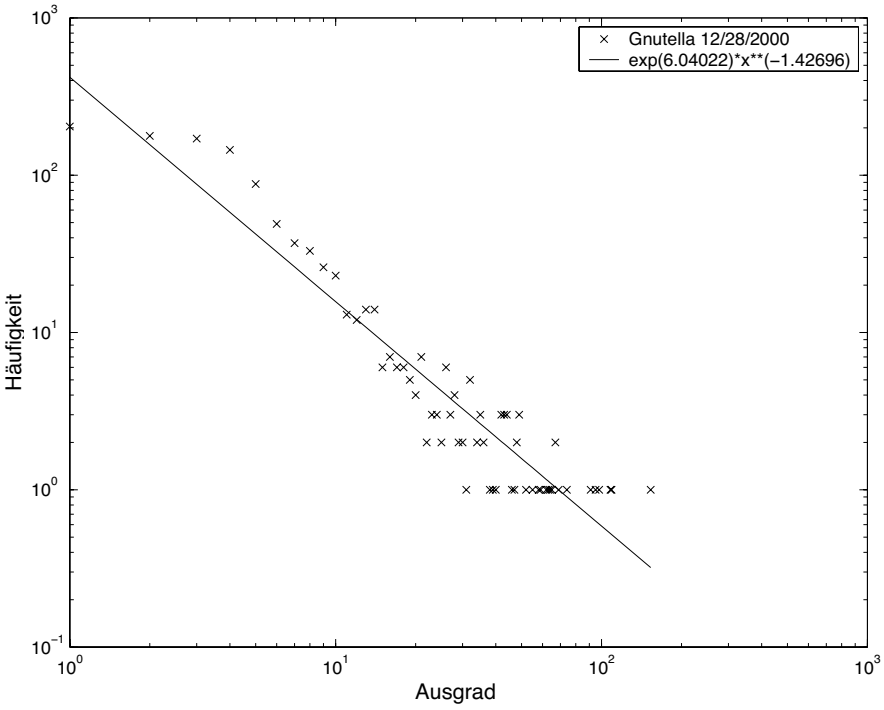


Abb. 9.1. Häufigkeit der Ausgrade im Gnutella-Netzwerk (Messung im März 2001) [54].

bezeichnet. Zu beachten ist, dass die obige Gleichung keine mathematisch exakte Beschreibung des Grades ist, sondern lediglich eine Formel, die die Beobachtung ganz gut annähert.

Die *diskrete Pareto-Verteilung* ist für $x \in \{1, 2, 3, \ldots\}$ definiert als

$$Pr[X = x] = \frac{1}{\zeta(a)x^{\alpha}} ,$$

wobei der konstante Faktor $\zeta(a) = \sum_{i=1}^{\infty} \frac{1}{i^{\alpha}}$ auch als *Riemannsche Zeta-Funktion* bekannt ist.

Pareto-Verteilungen besitzen die so genannte *Heavy Tail*-Eigenschaft. Damit wird beschrieben, dass für große Werte die Wahrscheinlichkeiten im Vergleich zu Poisson-Verteilungen oder geometrischen Verteilungen nur sehr langsam abnehmen. Im Gegensatz zu diesen sind nämlich nicht alle Momente $E[X^k]$ einer Pareto-Verteilung definiert. Auch existiert der Erwartungswert einer Pareto-Verteilung nur, wenn $\alpha > 2$ gilt. Die Varianz und das zweite Moment $E[X^2]$ existieren genau dann, wenn $\alpha > 3$ ist und $E[X^k]$ ist genau dann definiert, wenn $\alpha > k + 1$.

Der Vollständigkeit halber geben wir hier noch die Dichtefunktion der *kontinuierlichen Pareto-Funktion* an. Sie ist für $x \geq x_0$ definiert als

$$f(x) = \frac{\alpha - 1}{x_0} \left(\frac{x_0}{x}\right)^{\alpha} .$$

Einen Graph, dessen Knotengrade Pareto-verteilt sind, bezeichnen wir im Weiteren auch kurz als *Pareto-Graph*. Wir werden nun sehen, welche Aussagen über die Struktur eines Graphen sich allein durch die Pareto-Verteilung der Grade treffen lassen. Aiello, Chung und Lu untersuchen in [56] Graphen, deren Grade gemäß einer Pareto-Verteilung mit Exponent α gewählt wurden. Diese Graphen werden durch den folgenden Zufallsprozess erzeugt. Zuerst wählt man die Anzahl der Nachbarn in einem Pareto-verteilten Zufallsprozess mit Exponenten α. Nachdem man auf diese Weise die Anzahl der Nachbarn festgelegt hat, werden diese nun zufällig zugelost.

Für genügend große Graphen mit n Knoten lassen sich in Abhängigkeit des Exponenten α folgende Aussagen über Pareto-Graphen treffen:

- Ist $\alpha < 1$, dann ist der resultierende Graph mit Wahrscheinlichkeit $1 - o(1)$ zusammenhängend.
- Für $\alpha > 1$ sind Pareto-Graphen mit Wahrscheinlichkeit $1 - o(1)$ nicht zusammenhängend.
- Für $1 < \alpha < 2$ gibt es eine Zusammenhangskomponente der Größe $\Theta(n)$.
- Für $2 < \alpha < 3,4785\ldots$ gibt es eine Zusammenhangskomponente der Größe $\Theta(n)$ und sonst nur kleinere Zusammenhangskomponenten der maximalen Größe $O(\log n)$.
- Für $\alpha > 3.4785$ gibt es keine große Zusammenhangskomponente der Größe $\Theta(n)$.

Der Fall $\alpha < 1$ ist auf den ersten Blick irritierend, denn für solche Koeffizienten ist der normalisierende Faktor für beliebige Größen von Graphen keine Konstante mehr. Man behilft sich hier, indem man einen normalisierenden Faktor in Abhängigkeit der Größe des Graphen zulässt.

Man kann also aus dem Grad eines Pareto-Graphen schon erstaunlich viel Strukturinformation herauslesen. Pareto-Verteilungen sind ein sehr typisches Phänomen in sozialen Prozessen. 1897 beobachtete Pareto [57], dass die Verteilung des Wohlstands in der Bevölkerung einer solchen Verteilung folgt. Bis heute wurde diese These immer wieder mit den aktuellen Daten überprüft. Das Ergebnis ist, dass sich zwar die Koeffizienten der Pareto-Verteilung ändern, dass aber die Ergebnisse weiterhin Gültigkeit haben. (Oftmals versteht man unter der Pareto-Verteilung nur eine verballhornte Form: 20% der Bevölkerung haben 80% des Einkommens, und 80% Prozent der Bevölkerung haben 20% des Einkommens. Diese Darstellung übersimplifiziert jedoch Paretos Beobachtung.)

Andere Beispiele für Pareto-Verteilungen

Im Jahr 1932 untersuchte der Germanist George Kingsley Zipf die Worthäufigkeiten in Sprachen. So stellte er aufgrund empirischer Untersuchungen die Behauptung auf, dass das n-häufigste Wort einer Sprache mit relativer Häufigkeit $f(n)$ proportional zu $\frac{1}{n}$ auftaucht. Wie schon zuvor bei der Pareto-Verteilung für Koeffizienten $\alpha \leq 1$

ist diese Aussage nur sinnvoll, wenn man eine endliche Verteilung betrachtet (da für die harmonische Reihe $\sum_{i=1}^{n} = H_n$ gilt: $H_n = \ln n + O(1)$ — sie strebt also für wachsendes n gegen unendlich). Diese Beobachtung wurde von Yule [58] bestätigt und in [59] verallgemeinert.

Später wurde der Begriff der *Zipf-Verteilung* ausgedehnt und auch für andere Koeffizienten definiert. Der entscheidende Unterschied zwischen Zipf- und Pareto-Verteilung ist, dass sich die Zipfverteilung auf den Rang und nicht den Wert bezieht, d.h., eine Folge $x_1 < x_2 < x_3 < \ldots$ ist Zipf-verteilt mit Exponenten α, wenn

$$Pr[X = x_i] = \frac{c}{i^{\alpha}}$$

gilt und die Wahrscheinlichkeit für andere Werte 0 ist. Somit stellen Zipf-Verteilungen eine Verallgemeinerung der Pareto-Verteilungen dar.

Ein Beispiel für eine Zipf-Verteilung ist die Verteilung der Größe von Städten [60, 61]. Hierfür liegen ausführliche Untersuchungen vor, in denen man mittels des Zipf-Exponenten den Grad der Verstädterung einer Region erkennen kann.

Im Internet können Zipf- und Pareto-Verteilungen nicht nur bei Gnutella beobachtet werden. So untersuchte man in [62] den Verbindungsgrad von Internet-Routern, siehe Abbildung 9.2. In [63, 64] wurde der Ein- und Ausgrad von Web-Seiten im WWW untersucht, siehe Abbildung 9.3. Der Ausgrad einer Webseite ist hierbei einfach die Anzahl ihrer Links, der Eingrad ist die Anzahl anderer Web-Seiten, die auf diese zeigen.

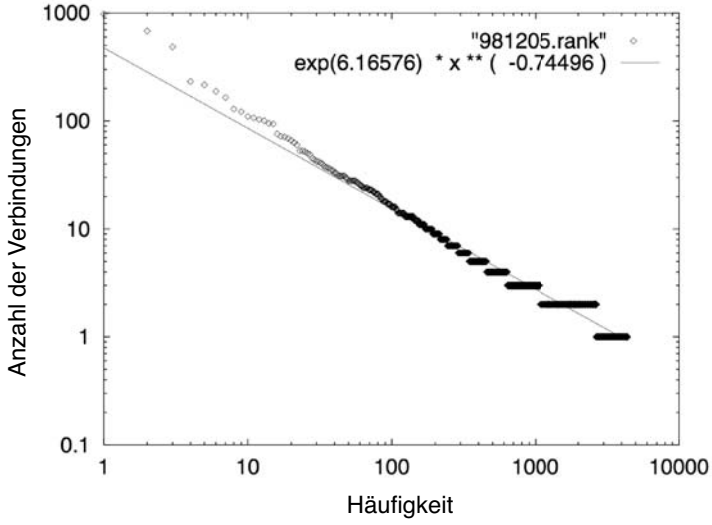


Abb. 9.2. Anzahl der Verbindungen von Internet-Routern (vertikale Achse), sortiert nach dem Rang (horizontale Achse) im Dezember 1998 [62].

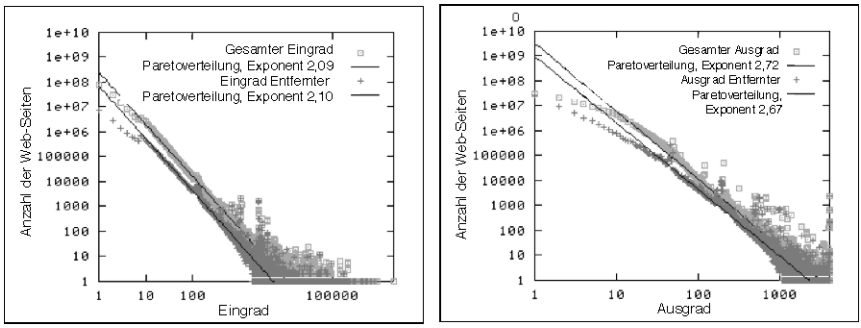


Abb. 9.3. Anzahl und Häufigkeit eingehender und ausgehender Links auf Web-Seiten im Mai 1999 [65].

Abbildung 9.3 zeigt, wie gut eine Pareto-Verteilung diese Häufigkeitsverteilung annähert. Betrachtet man die Anzahl der Web-Seiten einer Web-Site, die Anzahl der Benutzer, die Anzahl der Links auf eine Web-Site und von einer Web-Site, so findet man wieder eine Pareto-Verteilung [66]. Auch die Anzahl der Zugriffe auf eine Web-Seite unterliegt einer Zipf-Verteilung [67]. Interessanterweise ist auch die Größe einer Web-Seite (die Anzahl der Bytes) Pareto-verteilt [67]. Diese Beobachtung kannte man schon aus der Größenverteilung von Unix-System-Dateien.

9.1.1 Der Durchmesser des Gnutella-Netzwerks

In der schon zitierten Arbeit von Jovanovic [54] wird neben der Gradverteilung auch der Durchmesser von Gnutella untersucht. Gerade dieser Parameter ist wichtig für eine erfolgreiche Suche in einem unstrukturierten Peer-to-Peer-Netzwerk. Die Untersuchung geschah anhand von fünf Messungen im Jahr 2000 im Gnutella-Netzwerk. Dabei zeigte sich, dass der Durchmesser eines Gnutella-Netzwerks mit rund 1000 Teilnehmern im Bereich von acht bis zwölf liegt (siehe Tabelle 9.1). Der durchschnittliche Grad der Peers lag zwischen 1,7 und 4.

Tabelle 9.1. Anzahl Peers und Durchmesser des Gnutella-Netzwerks. Fünf Messungen aus dem Jahr 2000 [54].

Datum der Messung	Knoten	Kanten	Durchmesser
13.11.2000	992	2465	9
16.11.2000	1008	1782	12
20.12.2000	1077	4094	10
27.12.2000	1026	3752	8
28.12.2000	1125	4080	8

Der Durchmesser des Gnutella-Netzwerks ist somit erstaunlich klein. Zum Vergleich: Ein zweidimensionales Gitter mit Grad vier hat bei 1000 Teilnehmern einen Durchmesser von rund 62. Ein ebensolcher Torus hat einen Durchmesser von 31. Einen so geringen Durchmesser wie beim Gnutella-Netzwerk erreicht man sonst nur durch Strukturen wie einem vollständigen Baum, der bei einem Grad von drei einen Durchmesser von 20 besitzt. Diese Beobachtung wirft die Frage auf, wie sich der geringe Durchmesser des Gnutella-Netzwerks erklären lässt. Bevor wir eine mathematische Erklärung liefern, möchten wir andere soziale Netzwerke vorstellen, die solch einen geringen Durchmesser besitzen.

9.1.2 Small-World-Netzwerke

In den Sechzigern hat der Psychologe Stanley Milgram das folgende Experiment durchgeführt [68]: Er händigte 60 zufällig ausgewählten Teilnehmern in Omaha, Nebraska, jeweils einen Brief aus, den diese einer Adresse in Sharon, Massachusetts, zukommen lassen sollten. Sie durften den Brief aber nur Personen zusenden, die sie persönlich kannten (und nicht etwa direkt an die Ziel-Adresse). Außerdem sollten die Personen, die nun als Zwischenstationen ausgewählt wurden, sich wieder an diese Bedingungen halten. Tatsächlich kam die Mehrzahl der Briefe am Bestimmungsort an und die Teilnehmer hielten sich an die Anforderungen. Die meisten Briefe kamen mit nur sechs Zwischenstationen zum Ziel, was Milgram zu der Vermutung brachte, dass jeder Mensch jeden anderen über maximal sechs Zwischenstationen kennt (*six degrees of separation*) [68].

Dieses Phänomen wird als *Small-World*-Phänomen bezeichnet und soziale Netzwerke mit einem so geringen Durchmesser bezeichnet man als *Small-World-Netzwerke* (*Small-World-Network*). Bis heute ist außer dem geringen Durchmesser noch keine genauere Beschreibung oder Charakterisierung dieser Netzwerke erfolgt. Wie entstehen Small-World-Netzwerke? Wir werden im Folgenden drei Ansätze für die Modellierung von Small-World-Netzwerken vorstellen.

Das Modell von Watts und Strogatz

Watts und Strogatz [69] gehen von der Beobachtung aus, dass in sozialen Netzwerken *Cliquen* (das sind vollständige Teilgraphen: eine Menge von Knoten, in der jeder Knoten mit jedem anderen verbunden ist) weitaus häufiger vorkommen als zum Beispiel in Zufallsgraphen. Ihr Ziel war es, eine Netzwerkbeschreibung zu liefern, die durch Wahl eines Parameters $p \in [0, 1]$ variabel und kontinuierlich aus einem strukturierten Netzwerk ein vollkommen zufälliges entstehen lässt. Hierzu betrachteten sie einen Ring aus n Knoten, in dem jeder der Knoten mit den $k/2$ nächsten Knoten zur linken und zur rechten verbunden ist (also ähnlich wie bei Pastry, nur ohne die Menge M lokaler Nachbarn und ohne die Routing-Tabelle). Ein solches Ring-Netzwerk ist in Abbildung 9.4(a) dargestellt.

Ausgehend von diesem Ring-Netzwerk wird nun jede der ursprünglichen Kanten mit Wahrscheinlichkeit p durch eine Kante zu einem zufälligen Nachbarn ersetzt. Die dabei entstehenden Netzwerke sind für verschiedene Parameter p in Abbildungen 9.4 dargestellt. So bleibt das Ring-Netzwerk für $p = 0$ unverändert und wird für

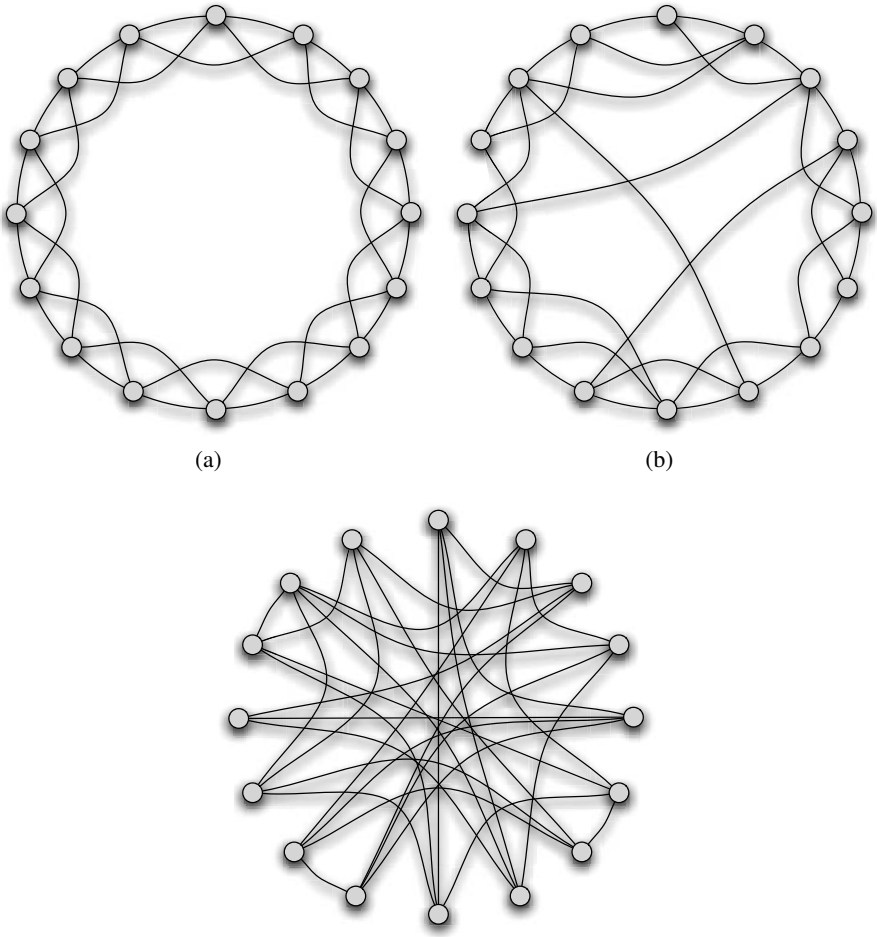


Abb. 9.4. Small-World-Netzwerke nach dem Modell von Watts und Strogatz. Dargestellt werden Netzwerke mit $k = 4$ sowie Parameter $p = 0$ (a), $0 < p < 1$ (b) und $p = 1$ (c) [69].

$p = 1$ zu einem Zufalls-Netzwerk. Für $p = 0.1$ zum Beispiel bleiben jedoch relativ viele Cliquen des ursprünglichen Ring-Netzwerks bestehen und durch die zufälligen Kanten wird der Durchmesser des Netzwerks stark verringert.

Das Modell von Kleinberg

Die Small-World-Netzwerk-Modellierung von Watts und Strogatz wurde von Kleinberg in [70] aufgegriffen. Kleinberg zeigte, dass Milgrams Beweis sogar konstruktiv war. Jeder der Teilnehmer hatte also eine Intuition, wem man das Paket zu geben hat, damit dieses möglichst schnell ankommt. In Kleinbergs Modell [70] sind die Teil-

nehmer in einem euklidischen Gitternetzwerk angeordnet. Die Verbindungen dieser Gitter-Struktur werden als *Lokal*-Verbindungen bezeichnet (siehe Abbildung 9.5). Mit Hilfe der Gitterstruktur kann bereits Routing durchgeführt werden, wenn die Koordinaten des Ziels im Gitter bekannt sind (siehe CAN, Kapitel 4).

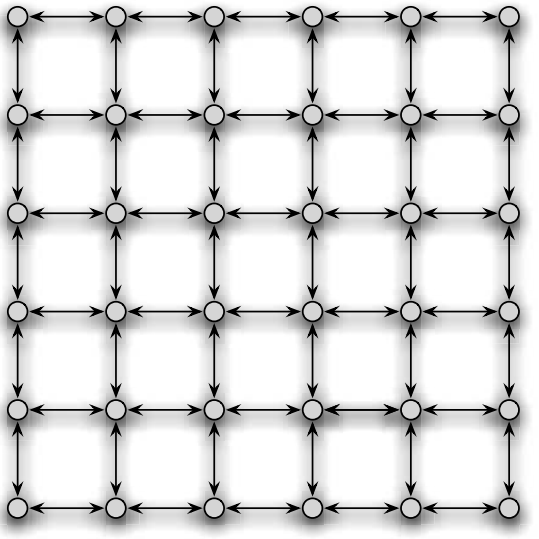


Abb. 9.5. Die Lokal-Verbindungen in Kleinbergs Modell [70].

Zusätzlich zu den Lokal-Verbindungen erhält jeder Knoten noch einige Kanten zu weiter entfernten Knoten, die dann als Abkürzungen beim Routing dienen (siehe Abbildung 9.6). Diese Verbindungen werden als *Fern-Verbindungen* bezeichnet. Durch eine vorsichtige Wahl dieser Fern-Verbindungen mittels einer über dem Abstand gewichteten Wahrscheinlichkeitsverteilung konnte Kleinberg zeigen, dass in diesen Graphen jedes Paket mit $\mathcal{O}(\log^2 n)$ Schritten zum Ziel gesendet werden kann. Das Modell von Kleinberg ist insbesondere in der Auswahl der Fernverbindungen komplexer und flexibler als das Modell von Watts und Strogatz. Wir verzichten hier jedoch auf die detaillierte Beschreibung der Wahrscheinlichkeitsverteilung zur Auswahl ebendieser. Die Details hierzu finden sich in [70].

Das Modell von Barabasi und Albert

Einen völlig anderen Weg zum Aufbau eines Small-World-Netzwerks gehen Barabasi und Albert in [63]. Ausgehend von einem kleinen Startgraphen werden sukzessive Knoten mit jeweils m Kanten eingefügt. Nun wird Paretos Beobachtung verwendet: „rich gets richer“. So wählt ein neuer Knoten seine Nachbarn mit einer Wahrscheinlichkeit, die proportional zum Grad dieser Knoten ist. Somit erhalten Knoten, die bereits einen hohen Grad haben, mehr neue Kanten (und damit einen noch höheren

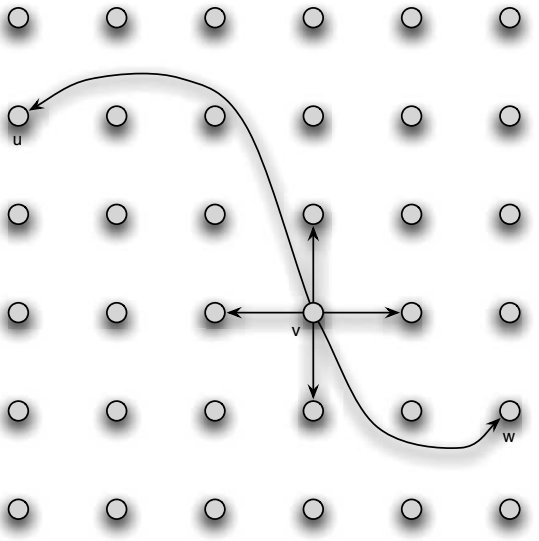


Abb. 9.6. Die Lokal- und Fernverbindungen eines Knotens v in Kleinbergs Modell [70].

Grad) als Knoten mit geringem Grad (siehe Abbildung 9.7). Es zeigt sich, dass das entstehende Netzwerk einen logarithmischen Durchmesser besitzt und dass die Gradverteilung einer Pareto-Verteilung mit Exponent $2,9 \pm 0,1$ genügt. Auch hier verzichten wir auf weitere Details des Modells.

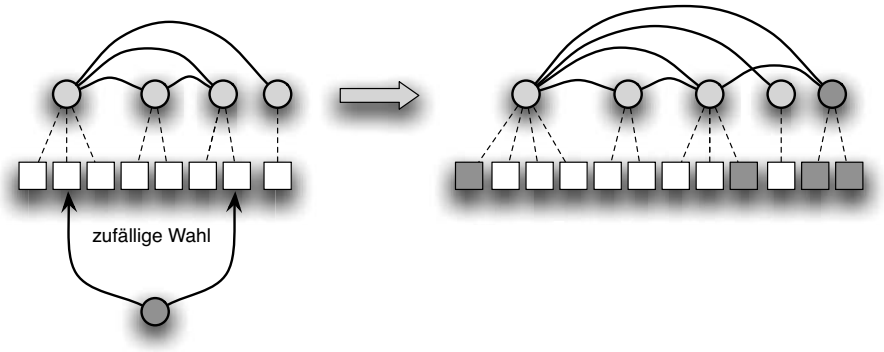


Abb. 9.7. Einfügen eines Knotens im Small-World-Netzwerk Modell von Barabasi und Albert [63] ($m = 2$).

9.1.3 Vergleich von Gnutella und Small-World-Netzwerken

Wir werden nun die drei vorgestellten Modelle für Small-World-Netzwerke mit dem Gnutella-Netzwerk anhand ihrer *charakteristischen Pfadlänge* vergleichen. Die charakteristische Pfadlänge ist der durchschnittliche Abstand zweier Knoten bzw. Peers in einem Netzwerk. Tabelle 9.2 zeigt die charakteristische Pfadlänge des Gnutella-Netzwerks (hier handelt es sich wiederum um fünf Messungen aus dem Jahr 2000 [54]) verglichen mit derjenigen, der eben vorgestellten Small-World-Netzwerke und der eines Zufallsgraphen. Die Parameter der einzelnen Modelle wurden hierbei (soweit möglich) der Größe und dem Grad von Gnutella angepasst. Es zeigt sich, dass sich die beste Übereinstimmung mit dem Modell von Barabasi und Albert ergibt. Mit dem Modell von Watts-Strogatz sowie dem Zufallsgraph mit n Knoten und Kantenwahrscheinlichkeit p kann eine gewisse Nähe festgestellt werden. Die schlechteste Beschreibung des Gnutella-Netzwerks liefert das zweidimensionale Gitter aus Kleinbergs Ansatz.[1]

Tabelle 9.2. Vergleich der charakteristischen Pfadlänge von Gnutella, Small-World-Netzwerken und Zufallsgraphen [54].

Datum	Gnutella	Barabasi-Albert	Watts-Strogatz	Zufallsgraph	Kleinberg
13.11.2000	3,72299	3,47491	4,59706	4,48727	20,6667
16.11.2000	4,42593	4,07535	4,61155	5,5372	21,3333
20.12.2000	3,3065	3,19022	4,22492	3,6649	22
27.12.2000	3,30361	3,18046	4,19174	3,70995	21,3333
28.12.2000	3,32817	3,20749	4,25202	3,7688	22,6667

Dieses Ergebnis ist nicht sehr überraschend, wenn man bedenkt, wie ähnlich das Einfügen von Knoten im Modell von Barabasi und Albert und der Netzwerkaufbau von Gnutella funktionieren. Man kann also schlussfolgern, dass obwohl in Gnutella die Netzwerkstruktur ungeplant aufgebaut wird, sich durch das Prinzip der Selbstorganisation ein Pareto-verteiltes Small-World-Netzwerk (ähnlich dem des Modells von Barabasi und Albert) mit geringem Durchmesser ergibt.

9.2 Selbstorganisierende Zufalls-Netzwerke

Wie wir gesehen haben, besitzen Pareto-Netzwerke sehr gute strukturelle Eigenschaften. Wenn sie jedoch als Verbindungsstruktur eines Peer-to-Peer-Netzwerks dienen sollen, in dem die Peers typischerweise vollkommen gleichwertig behandelt werden sollen, so ist die ungleichmäßige Verteilung der Grade nicht akzeptabel. Es ist dann vielmehr erstrebenswert, ein Netzwerk mit gleichem Grad, kleinem

[1] Hierbei ist anzumerken, dass es nicht das Ziel der drei Small-World-Graph-Modelle ist, eine mathematische Modellierung des Gnutella-Netzwerks zu finden.

Durchmesser und starkem Zusammenhang zu haben. Darüber hinaus sollen sich diese Netzwerke noch selbst organisieren können wie Pareto-Netzwerke.

Eine Alternative sind Zufalls-Netzwerke. Wie der Name schon sagt, sind dies Netzwerke mit zufälliger Verbindungsstruktur. Zufalls-Netzwerke haben viele Eigenschaften, die sie für den Einsatz im Bereich der Peer-to-Peer-Netzwerke interessant machen. So besitzen sie z.B. mit hoher Wahrscheinlichkeit einen logarithmischen Durchmesser, bleiben auch beim Ausfall vieler Netzwerkteilnehmer mit hoher Wahrscheinlichkeit zusammenhängend, lassen sich beim Ausfall einiger Netzwerkteilnehmer leicht reparieren, da lediglich ein neuer zufälliger (d.h., beliebiger) Nachbar gewählt werden muss, etc.

Es stellt sich die Frage, wie wir Zufalls-Netzwerke erzeugen und unterhalten können und welche Typen von Zufalls-Netzwerken es gibt. Hierfür werden wir im Folgenden zunächst mathematische Modelle für Zufallsgraphen vorstellen. Dann werden wir untersuchen, wie wir Zufalls-Netzwerke, die diesen Klassen entsprechen, verteilt als Verbindungsstruktur eines Peer-to-Peer-Netzwerks unterhalten können.

9.2.1 Standardmodelle für Zufallsgraphen

Zufallsgraphen spielen in vielen Gebieten der Informatik und Mathematik eine wichtige Rolle, so dass einige Modelle zum Generieren eines solchen Graphen existieren. Ein sehr bekanntes Modell für Zufallsgraphen wird als $\mathcal{G}_{n,p}$ bezeichnet. Hierbei benennt n die Anzahl der Knoten und p ist eine feste Wahrscheinlichkeit, mit der jede der möglichen Kanten zwischen den n Knoten existiert. Will man solche Graphen als Netzwerk einsetzen, muss man zuerst berücksichtigen, dass ein Graph aus $\mathcal{G}_{n,p}$ für $p \in o(\frac{\log n}{n^2})$ mit Wahrscheinlichkeit $1 - o(1)$ nicht zusammenhängend ist. Dies folgt bereits aus der Wahrscheinlichkeit, dass ein Knoten mit überhaupt keinen Kanten inzident sein kann. Solange also der durchschnittliche Grad nicht mindestens logarithmisch ist, sind solche Zufallsgraphen nicht mit hoher Wahrscheinlichkeit zusammenhängend, was für die Anwendung als Peer-to-Peer-Netzwerk nicht akzeptabel ist.

Es gibt eine einfache Operation, die zufällige Graphen aus $\mathcal{G}_{n,p}$ gleichwahrscheinlich erzeugt. Hierzu wählt man wiederholt ein zufälliges Knotenpaar aus den n Knoten und fügt eine Kante zwischen diesen mit Wahrscheinlichkeit p ein und entfernt diese mit Wahrscheinlichkeit $1 - p$. Dabei spielt es keine Rolle, ob die Kante bereits existiert hat, oder ob das zufällige Knotenpaar schon einmal zuvor gewählt wurde. Der resultierende Graph ist ein echt zufälliger Graph der Klasse $\mathcal{G}_{n,p}$, sobald jedes der möglichen Knotenpaare mindestens einmal gewählt wurde.

Leider ist es schwierig, die beschriebene Operation in einem Peer-to-Peer-Netzwerk umzusetzen. Ein Problem ist die zufällige Wahl eines Knotenpaares. Hierzu müsste ein Peer alle anderen im Netzwerk existierenden Peers kennen (z.B. durch Fluten mit Nachrichten). Eine weitere Möglichkeit wäre es, einen sehr langen *Random Walk* im Netzwerk durchzuführen. Bei einem Random Walk wandert man durch das Netzwerk, in dem man an jedem Knoten bzw. Peer eine zufällige Kante wählt und dieser folgt. Ist ein solcher Random Walk sehr lang, wird irgendwann jeder andere Knoten mit gleicher Wahrscheinlichkeit erreicht. Allerdings ist die Länge des hierfür

benötigten Random Walks abhängig von der (unbekannten) Struktur des Netzwerks und mitunter viel zu lang, um praktisch umgesetzt zu werden.

Ein weiteres Problem ist, dass das Netzwerk durch diese Operationen auseinanderfallen kann. Und ist das Peer-to-Peer-Netzwerk erst in zwei Komponenten zerfallen, kann es nicht wieder zusammengefügt werden — zumindest nicht ohne zusätzliche Informationen, die in einem Peer-to-Peer-Netzwerk nicht vorhanden sind.

In einem Zufallsgraph des $\mathcal{G}_{n,p}$-Modells haben die Knoten zudem unterschiedliche Grade; wir hatten jedoch eingangs festgehalten, dass die Knoten möglichst den gleichen Grad haben sollten. Nun existiert jedoch mit $\mathcal{G}_{n,d}$ auch ein weit verbreitetes Modell für solche *regulären Zufallsgraphen*. Hier bezeichnet n wieder die Anzahl der Knoten und d den Grad ebendieser. Der Wechsel zu d-regulären Graphen hilft zugleich, das Problem des mangelnden Zusammenhangs abzuschwächen. Denn für $d \geq 3$ ist ein d-regulärer Zufallsgraph mit Wahrscheinlichkeit $1 - o(1)$ zusammenhängend und für höhere konstante Grade gilt der Zusammenhang mit Wahrscheinlichkeit $1 - n^{-c}$, also mit polynomiell hoher Wahrscheinlichkeit [71].

Es existiert auch eine einfache verteilte Operation, die in der Lage ist, ausgehend von einem beliebigen d-regulären Graphen G aus $\mathcal{G}_{n,d}$ jeden anderen Zufallsgraphen aus $\mathcal{G}_{n,d}$ schnell und mit uniformer Wahrscheinlichkeit zu erzeugen. Diese Operation nennt sich *Simple Switching* und wurde eingeführt von McKay [72] (und verwendet in [73, Seite 215] unter dem Namen *Rewiring*). Die Simple Switching-Operation ist wie folgt definiert:

Definition 9.1 (Simple Switching). *Sei $G = (V, E)$ ein ungerichteter d-regulärer Graph. Wähle zwei Kanten $\{v_1, v_2\}$, $\{v_3, v_4\}$ zufällig und gleichwahrscheinlich aus E. Ersetze diese Kanten durch Kanten $\{v_1, v_3\}$, $\{v_2, v_4\}$ oder $\{v_1, v_4\}$, $\{v_2, v_3\}$, falls der resultierende Graph keine Mehrfach-Kanten besitzt.*

Zur Veranschaulichung wird die Simple Switching-Operation in Abbildung 9.8 dargestellt. In [74] wird gezeigt, dass lediglich eine polynomielle Anzahl von Simple Switching-Operationen erforderlich ist, um einen einen echt zufälligen Graph aus $\mathcal{G}_{n,d}$ zu erzeugen. Es wiederholen sich hier allerdings die Probleme, auf die wir schon bei der verteilten Umsetzung einer Operation für Graphen des $\mathcal{G}_{n,p}$-Modells gestoßen sind. So ist es zum einen schwierig, zwei zufällige Kanten mit uniformer Wahrscheinlichkeit zu bestimmen. Zum anderen besteht immer noch die Gefahr, dass der Graph in zwei oder mehr Zusammenhangskomponenten zerfällt, wenngleich die Wahrscheinlichkeit hierfür nun deutlich geringer als zuvor ist.

Eine andere Technik für die Umsetzung d-regulärer Zufallsnetzwerke wird in [75] vorgestellt. Wir werden diesen Ansatz jedoch nicht im Detail besprechen und betrachten im Folgenden nur die grundlegende Idee. Das Netzwerk [75] verwendet einen zentralen Host-Server, der einen Cache-Speicher mit konstant vielen zufälligen Peers im Netzwerk unterhält. Jeder Peer, der sich neu anmeldet, wird aus diesem Cache mit zufälligen Nachbarn verbunden. Das Hauptaugenmerk des Artikels gilt der Beschreibung und Analyse der Technik, diesen konstant großen Cache zu unterhalten. Jeder Knoten, der dem Netzwerk beitritt, erhält eine Chance, in den Cache zu gelangen.

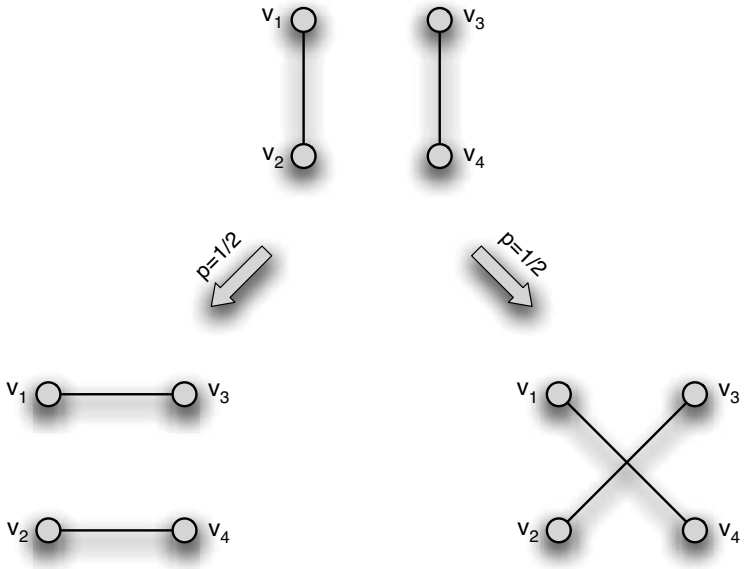


Abb. 9.8. Die Simple Switching-Operation.

Dieser Ansatz liefert eine effiziente Methode, um Zufalls-Netzwerke aufzubauen. Aber die Verwendung eines zentralen Servers steht natürlich im krassen Gegensatz zu den elementaren Grundsätzen eines Peer-to-Peer-Netzwerks, und es stellt sich die Frage, warum dieser Server nicht gleich die gesamte Knotenmenge inklusive der Kanten verwaltet. Ersetzt man den zentralen Server durch eine Reihe von Servern, so erhält man das Äquivalent eines hybriden Netzwerks wie *Fast-Track* oder *Gnutella-2,* die wir in Kapitel 13 kennenlernen werden.

9.2.2 Ungerichtete reguläre Zufallsgraphen

Wir werden nun sehen, wie sich ein Peer-to-Peer-Netzwerk, auf Basis regulärer zusammenhängender Zufallsgraphen, verteilt, d.h., ohne Zuhilfenahme zentraler Mechanismen, unterhalten lässt. Hierzu betrachten wir nicht länger einen zufälligen Graphen aus der Klasse $\mathcal{G}_{n,d}$, sondern wir schränken uns auf die Teilmenge der zusammenhängenden Graphen aus $\mathcal{G}_{n,d}$ ein. Diese Teilmenge der zusammenhängenden regulären Graphen bezeichnen wir fortan als $\mathcal{CG}_{n,d}$.

Die 1-Flipper Operation

In [76] wird eine einfache verteilte Operation beschrieben, die jeden Graphen aus $\mathcal{CG}_{n,d}$ mit uniformer Wahrscheinlichkeit generieren kann. Diese Operation wird als

1-*Flipper*-Operation bezeichnet und besitzt Ähnlichkeit mit der schon vorgestellten Simple Switching-Operation. Aus Graphen-theoretischer Sicht ist die 1-Flipper-Operation wie folgt definiert:

Definition 9.2 (1-Flipper). *Betrachte einen ungerichteten, zusammenhängenden und d-regulären Graphen $G = (V, E)$ sowie vier Knoten $v_1, v_2, v_3, v_4 \in V$, die einen Pfad $P = (v_1, v_2, v_3, v_4)$ in G definieren. Falls $\{v_1, v_3\}$, $\{v_2, v_4\} \notin E$, dann transformiert die 1-Flipper-Operation F_P^1 den Graphen G zu $F_P^1(G) = (V, E')$ mit*

$$E' := (E \setminus \{\{v_1, v_2\}, \{v_3, v_4\}\}) \cup \{\{v_1, v_3\}, \{v_2, v_4\}\} \ .$$

Die 1-Flipper-Operation wird in Abbildung 9.9 veranschaulicht. Wir bezeichnen die Kanten $\{v_1, v_3\}$, $\{v_2, v_4\}$ auch als *Flip-Kanten* und die Kante $\{v_2, v_3\}$ als *Hub-Kante* der 1-Flipper-Operation.

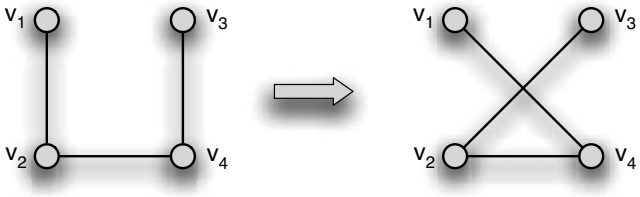


Abb. 9.9. Die 1-Flipper-Operation [76].

Ein wichtiger Unterschied zwischen der 1-Flipper- und Simple Switching-Operation ist, dass die 1-Flipper-Operation den Zusammenhang des Graphen garantiert. Dies lässt sich leicht durch das Betrachten der Abbildungen 9.9 und 9.8 feststellen: Im Fall der 1-Flipper-Operation bleiben alle beteiligten Knoten miteinander verbunden, im Fall von Simple Switching jedoch nicht.

Wir werden nun sehen, dass, ausgehend von einem beliebigen d-regulären, zusammenhängenden Graph mit n Knoten, jeder andere Graph aus $\mathcal{CG}_{n,d}$ durch wiederholtes Anwenden zufälliger 1-Flipper-Operationen mit uniformer Wahrscheinlichkeit erzeugt wird. Zufällige 1-Flipper-Operation bedeutet hierbei, dass der Pfad P zufällig gewählt wird, d.h., es wird ausgehend von einem zufälligen Startknoten ein Random Walk über drei Kanten durchgeführt. Werden bei diesem Random Walk nicht vier unterschiedliche Knoten besucht, wird die Operation abgebrochen.

Um nun zu zeigen, dass die 1-Flipper-Operation Graphen aus $\mathcal{CG}_{n,d}$ uniform generiert, betrachten wir zunächst einige wichtige Eigenschaften der Operation.

Lemma 9.3. *Die Wahrscheinlichkeit, dass ein d-regulärer Graph G durch eine zufällige 1-Flipper-Operation zu einem d-regulären Graphen G' transformiert wird, ist identisch mit der Wahrscheinlichkeit, dass G' durch eine zufällige 1-Flipper-Operation zu G transformiert wird. Das heißt, es gilt*

$$Pr[G \xrightarrow{F^1} G'] = Pr[G' \xrightarrow{F^1} G].$$

Diese Eigenschaft der 1-Flipper-Operation bezeichnen wir auch als *Symmetrie*. Der Beweis des Lemmas ist nicht schwierig und wird hier nicht geführt. Er kann in [76] nachgelesen werden. Eine weitere Grundvoraussetzung für die uniforme Generierung von Graphen aus $\mathcal{CG}_{n,d}$ liefert das folgende Lemma:

Lemma 9.4. *Für alle Paare G, G' von Graphen aus $\mathcal{CG}_{n,d}$ mit $d \geq 2$ und d gerade, existiert eine Folge von 1-Flipper-Operationen, die G zu G' transformiert.*

Diese Eigenschaft bezeichnen wir auch als *Erreichbarkeit*, da jeder Graph aus $\mathcal{CG}_{n,d}$ zu jedem anderen aus $\mathcal{CG}_{n,d}$ transformiert werden kann. Kombiniert man nun die Aussagen von Lemma 9.3 und Lemma 9.4, folgt bereits die Eigenschaft, dass eine Folge von 1-Flipper-Operationen auf lange Sicht jeden Graphen aus $\mathcal{CG}_{n,d}$ mit der gleichen Wahrscheinlichkeit erzeugt. Um dies zu sehen, betrachten wir den *Markov-Prozess*, den die 1-Flipper-Operation über den Graphen aus $\mathcal{CG}_{n,d}$ beschreibt. Jeder Graph aus $\mathcal{CG}_{n,d}$ entspricht einem Zustand dieses Markov-Prozesses. Lemma 9.3 besagt, dass die zugehörige *Markov-Matrix* symmetrisch ist, und Lemma 9.4 zeigt, dass jeder Zustand des Markov-Prozesses erreichbar ist. Elementare Markov-Prozesstheorie besagt nun, dass im Limes jeder Startvektor gegen den uniformen Vektor mit gleichen Einträgen konvergiert, welcher der uniformen Wahrscheinlichkeitsverteilung entspricht. Wir halten dieses Ergebnis im folgenden Theorem fest.

Theorem 9.5. *Sei G ein d-regulärer zusammenhängender Graph mit n Knoten und $d \geq 2$ und d gerade. Dann wird durch wiederholtes Anwenden von zufälligen 1-Flipper-Operationen jeder d-reguläre zusammenhängende Graph mit n Knoten mit gleicher Wahrscheinlichkeit erzeugt, d.h.,*

$$\lim_{t\to\infty} Pr[G \xrightarrow{t} G'] = \frac{1}{|\mathcal{CG}_{n,d}|}.$$

Die 1-Flipper-Operation kann ohne weiteres verteilt angewendet werden, wenn die Menge der Peers gleich bleibt oder wächst (und der Grad gerade ist). Kommt ein Peer hinzu, so bestimmt er $d/2$ knotendisjunkte Kanten, löscht diese und setzt eine Kante zu jedem der d Endpunkte. So bleibt das Netzwerk d-regulär und zusammenhängend. Schwieriger wird es, wenn ein Peer das Netzwerk verlässt bzw. ausfällt und somit bei seinen d Nachbarn den Grad $d-1$ erzeugt. Diese Nachbarn müssten sich jetzt auf die Suche nach anderen Peers mit demselben Problem machen. Das ist aber in einem unstrukturierten Netzwerk sehr aufwändig. Eine einfache Lösung für das Problem ist, wenn Peers mit zu geringem Grad diesen erst dann erhöhen, wenn ihnen mindestens zwei Nachbarn fehlen. In dem Fall kann dann einfach wie beim Einfügen eines neuen Peers vorgegangen werden, d.h., der Peer setzt sich in die Mitte einer zufälligen Kante des Netzwerks. Dieses Vorgehen hat zur Folge, dass die resultierenden Graphen bzw. Netzwerke zwar nicht mehr exakt d-regulär sind, die Grade der Knoten liegen jedoch immer zwischen d und $d-1$, was in der Praxis nicht weiter tragisch ist.

9.2.3 Gerichtete Zufallsgraphen mit regulärem Ausgrad

Die im vorigen Abschnitt beschriebene 1-Flipper-Operation ist bereits sehr gut geeignet, um Zufalls-Netzwerke zu unterhalten. Aus praktischer Sicht lässt sich diese Operation jedoch noch weiter verbessern. Ansatzpunkt für die Verbesserung sind die ungerichteten Kanten. Um eine ungerichtete Kante zu unterhalten und ggf. zu ändern müssen immer beide zu dieser Kante adjazenten Knoten miteinander kommunizieren. Dieser Aufwand entfällt bei Verwendung gerichteter Graphen: Um eine Kante in einem gerichteten Graphen zu ändern, muss lediglich derjenige Knoten, der den Startpunkt der Kante definiert, den Zielpunkt in seiner Nachbarschaftsliste ändern. Stellt dieser Graph nun ein Netzwerk dar, bedeutet dies, dass das Ändern einer Kante nur noch eine lokale Operation ist und keine teure Kommunikation im Netzwerk stattfinden muss. Wenn wir also eine ähnliche Operation für gerichtete Graphen mit regulärem Ausgrad finden, können wir unter Umständen die Anzahl der Kommunikationsschritte je Operation deutlich verringern.

Wir überlegen uns zunächst, welche grundlegenden Möglichkeiten es gibt, die Kanten eines gerichteten Graphen $G = (V, E)$ zu ändern. Betrachten wir einen Knoten $v_1 \in V$, so gibt es zwei prinzipielle Möglichkeiten, die durch die beiden folgenden Operationen beschrieben werden:

Pointer-Push: Verändere die Menge der ausgehende Kanten von v_1, d.h., führe einen Random Walk v_1, v_2, v_3 in G durch und ersetze die Kante (v_1, v_2) durch die Kante (v_1, v_3) (siehe Abbildung 9.10).

Pointer-Pull: Verändere die Menge der auf v_1 zeigenden Kanten, d.h., führe einen Random Walk v_1, v_2, v_3, v_4 in G durch und ersetze die Kante (v_3, v_4) durch die Kante (v_3, v_1) (siehe Abbildung 9.11).

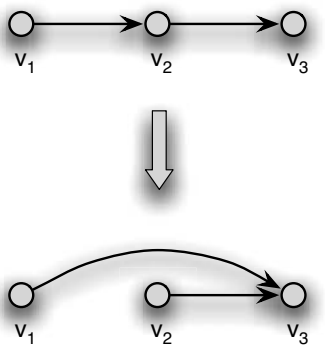


Abb. 9.10. Die Pointer-Push-Operation.

Die Einfachheit dieser elementaren Operationen bringt mit sich, dass die resultierenden Graphen *gerichtete Multi-Graphen* sein können, d.h., Kanten können mehr-

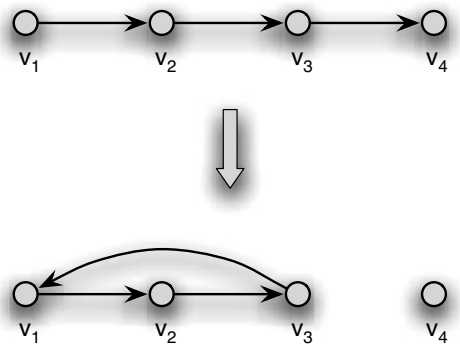


Abb. 9.11. Die Pointer-Pull-Operation.

fach existieren und Knoten können auf sich selbst zeigen. Dies ließe sich sicherlich durch einfache Abfragen beheben, wir werden später in diesem Kapitel jedoch noch sehen, dass wir Multi-Kanten tatsächlich benötigen. Deshalb betrachten wir von nun an gerichtete zusammenhängende Multigraphen mit regulärem Ausgrad und bezeichnen die Menge dieser Graphen als $\mathcal{MDG}_{n,d}$. Auch hier benennt n wieder die Anzahl der Knoten und d den Ausgrad ebendieser.

Bei näherer Analyse der beiden elementaren Operationen zeigt sich jedoch, dass weder Pointer-Push- noch Pointer-Pull-Operation tatsächlich für die Unterhaltung zufälliger Graphen aus $\mathcal{MDG}_{n,d}$ geeignet sind. Zwar behält der Graph G in beiden Fällen seinen regulären Ausgrad d, im Fall der Pointer-Push-Operation können jedoch Schleifen an einem Knoten erzeugt werden. Dies geschieht, wenn der Start-Knoten der Operation auch der End-Knoten der Operation ist. Genau das wird problematisch, wenn alle ausgehenden Kanten eines Knotens v Schleifen sind. Diese Schleifen können dann nicht mehr durch Pointer-Push-Operationen entfernt werden. Gleiches gilt dann für Kanten, die auf v zeigen. Der Knoten v wirkt dann wie eine Senke und wird im Laufe der Zeit immer mehr Kanten an sich ziehen. So wird der Graph G durch wiederholtes Anwenden der Pointer-Push-Operation zu einem sternförmigen Graphen oder im Fall mehrerer Senken zu einer Menge verbundener Sterne konvergieren (siehe Abbildung 9.12). Ein solcher Graph wird sich auch durch weitere Pointer-Push-Operationen nicht mehr verändern.

Im Fall der Pointer-Pull-Operation ist leicht zu sehen, dass diese nicht den Zusammenhang des Graphen garantiert. So würde ein Graph G durch wiederholtes Anwenden dieser Operation zu einem Graphen aus lauter einzelnen Knoten mit Schleifen transformiert (siehe Abbildung 9.13). Bei genügend hohem Grad kann es zwar sehr lange dauern, bis der Graph tatsächlich nicht mehr zusammenhängend ist, da das Auseinanderfallen in einem Netzwerk jedoch irreversibel ist, disqualifiziert dies die Pointer-Pull-Operation für den praktischen Einsatz.

So sind beide Elementaroperationen nicht für die Unterhaltung gerichteter Zufallsnetzwerke geeignet. Wir werden im folgenden Abschnitt jedoch sehen, dass eine Kombination dieser Operationen die beschriebenen Probleme beheben kann.

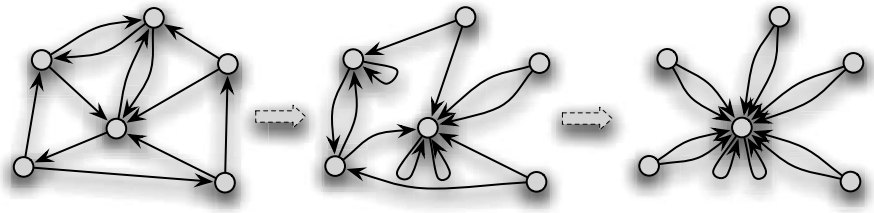


Abb. 9.12. Beim wiederholten Anwenden von Pointer-Push-Operationen konvergiert ein Graph gegen einen sternförmigen Multi-Graphen.

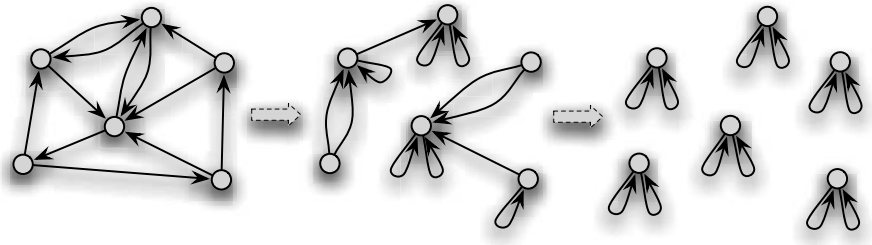


Abb. 9.13. Wiederholtes Anwenden von Pointer-Pull-Operationen zerlegt einen Graphen in einzelne Knoten mit Schleifen.

Die Pointer-Push&Pull-Operation

Eine geschickte Kombination von Pointer-Push- und Pointer-Pull-Operation wird in [77] vorgestellt. Diese Operation wird als *Pointer-Push&Pull* bezeichnet und ist formal wie folgt definiert:

Definition 9.6 (Pointer-Push&Pull). *Sei $G = (V, E)$, $G \in \mathcal{MDG}_{n,d}$, ein gerichteter und zusammenhängender Multi-Graph mit regulärem Ausgrad d. Seien des Weiteren $v_1, v_2, v_3 \in V$ Knoten, die einen gerichteten Pfad $P = (v_1, v_2, v_3)$ in G bilden. Dann transformiert die Pointer-Push&Pull-Operation $\mathcal{PP}_P$ den Graphen G zum Graphen $\mathcal{PP}_P(G) = (V, E')$, wobei*

$$E' = (E \setminus \{(v_1, v_2), (v_2, v_3)\}) \cup \{(v_1, v_3), (v_2, v_1)\} \; .$$

Die Pointer-Push&Pull-Operation wird in Abbildung 9.14 dargestellt. Anhand der Abbildung lässt sich leicht sehen, dass diese Operation vom Prinzip her einer Pointer-Pull-Operation zwischen den Knoten v_1 und v_2 und einer Pointer-Push-Operation zwischen den Knoten v_1 und v_3 entspricht.

Wird diese Operation wiederholt auf zufällige Pfade im Graphen angewendet, so hat sie vergleichbare Eigenschaften wie die 1-Flipper-Operation. Es ist allerdings noch festzulegen, wie der Random Walk einer Pointer-Push&Pull-Operation durchgeführt wird. Am sinnvollsten ist es, in jedem Schritt einer zufälligen der d Kanten

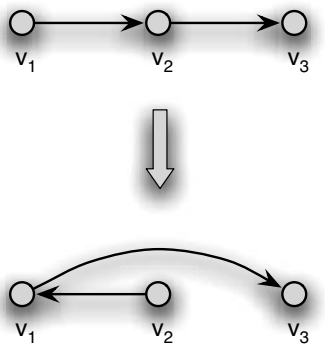


Abb. 9.14. Die Pointer-Push&Pull-Operation

eines Knotens zu folgen. Wir bezeichnen dies als *kantenorientierten* Random Walk. Alternativ könnte bei einem *knotenorientierten* Random Walk in jedem Schritt einer der höchstens d Nachbarknoten gewählt werden.

Kommen wir zurück zu den Eigenschaften der Pointer-Push&Pull-Operation. Angewendet auf einen Graphen $G \in \mathcal{MDG}_{n,d}$, bewahrt sie offenbar den Zusammenhang von G, da alle beteiligten Knoten miteinander verbunden bleiben. Weiterhin ist die Operation symmetrisch[2], d.h., für Graphen $G, G' \in \mathcal{MDG}_{n,d}$ gilt

$$Pr[G \xrightarrow{\mathcal{PP}} G'] = Pr[G' \xrightarrow{\mathcal{PP}} G] \; .$$

Da auch jeder Graph $G \in \mathcal{MDG}_{n,d}$ durch eine Folge von Pointer-Push&Pull-Operationen erreicht werden kann, gilt — ähnlich zur 1-Flipper Operation — folgendes Theorem:

Theorem 9.7. *Sei $G' \in \mathcal{MDG}_{n,d}$ ein beliebiger zusammenhängender gerichteter Multi-Graph mit regulärem Ausgrad d. Dann wird durch wiederholtes Anwenden zufälliger Pointer-Push&Pull-Operationen mit knotenorientiertem Random Walk im Limes jeder Graph $G \in \mathcal{MDG}_{n,d}$ mit der gleichen Wahrscheinlichkeit erzeugt, d.h.,*

$$\lim_{t \to \infty} P[G' \xrightarrow{t} G] = \frac{1}{|\mathcal{MDG}_{n,d}|}.$$

Die Pointer-Push&Pull-Operation erzeugt also echt zufällige Graphen aus der Menge $\mathcal{MDG}_{n,d}$. Wird anstelle des knotenorientierten Random Walk ein kantenorienter Random Walk verwendet, so werden die Graphen aus $\mathcal{MDG}_{n,d}$ nicht mehr mit uniformer Wahrscheinlichkeit erzeugt, sondern Graphen aus $\mathcal{MDG}_{n,d}$ mit wenig oder keinen Multi-Kanten, und Schleifen treten mit etwas höherer Wahrscheinlichkeit auf (für Details siehe [77]).

[2] Tatsächlich gilt die Symmetrie nur im Fall des knotenorientierten Random Walks. Betrachtet man jedoch kantennummerierte Multi-Graphen, so gilt die Symmetrie für den Fall des kantenorientierten Random Walks. Für Details siehe [77].

Aus praktischer Sicht, d.h., wenn ein solcher Multi-Graph als Netzwerk eingesetzt wird, erscheint das mögliche Auftreten von Multi-Kanten und Schleifen als Verschwendung von Ressourcen, da durch diese z.B. weder der Zusammenhang des Graphen vergrößert, noch der Durchmesser verringert wird. Tatsächlich ist der Effekt der Mulit-Kanten und Schleifen jedoch beschränkt. Betrachtet man nämlich einen einzelnen Knoten v eines zufälligen Graphen $G \in \mathcal{MDG}_{n,d}$, so ist die Wahrscheinlichkeit, dass v weder Schleifen besitzt noch Startpunkt von Multi-Kanten ist, mindestens $1 - \frac{d}{n-d-1}$. Da wir in der Praxis $d \ll n$ annehmen können, ist die Wahrscheinlichkeit, dass ein bestimmter Knoten von Schleifen oder Multi-Kanten betroffen wird, sehr klein — und auch wenn das der Fall sein sollte, so ist dies nur von kurzer Dauer, da der Graph ständig verändert wird.

Betrachten wir nun die Umsetzung der Pointer-Push&Pull-Operation in einem Peer-to-Peer-Netzwerk. Jeder Peer wird zu zufälligen Zeitpunkten eine Pointer-Push&Pull-Operation initiieren. Sei p_1 dieser Peer. Der Ablauf der Operation geschieht dann in drei Schritten:

Schritt 1: Peer p_1 kontaktiert einen zufälligen Peer p_2 seiner Nachbarschaftsliste und teilt ihm den Beginn der Operation mit.

Schritt 2: Peer p_2 empfängt die Nachricht von p_1 und wählt einen zufälligen Peer p_3 aus seiner Nachbarschaftsliste. Weiterhin ersetzt p_2 den Eintrag p_3 durch p_1 an der entsprechenden Stelle in seiner Nachbarschaftsliste und sendet als Antwort die Adresse des Peers p_3 zurück an p_1.

Schritt 3: Peer p_1 empfängt die Antwort von p_2 und ersetzt seinerseits den Eintrag p_2 durch p_3 in seiner Nachbarschaftsliste.

Innerhalb dieser drei Schritte werden lediglich zwei Nachrichten zwischen den Peers p_1 und p_2 ausgetauscht. Des Weiteren enthalten diese Nachrichten lediglich die Adresse eines Peers. Somit ist eine Pointer-Push&Pull-Operation vom Kommunikationsaufwand nicht teurer als die in dynamischen Netzwerken obligatorische periodische Überprüfung der Nachbarschaft. Ferner lässt sich die Pointer-Push&Pull-Operation mit dieser Überprüfung kombinieren, so dass letztendlich kein oder kaum zusätzlicher Netzwerkverkehr entsteht.

Auch das Einfügen und Entfernen von Peers in gerichteten Multi-Graphen mit regulärem Ausgrad gestaltet sich viel einfacher als im Fall von ungerichteten Graphen. Tritt ein neuer Peer dem Netzwerk bei, so reicht es im Prinzip, wenn er einen Peer in seine Nachbarschaftsliste aufnimmt und diesen Eintrag $d - 1$-mal dupliziert. Durch Anwenden der Pointer-Push&Pull-Operation wird dennoch ein zufälliges Netzwerk entstehen. Auch wenn ein Peer ohne Vorankündigung ausfällt, lässt sich das einfach lösen. Sobald ein Nachbar sich als tot herausstellt, wird dieser mit der Kopie eines anderen Eintrags der Nachbarschaftsliste ersetzt. Wieder wird eine Folge von Pointer-Push&Pull-Operationen beim simultanen Ausfall vieler Peers entstehende Probleme bereinigen.

Simulationen legen nahe, dass sowohl die 1-Flipper-Operation als auch die Pointer-Push&Pull-Operation schnell zur uniformen Wahrscheinlichkeitsverteilung

konvergiert. Schnell heißt hier, dass $\mathcal{O}(dn \log n)$ parallele verteilte Operationen ausreichen, bis der Graph rein zufällig wird. Ein mathematischer Beweis hierfür wurde bislang noch nicht erbracht. Bewiesen ist jedoch, dass mit Hilfe solch einfacher Operationen robust und selbstorganisierend Zufalls-Netzwerke unterhalten werden können. So könnte diese Operation zum Beispiel ohne Weiteres in Gnutella für den Topologieaufbau bzw. die Unterhaltung der Topologie verwendet werden.

9.3 Topologie-Management durch Selbstorganisation

Durch einen ähnlichen Ansatz kann durch Selbstorganisation der Verbindungsstruktur auch so genanntes *Topologie-Management* (*T-Man*) durchgeführt werden. Dieser allgemeine Ansatz stammt von Ozalp Babaoglu, Mark Jelasity und Alberto Montresor [78, 79].

Die Methode unterscheidet einen aktiven und einen passiven Prozess (*Thread*). Im aktiven Thread wird ein Peer ausgewählt, der seinem Nachbarn eine Auswahl seiner Nachbarliste zusendet. Dieser empfängt diese Liste im passiven Thread und untersucht sie dahingehend, ob er dadurch seine Situation im Netzwerk verbessern kann. Dann wählt der Peer gewisse Knoten aus und sendet sie dem aktiven Peer, der aus dieser Liste wiederum versucht, seine Verbindungsstruktur zu verbessern.

Ein Beispiel ist die Erstellung eines Rings, der die Knoten nach Winkeln sortiert. Hierzu versucht ein Peer immer den kleinsten relativen Winkel nach links und rechts zu erhalten. Eine weitere Aufgabe kann der Bau eines vollständigen Baumes oder eines Gitter- oder Torusnetzwerks sein. In Abbildung 9.15 ist dargestellt, wie T-Man einen Torus aufbaut.

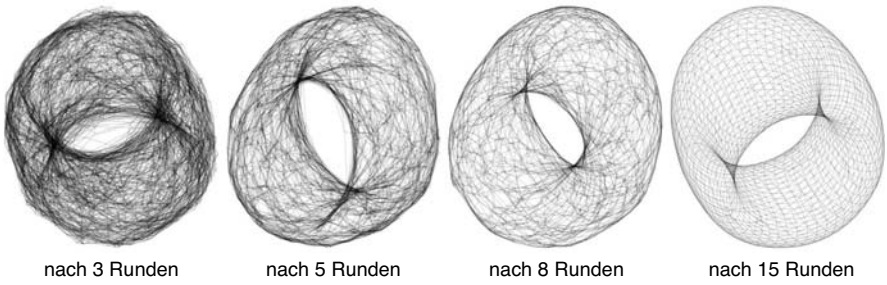


Abb. 9.15. T-Man erzeugt durch Selbstorganisation die zugrunde liegende Torusmetrik [79].

In Abbildung 9.16 werden die verschiedenen Konvergenzzeiten von T-Man für den Binärbaum, Ring und Torus verglichen. Dieser Ansatz ist jedoch nicht auf die Erzeugung solcher Strukturen beschränkt. In [80] zeigen die Autoren, wie T-Man einen selbstorganisierenden Prozess darstellt, der den Aufbau und Unterhalt eines Chord-Peer-to-Peer-Netzwerks mit Hilfe dieser Operationen sicherstellt. Somit wird keine

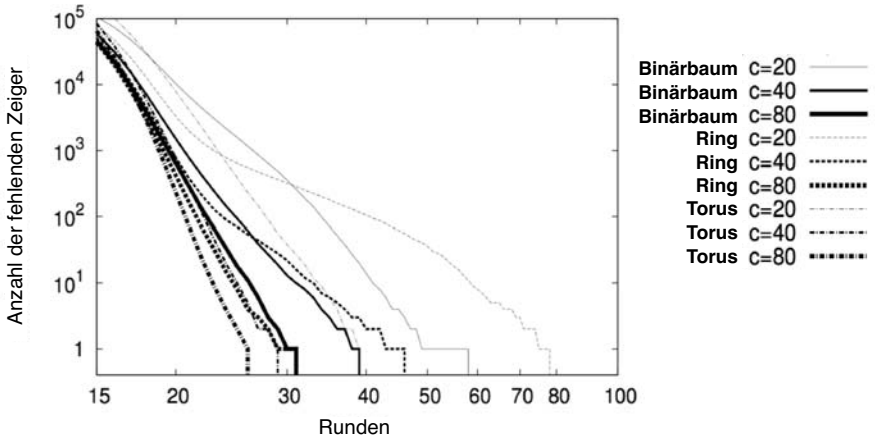


Abb. 9.16. In Abhängigkeit von der gesuchten Metrik und der Größe c des *Views* variiert die Konvergenzzeit von T-Man für $N = 2^{17}$ Knoten [79].

dedizierte Operation zum Einfügen oder Reparieren des Chord-Netzwerks mehr notwendig.

Leider basieren die beschriebenen Methoden noch vollkommen auf einer empirischen Analyse mit Simulationen. Für eine solch wichtige Aufgabe wäre es wünschenswert, dass Operationen für diese beweisbar die gewünschte Grapheigenschaft konstruieren.

10

Sicherheit

Please accept my resignation. I don't want to belong to any club that will accept me as a member.
Groucho Marx.

Bevor wir uns den spezifischen Sicherheitsaspekten von Peer-to-Peer-Netzwerken zuwenden, werden wir einen sehr kurzen Überblick über elementare Methoden der Kryptographie geben.

10.1 Methoden der Kryptographie

Die klassische Verschlüsselungsmethode ist das *symmetrische Verschlüsselungsverfahren*. Bekannte Verfahren sind hierbei z.B. *Cäsars Code*, DES (*Digital Encryption Standard*) und AES (*Advanced Encryption Standard*). Solche Verfahren bestehen aus den Funktionen f und g sowie einem geheimen Schlüssel S. Ein Text T wird nun zum Code C verschlüsselt durch Anwenden von f:

$$f(S,T) = C \ ,$$

wobei aus C ohne die Kenntnis von S nicht der Originaltext rekonstruiert werden kann. Die Entschlüsselung geschieht durch Anwenden der Funktion g mit Kenntnis desselben Schlüssels k:

$$g(S,C) = T \ .$$

In der Regel werden für f und g bekannte Verfahren verwendet, so dass die Sicherheit dieser symmetrischen Verschlüsselungsverfahren ausschließlich auf der Geheimhaltung des Schlüssels S beruht. Dieser muss auf einem separaten sicheren Weg übertragen werden.

Anders verhält es sich hier bei den *asymmetrischen Verschlüsselungsverfahren*, wie z.B. *RSA* von Ronald Rivest, Adi Shamir, Lenard Adleman [81] oder dem *Diffie-Hellman*-Schema [82] oder *PGP* (*Pretty good privacy*) [83]. Hier gibt es zwei Schlüssel: einen geheimen Schlüssel S und einen öffentlichen Schlüssel P.

Der geheime Schlüssel und der öffentliche Schlüssel werden von dem (zukünftigen) Empfänger einer verschlüsselten Nachricht zueinander passend erzeugt. Dann wird der öffentliche Schlüssel allen potenziellen Sendern mitgeteilt. Diese verwenden ihn dann zur Verschlüsselung der Nachricht T in den Code C:

$$f(P,T) = C \ .$$

Der Empfänger kann dann als einziger durch die Kenntnis des geheimen Schlüssels S diese Nachricht entschlüsseln :

$$g(S,C) = T \ .$$

Hierbei sind die Verschlüsselungsfunktion f und die Entschlüsselungsfunktion g in der Regel veröffentlicht. Die Sicherheit des Verfahrens beruht hier auf der Schwierigkeit den geheimen Schlüssel aus der Kenntnis des öffentlichen Schlüssels zu rekonstruieren (oder aus einer Menge von Text/Code-Paaren eine Gesetzmäßigkeit in der Entschlüsselungsfunktion aufzudecken).

Eine weitere wichtige Methode der Kryptographie sind die so genannten *digitalen Unterschriften* (*Digital Signatures*). Diese beruhen zumeist auf asymmetrischen Verschlüsselungsverfahren, wie z.B. *RSA*. Der *Unterzeichner* erzeugt auch hier einen geheimen Schlüssel S und einen öffentlichen Schlüssel P sowie das Komprimat K des Texts T, z.B. durch eine kryptographische Hash-Funktion h:

$$K = h(T) \ .$$

Nun berechnet der Unterzeichner die Funktion

$$g(S,K) = U \ .$$

Der Empfänger kann dann mittels des *öffentlichen Schlüssels* P und des *Textes* T die *Unterschrift* U verifizieren, indem er die folgende Gleichheit überprüft:

$$f(P,U) = h(T) \ .$$

Ein Problem hierbei ist, dass die Sicherheit des Verfahrens durch die Veröffentlichung von Text/Code-Paaren gefährdet ist, insbesondere wenn der Angreifer gewählte Dokumente zur Unterschrift vorlegt (*chosen message attack*). Dieses Problem wurde durch ein Signaturschema von Shafi Goldwasser, Silvio Micali und Ronald Rivest [84] gelöst, das unter bestimmten Annahmen nachweisbar sicher ist.

Eine *kryptographische Hash-Funktion* hat weder öffentliche noch private Schlüssel. Sie bildet ein Dokument auf eine eindeutige Ausgabe fester Länge ab. Eine kryptographische Hash-Funktion h ist genau dann sicher, falls für ein Dokument T kein anderes Dokument T' gefunden werden kann, so dass $h(T) = h(T')$. Mathematisch gesehen ist das natürlich unmöglich, da es beliebig viele Dokumente gibt und nur endlich viele Hash-Werte, da deren Länge begrenzt ist. In der Praxis sieht das aber anders aus, da der Hash-Bereich größer als 128 Bit gewählt wird. Damit ist es für viele Funktionen praktisch unmöglich, verschiedene Dokumente mit gleichen Hash-Wertene zu finden. In einem solchen Fall spricht man auch von einer Kollision.

Geläufige kryptographische Hash-Funktionen sind MD-5 [36] (*Message Digest*), für welche kürzlich ein Algorithmus zur Erzeugung von Kollisionen entwickelt wurde [85]. Diese Attacke ist tatsächlich auch praktisch relevant. Damit ist MD-5 für den praktischen Einsatz nicht mehr zu empfehlen. SHA-2 (*Secure Hash Function*) wird momentan (noch) als sicher angesehen [38], SHA-1 [37] dagegen nicht mehr.

10.2 Sicherheitsanforderungen in Peer-to-Peer-Netzwerken

Zumeist sind Peer-to-Peer-Netzwerke darauf ausgelegt, offen und autonom zu sein. Dies scheint in einem Widerspruch zu Sicherheitsanforderungen zu stehen. Schließlich arbeiten Peer-to-Peer-Netzwerke in einem potenziell feindlichen Umfeld: dem Internet.

Als Anforderungen an ein sicheres Peer-to-Peer-Netzwerk kann man folgende Punkte formulieren:

- *Verfügbarkeit*
  Ein großes Problem für Peer-to-Peer-Netzwerke stellen Dienstverweigerungsangriffe dar (*Denial of Service Attack* — DOS). Hier „bombardiert“ ein Peer oder eine große Menge von koopierenden Peers (*Distributed Denial of Service Attack* — DDOS) ausgewählte Peers mit Anfragen nach einem Dokument. Dies kann zum Beispiel in Client-Server-Architekturen zu einem Kollaps des Servers führen (*chosen victim attack*).
  Eine andere Angriffsvariante ist, dass ein bösartiger Peer sich in das Netzwerk einreiht. Dann erzeugt er eine Menge von Scheinanfragen und blockiert so das ganze Netzwerk. Typisch ist auch ein Angriff durch Ausnutzung von Protokollfehlern und -schwächen. Auf solche Angriffe kann mit der Nachbesserung der Netzwerksoftware reagiert werden.
  Für die Lösung der *Infiltration bösartiger Peers* verweisen wir auf die nachfolgende Diskussion von Sybil-Attacken und Byzantinischen Generälen.
- *Dokumentauthentifikation*
  Wenn man in einem Peer-to-Peer-Netzwerk ein Dokument bezieht, dann kann es sein, dass dieses Dokument in mehrfacher Kopie mit leichten Veränderungen vorhanden ist. Welches Dokument ist dann authentisch, welches ist gefälscht, nachgemacht oder verfälscht? Eine Lösung kann es sein, *digitale Unterschriften* zu verwenden. Diese beantworten aber nicht die Frage, welches Dokument zuerst da war.
  Wie kann man nachweisen, dass eine Datei älter ist als eine andere? In der wirklichen Welt hilft die Physik weiter (C14-Methode, Vergilbung). In der digitalen Welt funktioniert das natürlich nicht. Man kann mit herkömmlichen Methoden höchstens nachweisen, dass ein Dokument jung ist (aber nicht, dass es alt ist), zum Beispiel durch Referenz auf aktuelle, nicht vorhersehbare Ereignisse. (Es gibt aber durchaus Ansätze für Systeme, die das Alter von Dokumenten nachweisen können.)

- *Anonymität*
  Eine Motivation für die Anonymität ist nicht nur die Verhinderung des berechtigten Zugriffs staatlicher Verfolgungsbehörden gegen die Verletzung von Urheberschutzgesetzen. Das Internet ist weltumspannend und oftmals das einzige Medium, das der Zensur und Verfolgung in Diktaturen entzogen ist. Der Aufrechterhaltung der Anonymität in Peer-to-Peer-Netzwerken wird hier ein eigenes Kapitel gewidmet.
- *Zugangskontrolle*
  Peer-to-Peer-Netzwerke werden aber auch in geschützten Bereichen, wie in Unternehmen und beim Militär, eingesetzt, die über das Internet eine sichere, verteilte und robuste Netzwerkstruktur unterhalten wollen. Eine mögliche Anwendung sind zum Beispiel Peer-to-Peer-Netzwerke mit kostenpflichtigen Inhalten. Eine Lösung kann hierzu eine zentrale Authorisierung sein. Auch kann man virtuelles Geld verwenden (das ebenfalls zentral authentifiziert ist). Bei verteilten Lösungen ergeben sich die Probleme der Identifizierung und eine Interessenskollision mit dem Prinzip der Anonymität.

Als Maßnahmen gegen Attacken unterscheiden wir die Entdeckung der Attacke, ihre Handhabung, die Systemwiederherstellung nach einer Attacke und natürlich im besten Falle die Verhinderung einer Attacke.

10.3 Die Sybil-Attacke

Mit dem Namen „Sybil“ kann kaum jemand in Europa etwas verbinden — ganz im Unterschied zu den USA. Im Jahr 1973 erschien das Buch *Sybil* von Flora Rheta Schreiber, das von einer Frau mit 16 separaten Persönlichkeiten handelte (mit 16 unterschiedlichen Namen): Sybil war eine Aushilfslehrerin mit Zeitaussetzern, Peggy war ein neun Jahre altes, wütendes, verängstigtes Mädchen, Vicki sprach fließend französisch, Vanessa spielte Klavier und war mit den Nachbarn befreundet, Marsha war eine dunkle Persönlichkeit mit Selbstmordabsichten usw. Das Buch und der darauf beruhende Film aus dem Jahr 1976 geht auf einen tatsächlichen Fall von multipler Persönlichkeitsstörung (mehrfacher Schizophrenie) zurück. Bis heute ist umstritten, inwieweit diese Krankheit bei Menschen wirklich existiert.

Bei Peer-to-Peer-Netzwerken ist eine absichtliche Persönlichkeitsstörung jedenfalls ein probates Mittel, um die Strukturen zu attackieren. Hierzu gibt sich ein Peer als eine Vielzahl virtueller Peers aus und wird dadurch an mehreren Stellen im Netzwerk aktiv.

Was kann eine Sybil-Attacke bewirken?

Ein Netzwerk kann durch eine Sybil-Attacke den Zusammenhalt verlieren. Dies betrifft beispielsweise CAN, Chord, Viceroy, Pastry und Tapestry, aber nicht zwingend Gnutella. Außerdem können Mehrheitsabstimmungen über den Zustand des Netzwerks gestört werden. Das betrifft insbesondere die Mehrheitsfrage: Verhält sich ein

Peer korrekt? Diese Frage ist entscheidend für die Lösung des Problems der *Byzantinischen Generäle*.

Außerdem können durch eine Sybil-Attacke Anfragen im Netzwerk weitestgehend observiert werden. Das Netzwerk kann absichtlich verlangsamt oder gar ganz zerstört werden. Natürlich lässt sich mittels einer Sybil-Attacke auch einen Denial-of-Service-Angriff starten.

Wir halten daher fest: Sybil-Attacken greifen Peer-to-Peer-Netzwerke wirkungsvoll an.

Wie kann man Sybil-Attacken abwehren?

Ein naheliegender Ansatz ist eine zentrale Authorisierungsstelle, welche die Identität der teilnehmenden Peers bestätigt. Diese zentrale Instanz authentifiziert die Existenz eines Teilnehmers und die Gültigkeit seiner öffentlichen Schlüssel durch eine digitale Unterschrift. Jeder Peer kann sich dadurch von der Existenz des Peers überzeugen. Damit kann natürlich kein anonymes Peer-to-Peer-Netzwerk betrieben werden.

Problematisch sind dezentrale Authorisierungsstrategien. Erlaubt man z.B. einem Peer auch nur eine kleine Menge von anderen Peers zu authorisieren, dann kann Peer 1 den Peer 2 authorisieren. Dieser authorisiert Peer 3, der dann Peer 4 usw. Damit steht einer Sybil-Attacke nichts mehr im Wege.

Douceurs Ansatz zur Abwehr

Ein interessanter Ansatz zur Abwehr von Sybil-Attacken wurde von John Douceur [86] entwickelt. Die Annahmen sind hier, dass Peers einzelne Rechner sind, die einer Person unterstehen. Außerderm verfügen Einzelpersonen nicht über unbeschränkte Rechenressourcen.

Die Lösungsstrategie ist nun, dass alle Teilnehmer des Peer-to-Peer-Netzwerks ein bestimmtes mathematisches Problem (*Challenge*) lösen müssen. Zum Beispiel müssen sie für verschiedene Werte y den Wert x finden mit $h(x) = y$, wobei h eine kryptographisch sichere Hash-Funktion ist. Hierbei wurde $y = h(x)$ von einem herausfordernden Peer (*Challenger*) gewählt. Die Schwierigkeit der Aufgabe lässt sich durch die teilweise Bekanntgabe der Bits von x einstellen. Innerhalb einer gewissen Zeit kann jeder virtuelle Peer nur eine bestimmte Anzahl von solchen Challenges lösen.

Der Vorteil dieser Strategie ist, dass hiermit ein Ansatz vorhanden ist, überhaupt Sybil-Attacken zu begegnen.

Nachteile

Der Hauptnachteil ist sicherlich die unglaubliche Verschwendung von Rechenressourcen (Energie, Zeit, Rechenleistung etc.). Andererseits stehen einem Angreifer eventuell durch Einbruch in fremde Rechner enorme Rechenkapazitäten zur Verfügung. Daneben sind die den Benutzern zur Verfügung stehenden Rechenressourcen enorm heterogen. Studenten an Universitäten können durch Pool-Recher über große Rechenkapazitäten verfügen, staatliche Einrichtungen verfügen über

noch größere Ressourcen (siehe Motivation). Wenig leistungsfähige Peers können durch das Challenge überfordert werden, wie zum Beispiel ältere PCs, Pocket-PCs etc. Das Challenge selbst ist eine institutionalisierte Form des Denial-of-Service-Angriffs.

10.4 Das Problem der Byzantinischen Generäle

Ein ganz typisches Problem im Netzwerkbereich ist das so genannte *Problem der Byzantinischen Generäle* [87]. Im oströmischen Reich besaßen die Generäle immer Ambitionen Kaiser zu werden und verfolgten daher mit ihren Armeen mitunter ausschließlich eigennützige Ziele. Daher konnten sich Generäle im Falle eines Krieges nicht unbedingt auf ihre Kollegen verlassen. Wir modellieren hier den Fall von drei Generälen.

Drei Byzantinische Armeen stehen unter dem Kommando dreier Generäle bereit, die gegnerische Burg zu erobern. Die Armeen sind räumlich getrennt und kommunizieren nur über Boten. Die militärische Lage ist die Folgende: Greift nur eine Armee an, so verliert sie. Greifen zwei an, so gewinnen diese. Greift keine an, so gewinnen die Armeen auch, da durch die fortwährende Belagerung die Burg ausgehungert wird. Jetzt ist aber ein General übergelaufen und man weiß nicht, welcher es ist. Insbesondere kann dieser General durch widersprüchliche Botschaften die anderen verwirren.

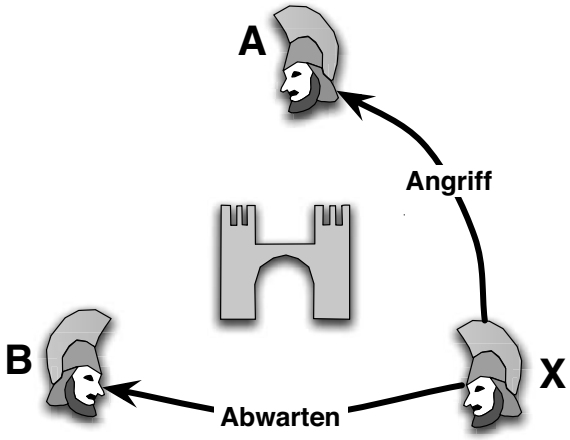


Abb. 10.1. Der übergelaufene General X gibt widersprüchliche Anweisungen.

Wie in Abbildung 10.1 gezeigt, versucht der übergelaufene General X den General A zum Angriff zu überreden und General B zum Abwarten. Wenn nun B diesen Befehl A übermittelt und A den gegenteiligen Befehl an B, so können die beiden Generäle diesen Widerspruch nicht auflösen. Schlimmer wiegt der Umstand, dass weder

B noch A den General X als Überläufer überführen können oder als Übermittler widersprüchlicher Befehle, da dieser jeweils einen der anderen Generäle beschuldigen kann.

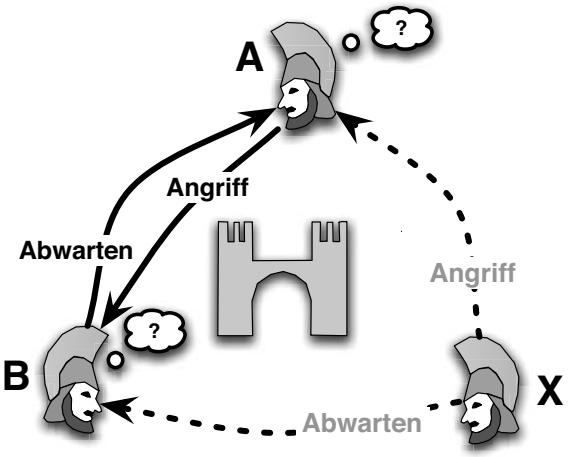


Abb. 10.2. Unauflösbare Verwirrung unter den Generälen.

Theorem 10.1. *Das Problem der drei Byzantinischen Generäle kann nicht gelöst werden. Für vier Generäle ist das Problem lösbar.*

Wir werden die Strategie für vier Generäle kurz skizzieren. Im Falle von vier Generälen müssen alle drei ehrlichen Generäle angreifen oder alle drei abwarten. Dieses Problem kann auf das Ein-General-und-drei-Offiziere-Problem reduziert werden, siehe Abbildung 10.3.

Nun sei der General loyal und einer der drei Offiziere ein Überläufer. So gibt der General konsistente Anweisungen für den Angriff. Diese geben die Information an alle anderen Offiziere weiter. Die wiederum berechnen eine Mehrheitsentscheidung, die durch einen der drei Offiziere nicht behindert werden kann, siehe Abbildungen 10.3 und 10.4.

Ist der General nicht loyal, so werden seine (widersprüchlichen) Befehle korrekt unter den Offizieren ausgetauscht. Diese folgen dann der Mehrheitsentscheidung wie bei einem loyalen General, der damit ein gemeinsames Handeln der Offiziere nicht verhindern kann, siehe Abbildung 10.5.

Im Allgemeinen gilt das folgende Theorem.

Theorem 10.2. *Falls m Generäle Überläufer sind, müssen mindestens $2m + 1$ Generäle ehrlich sein, damit das Problem der Byzantinischen Generäle lösbar ist.*

Diese Schranke ist genau, wenn keine kryptographischen Annahmen gemacht werden, wenn also asymmetrische Kryptographieverfahren nicht verfügbar sind. Im

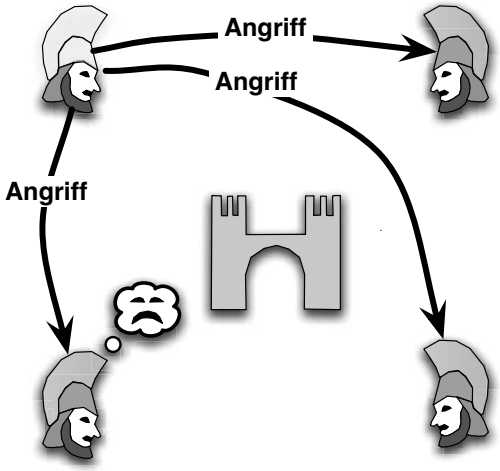


Abb. 10.3. Der loyale General gibt konsistente Befehle.

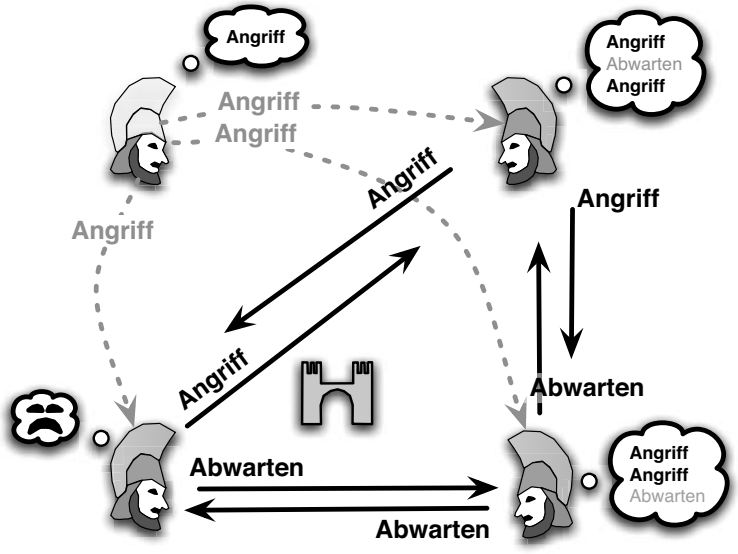


Abb. 10.4. Eine Mehrheitsentscheidung unter den drei Offizieren kann von einem Überläufer nicht gestört werden.

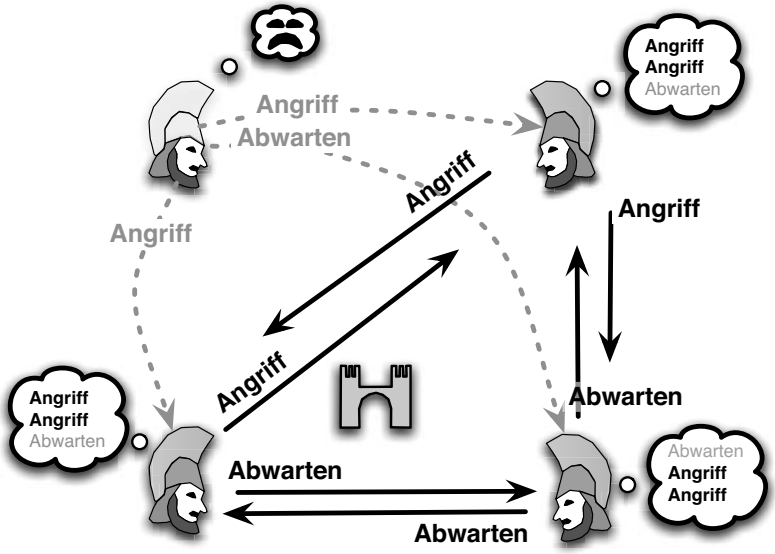


Abb. 10.5. Selbst ein verräterischer General kann das gemeinsame Handeln der Offiziere nicht verhindern.

strengen Berechenbarkeitsbegriff der Rekursionstheorie ist jedes asymmetrische Verfahren unsicher, da man „nur“ jeden geheimen Schlüssel ausprobieren muss, der zum öffentlichen Schlüssel passen könnte. Das berücksichtigt aber nicht die kombinatorische Vielfalt der verschiedenen möglichen Schlüssel und das Unvermögen von Rechenmaschinen, jede Möglichkeit auszuprobieren. Andererseits beruht die Annahme der asymmetrischen Kryptographie auf der Annahme, dass die Komplexitätsklassen $\mathcal{P}$ und $\mathcal{NP}$ verschieden sind. Solange diese Annahme nicht bewiesen wurde, muss man befürchten, dass jeder kryptographische Code geknackt werden kann.

Theorem 10.3. *Steht ein digitales Signaturschema zur Verfügung, dann kann eine beliebige Anzahl von falschen Generälen verkraftet werden.*

Beweis. Hierbei unterschreibt jeder General seinen Befehl und in der folgenden Runde gibt jeder General alle Befehle mit Unterschriften an alle anderen Generäle weiter. Hat nun jeder den korrekten öffentlichen Schlüssel, kann jeder inkonsistente Befehl oder jede falsche Weitergabe aufgedeckt werden. □

Durch den Einsatz von digitalen Unterschriften reduziert sich das Problem auf den (bösartigen) Ausfall von Peers in einem Netzwerk, wenn die Nachbarn die Kommunikation rekonstruieren können. Das Hauptproblem ist, dass dafür $\mathcal{O}(n)$ Nachrichten pro Peer verwendet werden müssen.

10.5 Ein zensorresistentes Peer-to-Peer-Netzwerk

Amos Fiat und Jared Saia stellen in [88] ein Peer-to-Peer-Netzwerk vor, das in der Lage ist, den Ausfall von 50% aller Peers durch einen Angriff zu verkraften. Voraussetzung hierfür ist, dass der Angreifer nichts über die interne Netzwerkstruktur weiß. Weist man den Peers die Aufgaben zufällig zu, so fällt jeder Peer mit Wahrscheinlichkeit $\frac{1}{2}$ aus.

Grob beschrieben besteht das Netzwerk aus einer Butterfly-Struktur, die wir schon bei Viceroy (Seite 126) kennen gelernt haben. Jeder Knoten des Netzwerks wird jetzt durch eine Menge von Peers realisiert, die als Cluster bezeichnet werden. Die Cluster sind so genannte bipartite *Expander*-Graphen der Größe $c \log n$, welche wie folgt definiert sind.

Definition 10.4. (Bipartiter Graph) *Ein bipartiter Graph besteht aus zwei disjunkten Knotenmengen A und B, so dass jede Kante einen Knoten aus A und B verbindet.*

Definition 10.5. (Bipartiter Expander-Graph) *Ein bipartiter Graph $G = (V, E)$ mit der Zweiteilung $V = A \dot\cup B$ ist ein bipartiter Expander-Graph mit Parametern $c, k \in (0, 1)$, wenn für jede Teilmenge $X \subseteq A$ mit $|X| \leq c|A|$ die Menge der Nachbarknoten Y in B mindestens die Kardinalität $|Y| \geq k|X|$ hat.*

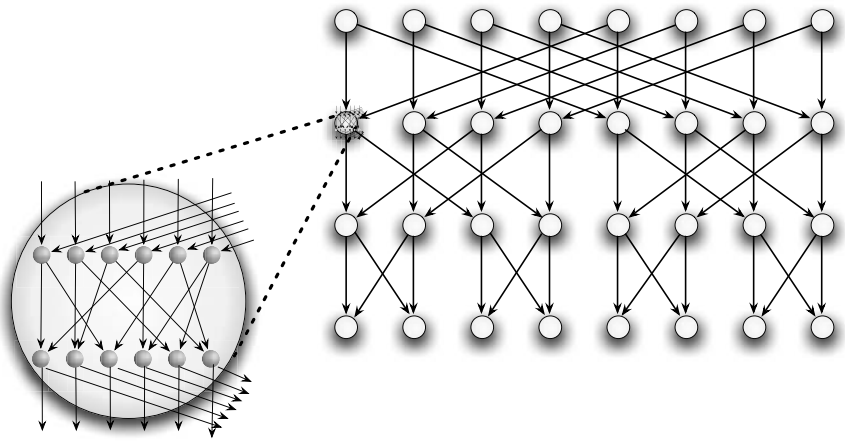


Abb. 10.6. Struktur des Peer-to-Peer-Netzwerks von Fiat und Saia[88].

Abbildung 10.6 skizziert den Aufbau des Netzwerks. Die Knotenmengen A und B sind jeweils für die Kommunikation zur nächsthöheren (A) und nächsttieferen Schicht (B) zuständig. Das Routing ist analog zum Routing im Butterfly-Netzwerk. Der Aufbau von Expander-Graphen ist ein eigener Forschungsbereich. Hierzu gibt es eine Reihe von Möglichkeiten, wie z.B. Zufallsgraphen und komplexe deterministische Konstruktionen.

Der Nutzen der Expander-Eigenschaft ergibt sich dann, wenn parallel eine Anfrage in einem Cluster bearbeitet wird. Dann findet sich zu der Teilmenge von A, die diese Anfrage erhalten hat, immer eine genügend große Teilmenge von B, so dass ein Angreifer diese nicht mit hoher Wahrscheinlichkeit kompromittiert haben kann. Hierzu wird die vorausgesetzte Unkenntnis des Angreifers über die Struktur des Netzwerks ausgenutzt. Von diesem Peer in B wird die Suchanfrage jetzt in die nächste Ebene weitergeleitet, wo sie von einer Teilmenge von korrekten Peers von A wieder entgegengenommen wird.

Auf diese Weise werden alle korrekt arbeitenden Peers eines Clusters im Routing an der Suche beteiligt, um eine Lösung des lokalen Byzantinischen Generäle-Problems zu erreichen.

Diskussion

Mit diesem Netzwerk hat man zugleich ein sehr effizientes, robustes und einfaches Verfahren. Auch lässt sich das Funktionsprinzip auf alle bisher diskutierten Verfahren verallgemeinern.

Problematisch ist nur, dass der Angreifer wenig über die interne Struktur wissen darf. Somit funktioniert dieses Verfahren wohl nur bei Systemen mit Zugangskontrolle. Denn falls der Angreifer die Struktur kennt, dann kann er jedes System mit konstantem Grad durch einen *Denial-of-Service-Angriff* auf wenigen Peers aushebeln.

Bis heute ist die Sicherheit in Peer-to-Peer-Netzwerken noch ein weites Feld mit vielen offenen Fragen, so dass dieses Kapitel nur den aktuellen Stand der Entwicklung festhalten kann.

11

Anonymität

Man is least himself when he talks in his own person. Give him a mask, and he will tell you the truth.
Oscar Wilde

Das Internet ist weltumspannend und oftmals das einzige Medium, das der vollständigen Zensur und Kontrolle in Diktaturen entzogen ist. Dennoch überwachen die dortigen Behörden den Nachrichtenverkehr. So ist die Wahrung der Anonymität für Bürger dort die einzige Möglichkeit ihr Menschenrecht, auf freie Meinungsäußerung wahrzunehmen.

11.1 Arten der Anonymität

Man kann folgende Arten der Anonymität unterscheiden in Anlehnung an [89]:

- *Autor*: Hierbei muss die Identität des Autors eines Dokuments geschützt werden. Dennoch kann es vorteilhaft sein, dass alle Dokumente eines Autors seinem Pseudonym zugeordnet werden können. Auf diese Weise ist er vor Verleumdung durch ihm zugeschriebene Dokumente geschützt.
- *Server*: Selbst wenn der Server mit den Dokumenten sich im sicheren Ausland befindet, ist es möglich, den Zugriff auf diesen zu unterbinden. Hierzu genügt es, die Pakete zu diesem Server aus dem Netzwerkverkehr herauszufiltern.
- *Leser*: Auch die Identität der Leser muss geschützt werden. Somit ist schon die Anfrage gefährlich. Schließlich offenbaren die Suchschlüssel eventuelle Absichten.
- *Dokument*: Sollten bei einer Hausdurchsuchung die Speichermedien dem Gegner in die Hände fallen, so muss die Natur der Dokumente unklar bleiben. Natürlich dürfen sie bei der Übertragung auch nicht entschlüsselbar sein.
- *Peer*: Auch die Teilnahme an dem Netzwerk kann gefährlich werden. Einige Systeme verwenden ein Ping-Pong-System zur Verschleierung der Wege. Ist der erste Rechner als Eintrittspunkt bekannt, so kann versucht werden, gegen diesen

vorzugehen. Auch ist in einigen Ländern die bloße Verwendung von kryptographischen Methoden strafbar.

11.2 Methoden

Bevor wir uns anonymisierten Peer-to-Peer-Netzwerken zuwenden, beschreiben wir elementare Methoden zur Anonymisierung.

Secret Sharing

Ein Grundbaustein für einige Peer-to-Peer-Netzwerke ist das so genannte *Secret-Sharing*. Hierbei wird ein Dokument auf n Personen verteilt. Dieses ist so kodiert, dass keine der Personen es alleine lesen kann. Zur Dekodierung müssen eine Mindestanzahl von k Personen mitwirken. Solch ein System nennt man dann ein *k-aus-n Secret-Sharing-System*.

Als Beispiel für ein solches System skizzieren wir kurz das *Secret-Sharing*-System von Blakley [90]. Angenommen, die kodierte Information wäre ein Punkt im zweidimensionalen euklidischen Raum. Jeder der n Teilnehmer erhält nun eine Gerade durch diesen Punkt, jedoch mit jeweils unterschiedlicher Richtung. Jeder der Teilnehmer kann alleine nicht bestimmen, welcher Punkt gemeint ist. Wenn aber zwei Teilnehmer ihre Information austauschen, dann können sie den Punkt als Schnittpunkt errechnen. Hierbei handelt es sich offensichtlich um ein *zwei-aus-n-Secret-Sharing-System*.

Will man ein *3-aus-n-Secret-Sharing* nach dieser Methode erhalten, so wählt man einen Punkt aus dem dreidimensionalen Raum und verteilt an die n Mitspieler jeweils verschiedene Ebenen. Kommen nur zwei Mitspieler zusammen, so können sie höchstens eine Gerade bestimmen, die durch den gesuchten Punkt führt. Drei Spieler können aber den Punkt bestimmen.

Dieses System ist recht anschaulich, aber zur eigentlichen Kodierung ungeeignet, da reelle Zahlen mit ihren auf Computern immanenten Rundungsfehlern zu Fehlern führen.

Besser ist es, wenn man die beschriebene Methode auf einen endlichen Körper der Größe 2^k betrachtet, wie sie *Evariste Galois* entdeckte. Beispielsweise kann man einen Zahlenraum der Größe 256 betrachten (beschrieben durch $F[256]$), so dass diese einem Byte entsprechen. Allgemeiner gibt es einen endlichen Körper für jede ganze Zahl. Am bekanntesten sind sicherlich die endlichen Körper der booleschen Zahlen mit $k = 0$.

Auf diesen Zahlen kann man addieren, subtrahieren, multiplizieren und dividieren, wie man es von den reellen Zahlen her kennt. Es gibt ein neutrales Element für die Addition: 0, eines für die Multiplikation: 1. Es gelten Kommutativgesetz und Distributivgesetze. Man kann auch Matrix- und Vektorrechnungen wie gewohnt durchführen. Nur haben die Zahlen selbst keine Interpretation.

Da man nur in einem endlichen Raum agiert, kann man die Ergebnisse der Addition und der Multiplikation in Tabellen speichern. Will man so einen endlichen

Körper $F[2^k]$ explizit konstruieren, so kann man diesen durch Polynome mit einer Variablen vom Grad $k-1$ beschreiben, wobei die Koeffizienten 0 oder 1 sind. Die Addition wird durch Polynomaddition beschrieben, wobei das Ergebnis modulo 2 genommen wird. Bei der Multiplikation wird das Ergebnis (das ein Polynom vom Grad $2k$ ergeben kann) durch ein festgewähltes irreduzibles Polynom dividiert und der entstehende Rest modulo 2 geteilt. In Tabelle 11.1 ist die Multiplikationstabelle für $F[4]$ angegeben.

Tabelle 11.1. Additions- und Multiplikationstabelle für $F[4]$.

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Die Einzelheiten dieser Berechnung sind hier von nachrangiger Bedeutung (hierfür siehe Seite 261). Im Moment ist es völlig ausreichend zu berücksichtigen, dass man in diesen Zahlenräumen wie in den bekannten Zahlenräumen rechnen kann.

Man kann nun eine Ebene im n-dimensionalen Raum beschreiben durch das Produkt $V^T \cdot X = A$, wobei V und A feststehende Vektoren sind und X der Koordinatenvektor ist. Wenn man nun n dieser Gleichungen kombiniert, erhält man die Gleichung $M \cdot X = T$ für zwei Matrizen M und T, die aus den Einzelvektoren zusammengefasst wurden. Der gesuchte Vektor bestimmt sich dann aus $M^{-1}T$, wobei M genau dann invertierbar ist, wenn die Ebenen paarweise unabhängig sind. Man kann zeigen, dass in einem endlichen Feld der Größe 2^k bis zu 2^{k-1} voneinander unabhängige Ebenen existieren können.

Eine Alternative ist die Methode von Shamir [91], die auf den Stützstellen eines Polynoms beruht.

Dining Cryptographers

Mit der Methode der *Dining Cryptographers* wird ein einfaches Verfahren beschrieben, den Urheber einer Nachricht zu verschleiern. Wir wollen zuerst ein anderes Problem und seine Lösung als Einführung vorstellen: Die Bestimmung des Gesamtgehalts einer Gruppe von mindestens drei Leuten, ohne dass einer sein Gehalt preisgeben muss.

Hierzu ordnet man n Mitspieler kreisförmig an. Zuerst wählt der erste Mitspieler eine völlig willkürlich gewählte (zufällige) Zahl r. Auf diese addiert er sein Gehalt h_1 und gibt sie Spieler 2. Nun addiert jeder folgende Spieler jeweils sein Gehalt auf die Summe und gibt diese dem nächsten. Hierbei ist zu beachten, dass zwei benachbarte Spieler die Weitergabe der Zahl unbeobachtet von den anderen Spielern durchführen. (Ansonsten können diese natürlich durch Bestimmung der Differenz

der ausgehenden Zahl mit der eingehenden Zahl das geheimzuhaltende Gehalt aufdecken.) Wenn diese Summe jetzt beim ersten Spieler ankommt, so sieht der die Zahl $r + h_1 + h_2 + \ldots + h_n = g$. Er zieht für sich die Zahl r, die nur er kennt, ab und erhält die gesuchte Summe $g - r$.

Möchte man das Ergebnis den anderen auch mitteilen, so addiert jeder abermals sein Gehalt auf g und gibt diese Summe wieder reihum durch. Die Differenz der beiden empfangenen Werte entspricht dem Gesamtgehalt aller Spieler. Diese erhält jetzt jeder Spieler ohne ein einziges Einzelgehalt zu kennen, siehe auch Abbildung 11.1.

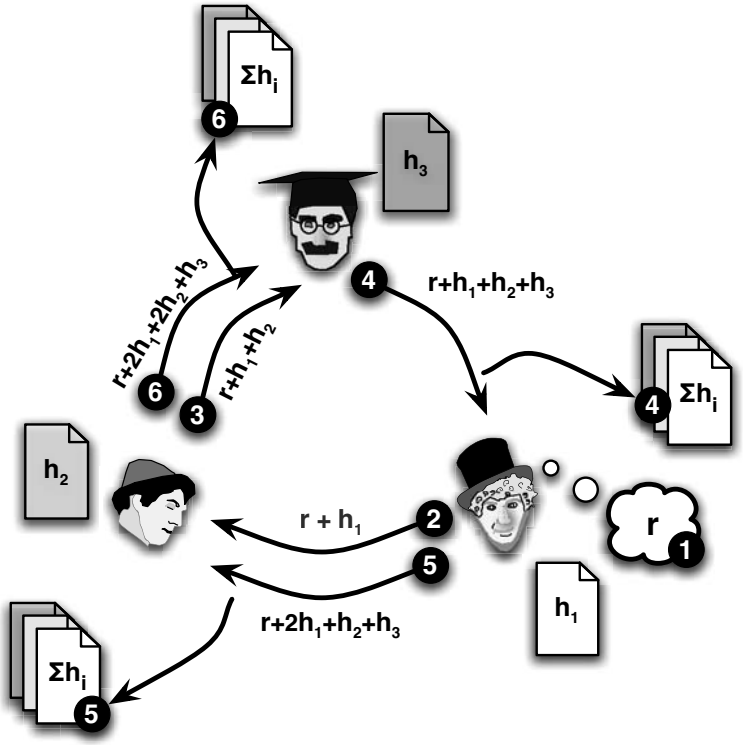


Abb. 11.1. Private Computing: Addition von n Zahlen.

Diese Technik, eine Berechnung ohne Kenntnis der Einzelergebnisse durchzuführen, nennt man auch *Private Computing*, ein Teilgebiet der Kryptographie und des Verteilten Rechnens (*Distributed Computing*).

Das Verfahren der Dining Cryptographers funktioniert nun ganz ähnlich. Die Aufgabe ist, dass einer der n Teilnehmer eine Nachricht veröffentlichen möchte. Diese wird am Ende allen bekannt sein, es soll aber unklar sein, wer die Nachricht ursprünglich verschickt hat. Hierzu ordnet man die Teilnehmer wieder kreisförmig an einem Tisch an und fordert, dass sämtliche Kommunikation nur zwischen zwei

benachbarten Teilnehmern stattfindet und dass der Inhalt zwischen den Beteiligten geheim gehalten wird.

Zu Beginn bestimmt jeder Teilnehmer i unabhängig eine zufällige Zahl x_i. Jeder Teilnehmer gibt diese Zahl seinem rechten Tischnachbarn. Dieser berechnet die Differenz aus seiner Zahl und der eben erhaltenen Zahl und erhält $x_i - x_{i-1}$. Möchte nun einer der Teilnehmer eine Nachricht m veröffentlichen, so addiert er diese auf die Differenz und erhält $x_i - x_{i-1} + m$, siehe Abbildung 11.2.

Nun veröffentlicht jeder Teilnehmer das Ergebnis dieser Rechnung, wobei natürlich m und die ursprünglichen Werte x_i weiterhin geheim gehalten werden. Jetzt kann jeder die Summe aller Werte aufaddieren, und da jeder Term x_i einmal positiv und einmal negativ auftauchten, fallen all diese heraus und es bleiben nur noch die Summen der Nachrichten übrig.

Hat nun niemand eine Nachricht senden wollen, so ist das Ergebnis 0. Hat nur einer eine Nachricht senden wollen, so können alle diese Nachricht sehen und keiner kann den Erzeuger identifizieren, solange nicht die geheimzuhaltenden Informationen aufgedeckt werden.

Natürlich funktioniert dieses Verfahren erst ab drei Mitspielern, da im Falle von zwei Mitspielern jeder weiß, ob er selbst Urheber der Nachricht ist. Außerdem müssen mindestens zwei benachbarte Mitspieler ehrlich sein, da ansonsten der Urheber der Nachricht herausgefunden werden kann.

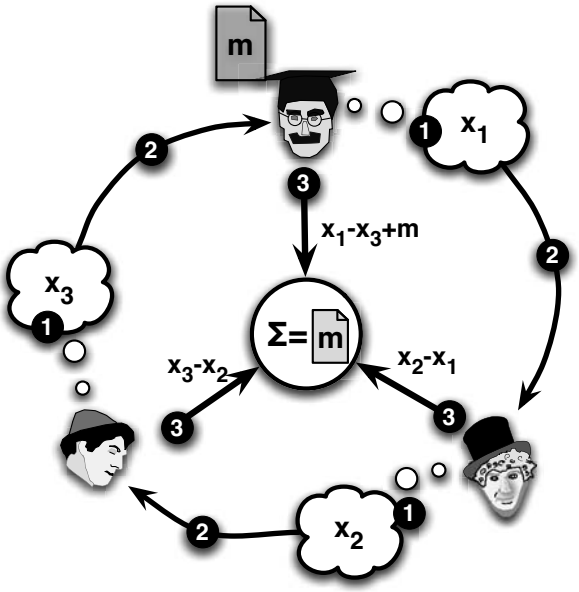


Abb. 11.2. Dining Cryptographers.

Onion Routing

Das Prinzip des *Onion Routing* von [92] geht auf die so genannten *Mix-Networks* oder *Mix-Cascades* von David Chaum [93] zurück.

Das Ziel ist jeweils der Schutz der Anonymität von Sender und Empfänger einer Nachricht. Die Angreifer sind in der Lage, den Nachrichtenverkehr zwischen den Teilnehmern im Internet zu beobachten. Zusätzlich soll die übermittelte Nachricht geheim gehalten werden.

Würde nun der Sender die Nachricht dem Empfänger direkt zusenden, so kann der Angreifer sie direkt lesen. Angenommen, der Sender verschlüsselte diese Nachricht mit dem öffentlichen Schlüssel eines asymmetrischen Kryptographieschemas des Empfängers, so könnte der Angreifer diese Nachricht zwar nicht lesen, jedoch schon die Tatsache, dass der Sender und der Empfänger direkt miteinander kommunizieren, könnte dem Angreifer genügen. Onion Routing möchte also schon das Vorhandensein der Kommunikation möglichst gut verschleiern. Hierzu werden eine Reihe von hilfreichen Servern verwendet. Außerdem wird vorausgesetzt, dass eine Reihe von Nachrichten andauernd versendet wird. Auf diese Weise soll sichergestellt werden, dass der gefährdete Nachrichtenverkehr im Hintergrundrauschen untergeht.

Wir gehen davon aus, dass die helfenden Zwischenstationen allgemein bekannt sind und dass alle ein asymmetrisches Kryptographieschema etabliert haben. Der Sender p_1 wählt jetzt eine Folge $r_1, \ldots, r_k$ von Hilfsservern zum Empfänger p_2. Hierzu kodiert er zuerst die Nachricht mit dem öffentlichen Schlüssel des Empfängers p_2.

Dann kombiniert er diese Nachricht mit der Adresse von p_2 und kodiert das Resultat mit dem öffentlichen Schlüssel von r_k. Nun fügt er die Adresse von r_k an und kodiert das Ganze mit dem öffentlichen Schlüssel von r_{k-1}. Dies wird solange wiederholt, bis die Nachricht mit dem öffentlichen Schlüssel von r_1 kodiert wird. Nun sendet der Sender die Nachricht an r_1.

Der beobachtende Angreifer kann nur eine verschlüsselte Nachricht sehen. Er weiß nichts von dem Empfänger oder von den Zwischenstationen. Genauso kennt r_1 weder den Empfänger noch den Inhalt der Nachricht. Er kann aber die Nachricht entschlüsseln und erhält so die Information, welcher Server als nächstes zu verwenden ist. An diesen Server verschickt jetzt r_1 die Nachricht. Server r_2 kann nun eine weitere Schale entfernen (daher auch der Name Onion Routing). Dieser zweite Server weiß nun nicht, wer der ursprüngliche Sender war. Auch weiß er nicht, wer die Nachricht empfangen soll (falls $k > 2$). Somit reicht er die Nachricht an den nächsen Router weiter und dieser Prozess wird iteriert, bis der Router r_k schließlich den Empfänger entschlüsselt und an diesen die für ihn verschlüsselte Nachricht schickt. Der Empfänger kann nun die Nachricht entgegennehmen und sie entschlüsseln, siehe Abbildung 11.3.

Man beachte, dass der Empfänger nicht weiß, wer ihm diese Nachricht geschickt hat. Wünscht der Sender jetzt eine anonyme Antwort, so kann er dem Empfänger eine vorgefertigte Rückantwort mitgeben, welche wieder zwiebelschalenförmig verschlüsselt ist und nur den nächsten Sprung offenbart. Der Sender würde dieser Nach-

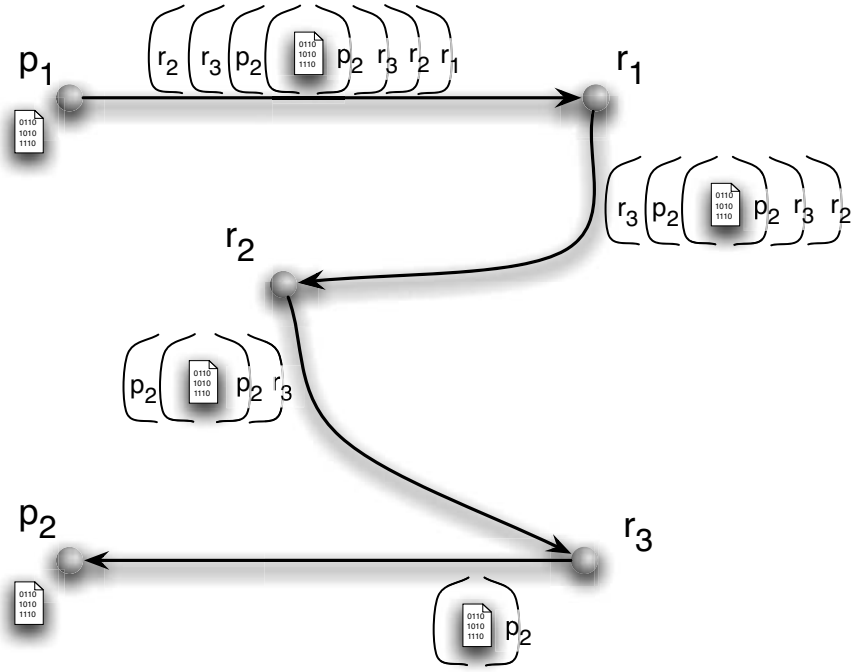


Abb. 11.3. Onion Routing.

richt noch eine andere Nachricht anhängen, die mit einem symmetrischen Schlüssel kodiert ist, die der Empfänger p_2 gerade erst erhalten hat.

Auf diese Weise kann zum Beispiel ein Client-Server-Protokoll etabliert werden, wobei der Server die Anfragen des Clients beantwortet, aber keinerlei Information über die Adresse des Clients erhält. Sicherlich wäre es in der Praxis besser, wenn man hierfür einen vertrauenswürdigen Proxy benützte. Solch ein Ansatz genügt aber bei weitem nicht den hoch gesteckten Ansprüchen an die Anonymität.

Zusätzlich zu der Route, die der Sender vorgibt, kann nun jeder Router dieses Schemas die Route einer Nachricht dadurch verwischen, dass er willkürlich weitere Router in den Zwiebelcode hineinpackt. Auf diese Weise geht die Nachricht weitere Umwege, die noch nicht einmal der Sender im Vorfeld antizipieren kann.

Im Übrigen nützte es dem Angreifer auch wenig, wenn er selbst Router in das Netzwerk einschleust. Zwar kann er erfahren, welche anderen Router Nachrichten übermitteln, jedoch erhält er immer nur verschlüsselte Informationen. Er kann sogar nicht einmal bestimmen, ob die Nachricht vom Sender kommt oder ob sie nun den Empfänger erreicht, vorausgesetzt Sender und Empfänger tauchen überhaupt in der Liste der möglichen Onion Router auf.

Es gibt verschiedene solcher Overlay-Netzwerke im Internet, die das Prinzip des Onion-Routings umsetzen. Die Betreiber verstehen diese Netzwerke aber nicht als Peer-to-Peer-Netzwerke, obgleich sie streng formal als solche anzusehen sind, son-

dern eher als Infrastrukturerweiterung des Internets, um die fehlende Fähigkeit der Anonymisierung und Verschleierung von Daten zu ermöglichen.

Das eigentliche Software-Projekt wurde in *TOR* (*The Onion Router*) umbenannt [94]. Es gibt für die wichtigsten Rechnerplattformen Client- und Server-Software. Natürlich ist TOR im eigentlichen Sinne kein Peer-to-Peer-Netzwerk, weil es eine Client-Server-Struktur gibt. Auch wird bei den Servern auf Vertrauenswürdigkeit geachtet (die aber nicht garantiert werden kann). Anfragen sind nicht anonym. Aber die Autoren, die Leser, die Speichernden und die Dokumente werden in ihrer Anonymität geschützt. Die Techniken von TOR können jedoch sehr gut in Peer-to-Peer-Netzwerken angewendet werden.

Friend-to-Friend-Netzwerke

Mittlerweile haben sich eine Reihe von Firmen darauf spezialisiert, die Nutzer von Peer-to-Peer-Netzwerken ausfindig zu machen. In einigen Ländern ist alleine die Teilnahme an einem Peer-to-Peer-Netzwerk strafbar. Wenn nun also nachgewiesen wird, dass die IP-Adresse im Zusammenhang mit einer Peer-to-Peer-Netzwerk-Kommunikation auftaucht, so ist unabhängig von der Verschlüsselung der Nachricht für den Teilnehmer mit Repressalien zu rechnen. So könnte dies zum Beispiel eine Beschlagnahmung der privaten Rechner oder die Überprüfung des Rechnerzugangs in einer Institution zur Folge haben.

Es gibt aber eine Möglichkeit dieser Art der Verfolgung zu entgehen. Der Ansatz ist das so genannte *Friend-to-Friend-Netzwerk* (auch *F2F* genannt). Hierbei unterhält man nur Netzwerkverbindungen zu Kommunikationspartnern, denen man uneingeschränkt vertraut. IP-Adressen werden nicht über diese hinaus weiter verbreitet. Wenn nun die These der kurzen Verbindungen in sozialen Netzwerken, die wir im Kapitel 9 Selbstorganisation besprochen haben, stimmt, dann kann solch ein Netzwerk mit großem Zusammenhang und kleinem Durchmesser aufgebaut werden. Damit eine netzwerkweite Kommunikation möglich wird, muss jeder Teilnehmer eine geheime Identität annehmen, unter der er erreichbar ist. Außerdem müssen die Nachrichten auf der gesamten Route kodiert werden. Denn schließlich sind auch die Angreifer im sozialen Netzwerk des Globus verbunden. Nur sind sie (hoffentlich) einige Hops von den Kommunikationspartnern entfernt.

In einem Friend-to-Friend-Netzwerk sind sicherlich längere Wege notwendig. Der Vorteil ist aber die bestmögliche Sicherheit, die nur durch sozialen Betrug, wie zum Beispiel falsche Freunde, korrumpiert werden kann.

Dark-Net

Eine Erweiterung von Friend-to-Friend-Netzwerken ist das so genannte *Dark-Net*. Solche Netzwerke sind vergleichbar mit privaten Clubs, in die nur eine ausgewählte Klientel Zutritt erlangt. Hier vertrauen die Teilnehmer allen Clubmitgliedern. Der Unterschied zum Friend-to-Friend-Netzwerk ist, dass nun benachbarte Peers nicht

unbedingt befreundet sind. Es wird aber vorausgesetzt, dass ein sozialer Mechanismus in Kraft ist, der dafür sorgt, dass alle Teilnehmer des Dark-Net diskret und zuverlässig agieren. Genauso wie bei Friend-to-Friend-Netzwerken hängt die Sicherheit von Dark-Nets nur von diesem Mechanismus ab. Der Zugang kann dann geregelt über geheim gehaltene Zugangsadressen oder spezielle Software werden. Häufiger wird aber eine Authentifikation über ein Passwort gewählt oder eine zentrale Authentifikation.

Steganographie

Natürlich sind diese vorgestellten Methoden den Gegnern anonymer Kommunikation nicht unbekannt und so erscheint es ihnen als der einfachste Weg, allen Internetnutzern, die in ihrem Kontrollbereich liegen, sämtliche kryptographischen Methoden zu verbieten. So ist es Häftlingen in Deutschland verboten, verschlüsselte Briefe zu empfangen oder zu verschicken. In China, Iran und bis vor kurzer Zeit auch in Frankreich ist es einfachen Bürgern nicht gestattet, kryptographische Methoden zu verwenden. Insbesondere kann man von China aus nicht zu bestimmten Internetadressen Pakete schicken und der Inhalt wird sicherlich kontrolliert. In solchen Ländern ist also allein das Entdecken verschlüsselter Kommunikation für die Anwender ein Problem und die Kommunikationsschnittstellen werden beaufsichtigt und kontrolliert. Dennoch gibt es einen Ausweg, in solch einem Umfeld geheim und dennoch unentdeckt zu kommunizieren.

Die Wissenschaft und Kunst der verborgenen Speicherung und Übermittlung von Informationen ist die *Steganographie*. Sie beschäftigt sich mit dem Verstecken von Botschaften und der Überprüfung der Urheberschaft von Dokumenten (durch absichtlich versteckte Botschaften). In realen Dokumenten kann man Botschaften durch Wasserzeichen oder durch Mikropunkte übermitteln, was auch eine Möglichkeit zum Nachweis der Urheberschaft darstellt.

In elektronischen Dokumenten erscheint die Aufgabe ungleich schwieriger. Es gibt aber auch hier eine Vielzahl von Möglichkeiten. Eine davon ist, dass man in einem Text unter einer Vielzahl von Möglichkeiten eine passende wählt. Hier ein Beispiel:

$$\left\{\begin{array}{c}\text{Eine}\\ \text{Die}\end{array}\right\} - \left\{\begin{array}{c}\text{Möglichkeit}\\ \text{Alternative}\end{array}\right\} - \left\{\begin{array}{c}\text{ist}\\ \text{war}\end{array}\right\} \text{unter} \left\{\begin{array}{c}\text{einer}\\ \text{der}\end{array}\right\} - \left\{\begin{array}{c}\text{Vielzahl}\\ \text{Mehrzahl}\end{array}\right\} \text{von}$$
$$\left\{\begin{array}{c}\text{Texten}\\ \text{Zitaten}\end{array}\right\} - \left\{\begin{array}{c}\text{eine}\\ \text{die}\end{array}\right\} - \left\{\begin{array}{c}\text{auszuwählen}\\ \text{zu bestimmen}\end{array}\right\}, \text{um} \left\{\begin{array}{c}\text{eine}\\ \text{die}\end{array}\right\} - \left\{\begin{array}{c}\text{geheime}\\ \text{verborgene}\end{array}\right\}$$
$$\left\{\begin{array}{c}\text{Information}\\ \text{Bedeutung}\end{array}\right\} \text{zu} \left\{\begin{array}{c}\text{chiffrieren}\\ \text{kodieren}\end{array}\right\}.$$

Es ergeben sich also für diesen Satz 2^{12} mehr oder weniger gleichbedeutende, aber alles in allem sinnvolle Sätze. Wenn nun für jedes Wort entsprechende Synonyme mit entsprechender Interpretation vereinbart worden sind, kann man sich leicht vorstellen, wie in einem unscheinbaren Text völlig unbemerkt ein anderer Text versteckt wird.

Eine andere Möglichkeit ist die Manipulation von elektronischen Audiodateien. Verändert man nämlich alle Werte, so dass sich unmerkliche Amplitudenveränderungen ergeben, wird das im allgemeinen Hintergrundrauschen, das praktisch jeder Audioaufzeichnung zu Grunde liegt, untergehen. Aber der Empfänger kann hieraus wieder die verborgene Information entnehmen.

Genauso kann man in Bilddateien durch eine geringe Manipulation der Helligkeits- oder Farbwerte verborgene Information übermitteln. Diese Technik wird bereits eingesetzt, um die Urheberrechte von Fotografen an ihren Bildern nachweisbar zu schützen. Den Bildern ist das Wasserzeichen nicht anzumerken. Aber mit der richtigen Software kann die unerlaubte Kopie eines Bildes nachgewiesen werden.

Der Vorteil der Steganographie im Bereich der Peer-to-Peer-Netzwerke liegt auf der Hand: Der Datenaustausch zwischen Peers kann unbemerkt in einer harmlosen Kommunikation versteckt werden.

Der Nachteil liegt in der Aufblähung des Datenvolumens. So kann in einer Audiodatei bei Veränderung des am wenigsten signifikanten Bits weniger als $0,5\%$ Datendichte erreicht werden. Oder anders formuliert: Um Daten zu verschleiern ist die 200-fache Datenmenge notwendig. Bei den oben skizzierten Textverschleierungsschemata ist das Verhältnis sogar noch schlechter.

Verschlüsselte Daten

Es bringt einen entscheidenden Sicherheitsvorteil, wenn der Autor seine Dokumente nicht selbst verteilen muss. Ansonsten kann man mittels der Adressierung und einer Verkehrsanalyse herausfinden, wer der Urheber war. Selbst wenn er das Routing verstecken kann, so muss er doch die Information verteilen. So kann bei vollständiger Überwachung des Netzwerks der Autor eingekreist werden. Wenn nun die Dokumente auf fremden Servern liegen, dann besteht zudem die Gefahr, dass der Server in fremde Hände gerät oder dass er gar vom Angreifer betrieben wird.

Daher kann man die Sicherheit erhöhen, indem man die Daten so verschlüsselt auf den Datenservern ablegt, dass sie auf dem Server ohne Zusatzinformation nicht gelesen werden können. Hierzu werden die Daten vom Autor verschlüsselt und die Information zum Entschlüsseln auf getrennten Servern abgespeichert. Zusätzlich kann diese verschlüsselte Datei noch mit der geheimen Identität des Autors unterschrieben werden. Damit verhindert man, dass ein fremder Autor den Auftrag zur Löschung erteilt, die Daten ändert oder die Glaubwürdigkeit des Autors mit widersprechenden Dateien erschüttert.

Diese verschlüsselten Dateien werden so vom Leser geladen und können nur entschlüsselt werden, wenn dieser sich über einen anderen Weg den Schlüssel für diese Datei geben lässt. Das kann über ein geheimes Routing direkt vom Autor oder durch Abfrage eines Indexeintrags geschehen. Die Voraussetzung hierfür ist, dass der Anfrager eine gewisse Vorinformation zu diesem Dokument hat, die dem Server nicht zur Verfügung steht. Dies kann durch Kommunikation über einen anderen Kanal geschehen, zum Beispiel über ein schwarzes Brett in einer Universität oder durch Erraten des Suchbegriffs „*Freedonian War*“.

Ist ein zu erratender Suchbegriff das fehlende Bindeglied zur Entschlüsselung, so besteht die Gefahr einer Wörterbuchattacke. Hierzu probiert der Angreifer alle relevanten Suchbegriffe aus und erhält so irgendwann den passenden Schlüssel.

Ein Nachteil der Verschüsselung von Inhalten ist, dass der Server die Wichtigkeit von Daten nicht beurteilen kann. Es könnte sich um ein vielfach vorhandenes Replikat handeln. Es könnte auch von einem Angreifer erzeugt worden sein, nur um den vorhandenen Speicherplatz mit unsinnigen Dokumenten zu belegen, so dass weniger Platz für sinnvolle Dokumente übrigbleibt. Kurzum, ein Server kann nicht ohne weiteres entscheiden, ob Dokumente gelöscht werden können.

Eine Lösungsstrategie ist die Verwendung von Haltbarkeitsdaten für Dokumente, nach deren Ablauf das Dokument gelöscht werden kann. Eine andere ist ein *Trust Management System* (Vertrauensbewertungssystem). Hierbei erhalten Benutzer die Erlaubnis, selbst Dokumente zu veröffentlichen, wenn sie ihrerseits positiv zum Erhalt des Systems beitragen.

Verschlüsselte unterschriebene Indizes

Wir haben bereits gesehen, dass die Speicherung von Sucheinträgen die Zuordnung von Leser und Datenspeicherer einfach und effizient lösen kann. Diese Indexeinträge sind aber ein kritischer Punkt für die Sicherheit. Wenn der Autor hier unverschlüsselt speichert, auf welchem Server die Information abgelegt wird, so kann der Peer, der diesen Indexeintrag verwaltet, das gesamte System überwachen. Er kann sehen, wer die Daten ablegt, wo die Daten liegen und wer danach sucht.

Nun kann man die Identität des Autors und des Abfragers durch Onion Routing verheimlichen. Der Index kann aber immer noch den Speicherort und den Inhalt verraten. Deswegen macht es Sinn, auch den Indexeintrag zu verschlüsseln. Der Suchende muss dann über eine Zusatzinformation diesen entschlüsseln. Das kann eine geheim übermittelte Nachricht oder der passende Suchbegriff sein. Der Indexserver kann mit seinen Indizes dann weder herausfinden, was gespeichert wird, noch wo die Information vorhanden ist. Zusätzlich unterzeichnet der Autor mit seiner geheimen Identität. Sollte der Index sich verändern, weil z.B. der Speicherort des Dokuments verändert wird, dann wird der speichernde Peer nur solche neuen (verschlüsselte) Indexeinträge akzeptieren, die von dem gleichen Unterzeichner stammen.

Eine elegante Möglichkeit ist, den Suchbegriff selbst zur Verschlüsselung des Sucheintrags zu verwenden. (Dies geschieht zum Beispiel bei Freenet.) Dann berechnet man, ausgehend vom Suchbegriff, den verschlüsselten Standort des Index und kann mit dem Suchbegriff selbst den Index entschlüsseln. Der entschlüsselte Index zeigt dann, wo das Dokument gespeichert ist, das man dann mit Onion Routing herunterladen kann. Natürlich besteht bei diesem Verfahren wieder die Gefahr der Wörterbuchattacken.

11.3 Free-Haven

Das *Free-Haven*-System ist ein real existierendes Netzwerk, das von Roger R. Dingledine, Michael Freedman, David Molnar, Brian Sniffen und Todd Kamin im

Jahr 2000 entwickelt wurde [89, 95]. Das Ziel ist ein verteilter Datenspeicher, der robust ist gegen Angriffe von sehr mächtigen Gegnern. Das mutmaßliche Ziel dieser Angreifer ist es, die Daten zu zerstören oder den Zugriff auf diese Daten zu unterbinden.

Das Free-Haven-System verwendet hierzu eine Gemeinschaft von Servern, die sich gegenseitig Speicherplatz bereitstellen. Die Server sind untereinander vertrauenswürdig. Es besteht aber die Gefahr, dass der Angreifer versucht, durch übermäßige Anfragen die Server lahmzulegen. Es gilt also den tatsächlichen Speicherort zu verheimlichen, so dass jede Attacke, die sich auf wenige Server konzentriert, ins Leere läuft.

Hierzu werden die Dokumente gemäß des bereits vorgestellten Secret-Sharing-Schemas auf mehrere Server verteilt. Diese Teile werden im Hintergrund zwischen den Servern ausgetauscht. Für eine Abfrage fragt ein Client einen Server mittels geschützter Kommunikation nach dem Dokument. Der Server teilt dem Anfrager den Standort einer Mindestmenge an Servern zur Rekonstruktion der Datei mit. Die Kommunikation erfolgt über Onion Routing, so dass ein Angreifer aus der Beobachtung des Verkehrs keinen Zusammenhang zwischen Anfrager und speicherndem Server konstruieren kann.
Folgende Operationen werden zur Verfügung gestellt:

- *Einfügen von Dokumenten*
  Der Autor erzeugt in einem asymmetrischen Kryptographieschema je einen öffentlichen und einen privaten Schlüssel. Dann wird das Dokument gemäß des k*-aus-*n*-Secret-Sharing-Systems* in n Teile zerlegt (n entspricht nicht unbedingt der Gesamtzahl der Server). Jeder Teil, genannt *Share*, erhält seinen eigenen öffentlichen Schlüssel. Die Daten sind mit diesem Schlüssel kodiert. Die Teile werden nun mit einem Verfallsdatum versehen und vom Autor unterzeichnet. Dann werden sie per Onion Routing auf eine vom Autor ausgewählte Menge von Servern verteilt.
  Um überhaupt die Erlaubnis für die Speicherung zu erlangen, muss ein externer Mechanismus die Erlaubnis der Publikation erteilen, da sonst ein Denial-of-Service-Angriff möglich wird. Denkbar wäre hier zum Beispiel, dass der Autor selbst einen Server betreibt.
- *Suchen und Herunterladen von Dokumenten*
  Über einen separaten Kanal (außerhalb von Free-Haven) erhält der Anfrager die Informationen zum Abrufen der Datei. Dies sind die privaten Schlüssel der Shares. Um diese Shares abzurufen, wird nun eine *Broadcast*-Anfrage mit den öffentlichen Schlüsseln über Onion-Routing mit Möglichkeit zur Rückantwort abgesetzt. Daraufhin treffen die Shares beim Anfrager ein. Dieser dekodiert zuerst alle Shares und konstruiert daraus das Dokument.
- *Widerruf und Ablauf eines Dokuments*
  Da das Verfallsdatum unverschlüsselt gespeichert wird, kann jeder Server bestimmen, ob Daten gelöscht werden können. Dokumente können auch widerrufen werden. Hierzu sendet der Autor eine signierte Aufforderung per Onion Routing an alle Server, die entsprechenden Teile zu löschen.

- *Server-Verwaltung*
  Da die Teile nicht mit einem Server identifiziert werden und da die Suche über Broadcast implementiert wird, können zwischen den Servern die Shares beliebig ausgetauscht werden. Free-Haven stellt sicher, dass die Server die Daten nicht lesen können und nicht wissen, wer welche Daten wo speichert. Der Ausfall inaktiver Server kann durch die Verwendung des k-aus-n-Secret-Sharing-Systems leicht verkraftet werden.

11.4 Free-Net

Im Gegensatz zu Free-Haven ist *Free-Net* [4] ein echtes Peer-to-Peer-Netzwerk. Es wurde von Ian Clarke, Oskar Sandberg, Brandon Wiley und Theodore Hong im Jahr 2000 gestartet. Das Ziel von Free-Net ist ein Netzwerk, das Veröffentlichung, Replikation und Suche von Dateien erlaubt und hierbei die Anonymität der Autoren und Leser wahrt.

Die Dateien werden unabhängig vom Speicherort referenziert durch verschlüsselte und unterzeichnete Indexdateien. Dadurch kann der Autor nicht zurückverfolgt werden. Diese Technik verhindert das unbefugte Überschreiben und Löschen.

Die Dateien werden (im Gegensatz zu Free-Haven) nur an einem Ort gespeichert. Zuvor werden sie verschlüsselt, so dass deren Inhalt nur durch Kenntnis der andernorts abgelegten Indexdatei in Kombination mit dem Suchbegriff entschlüsselt werden kann. Diese Dateien können außerdem noch repliziert werden, um häufige Anfragen besser zu bedienen. Außerdem werden Dateien gelöscht, wenn sie lange nicht mehr angefragt wurden (*Least Recently Used — LRU*).

Die Netzwerkstruktur von Free-Net ist stark verwandt mit der von Gnutella. Der Netzwerkaufbau geschieht durch Vermittlung der Nachbarn des Eintrittspeers. Dieser Eintrittspeer muss einem Peer, der dem Netzwerk beitreten möchte, die Mitgliedschaft in Free-Net anbieten. Bei Free-Net handelt es sich aber nicht um ein Friend-to-Friend-Netzwerk, da bei der Suche noch zusätzliche Abkürzungen gewählt werden können. Genauso wie in Gnutella ist der resultierende Grad Pareto-verteilt. Man kann sich leicht vorstellen, wie durch Infiltration, durch Inbesitznahme oder einfach durch Änderung der Motivation des Peers dieses Netzwerk beschädigt werden kann.

Speichern von Dateien

Jede Datei kann durch den kodierten Address-String und den signierten Indexschlüssel (*Signed Subspace Key*) gefunden, entschlüsselt und gelesen werden. Jede Datei kann mit der Information des Indexschlüssels zwar gefunden werden, lässt sich aber nicht ohne Adress-String dekodieren. Dadurch weiß der Server nicht, welche Daten er speichert, denn er kennt weder die Indexdatei noch den Address-String. Sollte er durch Zufall die Indexdatei finden, so könnte er diese höchstens mit einer Wörterbuchattacke entschlüsseln.

Speichern der Indexdateien

Der Address-String wird durch eine kryptographische Hash-Funktion kodiert. Der Hash-Code ist dem Peer bekannt, der die Indexdatei bestehend aus dem Address-String und dem signierten Indexschlüssel erzeugt. Diese Information ist nur durch den Hash-Code des Address-Strings zu entschlüsseln. Mit diesen Indexdateien kann die Datei gefunden und entschlüsselt werden. In Abbildung 11.4 wird das Funktionsprinzip von Free-Net dargestellt.

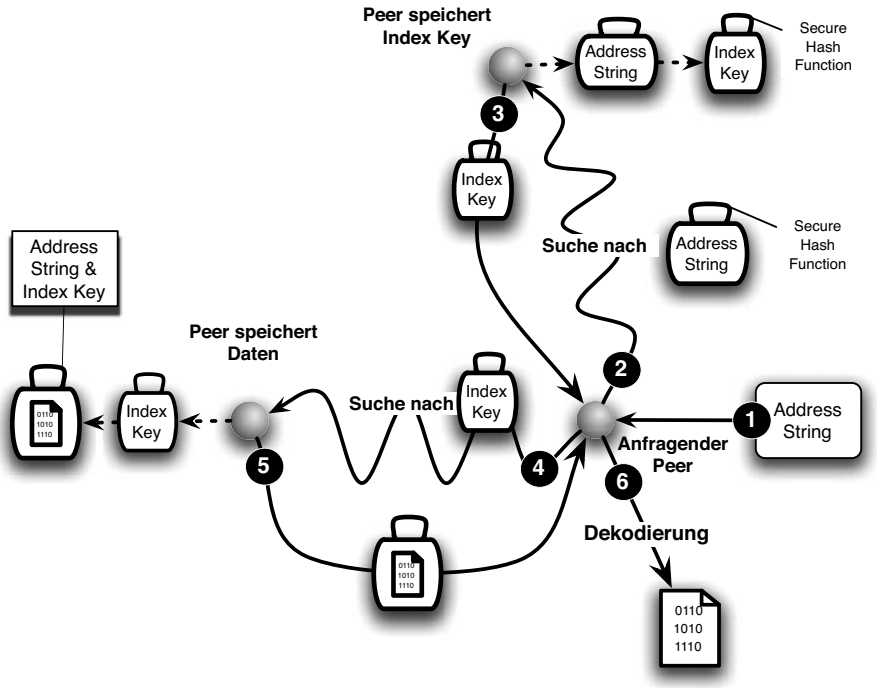


Abb. 11.4. Suche in der Datenstruktur von Free-Net.

Suche

Die Suche nach der Indexinformation und der Datei geschieht nach dem *Steepest-Ascent-Hill-Climbing*-Prinzip (Prinzip des steilsten Anstiegs auf einen Hügel). Hierzu wandern die Indexdaten und die Daten auf Peers, deren Identifikation den Daten am ähnlichsten ist. Die Suche ist im Wesentlichen eine *Tiefensuche* (*Depth-First-Search — DFS*), wobei bei mehreren Verästelungen derjenige Peer bevorzugt wird, dessen Identifikation dem Suchbegriff am ähnlichsten ist. Bei der Rücksendung werden dann von dem gesuchten Objekt Kopien auf dem Anfragepfad abgelegt, so dass bei der nächsten Suche diese Kopien zur Verfügung stehen.

Selten angefragte Kopien werden mit der Zeit gelöscht. Dadurch wird die Suchzeit erheblich verkürzt. In Abbildung 11.5 sieht man in dem jeweils logarithmisch aufgetragenen Diagramm der $\frac{1}{2}$-, $\frac{1}{4}$- und $\frac{3}{4}$-Quantile der Zugriffszeit in Abhängigkeit der Netzwerkgröße ein Anwachsen, das im wesentlichen proportional zur vierten Wurzel der Netzwerkgröße ist.

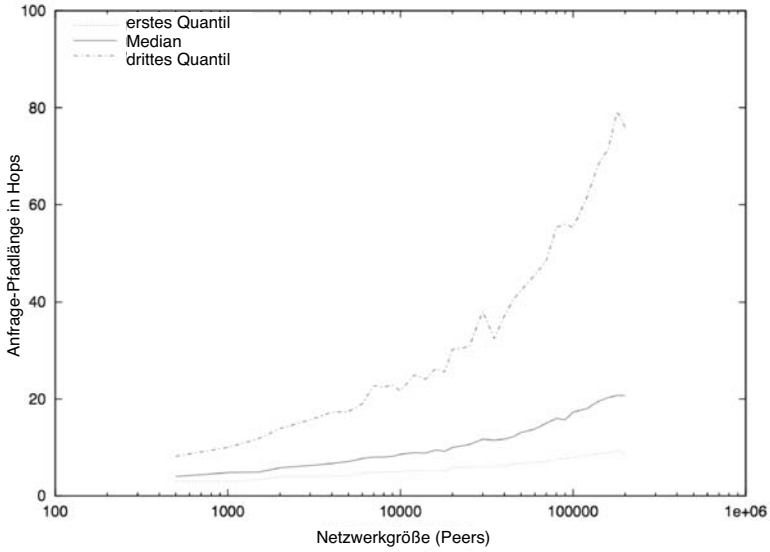


Abb. 11.5. Gemessene Suchzeit in Free-Net in Abhängigkeit der Netzwerkgröße [96].

Free-Net ist das erste Peer-to-Peer-Netzwerk, das eine anonyme Speicherung von Dokumenten zulässt und sowohl Autor, Server, Leser, aber auch das Dokument selbst schützt. Problematisch ist das Fehlen eines Verfallsdatums und die auf Popularität basierende Replikation. Dadurch kann die Verfügbarkeit und Widerrufbarkeit eines Dokuments wie in Free-Haven nicht sichergestellt werden. Außerdem können die Suchanfragen zurückverfolgt und das Netzwerk durch Infiltration korrumpiert werden.

11.5 Gnu-Net

Ziel des im Jahr 2006 von Krista Bennett, Christian Grothoff, Tzwetan Horozov, Ioana Patrasca und Tiberiu Stef entwickelten Netzwerks *Gnu-Net* [97] ist vertrauenswürdiges und anonymes verteiltes File-Sharing. Zusätzlich soll der Nachrichtenverkehr und der Berechnungsaufwand minimiert werden. Als mögliche Angreifer werden teilnehmende bösartige Peers betrachtet.

Gnu-Net beruht darauf, Dateien in Blöcke zu zerlegen, die in einem baumförmigen Code über das gesamte Netzwerk verteilt werden. Kodierte Knoten beschreiben hierbei den Hash-Wert der Kinder im Baum, so dass keine neuen Teilbäume manipuliert werden können. Wie wir bei Bittorrent gesehen haben, bringt die Verteilung von Dateien einen effizienteren Download.

Um den Download zu verbessern und Knoten anzuregen, Daten für andere Peers zu speichern, verwendet Gnu-Net ein Trust-Management-System, vergleichbar mit dem von Bittorrent. Knoten können dem Netzwerk jederzeit ohne zentrale Kontrolle beitreten. Diese neuen Knoten starten mit geringem Vertrauen (*untrusted*). Erst durch positive Mitwirkung wird das Vertrauen der anderen in diese Peers erhöht. Vertrauen kann man sich dadurch verdienen, dass man Routing-Anfragen weiterleitet und dass man Dateien für andere speichert. Je größer das Vertrauen gewachsen ist, desto mehr Anfragen und Daten darf der Peer selbst im Netzwerk produzieren.

11.6 Zusammenfassung

Neben Free-Net und Free-Haven gibt es ein Reihe anderer Peer-to-Peer-Netzwerke, die versuchen die Anonymität zu schützen. Diese verwenden ähnliche Techniken, wie wir sie bereits bei Free-Haven und Free-Net vorgestellt haben.

Es zeigt sich, dass die Methodik zur Anonymisierung schon sehr wirkungsvoll funktioniert. Es bleiben aber inhärente Schwachstellen. So ist im Moment sehr wohl nachweisbar, dass ein Peer an solch einem Netzwerk teilnimmt. Damit ist die Peer-Anonymität nicht gewährleistet. Außerdem stellt sich in diesem Zusammenhang auch die rechtliche und moralische Frage der Anonymisierung. Beispielsweise werden Betreiber von TOR-Routern angeklagt, der Verbreitung von Kinderpornographie Vorschub zu leisten. Die Speicherung solcher Daten ist in Deutschland strafbar und somit stellt der erste Router, der diese unverschlüsselt einem Anfrager zur Verfügung stellt, die einzige Möglichkeit zur Bekämpfung dar. Es bleibt ein ungelöstes gesellschaftliches Problem, wie man gleichzeitig die berechtigte Anonymität von Benutzern als auch die rechtliche Frage der Verfolgung Krimineller, wie zum Beispiel Terroristen, miteinander vereinbaren kann.

12

Datenzugriff: Der schnelle Download

Time is the worst place, so to speak, to get lost in ...
Douglas Adams

Peer-to-Peer-Netzwerke sind für viele ein Synonym für *File-Sharing*. In einem *File-Sharing*-Netzwerk stellt jeder Teilnehmer einen Teil seiner Dateien der Allgemeinheit der Peers zur Verfügung, die sich dann nach Belieben eine Kopie dieser Datei anlegen können. Wir haben uns sehr ausführlich mit der Suche beschäftigt. Wesentlich zeitraubender ist es, aber die Datei zu erhalten.

Zwischen zwei Peers im Internet gibt es eine maximale Übertragungsgeschwindigkeit, genannt *Bandbreite*. Ein Grundprinzip des Internets ist, dass für jede Punkt-zu-Punkt-Übertragung immer nur ein Pfad gewählt wird. Damit ist der Datendurchsatz auf diesem Pfad entscheidend geprägt durch das schlechteste Verbindungsstück. Die schlechtesten Verbindungen findet man in der Regel zwischen einem Peer und seinem *Internet Service Provider* (*ISP*). Für das einmalige Herunterladen (Download) einer Datei ist dieser Flaschenhals unvermeidbar. Wenn aber mehrere andere Peers dieselbe Datei lesen wollen, ergeben sich effizientere Methoden.

Vorab sollte man sich aber verdeutlichen, dass Peers sich jederzeit aus dem Netzwerk abmelden können. So kommt es gerade beim Download zu Unterbrechungen. Zwingend erforderlich ist die Möglichkeit, einen angefangenen Download fortzusetzen. Besser ist es, wenn man den Download auch von einem anderen Peer fortsetzen kann. Daraus entsteht die Notwendigkeit, für jede Datei kryptographisch sichere Hash-Signaturen mitzuliefern, da es sonst zu Inkonsistenzen kommen kann.

12.1 IP-Multicast

Der Begriff *Multicast* beschreibt die Datenübertragung von einer Quelle zu einer ausgewählten Menge anderer Geräte. Dieser Kunstbegriff wurde aus dem Begriff *Broadcast* abgewandelt. Hierunter versteht man die Verbreitung einer Information von einem Teilnehmer zu allen übrigen Teilnehmern, wie man es z.B. vom Fernsehen

oder Rundfunk her kennt. Unabhängig vom Bedarf wird hier ein Funkkanal für die Übertragung eines Programms durchgehend verwendet.

Im Internet ist IP-Multicast ein Randphänomen. Der Grund liegt darin, dass die in TCP/IP vorgesehene Funktionalität des IP-Multicast kaum von Internet-Service-Providern zur Verfügung gestellt wird. Das liegt zum einen daran, dass viele Internet-Service-Provider diesen Service nicht unterstützen. Das ist sehr zu bedauern, da von den hier vorgestellten Lösungen IP-Multicast die Beste wäre. So ist zwar die absolute Anzahl der IP-Multicast-fähigen autonomen Systeme im Internet gestiegen, der relative Prozentsatz stagniert aber seit Jahren unter 5%. In lokalen Netzwerken und in Forschungsnetzen spielt IP-Multicast eine gewisse Rolle.

Der Dienst von IP-Multicast ist die Verteilung einer Datei an jeden Teilnehmer einer festgelegten Gruppe. Zu einer solchen Gruppe kann man sich anmelden und (in der neueren Versionen von IGMP) auch wieder abmelden. Sie wird identifiziert durch besondere IPv4-Adressen, den so genannten Klasse-D-Adressen. Diese sind außerhalb des Adressraum des *CIDR* (Classless Interdomain Routing), das in Kapitel 2 vorgestellt wurde. Klasse-D-Adressen beginnen mit den Bits `1110`, so dass alle Adressen von `224.0.0.0` bis `239.255.255.255` für Multicasting reserviert sind. Bis auf ganz wenige Ausnahmen kann man sich zu jeder dieser Adressen weltweit anmelden und man erhält (soweit man sich in einem autonomen System befindet, welches IP-Multicast unterstützt) dann automatisch alle Nachrichten, die von der Quelle an diese Adresse geschickt werden.

Als Transportprotokoll kommt hier nur UDP in Frage, da TCP ausschließlich für Punkt-zu-Punkt-Verbindungen geeignet ist. Diese Verbindungen werden auch Unicast genannt. TCP ist insbesondere wegen der Fenster-Mechanismen und der Bestätigungen für Multicast-Verbindungen ungeeignet. Damit entfällt natürlich die Congestion-Control (Stauvermeidungsmechanismen).

Für die Steuerung von IP-Multicast ist das *Internet Group Management Protocol* (*IGMP*) zuständig. Zum Anmelden zu einer Multicast-Gruppe schickt der Host eine IGMP-Nachricht mit einem TTL-Wert (Time-to-Live) von 1 (erzwungenermaßen). Daher ist es unabdinglich, dass der erste Router *IP-Multicast*-fähig ist. Ist das nicht der Fall, so gibt es noch die Möglichkeit des *Tunnelns*. Hierfür wird eine Unicast-Verbindung zu einem Proxy aufgebaut, der sich in der Nähe eines Multicast-fähigen Router befindet. Der *Proxy* muss dann die Multicast-Pakete per Unicast an den Endknoten weiterleiten. Natürlich ist diese Methode nicht so effizient wie Multicast und sollte weitestgehend vermieden werden.

Technisch wird Multicast wie folgt gelöst: Ziel ist es, eine baumförmige Verteilungsstruktur aufzubauen, wie in Abbildung 12.2 dargestellt. Optimal ist eine solche Baumstruktur, wenn die Summe aller Kosten auf den Kanten minimiert wird. Dieses Problem ist die diskrete Version des so genannten *Steiner-Baum-Problems*, welches im kontinuierlichen euklidischen Raum $\mathcal{NP}$-vollständig ist. Für die lokale Optimierung kennt man gute Lösungen, bekannt durch das Seifenblasenexperiment [98]. Hierbei werden zwei parallele Glasplatten mit senkrecht verbindenden Stäben in Seifenwasser getaucht. Beim Herausholen bildet sich ein Seifenfilm zwischen den Stäben, der eine minimale Oberfläche anstrebt (siehe Abbildung 12.1). Mit etwas Glück bildet sich hierbei ein Steiner-Baum zwischen den Positionen der Stäbe.

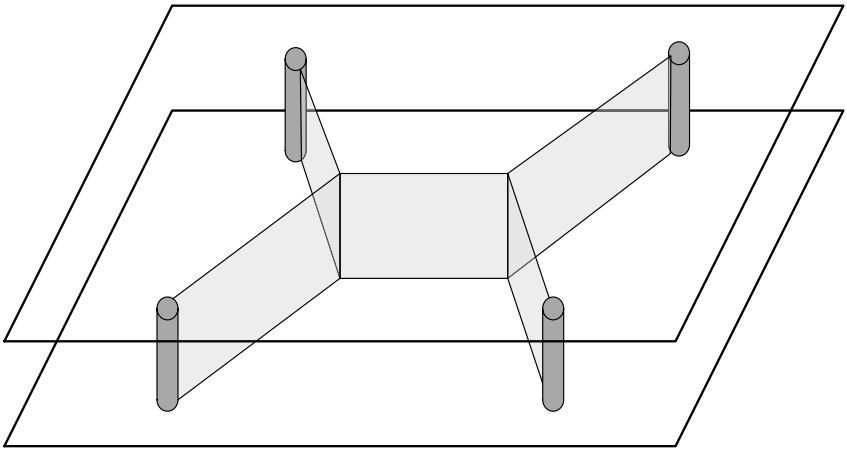


Abb. 12.1. Das Seifenblasenexperiment.

Die diskrete Variante ist, einen Baum in einem gegebenen ungerichteten Graphen zu finden, so dass die Summe der Distanzen des Baumes, der eine gegebene Menge von Knoten verbindet, minimiert wird. Während für das kontinuierliche Problem wenigstens effiziente Approximationsalgorithmen bekannt sind, ist das Problem im diskreten Fall schwieriger.

In den Router-Knoten ohne Verzweigung werden die Nachrichten wie bei einer *Unicast*-Verbindung einfach weitergereicht. An den Verzweigungsknoten des Baumes werden die Nachrichten dupliziert und in die beiden Teilbäume durchgereicht. An den Blättern des Router-Baumes werden schließlich alle angemeldeten Hosts mit Kopien der Pakete versorgt. Für die Etablierung eines solchen Multicast-Baumes gibt es verschiedene Protokolle.

Das erste war das *Distance-Vector-Multicast-Routing-Protocol* (*DVMRP*), das jahrelang im so genannten *MBONE*, einem Multicast-fähigen Teilnetzwerk des Internets, eingesetzt wurde. Mittlerweile hat sich das *PIM-SM* (*Protocol Independent Multicast – Sparse Mode*) [99] weitestgehend durchgesetzt. Während DVMRP noch eigene Multicast-Routing-Tabellen verwendet hat, ist PIM-SM weitgehend protokollunabhängig. Es verwendet die *Unicast*-Routeninformation, um auch den Multicast-Verkehr zu befördern. Hierzu ist mindestens ein designierter *Rendezvous-Punkt* (*RP*) in jeder Domain notwendig. Dieser dient als Anlaufstation für die lokale Organisation des Multicast-Baumes. Von der Quelle wird ein Kürzester-Wege-Baum zu den RPs der angemeldeten Hosts konstruiert. Ein Kürzester-Wege-Baum ist die Vereinigung aller kürzesten Wege von einem Knoten, hier der Quelle, zu einer Menge von Knoten, hier die RPs. In der Praxis kann es vorkommen, dass dieser Baum nicht aus kürzesten Wegen besteht, da die Informationen aus den „normalen“ *Unicast-Routing*-Tabellen verwendet werden.

Da TCP für Multicast nicht verwendet werden kann und UDP keine Sicherung gegen den Verlust von Paketen hat, können Multicast-Nachrichten nicht ohne wei-

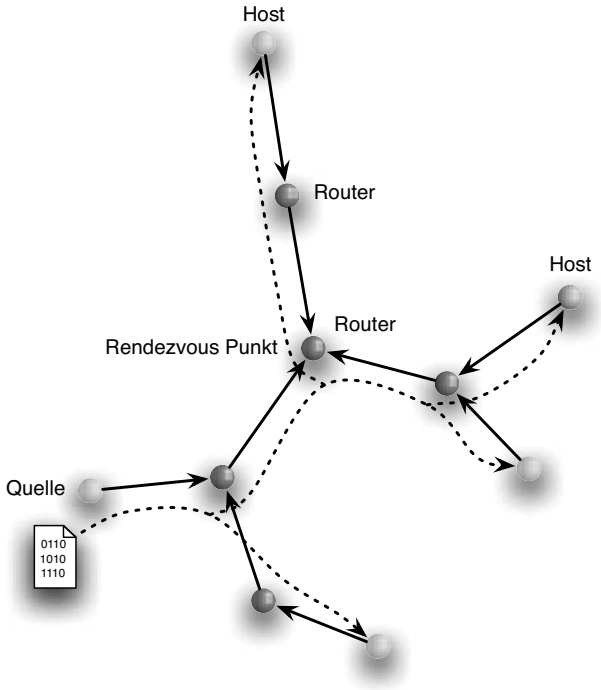


Abb. 12.2. Der IP-Multicast-Baum.

teres bestätigt werden. Um einen zuverlässigen Multicast zu erreichen, gibt es zwei Lösungen. Die erste arbeitet mit *Vorwärtsfehlerkorrektur* (*Forward Error Correction*). Hierbei werden die Nachrichten redundant kodiert, so dass der Verlust einer bestimmten Menge von Paketen kompensiert werden kann. Auf der anderen Seite kann man die Bestätigungen in den Routern überprüfen und so in der Vermittlungsschicht die Nachrichten absichern (und neu versenden). In diesem Zusammenhang ist Ciscos *PGM* (*Pragmatic General Multicast*) [100] zu erwähnen.

Die Adressmethode von Multicast kann auf der Mediumzugriffsebene (*Medium Access Control — MAC*) im lokalen Netzwerk zu Kollisionen führen. Außerdem besteht durch Multicast die besondere Gefährdung durch so genannte Denial-of-Service-Angriffe, die man gesondert abwehren muss. Im Vergleich zu *Unicast* ist der Wartungs- und Konfigurationsaufwand wesentlich größer, während diese Dienste beim Endkunden kaum nachgefragt werden. Somit ist der kommerzielle Druck auf die Internet-Service-Provider nicht groß genug, um die zusätzlichen Kosten zu rechtfertigen.

Es ist darüber hinaus nicht klar, ob der Aufwand eines IP-Multicasts zum Vertrieb einer einzelnen Datei an viele andere Teilnehmer gerechtfertigt ist. Zum einen ist die Anzahl möglicher Interessenten oftmals sehr gering. Zudem kommen die Anfragen nicht zeitgleich. Wird eine Übertragung dann initiiert und kommt später ein neuer

Interessent hinzu, so muss die Übertragung wiederholt werden. Eine andere Frage ist, ob IP-Multicast sich ohne weiteres mit den Problemen, die DHCP, NAT und Firewalls bereiten, vereinbaren lässt.

12.2 Scribe

Da IP-Multicast in der Regel nicht zur Verfügung steht, wurde nach gangbaren Alternativen gesucht. Eine hiervon nennt sich *Scribe* und wurde von Castro, Druschel und Kermarrec in [101] vorgestellt. Die Idee ist, einen Multicast-Verteilungsbaum auf einem Overlay-Netzwerk, hier Pastry [11], zu implementieren. Zwar entsteht im Internet immer noch der Mehrfachaufwand durch wiederholtes Senden der Nachrichten durch die Router, es hat sich aber herausgestellt, dass für den einfachen Internetbenutzer der letzte Link in das Netzwerk hinein der Flaschenhals ist. Der Grund ist, dass die weitverbreiteten DSL-, ISDN- oder analogen Modemverbindungen der Endkunden mit den Glasfaserverbindungen der Internet-Service-Provider nicht mithalten, selbst wenn eine Vielzahl der Benutzer diese Dienste verwendet.

Ähnliche Ansätze für andere Peer-to-Peer-Netzwerke wurden mit *CAN-Multicast* [102], basierend auf CAN [8], und *Bayeux* [103], basierend auf Tapestry [12], vorgestellt. Wir greifen hier Scribe beispielhaft auf, ohne die anderen Ansätze abwerten zu wollen.

Daneben gibt es mit Overcast [104] und Narada [105] ältere Ansätze zur Verwendung von Unicast-Verbindungen zum Aufbau eines Multicast-Baumes. In diesen Netzwerken ist es aber notwendig, alle Abstandsinformationen zwischen den Endknoten zu sammeln und auszuwerten. Wie wir bereits wissen, ist die optimale Lösung dieses Problems $\mathcal{NP}$-vollständig. Aber allein das Sammeln der Information übersteigt die Möglichkeiten eines Peer-to-Peer-Netzwerks. Schließlich können in Peer-to-Peer-Netzwerke mehrere Millionen Peers anwesend sein. Damit gilt es für jeden Peer, den Abstand zu Millionen anderen Peers zu sammeln. Auch Scribe findet keinen optimalen Multicast-Baum. Aber Scribe (wie CAN-Multicast und Bayeux) vermeidet es, solche Mengen an Informationen zu sammeln und auszuwerten.

Create

Scribe vergibt für jede Empfängergruppe eine so genannte Gruppen-ID (*Group-ID*). Diese wird per Hash-Funktion auf einen Peer abgebildet. Ansonsten ist die Funktionsweise von Scribe der von IP-Multicast sehr ähnlich.

Join

Jeder Peer, der sich einer Gruppe anschließen möchte, sendet einen *Lookup* zu dem *Root-Peer*, der die Group-ID verwaltet. Sobald auf dieser Route ein Peer gefunden wird, der zum Multicast-Baum dieser Gruppe gehört, wird der Pfad zu diesem Peer zum Multicast-Baum dieser Gruppe hinzugefügt. Die Information wird dann mittels dieses Baumes verteilt, siehe auch Abbildung 12.3.

Download

Die Nachrichten werden jetzt durch diesen Baum verteilt. Hierzu duplizieren die Knoten die Informationen.

Scribe benutzt eine Besonderheit des Pastry-Protokolls: Die Zeiger in den niedrigen Leveln führen zu näheren Knoten im Netzwerk, als Zeiger in hohen Leveln. Je größer der Baum wird und je mehr Knoten im Multicast-Baum sind, desto kürzere Kanten werden eingefügt.

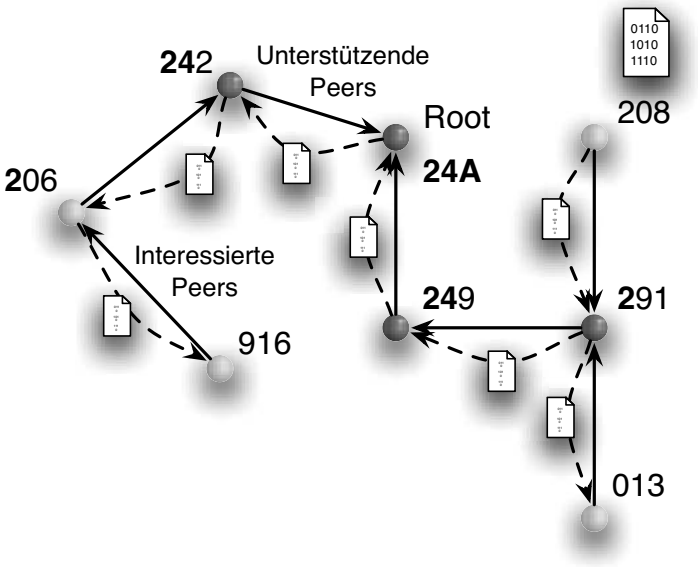


Abb. 12.3. Funktionsprinzip von Scribe.

Reparatur und Kontrolle

Da der Multicast-Baum wie der Rest des Peer-to-Peer-Netzwerks den dauernden Veränderungen durch hinzukommende oder verschwindende Peers ausgesetzt ist, bedarf es einer ständigen Kontrolle, ob der Multicast-Baum noch intakt ist.

Hierzu werden so genannte Herzschläge ausgesandt, das sind Kontrollnachrichten, die der Knoten, der die Group-ID verwaltet, an alle weiteren Knoten des Baums verschickt und weiterleitet. Wurde also ein Teil des Baums abgetrennt, so kann dies durch das Ausbleiben dieser regelmäßigen Kontrollnachrichten festgestellt werden. Der Multicast-Baum wird dann dadurch repariert, dass der Aufbaualgorithmus erneut ausgeführt wird.

Bottleneck-Remover

Wenn ein Knoten überlastet ist, dann wird die Gruppe mit der größten Last ausgewählt. In dieser Gruppe wird das in der Netzwerkmetrik am weitesten entfernte Kind gewählt, welches dann den Delay zwischen sich und den anderen Kindern misst. Darufhin wird das Kind unter den nächsten der hierbei gefundenen Knoten gehängt, siehe Abbildung 12.4.

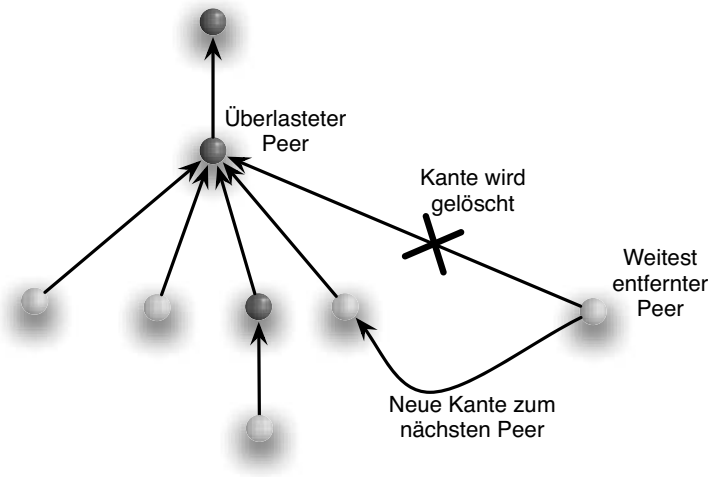


Abb. 12.4. Entfernen von Engpässen in Scribe.

Durch diese Mechanismen erhält man mit Scribe eine überraschend gute Bandbreite, die mit IP-Multicast vergleichbar ist.

12.3 Splitstream

Splitstream ist eine Verbesserung von Scribe von Castro, Druschel, Kermarrec, Nandi, Rowstron und Singh [106]. Die Struktur eines Multicast-Baumes bevorzugt die Blätter und benachteiligt die inneren Knoten. Diese müssen nämlich die empfangenen Daten mindestens zweimal wieder versenden. Die Blätter dagegen erhalten die Daten und haben keinerlei weiteren Aufwand.

Das Verhältnis der benachteiligten Peers in einem Binärbaum ergibt sich aus dem folgenden Lemma:

Lemma 12.1. *In jedem Binärbaum ist die Anzahl der Blätter b um eins größer als die Anzahl k der internen Knoten, d.h. $b = k + 1$.*

Beweis. Jeder Binärbaum kann durch sukzessives Anhängen von zwei Blättern an einen Knoten konstruiert werden. Der einfachste Baum besteht aus einem Blatt, hier gilt $b = 1$ und $k = 0$. Dies ist der Induktionsanfang. Wenn wir als Induktionsvoraussetzung annehmen, dass die Aussage für b Blätter korrekt ist, dann erhöht sich durch Anhängen von zwei Blättern die Anzahl der Blätter effektiv um eins (da ja ein bereits vorhandenes Blatt zum internen Knoten wird) und die Anzahl der internen Knoten erhöht sich auch um eins. Daraus folgt das Lemma. □

Dieses Problem lässt sich durch Erhöhung des Grades nicht verringern. Dann nimmt die Anzahl der internen Knoten ab, wie das folgende Lemma zeigt.

Lemma 12.2. *In jedem Baum mit Grad d gilt für die Anzahl der Blätter b und die Anzahl der internen Knoten k: $(d-1)k + 1 = b$.*

Beweis. Der Beweis ist völlig analog zum Beweis von Lemma 12.1. Jeder Baum mit Grad d kann durch sukzessives Anhängen von d Blättern an einem Knoten konstruiert werden. Der einfachste Baum besteht aus einem Blatt, hier gilt $b = 1$ und $k = 0$. Dies ist der Induktionsanfang. Angenommen, die Gleichung stimmt für b Blätter hängen wir jetzt d Blätter an, so erhöht sich die Anzahl der Blätter effektiv um $d - 1$ (da ja ein Blatt zum internen Knoten wird), und die Anzahl der internen Knoten erhöht sich um eins. Daraus folgt das Lemma. □

Man sieht also, dass sich der Aufwand der benachteiligten Knoten erhöht. Da Peer-to-Peer-Netzwerke auf Freiwilligkeit beruhen, ist davon auszugehen, dass sich solche Peers dieser Aufgabe durch Abmelden entziehen können (in der Hoffnung durch Neuanmelden zum Blatt im Baum zu werden).

Split-Stream löst dieses Problem durch die Kombination folgender Techniken: Zuerst wird jede Datei in Blöcke (*Stripes*) zerlegt. Außerdem ist von jedem Peer seine eingehende und ausgehende Bandbreite bekannt. Für jeden dieser Blöcke wird jetzt ein eigener Baum konstruiert. So übernimmt ein Peer für gewisse Blöcke die Funktion eines internen Knotens, d.h. eines Verteilers, und für andere Blöcke nur die Funktion des Blattes, d.h. eines Empfängers, siehe Abbildung 12.5. Hierbei wird beachtet, dass die Eingangs- und Ausgangsbandbreiten jedes Peers berücksichtigt werden. Es wird Scribe für jeden einzelnen Stripe verwendet. Dazu wird für jeden Stripe eine *Stripe-ID* angelegt. Außerdem gibt es lokale Optimierungsmechanismen, die eine Überschreitung der Eingangs- und Ausgangsbandbreite wieder korrigieren.

Splitstream löst die Ungleichbehandlung der Peers. Jedoch entsteht durch die explizite Verteilung jedes Blocks ein erheblicher Organisationsaufwand.

12.4 Bittorrent

Bram Cohen ist der Erfinder von *Bittorrent* [107]. Bittorrent ist ein sehr erfolgreiches Peer-to-Peer-Netzwerk, das in den letzten Jahren mehr als die Hälfte des Internetverkehrs verursacht hat. Der Erfolg des Netzwerks beruht in seiner Effektivität

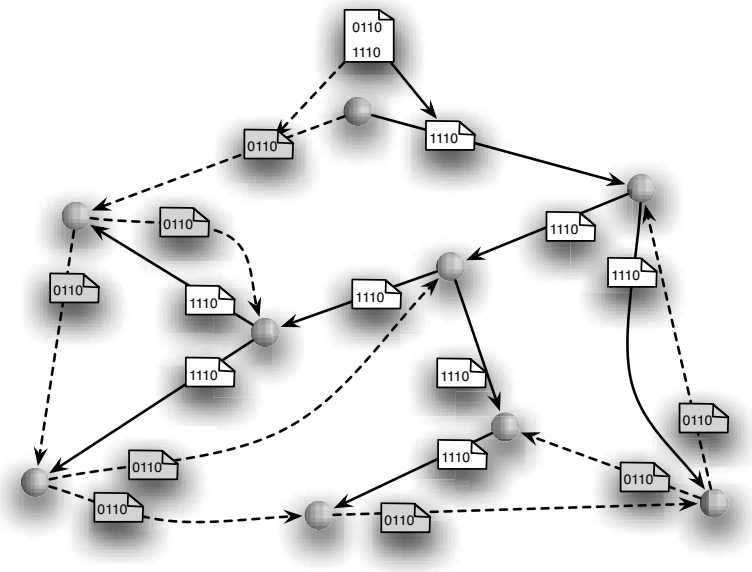


Abb. 12.5. Funktionsweise von Splitstream.

zur Verwendung des Uploads von Peers, die die Datei noch nicht vollständig herunter geladen haben. Andererseits besitzt Bittorrent keine Mechanismen zur Suche von Dateien. Die Beteiligten eines Teilnetzwerks konzentrieren sich auf den Download einer Datei. Dieser wird organisiert durch einen so genannten *Tracker-Host*, der die Verbindungen der interessierten Peers vermittelt. Bittorrent verwendet implizit Multicast-Bäume für die Verteilung der Daten, ähnlich wie Splitstream. Die Beschreibung von Bittorrent ist aber Peer-orientiert und nicht datenorientert. Anders formuliert, es wird nicht für jeden Teil der Datei explizit ein Baum konstruiert. Vielmehr wird für jeden Peer angegeben, welche Teile er als nächstes lädt oder weitergibt.

Die Ziele von Bittorrent lassen sich wie folgt zusammenfassen:

- Möglichst effizienter Download einer Datei unter Zuhilfenahme der Upload-Fähigkeit der Peers.
- Faires Verhältnis zwischen Upload und Download eines Peers. In der Praxis wird der Upload der Engpass. Das wird zum Beispiel durch die asymmetrische Protokollgestaltung von ISDN oder durch die asymmetrische Bitübertragungsschicht von DSL verursacht.
- *Fairness* zwischen den Peers. Kein Peer soll nur Daten herunterladen können, während andere Peers Daten hochladen.
- Verwendung verschiedener Quellen. Dies schließt sowohl Peers ein, die die Datei vollständig geladen haben, als auch Peers, die erst einige Teile der Datei geladen haben.

Technisch wird das in Bittorrent folgendermaßen realisiert: Die Verbindungsaufnahme erfolgt durch einen *Tracker-Host*. Ein Tracker-Host sammelt die Socket-Paare (IP-Adressen und Port-Nummer) der Peers, die sich für eine Datei interessieren. Diese werden dann bei einer Anfrage eines Peers zurückgeliefert. Daraufhin kann der Peer dann mit diesen Peers Kontakt aufnehmen. Außerdem werden der Hash-Wert der Datei und andere Kontrollinformationen mitgeliefert. Man beachte, dass Tracker-Hosts selbst keine Dateien bereitstellen. Trotzdem kann das Anlegen und Verwalten einer Tracker-Datei rechtliche Implikationen hinsichtlich des Urheberschutzes haben.

Die zu verteilende Datei wird in kleinere Teile unterteilt. Auch diese Information wird in der Tracker-Datei festgehalten. Sobald ein Teil vollständig heruntergeladen ist, kann er von diesem Peer wieder weiter verteilt werden. Betrachtet man nun jedes Teil für sich, so wird implizit ein Multicast-Baum erstellt.

Die Peers erhalten also die Informationen über andere teilnehmende Peers vom Tracker-Host. Daraufhin nehmen sie immer wieder Kontakt mit anderen Interessenten auf. Bei diesem Kontakt tauschen sie Informationen über ihre vorhandenen Teile aus. Somit erhält jeder Peer eine Statistik über die vorhandenen Teile einer Datei. Gemäß der Verteilungsstrategie von Bittorrent werden bestimmte Teile bevorzugt.

Das Hauptproblem bei der Verteilung ist das so genannte *Coupon-Collector-Problem*: Wird jeder der m Teile einer Datei zufällig verteilt und erhält jeder Peer nur einen Teil, so müssen $\Omega(m \ln m)$ Peers teilnehmen, damit mit konstanter Wahrscheinlichkeit jeder Teil wenigstens einmal vorhanden ist. Verteilt man die m Teile zufällig auf m Peers, so wird im Erwartungswert ein konstanter Teil (der gegen $1/e$ konvergiert) keinen einzigen Teil besitzen, während einige $\mathcal{O}(\log m)$ Teile besitzen.

Bittorrent versucht dieses Ungleichgewicht zwischen seltenen Teilen und häufigen Teilen durch die folgenden Maßnahmen zu verringern:

Auswahl der Teile

Jeder Peer versucht die fehlenden Teile der Datei durch Anfragen bei anderen Peers zu ergänzen. Hierzu kontaktiert er einen Peer aus seiner Gruppe und fragt nach dessen Teileliste. Aus der gesammelten Information dieser Teile erhält der Peer einen Überblick über die Verfügbarkeit verschiedener Teile und kann so insbesondere beurteilen, ob es im Moment überhaupt möglich ist, die Datei herunterzuladen. Das ist dann nicht möglich, wenn die ursprüngliche Quelle nicht mehr vorhanden ist und ein Teil auf den anderen Peers nicht existiert.

Die Auswahl des zu ladenden Teils wird wie folgt getroffen:

- *Rarest First* — Das Seltenste zuerst
  Jeder Peer versucht die seltensten Teile zuerst zu laden. Die Häufigkeit aller Teile im Gesamtnetzwerk erschließt sich aus der Kommunikation mit anderen Peers. Diese Strategie verringert das Risiko, dass eine Datei nicht mehr verteilt werden kann, wenn die Quelle nicht mehr vorhanden ist.
  Die Hoffnung ist hierbei, dass die Quelle, wenn sie jedes Teil einmal verteilt hat, obsolet wird. Diese Hoffnung trügt jedoch. Zwar verbessert sich die Situation gegenüber einem völlig unkoordinierten Download, was unweigerlich zum

Coupon-Collector-Problem und seinen eben diskutierten Folgen führt. Es entstehen aber neue Probleme: So können gerade beim Download der seltenen Teile Engpässe entstehen. Denn diese werden von vielen Peers nachgefragt. Dadurch wird die Bandbreite weiter verringert und die Zeit, bis die Datei wieder weiter verteilt werden kann, nimmt zu.

- *Random First* — Ein Zufälliges zuerst (Ausnahme für neue Peers)
  Insbesondere besteht die Gefahr, dass neue Peers, die eventuell nur „hineinschnuppern" wollen, das Herunterladen seltener Teile behindern und sich dann wieder verabschieden, ohne die Dateien weiter zu verteilen. Daher gibt es eine Ausnahme für neue Peers. Diese fragen nach einem zufälligen Teil der Datei an und balancieren so die Last.
- *Endgame Mode* — Endspielmodus
  Eine weitere Ausnahme greift, wenn ein Peer fast alle Teile geladen hat. Zumeist wird der Prozess dann von den Peers aufgehalten, die nur langsam übertragen. Dadurch kann sich der Download erheblich in die Länge ziehen. Da die Benutzer ungeduldig sind und vermuten, dass ein Prozess, der bei 99% zum Halten kommt, fehlerhaft ist, wird der so genannte Endspielmodus angewendet.
  Der Peer fragt zum Schluss also bei allen verbundenen Peers nach allen fehlenden Teilen. Das führt in der Regel zu einem schnellen Abschluss und es kann vermieden werden, dass ein langsamer Peer mit einem bestimmtem fehlenden Teil den Download am Ende noch herauszögert. Die Mehrbelastung lässt sich in der Regel vernachlässigen, da dies nur einen kurzen Zeitabschnitt betrifft.

Fairness

Ein wichtiges Ziel von Bittorrent ist *Fairness* zwischen den Peers. Man kann grob zwischen gutem und schlechtem Verhalten unterscheiden. Gutes Verhalten ist das Bereitstellen der Information, d.h. Upload. Solche Peers werden *Seeder* (Säher) genannt. Schlechtes Verhalten ist das Herunterladen von Information (Download). Diese so genannten *Leecher* (Sauger) beeinträchtigen das System, wenn sie sich nicht am Upload beteiligen.

Die Fairness wird verbessert, indem gutes Verhalten belohnt und schlechtes Verhalten bestraft wird. Die Belohnung ist das Bevorzugen eines Peers durch verbesserte Übertragung. Bestrafung geschieht durch Drosselung, d.h. Verweigern der Datenübertragung an einen Peer. Die Einschätzung von gutem und schlechtem Verhalten trifft jeder Peer selbst. Genauso belohnt jeder Peer für sich sein Umfeld aufgrund der vergangenen Erfahrungen.

Drosseln (Choke)

Jeder Peer unterhält eine schwarze Liste von Peers, die *gedrosselt* werden. Sämtliche Anfragen eines gedrosselten Peers werden nicht bedient. Peers können jedoch aus der Drosselliste wieder entfernt werden. Hierzu speichert jeder Peer eine Mindestanzahl von gedrosselten Peers (z.B. vier). In diese Liste werden nun diejenigen Peers aufgenommen, von denen der Download am schlechtesten war. Die Qualität eines Downloads wird hierbei durch die Bandbreite bestimmt, also die Anzahl

übertragener Bits pro Sekunde. Diese Bandbreite muss von Bittorrent durch Zählen über einem Testintervall von 20 Sekunden bestimmt werden, da diese Information nicht ohne weiteres aus dem TCP-Protokoll zu bestimmen ist. Verbessert sich die Bandbreite, so werden die gedrosselten Peers wieder entdrosselt (*unchoke*).

Damit hat jeder Peer ein Interesse nicht in diese Listen zu gelangen und versucht den Upload nicht zu vernachlässigen, weil sonst der Download ganz zusammenbricht. Was passiert aber, wenn ein Peer von allen anderen Peers gedrosselt wurde und daraufhin selbst alle anderen Peers drosselt? Ohne weitere Mechanismen kann der Peer nicht mehr aus dieser Situation entkommen.

Hierzu gibt es einen Mechanismus, das *optimistische Entdrosseln* (*optimistic unchoking*). Dabei wird ein zufälliger Peer aus der Drosselliste entfernt, so dass der Teufelskreis der gegenseitigen Drosselung durchbrochen werden kann. Außerdem gibt dieser Mechanismus denjenigen Peers eine Chance am Datenverkehr teilzunehmen, die sonst aufgrund ihrer schlechten Anbindung immer gedrosselt würden. Natürlich eröffnet sich auch ein Einfallstor für echte Leecher. Da aber diese zumeist immer noch gedrosselt werden, bleibt der Anreiz für solch unsoziales Verhalten gering.

In der Praxis haben sich diese Mechanismen bewährt, so dass Bittorrent als das effizienteste File-Sharing-System betrachtet wird. Kritik von den Nutzern erhält es hauptsächlich wegen der mangelnden Suchfunktion. Denn ohne Tracker-Host kann kein Teilnehmer eine bestimmte Datei herunterladen. Die Verbreitung dieser Information geschieht zumeist auf so genannten Tracker-Sites, in denen man per Suchfunktion bestimmte Dateien finden kann.

Gewissermaßen ist Bittorrent hier auch ein Opfer seines Erfolges, da wegen der stattfindenden Urheberrechtsverletzung diese Suchseiten und die Tracker-Hosts (wo es rechtlich möglich ist) abgestellt werden. Prominentes Beispiel ist zum Beispiel die Webseite `www.supernova.org`, die im Dezember 2005 von slowenischen Behörden vom Internet entfernt wurde, obgleich auf der Webseite selbst keine urheberrechtlich geschützte Information vorhanden war. Andere Bittorrent-Tracker wie zum Beispiel `ThePirateBay.org` mit Standort in Schweden sind weiterhin online.

Problematisch für die illegalen Downloader ist bei Bittorrent sicherlich die einfache Möglichkeit die IP-Adressen zu erhalten. Hierzu genügt eine Anfrage an den Tracker. Dann kann man sich direkt durch Kontaktaufnahme mit dem Peer davon überzeugen, dass er tatsächlich diese Datei herunterlädt. Ohne Anonymisierung der IP-Adresse lässt sich diese Schwäche wohl kaum abstellen. Hierzu kann man neuerdings Bittorrent mit dem TOR-Netzwerk (The Onion Router) kombinieren.

12.5 Redundante Kodierung

Durch die dezentrale Verteilung der Daten in Bittorrent besteht immer die Gefahr, dass ein Teil mit dem Abmelden einiger Peers verloren geht. Wenn nur wenige Teile fehlen, so kann man dies durch eine redundante Kodierung ausgleichen.

Hierzu geben wir ein einfaches Beispiel. Wenn zum Beispiel vier Teile aus je vier Bits bestehen, z.B. $x_1 = 0101$, $x_2 = 0011$, $x_3 = 1000$, und $x_4 = 0000$, dann kann man zusätzlich einen redundanten Fehlercode $z = x_1 + x_2 + x_3 + x_4 = 1110$ durch die XOR-Operation der einzelnen Bits berechnen. Dieser Fehler-Code wird nun zusätzlich verbreitet. Wenn zum Beispiel der zweite Teil verloren geht, so kann man durch Berechnung von $x_2 = z + x_1 + x_3 + x_4$ den fehlenden Teil wieder rekonstruieren.

Diese Technik wird in der Bitübertragungsschicht von Rechnernetzwerken als so genannte *Vorwärtsfehlerkorrektur* (*Forward Error Correction*) verwendet. Dort ist das Hauptproblem, dass einzelne Bits durch Störungen verloren gehen könnten. Zuweilen treten diese Störungen auch gehäuft auf, als so genannte *Bursts*. Für Kommunikation, die unter dieser Fehlerart zu leiden hat, sind die Kodierungen von Reed und Solomon besonders geeignet [108], welche eine begrenzte Menge von Bitfehlern kompensieren können. Die Situation in Bittorrent ist ähnlich. Nur gehen hier gleich ganze Blöcke verloren.

Reed-Solomon-Kodierungen

Betrachten wir eine Datei, die in n Blöcke zerlegt wurde. Dann muss man bei einer *Reed-Solomon-Kodierung* im Voraus festlegen, wieviele Blöcke davon maximal verloren gehen können. Sei diese Zahl k. Man kodiert nun statt n Blöcke $n + k$ Blöcke, und n beliebige dieser $n + k$ Blöcke genügen dann zur Rekonstruktion der Datei. Hierzu seien $X = (x_1, \ldots, x_n)^T$ die Originalblöcke. Diese Blöcke werden als Vektoren über einem Körper betrachtet, so dass man sie addieren und mit Variablen multiplizieren kann. Man kann sich für den Anfang vorstellen, dass es sich hier um einfache Zahlen handelt.

Dann konstruiert man die kodierten Blöcke $Y = (y_1, \ldots, y_{n+k})$, indem man die Matrixmultiplikation mit einer sorgfältig gewählten $(n + k) \times n$ Matrix durchführt:

$$M \cdot X = Y \ .$$

Hierbei muss jede $n \times n$ Teilmatrix, die man durch Streichen von k beliebigen Zeilen von M erhält, invertierbar sein. Sei M' eine Teilmatrix, die die korrespondierenden Zeilen des n-stelligen Teilcodes $(y_{i_1}, \ldots, y_{i_n})$ mit $i_1 < i_2 < \ldots < i_n$ erhält. Dann gilt

$$M' \cdot X = Y' \ .$$

Ist nun M' invertiert, so kann man durch

$$X = (M')^{-1} Y'$$

den Originalvektor rekonstruieren. Hierzu benötigt man dann nur einen n-stelligen Teilcode und die Indizes des entsprechenden Codes. Diese Kodierung wird $(n + k, n)$-Reed-Solomon-Code genannt. Wie bereits erwähnt, steht hier jeder Block für einen Vektor über einem endlichen Körper. Besonders geeignet ist zum Beispiel ein Vektor über den Galois-Körpern $F[256]$ oder $F[2^{16}]$, da diese den Wortlängen

gängiger CPUs entsprechen. Die Addition wird durch die XOR-Operation dargestellt und die Multiplikation kann mit Hilfe einer Multiplikationstabelle geschehen.

Für die Matrix kann man zum Beispiel eine *Vandermonde-Matrix* wählen, in der jede Zeile die Form $(1, x, x^2, \ldots, x^n)$ hat. Damit kann man für einen endlichen Körper $F[2^b]$ und $n + k \leq 2^b$ den Reed-Solomon-Code erzeugen. Hierzu gibt es auch weitere Möglichkeiten (z.B. durch zufällige Wahl der Einträge).

Da der Verlust von k Blöcken verkraftet werden kann, heißen diese Codes auch *Erasure Codes*. Bedauerlicherweise ist der Berechnungsaufwand in Reed-Solomon-Codes durch die Matrix-Multiplikation und die Invertierung der Matrix relativ hoch. Daher betrachtet man schneller berechenbare Erasure Codes, worunter jedoch die Redundanz der Codes leidet. In [109] wurde die Technik der Replikation von Daten mit Erasure Codes verglichen, wobei den Erasure Codes der Vorzug zu geben ist. Erasure Codes werden zum Beispiel in Dhash [39] und Oceanstore [110] eingesetzt.

12.6 Netzwerkkodierung

Bei der Vorwärtsfehlerkorrektur im vorigen Abschnitt musste im Vorhinein abgeschätzt werden, wie groß der auftretende Fehler ist. Mit der so genannten *Netzwerkkodierung* ist das nicht mehr nötig. Sie wurde von Ahlswede, Cai, Li und Yeung eingeführt [111] und betrachtet die Informationsverteilung in einem Flussgraphen von den Quellen zu den Senken. In Abbildung 12.8 kann man die Ausgangsproblemstellung sehen. Ziel ist die Verteilung der Bits x und y an die Senken. Jede Kante kann nur ein Bit weitergeben. Bei einer klassischen Informationsverteilung müsste man sich auf der mittleren Kante für x (Abb. 12.6) oder y (Abb. 12.7) entscheiden, woraufhin entweder links die Information x oder rechts die Information y fehlt. Das Prinzip der Netzwerkkodierung beruht nun darauf, dass auf jeder Kante eine passende Kodierung der vorhandenen Informationen weitergegeben wird.

In diesem Beispiel überträgt man also die Summe, d.h. das XOR, auf der mittleren Kante. Aus x und $x + y$ kann man nun x und y rekonstruieren (und analog aus y und $x + y$ ebenfalls x und y), siehe Abbildung 12.8.

Es wurde bereits in [111] bewiesen, dass dies im Allgemeinen durch eine zufällige Kodierung gelöst werden kann. In [112] wurde diese Idee aufgegriffen und auf Peer-to-Peer-Netzwerke angewendet.

Hierzu sieht man jeden Block b_i als einen Eintrag des Vektors $(b_1, \ldots, b_m)$, der das ganze Dokument darstellt. Anstatt einzelne Blöcke zu veröffentlichen, wird jetzt eine Linearkombination

$$(a_1 a_2 \ldots a_m) \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = a^T \cdot b = c$$

dieser Vektoreinträge samt der Faktoren versendet. Die Faktoren werden zufällig gewählt. Wenn nun der Empfänger m Blöcke $c_1, \ldots, c_m$ erhalten hat, so sei $A = (a_i j)_{i,j \in [m]}$ die zugehörige Variablenmatrix aus den einzelnen Faktoren der Blöcke. Dann gilt

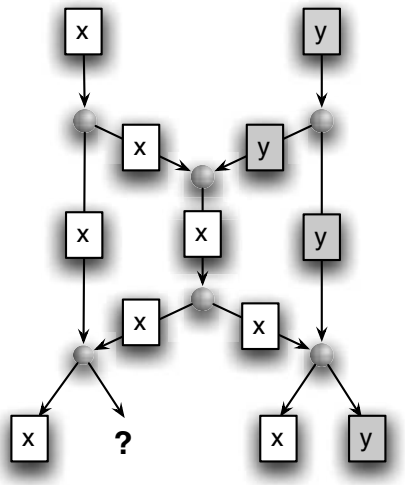


Abb. 12.6. Überträgt man auf der mittleren Kante nur x, so fehlt links das Bit y.

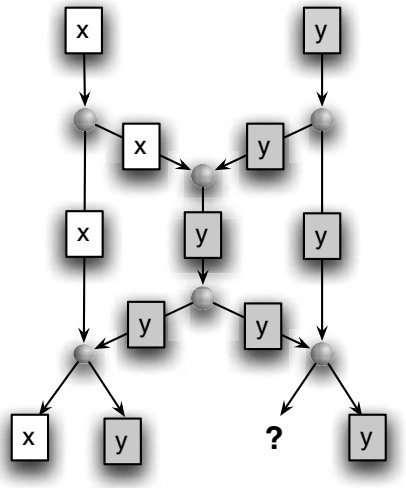


Abb. 12.7. Überträgt man auf der mittleren Kante nur y, so fehlt rechts das Bit x.

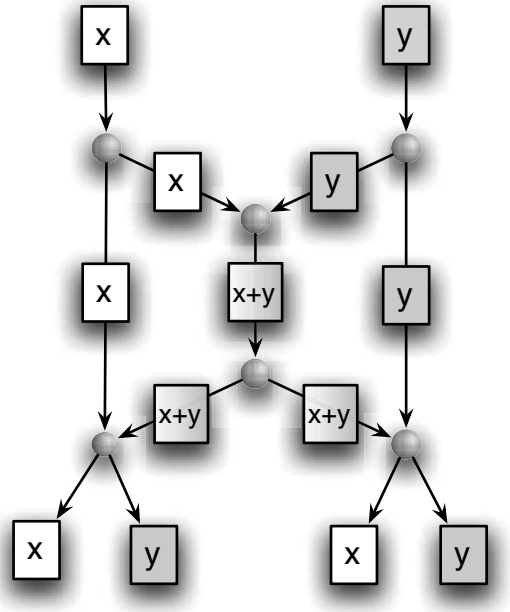


Abb. 12.8. Mit Hilfe der Netzwerkkodierung kann die Information A und B an den Senken des Netzwerks ankommen.

$$A \cdot b = \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,m} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_m \end{pmatrix} = c \,.$$

Ist nun die Matrix A umkehrbar, so erhält man die ursprüngliche Datei durch Berechnung von

$$A^{-1}c = b \,.$$

Die Blöcke werden hier wieder als Vektoren über einem Galois-Körper dargestellt. Damit ergeben sich die folgenden Aufgaben der Peers:

- Peers mit vollständiger Information (Quellen):
  Auf Anfrage eines anderen Peers wird eine Linearkombination über den ursprünglichen Peers wie eben beschrieben und der Variablenvektor zusammen mit dem Datenblock weitergegeben.
- Peers mit unvollständiger Information:
  Erhält ein solcher Peer eine Anfrage, dann berechnet er eine Linearkombination über den bereits erhaltenen Codes $c_1, \ldots, c_t$, wobei zum Code c_i die Variablen $a_i = a_{i,1}, \ldots, a_{i,m}$ gehören, d.h.

  $$\begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{t,1} & \cdots & a_{t,m} \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_t \end{pmatrix} \,.$$

  Nun wählt der Peer die Zufallszahlen $r_1, \ldots, r_m$ und übermittelt an den anfragenden Peer den Vektor

  $$r^T \cdot a = (r_1 \ \cdots \ r_t) \cdot \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{t,1} & \cdots & a_{t,m} \end{pmatrix} = (d_1 \ \cdots \ d_m) = d^T \,.$$

  Dann überprüft der anfragende Peer, ob dieser Vektor linear unabhängig ist von den bereits vorhandenen Variablenvektoren. Ist das der Fall, dann teilt er es dem Partner mit und dieser übermittelt dann den Vektor:

  $$\begin{aligned} r^T \cdot c &= (r_1 \ \cdots \ r_t) \cdot \begin{pmatrix} c_1 \\ \vdots \\ c_t \end{pmatrix} \\ &= (r_1 \ \cdots \ r_t) \cdot \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{t,1} & \cdots & a_{t,m} \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \\ &= (d_1 \ \cdots \ d_m) \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \,. \end{aligned}$$

Der Vorteil der Netzwerkkodierung gegenüber der Vorwärts-Fehlerkorrektur ist, dass die Peers keine Entscheidung treffen müssen, wie die Daten weiterzugeben sind. Sie erstellen eine Mischung aus allen Codes, die sie bisher erhalten haben.

12.7 Zusammenfassung

Die Netzwerkkodierung löst das *Coupon-Collector-Problem* bestmöglich. Ein gewisser Nachteil ergibt sich aus der Übertragung der Variablenvektoren, die zu einer Aufblähung des Datenvolumens führen. Beispielsweise sind für die Übertragung einer 4 GByte-Datei in Teilen von je einem MByte Vektoren der Größe 4.096 notwendig. Das ist ein Datenanteil von vier Promille. Wählt man kleinere Vektoren, so erhöht sich der Aufwand entsprechend. Ein weiteres Problem ist auch die Invertierung der Matrix. Bei 4096 Einträgen ist eine Matrix mit 16 Mio. Einträgen zu invertieren. Das bedingt einen erheblichen Berechnungsaufwand. Daher schneidet auf dem Papier die Netzwerkkodierung zwar besser ab, wenn leistungsstarke Rechner mit schlechter Bandbreite verbunden sind. In der Praxis hat sich die Netzwerkkodierung noch nicht durchgesetzt, da der Ansatz von Bittorrent mit erheblich geringerem Berechnungsaufwand die Verteilung sicherstellen kann. Zwar schneidet Bittorrent in den Vergleichen von [112] wesentlich schlechter ab, diese sind jedoch nicht unumstritten.

13

Peer-to-Peer-Netzwerke in der Praxis

In case of major discrepancy, it's always reality that's got it wrong.
Douglas Adams

Peer-to-Peer-Netzwerke sind seit ihrem ersten Auftreten ein Massenphänomen und werden in der Öffentlichkeit gleichgesetzt mit illegalem File-Sharing. In diesem Kapitel werden populäre Peer-to-Peer-Netzwerke betrachtet. Die Popularität ist hierbei schwer einzuschätzen. Denn die meisten Daten werden ohne Erlaubnis der Rechte-Inhaber verteilt, und diese gehen in letzter Zeit vermehrt dagegen vor. Auch versuchen zuweilen Netzwerkadministratoren und Internet-Service-Provider, die Teilnahme an Peer-to-Peer-Netzwerken zu unterbinden. Da das zumeist über die Identifikation von Ports geschieht, segeln die Peers nun oft unter wechselnden oder falschen Ports, d.h., die Netzwerke sind nicht mehr über eine einfache Port-Analyse identifizierbar, wie es IP eigentlich vorsieht. Also muss der Verkehr selbst analysiert werden, um den Anteil der Peer-to-Peer-Netzwerke zu bestimmen. Auf die Selbstauskunft der Teilnehmer ist wenig Verlass. Umfragen in Vorlesungen über Peer-to-Peer-Netzwerke zeigten dies deutlich.

13.1 FastTrack

Das *FastTrack*-Protokoll wurde von Kiklas Zennström, Janus Friies und Jaan Tallinn entwickelt. Dasselbe Team hat auch das Peer-to-Peer-Netzwerk für *Skype* entwickelt, das momentan eines der populärsten Internet-Telefonsysteme ist.

FastTrack folgt einer hybriden Peer-to-Peer-Netzwerk-Struktur. Peers werden in zwei Klassen unterteilt: normale Peers (*Nodes*) und besondere Peers (*Super-Nodes*). Diese Super-Nodes werden aufgrund ihrer besonderen Fähigkeiten ausgewählt, wenn sie besonders gute Netzwerkeigenschaften besitzen: hohe Bandbreite und Verfügbarkeit. Wie wir bereits in Kapitel 2 gesehen haben, sind viele Hosts im Internet nicht in der Lage, ohne weiteres Verbindungen miteinander aufzunehmen. Das liegt an der Verwendung von DHCP, NAT und Firewalls. Auch sorgen die

weit verbreiteten asymmetrischen Verbindungen, insbesondere DSL und ISDN, zu schlechter Upload-Bandbreite. Solche Peers sind als Super-Nodes ungeeignet.

Die Super-Nodes sorgen für die effiziente Suche. Wie diese Suche funktioniert, ist nicht bekannt, da die Server-Protokolle (die Kommunikation zwischen Super-Nodes) von FastTrack nicht veröffentlicht worden sind. Die Client-Server-Protokolle (die Kommunikation zwischen Super-Nodes und Nodes) sind inzwischen durch *Reverse-Engineering* analysiert worden.

Sämtliche Kommunikation wird über Port 80 abgewickelt, das ist der Port für das HTTP-Protokoll. Der Grund ist, dass dieser Port von den meisten Netzwerk-Administratoren nicht eingeschränkt wird, da die Endbenutzer mit einer Internetverbindung zuallererst eine Anbindung per HTTP zum WWW sehen. Es ist in dieser Hinsicht bemerkenswert, dass viele Netzwerkadministratoren elementare Protokolle, wie zum Beispiel SMTP (*Simple Mail Transfer Protocol*) oder SSH (*Secure Shell*), in Netzwerken unterbinden (insbesondere im W-LAN). Das Ergebnis dieser Einschränkung ist nicht etwa eine Erhöhung der Sicherheit, sondern nur ein Verlust an Transparenz, da nun falsche Ports (siehe FastTrack) zur Kommunikation verwendet werden.

Die offizielle Clientsoftware von FastTrack ist *Kazaa*. Da nun der HTTP-Port missbraucht wird, ist mit der Installation ein erheblicher Eingriff in die Netzwerkkomponenten des Betriebssystems verbunden. Von der Installation des Clients wird allgemein abgeraten, da dieser Eingriff zur Installation von *Malware* missbraucht wird. *Malware* (Malicious Software) sind schädliche Programme, die ohne Wissen des Benutzers unerwünschte Aktionen durchführen. Insbesondere sollen das bei Kazaa *Adware* und *Spyware* sein. Unter *Adware* (*Advertizing Supported Software*) versteht man Programme, die automatisch Werbung auf dem Bildschirm darstellen, während das Programm läuft. *Spyware* (*Spy Software*) sind Programme, die ohne Zustimmung des Benutzers Daten an Dritte weitergeben (z.B. durch Internet-Verbindungen). Da Kazaa durch Zugriff auf den HTTP-Port das Surfverhalten der Benutzer beobachten könnte, ist dies sicher nicht unproblematisch.

Nachdem die Funktionsweise der Kommunikation zwischen normalen Peers und Super-Nodes durch Reverse Engineering verstanden wurde, ist *Malware*-freie Software, wie z.B. *Kazaa-Lite*, verfügbar.

13.2 Gnutella-2

Der Nachfolger des ursprünglichen Gnutella-Netzwerks, das wir in Abschnitt 3.2 vorgestellt haben, ist *Gnutella-2*. Es wurde von Michael Stokes entwickelt und ist seit seiner Vorstellung Ende 2002 in der Peer-to-Peer-Netzwerk-Gemeinde sehr umstritten [113]. Im Gegensatz zu anderen P2P-Netzwerken werden die Protokolle von Gnutella und seinen Nachfolgern veröffentlicht und diskutiert.

Gnutella2 verwendet, wie auch FastTrack, eine hybride Netzwerkstruktur aus Peers und *Super-Peers*, hier Blätter (*Leaf*) und Naben (*Hub*) genannt. Auf diese Weise lassen sich Zuverlässigkeit und Effizienz der Suche verbessern, da die Indexdaten auf den Super-Peers gespeichert werden, siehe Abbildung 13.1.

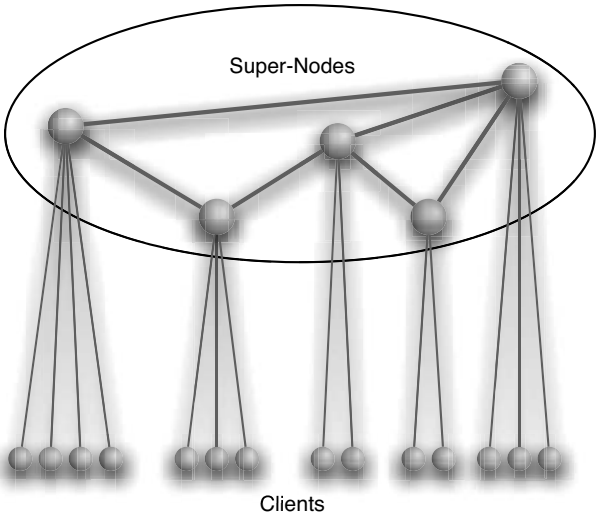


Abb. 13.1. Eine hybride Netzwerkstruktur mit Peers und Super-Peers.

Die Vorteile einer solchen Netzwerkstruktur sind die verbesserte Skalierbarkeit und geringere Latenzzeiten. Anderseits verlässt so Gnutella-2 die reine Lehre vom Peer-to-Peer-Prinzip. Dies ist nicht unproblematisch: So könnte sich zum Beispiel ein Client der Aufgaben eines Super-Peers verweigern.

Netzwerkstruktur

Die Peers unterhalten ein bis zwei Verbindungen zu Super-Peers. Super-Peers akzeptieren Hunderte von Peers und eine große Menge von Verbindungen zu anderen Peers, siehe Abbildung 13.1. Der Netzwerkaufbau ist fast analog zum ursprünglichen Gnutella-Protokoll, bezogen auf *Bootstrapping*.

Speicherung und Suche

Jeder Super-Peer speichert die Indexdateien aller Peers, mit denen er verbunden ist. Anfragen werden nur zwischen Super-Peers weitergeleitet, es sei denn, die Indexdatenbank des Super-Peers zeigt, dass der Peer das gesuchte Datum besitzt. Dann wird die Anfrage an diesen Peer weitergeleitet.

Bei der Anfrage selbst wird kein Fluten durchgeführt, sondern der zuständige Super-Peer eines Peers wählt eine Menge von anderen Super-Peers aus und kontaktiert diese hintereinander.

Dadurch wird die Suche berechenbarer und das Netzwerk robuster gegen Attacken. Schließlich kann die Suche von einem Super-Peer abgebrochen werden, sobald eine genügend große Treffermenge konstruiert worden ist.

In diesem Zusammenhang muss noch erwähnt werden, dass man in der letzten Erweiterung des ursprünglichen Gnutella-Protokolls ebenfalls eine hybride Struktur vorgesehen hat. Hier werden die Peers ebenfalls Blätter genannt, während die

Super-Peers hingegen *Ultra-Peers* heißen. Der Hauptunterschied ist neben einem unterschiedlichen Paketformat der, dass die Ultra-Peers weniger Peers verwalten und dafür sehr viele Verbindungen zu anderen Ultra-Peers unterhalten. Damit ist Gnutella-2 zentraler organisiert als Gnutella, das die Ultra-Peers als abkürzendes Rückgrat verwendet.

13.3 eDonkey

Das *eDonkey*-Netzwerk besteht aus einer Client-Server-Struktur und arbeitet ähnlich wie *Napster*. Es wurde von Jed McCaleb entwickelt. Auf dem Server läuft ein bestimmtes Server-System, zumeist *Lugdunum*. Die Teilnehmer verwenden als Client-Software z.B. eDonkey oder *eMule*. Die Aufgabe von eDonkey ist klassisches File-Sharing. Clients können Dateien zur Verteilung freigeben oder andere Dateien herunterladen. Die Server verwalten nur den Index der freigegebenen Datein und die Client-Adressen.

Bei der Anmeldung teilen die Clients dem Server mit, welche Daten zum Download zur Verfügung stehen. Diese Indexdateien speichert der Server, was insbesondere bedeutet, dass er keine (evtl. urheberrechtlich geschützten) Dateien speichert. Diese verbleiben beim Client. Zur Suche kontaktiert der Client den Server und erhält die Indexinformation, die zur Suchanfrage passt. Dann kontaktiert der Client den so gefundenen Client und kann die Datei direkt (ohne den Server) herunterladen. Dies ist die einzige Peer-to-Peer-Komponente im eDonkey-Netzwerk.

Die Client-Software erlaubt nun mehrere Server in eine Server-Liste aufzunehmen. So werden dann gleichzeitig mehrere Server nach der gewünschten Datei durchsucht. Dadurch erhöht sich die Ausfallsicherheit des Netzwerks. Zusätzlich kann auch jeder Teilnehmer selbst einen eDonkey-Server zur Verfügung stellen. Das ist aber nur bei sehr guter Netzwerkanbindung sinnvoll, die über der durchschnittlichen Netzwerkkapazität eines Benutzers liegt.

Durch die Server-Listen bei den Clients und die unterschiedliche Kapazität der Server gibt es einen gewissen Trend zur Konzentration bei eDonkey. Das führt zu Einbrüchen, wenn diese Server nicht mehr zur Verfügung stehen, wie z.B. im Fall des „Razorback 2.0"-Servers, der im Frühjahr 2006 von der belgischen Polizei abgeschaltet wurde. Zwar liegt die Urheberrechtsverletzung nicht beim Server vor, da dieser jedoch eine Vermittlerrolle spielt, gibt es juristische Gründe gegen die Server vorzugehen. Bei den Clients reicht schon die Erstellung der Links von Dateien, um eine Urheberrechtsverletzung zu begehen. Diese werden in letzter Zeit in einigen Ländern verfolgt (seit 2006 auch in Deutschland).

Neben echten Servern gibt es noch so genannte *Fake-Server*. Diese Server gaukeln den Clients eine große Nutzerzahl und Dateivielfalt vor. Sie zielen aber nur darauf ab, von den Nutzern Suchanfragen einzusammeln und diese dann an Ermittlungsbehörden oder Konzerne weiterzugeben. Das hat dann zur Folge, dass aufgrund der IP-Adresse die Endnutzer ermittelt werden können. Dann kann es vorkommen, dass bei einer Hausdurchsuchung die Rechner beschlagnahmt und der Inhalt ihrer Festplatten entsprechend untersucht werden.

13.4 Overnet und Kademlia

Das *Overnet*-Netzwerk wurde — wie schon das eDonkey-Netzwerk — von Jed McCaleb entwickelt. Overnet ist im Gegensatz zu eDonkey ein echtes Peer-to-Peer-Netzwerk ohne jegliche Server-Struktur und wurde im Jahr 2003 in den eDonkey-Client integriert. Dies sorgte für eine weite Verbreitung.

Als Routing-Algorithmus verwendet Overnet das *Kademlia*-Netzwerk, das von Petar Maymounkov und David Mazières als Peer-to-Peer-Netzwerk entwickelt wurde [14]. In Kademlia erhalten Peers durch Anwenden einer Hash-Funktion auf die IP-Adresse eine ID (und einen Zeitparameter). Jeder Peer unterhält Kanten zu den k-nächsten Peers bezüglich der XOR-Metrik auf diesen IDs. Die XOR-Metrik ist definiert als

$$\text{XOR-Distanz}(A, B) = A \oplus B \ .$$

Die bitweise XOR wird also als ganze Zahl interpretiert. Somit führen die 160-Bit-Adressen zu 2^{160} verschiedenen Adressen.

Für jedes Abstandsintervall $d \in [2^i, 2^{i+1} - 1]$ für $i \in \{0, \ldots, 159\}$ unterhält jetzt jeder Peer k Nachbarn, wobei diese nach dem *Least-Recently-Used*-Prinzip ausgewählt werden. Damit wird bei näherer Betrachtung genau das Plaxton-Routing-Netzwerk benutzt, siehe Seite 95. Hierbei wird das binäre Alphabet genutzt und k Nachbarn aus der primären Nachbarschaft gewählt. Neben dem Verzicht auf weitere Zeiger ist der Hauptunterschied, dass die IDs alle 24 Stunden erneuert werden, so dass alte Zeiger nicht aktualisiert werden müssen, sondern nach 24 Stunden einfach gelöscht werden können.

Der Grad von Kademlia ist daher maximal $160k$. Wenn die Routing-Tabelle korrekt aktualisiert worden ist, dann kann man nach einem Datum suchen, indem man wie beim Plaxton-Routing mit jedem Routing-Schritt ein Bit der Adresse anpasst. Die Suche nach einem Datum ist daher nach $\mathcal{O}(\log n)$ Sprüngen erfolgreich. Das Kademlia-Netzwerk hat somit einen erwarteten Durchmesser von $\mathcal{O}(\log n)$.

Der Unterhalt von Kademlia ist extrem einfach. Benachbarte Peers tauschen ihre Information über Nachbarn aus. Finden sich so nähere Nachbarn, wird der Link auf diese gesetzt und die Verbindung zu den entfernteren beendet. Mit diesem selbstorganisierenden Algorithmus repariert sich das Peer-to-Peer-Netzwerk.

13.5 Bittorrent

Die Funktionsweise von Bittorrent wurde im vorangegangenen Kapitel vorgestellt. Verschiedene Studien von Firmen (die sich der Bekämpfung von Peer-to-Peer-Netzwerk-Verkehr verschrieben haben) behaupten, dass Bittorrent in Deutschland im Jahr 2006 je nach Tageszeit einen Anteil zwischen 25% und 40% des Internet-Verkehrs besitzt [114, 6]. Diese Zahlen sind mit Vorsicht zu genießen, denn schließlich haben die Firmen ein Interesse daran, viel Peer-to-Peer-Netzwerk-Verkehr nachzuweisen. Der Gesamtanteil des Peer-to-Peer-Filesharing soll gemäß dieser Untersuchungen zwischen 50% tagsüber und 80% nachts sein.

Obgleich die Server, die die Verbindungen vermitteln, keinerlei Daten speichern, versteht die Rechtssprechung in einigen Ländern den Betrieb eines solchen Servers als unzulässige Hilfestellung zur Verletzung von Urheberschutzrechten. Zusätzlich bieten diese Server auch Suchfunktionen über alle Torrents an, so dass das Manko der mangelnden Suchfunktion von Bittorrent durch eine Server-Client-Struktur ausgeglichen werden kann. Seit 2004 kam es immer wieder zu Beschlagnahmungen und Stillegungen dieser Torrent-Server, angefangen in Finnland (*Finreactor*), den USA in 2005 (*EliteTorrents*, *LokiTorrent*) und dem bis dato größten Server in Slowenien (*Supernova*). Dennoch gibt es eine Reihe von aktiven Servern, die in Ländern wie den Niederlanden, Schweden und auch den USA stehen.

13.6 Skype

Als Internet-Telefonsystem hat sich die Peer-to-Peer-Netzwerk-Technologie in einem ganz anderen Bereich durchgesetzt. *Skype* wurde von Nikals Zennström und Janus Friis (den Gründern von Kazaa) entwickelt. Ähnlich wie bei Kazaa ist von *Skype* nicht bekannt, wie die Netzwerkstruktur genau funktioniert. Außerdem wird die Skype-Software fortwährend um neue Features, wie z.B. Videokonferenz, erweitert. Mittlerweile sind über fünf Millionen Teilnehmer gleichzeitig aktiv.

Skype bietet folgende wesentliche Grundfunktionalitäten: die direkte *VoIP*-Verbindung (*Voice over IP*) zwischen zwei Rechnern ohne Zwischenknoten. Aber auch auf Rechnern, die sich hinter einer Firewall oder NAT befinden, arbeitet Skype zumeist einwandfrei. Zusätzliche Funktionen sind eine Suchfunktion nach Teilnehmern und die Möglichkeit das bestehende Telefonnetz einzubinden, d.h., Anrufe von dort und dorthin abzuwickeln. Es gibt Konferenzschaltungen, Video-Verbindungen, Chats, Dateitransfer und zudem kann man Skype auf den Plattformen Windows XP, Linux und Mac OS betreiben. Das und die weitgehend robuste und kostenlose Verfügbarkeit sind ein Grund für den Erfolg von Skype.

Skype verschlüsselt die Gespräche und bestimmt wie Kazaa bestimmte Peers automatisch als Super-Peers. Wie diese Netzwerkstruktur aber im einzelnen funktioniert und wie sicher die Verschlüsselungen sind, lässt sich nur mutmaßen. Erste Erkenntnisse durch Nachrichtenanalyse wurden in [3] veröffentlicht. Auf der positiven Seite ist zu erwähnen, dass in Skype bisher keine Malware entdeckt worden ist.

14

Ausblick

Es reicht.
Immanuel Kant - Letzte Worte, 12. Februar 1804

Den Begriff der Peer-to-Peer-Netzwerke gibt es erst seit kurzem und er wird in der Öffentlichkeit verbunden mit illegalem File-Sharing. In diesem Buch haben wir gesehen, dass die Idee der Peer-to-Peer-Netzwerke älter ist. Schon die Erfinder des Internets haben eigentlich ein Peer-to-Peer-Netzwerk im Sinne gehabt. In diesem Ausblick werden wir uns mit der Frage beschäftigen, ob Peer-to-Peer-Netzwerke zwangsläufig etwas mit File-Sharing zu tun haben. Außerdem werden wir versuchen die Frage zu beantworten, wie verantwortungsbewusste Peer-to-Peer-Netzwerk-Benutzer und -Designer sich verhalten müssen, um nicht mit dem Gesetzgeber in Konflikt zu kommen. Da beide Autoren nur juristische Laien sind, kann für die Korrektheit und Richtigkeit der Aussagen hier keinerlei Garantie gegeben werden.

14.1 Anwendungen

File-Sharing

Die bekannteste Anwendung für Peer-to-Peer-Netzwerke ist File-Sharing. In der Regel kann ein Peer bestimmte Dateien oder ganze Unterverzeichnisse für andere Peers zum Kopieren zur Verfügung stellen. Zwar sieht man oft, dass File-Sharing mit Dateiaustausch übersetzt wird, tatsächlich trifft das aber den Sachverhalt nicht korrekt. Schließlich erwartet der *Sharer* keine Gegenleistung. Sharing kann auch *Mitteilen* bedeuten und File-Sharing bedeutet daher nichts anderes als Datei-Verteilung. File-Sharing an sich ist kein illegales Ziel. Es gibt eine große Bibliothek an Programmen, Bildern, Filmen und Musik, die entweder mit ausdrücklicher Erlaubnis der Autoren veröffentlicht werden dürfen oder bei denen die Urheberrechte verjährt sind oder aufgegeben wurden. Bei Software kennt man hierzu das *Freeware* und

Shareware-Konzept. Freeware (*Free Software*) sind Programme, die von den Autoren umsonst oder gegen eine freiwillige Spende zur Verfügung gestellt werden. Shareware ist dagegen keine kostenlose Software. Der Code ist zwar frei verfügbar, aber die Software läuft in einem eingeschränkten Demonstrationsmodus. Die volle Funktionalität bekommt man erst durch Lizensierung gegen Gebühr. Darüber hinaus gibt es auch noch das so genannte *Copyleft*-Konzept, das nicht nur den Anspruch auf Vergütung der Urheberschaft aufgibt, sondern auch alle zwingt, die das Softwareprodukt verwenden oder modifizieren, dies auch ohne *Copyright*-Anspruch zu tun. Hierunter fallen auch Programme, die unter der *GPL* (*Gnu Public License*) veröffentlicht wurden. Daneben existieren auch abgewandelte Lizenzbedingungen, die das Kopieren von Dokumenten ausdrücklich erlauben.

File-Sharing im Internet ist nichts Neues. Früher verwendete man hierfür FTP-Server (*File Transfer Protocol*) oder später dann auch Web-Server. Da aber 1999 Privatnutzer kaum eigene Web-Seiten hatten und der Raum für *Accounts* auf kostenfreien Web-Servern seinerzeit sehr beschränkt war, gab es mit Napster eine Lücke, die die Nutzer stark nachgefragt haben.

Der Schritt von verbotenem File-Sharing zu erlaubtem ist ohnehin sehr klein. Im Moment werden *Online-TV-Rekorder* angeboten, mit denen man Filme von einem Server aufzeichnen lassen kann und dann mit seinem Client-Rechner entweder gegen eine kleine Gebühr oder sogar ganz unentgeltlich herunterladen kann. Wenn dieses Angebot in ein Peer-to-Peer-Netzwerk integriert wird (wie zum Beispiel [115]), so fällt dieses File-Sharing-Modell unter die Bedingungen eines (wenn auch sehr seltsamen) Videorekorders, was unter gewissen Einschränkungen bis 2005 legal war (siehe hierzu auch Landgericht Leipzig, Aktenzeichen 05 O 4391/05). Mit dem Urteil vom Landgericht Braunschweig (Urt. v. 07.06.2006 - Az.: 9 O 869/06 (148)) ist dies wohl auch nicht mehr gegeben.

World-Wide-Web

Eine alternative Anwendung von Peer-to-Peer-Netzwerken ist das World-Wide-Web. Bisher sind die Web-Seiten auf bestimmten Web-Servern verfügbar, die von Web-Clients mit besonderer Software, den Web-Browsern, abgerufen werden können. Die Schwächen dieses Ansatzes sieht man in Extremsituationen. So können Web-Server unter dem Ansturm von legitimen oder illegitimen (Denial of Service Attack) Anfragen zusammenbrechen. Auch kann der Nutzerkreis der Web-Clients eingeschränkt werden. So konnte zum Beispiel kein europäischer Nutzer im US-amerikanischen Präsidentschaftswahlkampf (2004) auf die Web-Seite der republikanischen Partei zugreifen. Hierfür hat der Web-Server die IP-Adresse in den Paketen analysiert und Zugriffe aus europäischen Domains abgewiesen.

Es ist schon bekannt, dass verteilte Hash-Tabellen so genannte Hot-Spots im Internet entlasten können [9]. Dies ist ein Geschäftsfeld für Firmen wie zum Beispiel Akamai, die Server-Farmen anbieten, und somit Anfragespitzen abfedern. In diesen Fällen könnte aber auch ein Peer-to-Peer-Netzwerk Abhilfe schaffen.

Publius [116] stellt eine verteilte Version des HTTP (*Hypertext Transfer Protocols*) bereit, das die Server-Struktur durch ein Peer-to-Peer-Netzwerk ersetzt. Es hat

ebenfalls die Anonymisierung als Zielsetzung und verwendet zur Adressierung die URL-Adressen (*Uniform Resource Locator*), die man von HTTP her kennt.

Internet-Telefonie

Skype erfreut sich in der letzten Zeit zunehmender Popularität. Über 7 Millionen Nutzer waren Ende 2006 damit verbunden. Diese Tatsache lässt sich sicher auf die benutzerfreundliche Software, den kostenfreien Download der Software und den kostenlosen Dienst (für Gespräche im Internet) zurückführen. Was aber Skype gegenüber anderen Dienstanbietern hervorhebt, ist die Fähigkeit, selbst von Rechnern, die sich hinter einer Firewall befinden, Internet-Telefonverbindungen anzubieten. Weitere Anbieter in diesem Bereich sind *Gizmo* oder *Jajah*.

E-Mail

Normalerweise sind E-Mail-Dienste klassische Client-Server-Anwendungen, in denen ein E-Mail-Client von einem E-Mail-Server mit Protokollen wie POP3 (*Post Office Protocol Version 3*) oder IMAP (*Internet Mail Access Protocol*) die E-Mails abholt, während die E-Mail-Server per SMTP (*Simple Mail Transfer Protocol*) oder deren Nachfolgerprotokollen miteinander kommunizieren. Diese E-Mail-Server stellen nun einen künstlichen Flaschenhals dar. Denn wenn sie nicht verfügbar sind, kann der Nutzer nicht auf seine E-Mail zugreifen. Das Problem kann durch einen Peer-to-Peer-Ansatz gelöst werden. Mit *E-POST* [117, 118] wird ein Peer-to-Peer E-Mail-Dienst zur Verfügung gestellt, der diese Schwachstelle überwinden soll.

Grid-Computing

Ein den Peer-to-Peer-Netzwerken verwandtes Gebiet ist das *Grid-Computing*. Auch hier wird ein virtuelles Netzwerk zwischen gleichberechtigten Endknoten gespannt. Grid-Computing stammt ursprünglich aus dem Bereich des parallelen Rechnens. Die Idee des parallelen Rechnens ist eine Steigerung der Rechenleistung durch Zusammenschluss einer großen Anzahl von Rechnern, die dann eine komplizierte Berechnung unter den Rechnern aufteilen und gleichzeitig (parallel) lösen.

Seit den achtziger Jahren werden daher Parallelrechner oder Rechner-Cluster entwickelt und gebaut. Durch die Fortschritte in der Netzwerk-Technologie braucht es neuerdings keine besondere Hardware mehr. So können dezentral verteilte Institutionen ihre Rechenressourcen koppeln und zu einem verteilten Super-Computer kombinieren. Genau dies bezeichnet man mit Grid-Computing.

Es gibt viele Gemeinsamkeiten von Grid-Computing und Peer-to-Peer-Netzwerken. Der Hauptunterschied ist, dass Grid-Computer zentral administriert werden. Die Koordination der Berechnung erfolgt zentral durch einen Rechner und die Ausfallrate aller beteiligten Rechner ist wesentlich geringer. Außerdem kann man in der Regel von einer spezifizierten Hardware mit gleichartiger Programmumgebung ausgehen. Die beteiligten Rechner werden nicht egoistisch betrieben und sind durch sehr leistungsfähige Kommunikationsverbindungen miteinander vernetzt.

Man kann also beim Grid-Computing von einem Bruder der Peer-to-Netzwerke reden, weil beide aus den Erkenntnissen des verteilten und parallelen Rechnens

schöpfen. Letzthin ist aber das Anforderungsprofil völlig unterschiedlich, und nur sehr ausgewählte Probleme, die man mit Grid-Computing löst, kann man ohne weiteres auf Peer-to-Peer-Netzwerken berechnen.

Weitere Anwendungsmöglichkeiten

Weitere Anwendungsmöglichkeiten sind *Groupware*, z.B. *Groove* und *Distributed Content Management*, z.B. *Osprey* oder verteilte Web-Suchmaschinen, wie zum Beispiel *Yacy*.

Andere Anwendungsmöglichkeiten von Peer-to-Peer-Netzwerken sind Fernwartung von Programmen und Software-Updates. All diese Anwendungsmöglichkeiten nutzen das Fehlen von Schwachstellen und Engpässen in Peer-to-Peer-Netzwerken. Dadurch werden auch große Server-Farmen überflüssig, und Unternehmen können somit Kosten einsparen.

14.2 Juristische Situation

Das Internet spannt sich über den gesamten Globus. Während TCP, UDP und IP in allen Ländern die etablierten Kommunikationsprotokolle sind, ist die Rechtsprechung von Land zu Land verschieden. Noch nicht einmal die Länder der EU sind in der Rechtsprechung einheitlich.

Für die Endnutzer, Designer und Betreiber von Peer-to-Peer-Netzwerken gibt es eine Unzahl von möglichen juristische Fragestellungen. Diese betreffen

- die Erlaubnis des freien und unkontrollierten Meinungsaustauschs,
- den Besitz und die Weitergabe verbotener Dokumente,
- den Einsatz kryptographischer Protokolle,
- die Erlaubnis der Überwachung und Einsichtnahme in fremde Nachrichten,
- die illegale Nutzung fremder Dienste,
- die Urheberschutzbestimmungen.

Wir möchten von den ersten fünf Punkten ein paar Problemfelder benennen, bevor wir uns dem letzten Punkt ausführlicher zuwenden.

Alle Demokratien fußen auf dem freien und unkontrollierten Meinungsaustausch zur öffentlichen Meinungsfindung. Man darf aber nicht vergessen, dass durch das Internet auch Bürger von Diktaturen verbunden werden. Der damit einhergehende freie Informationsaustausch kann den Herrschenden gefährlich werden. Daher beschränken diese in solchen Ländern den Zugang zum Internet, kontrollieren die verfügbaren Dienste oder verbieten die Benutzung des Internets mitunter ganz. Das ist naturgemäß bei der Client-Server-Kommunikation einfacher als bei Peer-to-Peer-Verbindungen. Ein Peer-to-Peer-Netzwerk, das dahingehend konstruiert wurde, sicher und anonym Dokumente zu verteilen, wie zum Beispiel *Freenet*, ist daher für solche Regime äußerst unangenehm und deren Benutzung ist für die Bürger in solchen Ländern sehr gefährlich. Somit können auf der anderen Seite die Designer von sicheren Peer-to-Peer-Netzwerken nicht vorsichtig genug sein bei der Erstellung und

dem Unterhalt dieser Netzwerke, da unter Umständen Leben von der korrekten und sicheren Funktionsweise der Sicherheitsmechanismen abhängen.

Aber selbst in Demokratien darf nicht jedes Dokument gespeichert oder weitergegeben werden. In Deutschland ist es strafrechtlich verboten, volksverhetzende Schriften (§ 130 Strafgesetzbuch StGB), im Klartext Nazipropaganda, und Kinderpornographie (§ 184 b Strafgesetzbuch StGB) zu besitzen. Das hat in der Praxis schon dazu geführt, dass ein Verband, der es sich zur Aufgabe gemacht hat, die Polizei auf die Spur von Kinderpornographie im Internet zu führen, sich beim ersten Fund schon selbst strafbar macht. Insbesondere muss die Polizei dann Anklage gegen ihre eigenen Helfer erheben. Andere Beispiele von juristisch relevanten Feldern sind die Preisgabe von Staats- und Amtsgeheimnissen, Industriespionage, Verletzung des Steuergeheimnisses, Verletzung des Datenschutzes, Beleidigung und natürlich Verletzung des Urheberrechts. Wer also für andere Daten speichert und weiterleitet, muss sich darüber im Klaren sein, dass dies mit erheblichen juristischen Konsequenzen verbunden sein kann.

Es ist daher nicht verwunderlich, dass selbst demokratische Staaten, wie zum Beispiel Frankreich, neben nichtdemokratischen, wie zum Beispiel China und Saudi-Arabien, versuchen, die Verwendungen kryptographischer Methoden einzuschränken. Die Regelungen stammten oftmals aus einer Zeit, als Kryptographie nur mit Hilfe spezieller Geräte eingesetzt werden konnte. Spätestens mit der Entwicklung der RSA-Methode, bei der die Kryptographie mit den Grundrechenarten durchgeführt werden kann, ist jeder Rechner ein Ver- und Entschlüsseler und diese Gesetze haben sich überlebt. Wer Kryptographie verwenden will, der kann das unbemerkt tun (z.B. mit Hilfe der Steganographie). In Deutschland ist der Einsatz von Kryptographie (zumindest im Internet) erlaubt.

Das Deutsche Fernmeldegesetz verbietet die Überwachung und Einsichtnahme in fremde Nachrichten. Nur mit einer richterlichen Erlaubnis darf der Staat Nachrichten kontrollieren. Interessanterweise gelten diese Bestimmungen nicht für die Kommunikation mit oder über das Ausland. Da im Internet in der Regel nicht der Verkehr nach nationalen Gesichtspunkten geregelt wird, ist letztlich jede digitale Kommunikation über das Internet nicht vollständig vor dem Zugriff des Staates sicher. Eine andere Frage ist, ob man als Benutzer eines Peer-to-Peer-Netzwerks die Kommunikation anderer Benutzer kontrollieren darf oder sogar muss. Wenn sie juristisch als Nachricht nach dem Fernmeldegesetz zu sehen ist, dann darf sie nicht kontrolliert werden (noch nicht einmal durch Automaten, wie es sich Google-Mail in der Benutzervereinbarung vorbehält). Ist sie es nicht, so entsteht unter Umständen die Pflicht des Benutzers zu verhindern, dass verbotene Dokumente weitergegeben werden.

Viele Benutzer verwenden Firmen- oder Universitätsnetzwerke, um ihre Peer-to-Peer-Netzwerk-Clients mit dem Peer-to-Peer-Netzwerk in Verbindung treten zu lassen. Hierbei kann es sich um eine illegale Nutzung handeln. Dies hängt von den allgemeinen Nutzungsbedingungen des lokalen Netzwerk-Betreibers ab. Da diese Betreiber für illegale Nutzung zur Verantwortung gezogen werden könnten, verwenden sie oft eine sehr restriktive Strategie. Die Frage stellt sich hier nach der Überprüfbarkeit. Wir haben gesehen, dass die Port-Adressen im Internet ihre ursprüngliche Aufgabe weitgehend verloren haben, ironischerweise gerade durch

übervorsichtige Netzwerk-Administratoren und natürlich durch NAT und PAT. Die Überprüfung der Paketinhalte ist nach dem Fernmeldegesetz verboten. Es bleibt dem Netzwerkbetreiber nur die Möglichkeit der automatischen Dokumentation und der Benennung der jeweiligen Verursacher.

Hat der Dienstanbieter die Kontrolle über die verwendete Soft- und Hardware, so kann er natürlich eine nicht beabsichtigte Nutzung verhindern. Eine andere Möglichkeit wäre aber auch eine Veränderung der Kostenstruktur. So florieren Peer-to-Peer-Netzwerke eben gerade, wenn so genannte *Flat-Rates* zur Verfügung stehen. Schon bei minimalen Kosten pro übertragener Information scheuen viele Benutzer die Kosten. Andererseits sind diese Flat-Rates auch eine Reaktion auf die fehlende Kostenstruktur im Internet, die dem Empfänger und nicht dem Verursacher die Kosten unnötiger Datenmengen aufbürdet.

Urheberschutz

Peer-to-Peer-Netzwerke sind so populär geworden, weil sie eine kostenlose Alternative zum Erwerb von Musik-CDs darstellten. Neuerdings werden sie aber zum Großteil zum Verteilen von Spielfilmen, Fernsehserien, Softwareprogrammen und nicht zuletzt Computerspielen verwendet. Diese Verteilung geschieht ohne finanzielle Berücksichtigung der Schöpfer dieser Werke, die durch den so genannten Urheberschutz sichergestellt werden soll.

Historisch gesehen ist der Urheberschutz eine der Errungenschaften der Neuzeit, die die intellektuellen und kulturellen Leistungen von Komponisten, Autoren, Erfindern etc. schützen soll. Zuvor waren diese vor allem auf einzelne Förderer angewiesen, obgleich die Verbreitung ihrer Werke vielen diente. Seit der Erfindung des Buchdrucks ist die massenhafte Vervielfältigung keine große logistische Aufgabe mehr und mit Hilfe von Computern ist das kinderleicht. Das Problemfeld des Schutzes und der Würdigung von Autoren ist von allen Seiten anerkannt.

Seit 2003 werden die Endnutzer von Musik-File-Sharing in den USA und Europa von der RIAA (*Recording Industry Association of America*) und IFPI (*International Federation of the Phonographic Industry*) mit Klagen überzogen. In der Regel kann man eine solche Klage erst einreichen, wenn man den Endnutzer identifizieren kann. Dies kann mit Hilfe der IP-Adresse geschehen, wenn sie nicht durch DHCP oder NAT dynamisch vergeben wird. Da das für die meisten Kunden der Regelfall ist, hat diese Information nur der Internet-Service-Provider, also zum Beispiel in Deutschland T-Online, Arcor, Vodafone etc. Diese Firmen dürfen die Information nicht herausgeben, da sie dem Schutz des Fernmeldegeheimnisses unterliegen. Die Herausgabe kann aber durch richterliche Erlaubnis erreicht werden. Seit August 2003 machen sich Teilnehmer strafbar, wenn sie urheberrechtlich geschützte Inhalte anderen zum Herunterladen bereitstellen. Nach demselben Gesetz (1. Korb des deutschen Urheberrechtsgesetzes) ist das Herunterladen geschützter Inhalte in Deutschland nicht strafbar. Es sei denn, die Dateien stammen aus „offensichtlich illegaler Quelle."

Die Gesetzgebung in Europa ist unterschiedlich bezüglich der Verfolgung von Urheberrechtsverletzungen im Internet. Außerdem werden in verschiedenen Ländern zur Zeit die Regelungen verändert. Die rechtliche Lage ist sehr uneinheitlich und

wird zudem auch noch verschieden gehandhabt. In einigen Ländern ist der Betrieb von Rechnern verboten, die Index-Dateien bereitstellen. Dort finden sogar Polizeirazzien statt, wie zum Beispiel in Slowenien (`Supernova.org`). Andere Länder wie die Niederlande (*Kazaa*-Servers) oder Schweden (`Piratebay.org`) hingegen erlauben dies.

Der Endnutzer darf sich bei unerlaubter Veröffentlichung von urheberrechtlich geschütztem Material in jedem Fall nicht zu sicher fühlen. Manche Client-Server-Netzwerke laden zum Upload von Filmen und Musik ein und stellen Server-Kapazität zum Download zur Verfügung. Dazu wird noch geworben mit absoluter Sicherheit und Anonymität. Ob dieses Versprechen von den Unternehmen noch beherzigt wird, wenn sie als Server-Betreiber vor der Alternative stehen, ob sie oder der jeweilige Endnutzer für die Rechtsverletzung geradestehen, ist, gelinde gesagt, mehr als fraglich.

Konsequenzen für den Designer

Auch die Verfolger von Urheberrechtsverletzungen sind kein monolithischer Block. Die stärkste Rolle spielen hier die USA. Vielleicht weil ein Großteil der internationalen Musik- und Filmprodukte von dort stammen, vielleicht auch weil die USA der größte Markt mit einheitlicher Rechtsprechung ist. Das Mitfilmen im Kino ist dort mittlerweile ein Vergehen und ebenso die Verbreitung von so erhaltenen Filmmitschnitten (*The Family Entertainment and Copyright Act, 2005*). Im universitären Umfeld hört man hier häufig von studentische Wohngemeinschaften, deren Rechner konfisziert wurden wegen illegaler Verbreitung urhebergeschützter Musik. Mittlerweile ist es schon strafbar, ein Peer-to-Peer-Netzwerk zu erfinden und zu programmieren, das zuvorderst der Verletzung des Urheberschutzes dient.

Dieses allgemeine Klima der Verfolgung sorgt für eine Verunsicherung nicht nur bei den Endbenutzern, sondern auch bei den Designern und Betreibern. Fred von Lohmann bemüht sich daher in [119] um eine Klarstellung der rechtlichen Situation für Designer von Peer-to-Peer-Netzwerken oder verwandten Technologien. Er ist Anwalt und arbeitet bei der *Electronic Frontier Foundation* im Spezialgebiet Copyright und Peer-to-Peer-Filesharing. Bezüglich der Urheberrechtsverletzung gibt es in den USA zwei Szenarien:

Direct Infringement (Unmittelbare Urheberrechtsverletzung): Musik, Dokumente, Videos etc. werden ohne die Erlaubnis des Copyright-Besitzers zur Verfügung gestellt.

Secondary Infringement (Mittelbare Urheberrechtsverletzung): durch Peer-to-Peer-Netzwerkbetreiber oder -entwickler. Hierbei gibt es drei mögliche Unterszenarien:

Inducement (Herbeiführung): Voraussetzung für diesen Fall ist, dass eine (auch von Dritten herbeigeführte) Urheberrechtsverletzung stattfindet, diese vom Netzwerkbetreiber oder -entwickler unterstützt und mit Vorsatz betrieben wird. Erst wenn all diese drei Punkte beweisbar vorhanden sind, ist der Entwickler oder Betreiber wegen Herbeiführung haftbar zu machen.

Die Unterstützung kann in der Überredung oder Ermutigung der Copyright-Verletzung liegen. Mit Vorsatz wird diese Copyright-Verletzung insbesondere be-

trieben, wenn damit geworben wird, dass bestimmte Filme dort erhältlich sind. Auch kann das absichtliche Unterlassen von technischen Hilfsmitteln zur Unterbindung der Copyright-Verletzung als Vorsatz gelten.

Contributory infringement (Beihilfe): Beihilfe liegt vor, wenn eine (auch von Dritten herbeigeführte) Urheberrechtsverletzung stattfindet, der Betreiber/Entwickler davon wusste und materiell dies unterstützt hat, z.B. durch Rechner, Sites, Speicherplatz etc.

Vicarious liability (Haftung für das Verhalten Dritter): Auch hier sind drei Punkte notwendig: Urheberrechtsverletzung (auch von dritter Seite), das Recht und die Fähigkeit der Kontrolle dieser und ein direkter finanzieller Vorteil.

Ein Beispiel kann ein Unternehmer sein, der seine Mitarbeiter dazu auffordert, illegale Kopien bestimmter Software auf ihren Firmenrechnern zu installieren. Der finanzielle Vorteil des Unternehmers sind die eingesparten Lizenzgebühren. Das Recht und die Fähigkeit der Kontrolle leitet sich aus der Verwendung der Firmenrechner ab.

Circumvention technologies: Jede Kopiersperre oder Markierungsstrategie, die zur Wahrung der Urhererrecht angebracht worden ist, darf nicht zerstört oder umgangen werden.

Verteidigungsmöglichkeiten: Von Lohmann sieht für Entwickler von Peer-to-Peer-Netzwerken folgende Verteidigungsstrategien.

Die einfachste und sicherste Möglichkeit der Verteidigung ist das Fehlen von Urheberschutzverletzungen auch von Seiten Dritter. Das lässt sich im Allgemeinen kaum vermeiden, denn schon elementare Bausteine des Internets machen Kopiervorgänge möglich.

Ein wichtiger Pluspunkt für eine Verteidigung ist die Fähigkeit zur substanziellen legalen Nutzung. Das System dient einem legalen Zweck, Missbrauchsmöglichkeiten sind aber unvermeidbar darin enthalten.

Eine weitere Möglichkeit hält den Internet-Service-Provider (ISP) den Rücken frei. Dies sind so genannte *Safe Harbors* (Schutzhäfen). Die ISP dürfen für die Weiterleitung, Caching, Speichern im Auftrag von Nutzern (z.B. Web-Site), Unterhalt von Informationslokalisierungswerkzeugen (vulgo: Suchmaschine) urheberrechtgeschützte Informationen vervielfältigen. Damit sie sich aber als Safe Harbor qualifizieren, besteht eine Informationspflicht über die Beendigung des Vertrags bei einer Urheberschutzverletzung. Sie müssen einen *Copyright Agent* als Kontaktperson benennen und bei Beschwerden des Copyright-Besitzers das Material unmittelbar darauf entfernen. Ferner dürfen sie von Verletzungen nichts wissend (den Kopf in den Sand zu stecken, wird hierbei nicht akzeptiert) und es darf kein direkter finanzieller Vorteil aus evtl. trotzdem vorkommenden Urheberschutzverletzungen erwachsen. Um sich als Safe Harbor zu qualifizieren, müssen also von Anfang an gewisse Weichenstellungen gelegt werden. Diesen Prozess später einzuleiten, hat sich als nicht praktikabel erwiesen.

Eine weitere Strategie ist der Verzicht auf Kopien. Weder im RAM noch auf der Festplatte sollte die Beförderung von Nachrichten Kopien erzeugen, die als direkte Copyright-Verletzung angesehen werden kann. Als Betreiber eines Peer-to-Peer-Netzwerks ist das besonders einfach, da ja die Endbenutzer oftmals direkt kommunizieren und auf den Rechnern der Betreiber dann sowieso keine Kopien entstehen.

Jeder Entwickler von Peer-to-Peer-Netzwerk-Technologie sollte vollständig auf Werbung mit Copyright-Verletzungen verzichten. Neben den ethischen Gesichtspunkten kann dadurch Vorsatz oder zumindest Wissen nachgewiesen werden. Außerdem sollte zu Copyright-Verletzungen weder ermutigt noch sollten diese unterstützt werden. Besonders kritisch ist hier die Benutzerberatung. Wenn Kunden direkt fragen: „Ich versuche gerade die Musik von Frank Sinatra herunterzuladen und komme nicht weiter." Dann kann eine technische Auskunft vom Benutzer als Einverständnis in die Aktion interpretiert werden, vor allem wenn der Benutzer im Auftrag des Copyright-Besitzers der Musik von Frank Sinatra beim Kundencenter anruft. Wichtig ist auch, dass man als Unternehmer keinen finanziellen Nutzen aus Copyright-Verletzungen zieht.

Von Lohmann sieht nur zwei Möglichkeiten für den Netzwerk-Betreiber: totale Kontrolle oder das Fehlen jeglicher Kontrollmöglichkeiten. Bei einem Zwischenweg wird man früher oder später zur totalen Kontrolle gezwungen werden, was dann hinterher nicht mehr sehr einfach ist. Daher ist es auch besser die Software als Standalone-Lösung anzubieten statt als Dienstleistung. Bei Dienstleistungen wird man zu viel über das Nutzerverhalten erfahren, was rechtlich wieder problematisch werden kann. Er empfiehlt auch auf ein *End User Licence Agreement* (EULA) zu verzichten, da hier zwischen dem Hersteller und dem Kunden eine Vereinbarung geschlossen wird, die dem Hersteller der Peer-to-Peer-Software als Möglichkeit zur Kontrolle des Nutzers ausgelegt werden kann.

Ganz wichtig ist es, den legalen Nutzen des Peer-to-Peer-Netzwerks zu belegen. Es ist illusorisch zu hoffen, dass die Mehrzahl der Benutzer diese Funktionalität verwenden. Aber ohne legalen Nutzen wird man vor Gericht kaum eine Chance haben.

Hilfreich ist auch die Auslagerung von Funktionen. Als Beispiel kann der Prozess gegen Sony bei der Einführung von Videorekordern dienen. Dass Sony „nur" die Rekorder herstellte und nicht die Videobänder, war ein Schlüssel zum Gewinn des Prozesses.

Open-Source-Software scheint für Peer-to-Peer-Netzwerke die optimale Lösung zu sein. Der Hersteller hat dann weder Kontrolle noch finanziellen Nutzen. Die Software kann nicht vom Markt genommen werden, und kein Zugriff kann verhindert werden durch Filter, Zugangskontrollen oder ähnliches. Es stellt sich natürlich die Frage, wie man dann als Unternehmer ein Geschäftsmodell daraus entwickeln kann. Dieses ist nach der Meinung von von Lohmann durch Zusatzsoftware wie komfortable Bedienungsoberflächen, Suchmaschinen, Bandweitenoptimierer, Dateispeicherung etc. möglich.

14.3 Offene Fragen

Der Peer-to-Peer-Ansatz ist sowohl gesellschaftlich als auch in der Informatik älter als das, was man heute unter Peer-to-Peer-Netzwerken versteht. Wir stecken noch im ersten Jahrzehnt dieser Netzwerktechnologie und die Peer-to-Peer-Netzwerke sind gerade dabei den Kinderschuhen zu entwachsen. Langsam kristallisiert sich heraus, was Standardtechniken sind und welche Art von Problemen relevant sind.

Peer-to-Peer-Netzwerke haben eine große Anziehungskraft auf die Forschergemeinde ausgeübt. Einige Forscher mutmaßen daher, dass Methoden, die man im parallelen und verteilten Rechnen schon vor 20 Jahren entwickelt hat, nun einfach im Gewand der Peer-to-Peer-Netzwerke neu vermarktet werden. Das ist gerade nicht der Fall: Es hat sich im Bereich der Kommunikationsnetzwerke für Parallelrechner der Butterfly-Graph als besonders geeignet herausgestellt. In Peer-to-Peer-Netzwerken wurde dieser zwar in Viceroy auch eingesetzt, aber selbst ihre Erfinder sind inzwischen der Meinung, dass andere Graphkonstruktionen wie Distance-Halving oder das De-Bruijn-Netzwerk besser geeignet sind.

Die größte Konkurrenz von Peer-to-Peer-Netzwerken waren und bleiben Client-Server-Architekturen. Sie haben eine Reihe von Vorteilen gegenüber Peer-to-Peer-Netzwerken: Sie sind einfacher, besser kontrollierbar, besser zu steuern, und mit den Fortschritten in der Rechen- und Speicherkapazität kann man mit ihnen selbst sehr viele Clients bedienen. Es gibt aber Anwendungen, wo dieser Ansatz scheitert. Speziell wenn sehr robuste und selbstständig arbeitende Strukturen gefragt sind, dann eignen sich Peer-to-Peer-Netzwerke allerdings sehr gut.

Peer-to-Peer-Netzwerke leiden noch sehr unter den zumeist asymmetrischen Verbindungen der Endbenutzer, z.B. DSL oder ISDN. Damit besteht fortwährend das Verlangen der Endnutzer mehr Daten herunter- als heraufzuladen (Leecher versus Seeder) und damit sinkt die Bereitschaft der anderen Teilnehmer Daten zur Verfügung zu stellen.

Ein anderes großes Problem sind die Eingriffe der Netzwerkadministration in den Datenverkehr. Hier werden mitunter gezielt Pakete, die als Peer-to-Peer-Netzwerkverkehr identifiziert werden, herausgefiltert. Das führt dazu, dass neuere Netzwerk-Clients den Peer-to-Peer-Netzwerkverkehr als „normalen“ Internetverkehr (etwa HTTP) tarnen. Dadurch nimmt die Transparenz des Datenverkehrs weiter ab und „ehrliche“ Peer-to-Peer-Netzwerkprotokolle mit eigenen Port-Nummern werden abgestraft.

Wegen der Verfolgung der Nutzer des Internets wurden anonyme Peer-to-Peer-Netzwerke entwickelt. Wie in der Kryptographie kann man jetzt nicht mehr zwischen ehrenwerten und unmoralischen Benutzern unterscheiden. Wird hier anonym pornographisches Material ins Netz gestellt, treffen sich Dissidenten zum Gedankenaustausch oder nutzen Terroristen dieses Medium zur Absprache eines Anschlags?

Von der fachlichen Seite aus scheint die Steigerung der Sicherheit der Anonymität mit einem Verlust an Effizienz einherzugehen. Nachrichten müssen über mehr Stationen versandt und kryptographische Funktionen häufiger angewendet werden, möchte man die Sicherheit steigern. Im Moment ist noch nicht klar, ob dieser Zusammenhang zwangsläufig ist oder ob es nicht doch ein Peer-to-Peer-Netzwerk geben kann, das zugleich effizient und sicher ist.

Im Moment ist das größte Problem der rechtliche Widerspruch zwischen der Freiheit und Privatheit der Kommunikation im Internet einerseits und der Kontrolle der Verbreitung urhebergeschützter oder verbotener Inhalte andererseits. Hierzu zwei Standpunkte:

- Sind wir bereit, Software-Entwickler und Internet-Nutzer zu durchleuchten und zu kriminalisieren, um die Interessen der Musik- und Film-Industrie zu wahren?
- Soll die Kunst und die Fähigkeiten herausragender Kultur- und Wissensträger auf illegalen Peer-to-Peer-Netzwerken verramscht werden, um kriminelle Strukturen im Deckmäntelchen der Informationsfreiheit zu schützen?

Erst die weltweite Lösung dieses Widerspruchs werden die Peer-to-Peer-Netzwerke aus dem juristischen Sumpf befreien können. Manche behaupten, dass ein digitales Rechte-Management-System (DRM – *Digital Rights Management*) oder neuartige kryptographische Methoden der Verschlüsselung und digitale Wasserzeichen die Lösung des Problems sind. Man kann sich aber leicht überlegen, dass diese mittels Kamera und Mikrofon aufzuzeichnen sind und so jedes DRM ausgehebelt werden kann. Auf der anderen Seite haben digitale Wasserzeichen die Möglichkeiten zu beweisen, dass hier eine illegale Kopie vorliegt und wer der Empfänger dieser Kopie ist. Es bleibt das Problem herauszufinden, wer diese Kopie nun auf seinem Rechner hat.

Die Zukunft bringt sicher eine allgemeine Erhöhung der Bandbreiten aller Nutzer. Viele Nutzer erwarten von ihrer Internet-Verbindung, Filme in Echtzeit sehen zu können. Werden dann Peer-to-Peer-Netzwerke die Internet-Kapazität sprengen? Im Moment sieht es nicht danach aus. Der Grund ist, dass die Internet-Backbones auf Lichtleiterbasis arbeiten und derzeit kaum ausgenutzt werden. Im Gegensatz zu Kupferkabeln kann man hier technologisch aufrüsten, ohne das Medium (das Kabel) austauschen zu müssen. Nur durch Ersetzen der beiden Endstellen kann man die Datenkapazität vervielfachen. Es kann also davon ausgegangen werden, dass das Internet mit dem Peer-to-Peer-Netzwerk-Verkehr mitwachsen wird und der Einfluss von Peer-to-Peer-Netzwerken noch zunehmen wird.

A

Mathematische Grundlagen

A.1 Algebra

Ein algebraischer *Körper* ist ein kommutativer unitärer Ring, in dem jedes von Null verschiedene Element multiplikativ invertierbar ist, d.h., es müssen folgende Eigenschaften gelten:

1. Addition
   Für die Addition müssen die folgenden Gesetze gelten:
   a) Assoziativität: $a + (b + c) = (a + b) + c$
   b) Kommutativität: $a + b = b + a$
   c) Neutrales Element: Es gibt die Null, also $0 + a = a$
   d) Additives Inverses: Für jedes a gibt es ein Inverses $-a$ mit $a + (-a) = 0$
2. Multiplikation
   a) Assoziativität: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
   b) Kommutativität: $a \cdot b = b \cdot a$
   c) Neutrales Element: Es gibt die Eins, also $1 \cdot a = a$
   d) Zu jedem Element a, das nicht Null ist, gibt es multiplikativ Inverses a^{-1} mit $a \cdot a^{-1} = 1$
3. Es gilt das Distributivgesetz: $a \cdot (b + c) = a \cdot b + a \cdot c$

Aus der Schule kennt man die Körper der rationalen Zahlen, reellen Zahlen und komplexen Zahlen. Hier werden auch endliche Körper verwendet: die so genannten Galois-Körper (mit 2^k Elementen) und Restklassenringen über den Zahlen $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, in denen die Addition und Multiplikation bezüglich Rest p betrachtet wird.

Die Galois-Körper werden nicht über Restklassenringe definiert. Vielmehr betrachtet man hierzu ein Polynom vom Grad $k - 1$ als Zahl

$$p(a) = \sum_{i=1}^{k} a_i x^{i-1} \,,$$

die durch die Bitfolge $a = a_1, \ldots, a_k$ definiert wird. Die Variablen x^i werden niemals eingesetzt oder umgeformt. Sie bleiben als algebraische Rechensymbole stehen. Die Addition zweier Zahlen im Galois-Körper ergibt sich nun durch

$$a + b = p(a) + p(b) \bmod 2 = \sum_{i=1}^{k} 2^{i-1}((a_i + b_i) \bmod 2)$$

Die Multiplikation ergibt sich durch

$$a \cdot b = ((p(a) \cdot p(b)) \bmod q(x)) \bmod 2 \,,$$

wobei $q(x)$ ein irreduzibles Polynom ist, das keine Faktorzerlegung in Polynome $a(x), b(x)$ mit mindestens Grad Eins hat. Ein irreduzibles Polynom vom Grad 2 ist zum Beispiel: $q(x) = x^2 + x + 1$.

Bei der Polynommultiplikation werden jeweils die Variablen x mitmultipliziert:

$$p(a) \cdot p(b) = \sum_{i=0}^{2k-2} \left(\left(\sum_{k=0}^{i} a_{k+1} \cdot b_{i-k+1} \right) \bmod 2 \right) x^i \,.$$

Die Polynomdivision wird analog durchgeführt.

Logarithmen, Kombinatorik und Wachstumsklassen

Wir bezeichnen mit $\ln n = \log_e n$ den natürlichen Logarithmus zur Basis $e \approx 2.718281828\ldots$. Ferner bezeichnet $\log n = \log_2 n$ den Logarithmus zur Basis 2. Mit $\log^k n$ wird $(\log n)^k$ abgekürzt.

Für die Eulersche Zahl e gilt:

$$\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = 1/e$$

und insbesondere für alle $n \in \mathbb{N}$:

$$\left(1 - \frac{1}{n}\right)^n < \frac{1}{e} < \left(1 - \frac{1}{n}\right)^{n-1} \,.$$

Um das asymptotische Wachstum von Funktionen zu vergleichen, wird die so genannte $\mathcal{O}$-Notation verwendet. Für Funktionen $f, g : \mathbb{R}^+ \to \mathbb{R}^+$ wird definiert:

$$f \leq_{ae} g \qquad :\Longleftrightarrow \qquad \exists\, n_0 \in \mathbb{R} \quad \forall\, n \geq n_0 : \quad f(n) \leq g(n) \,.$$

$$\begin{aligned}
\mathcal{O}(g) &:= \{f \mid \exists\, k \in \mathbb{R} \quad f \leq_{ae} k \cdot g\} \,, \\
o(g) &:= \{f \mid \forall\, k \in \mathbb{R}^+ \;\; k \cdot f \leq_{ae} g\} \,, \\
\omega(g) &:= \{f \mid \forall\, k \in \mathbb{R}^+ \;\; k \cdot g \leq_{ae} f\} \,, \\
\Omega(g) &:= \{f \mid \exists\, k \in \mathbb{R} \quad g \leq_{ae} k \cdot f\} \,, \\
\Theta(g) &:= \mathcal{O}(g) \;\cap\; \Omega(g) \,.
\end{aligned}$$

Der Term $f(n) = O(g(n))$ besagt nun, dass die Funktion $f(n)$ bis auf einen konstanten Faktor höchstens so schnell wächst wie $g(n)$. Dieser Term ist äquivalent zu $g(n) = \Omega(f(n))$. Wenn $f(n)$ und $g(n)$ in derselben Wachstumsklasse liegen, dann sind sie bis auf einen Faktor in einer konstanten Schwankungsbreite gleich: $g(n) = \Theta(f(n))$.

Um Aussagen treffen zu können, dass eine Funktion echt stärker wächst als eine andere Funktion, wird nun $f(n) = \omega(g(n))$ verwendet, was das Gegenteil von $f(n) = \mathcal{O}(g(n))$ beschreibt. Für echt schwächer wird $f(n) = o(g(n))$ verwendet, was das Gegenteil von $f(n) = \Omega(g(n))$ beschreibt.

A.2 Graphtheorie

Das wichtigste mathematische Konzept für Rechnernetzwerke ist der Graph. Ein Graph $G = (V, E)$ besteht aus einer Knotenmenge V und einer Kantenmenge E. Man unterscheidet folgende Typen von Graphen:

- Der *ungerichtete Graph* hat als Kantenmenge zweielementige Mengen von Knoten. Der *Grad* eines Knoten im Graphen ist die Anzahl der Kanten in der Kantenmenge E, die diesen Knoten gemeinsam haben. Der *Grad eines Graphen* ist der maximale Grad eines Knotens.
  In Abbildung A.1 ist der Graph mit der Kantenmenge $\{\{1,2\}, \{1,3\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{3,5\}, \{4,5\}\}$ dargestellt.

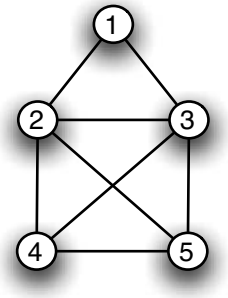


Abb. A.1. Ein ungerichteter Graph.

  Er kann zum Beispiel verwendet werden, um Netzwerke mit bidirektionalen Verbindungen wie etwa TCP-Verbindungen zu beschreiben.
- Der *gerichtete Graph* hat als Kantenmenge Paare (Tupel) von zwei verschiedenen Knoten, wie zum Beispiel (u, v). Zuweilen werden auch Selbstkanten erlaubt. Durch die Reihenfolge wird die Richtung angezeigt. Der erste Knoten zeigt auf den zweiten. Man unterscheidet in gerichteten Graphen *Eingrad* und *Ausgrad*. Der Ausgrad ist die Anzahl der ausgehenden Kanten eines Knotens; der Eingrad

ist die Anzahl der eingehenden Kanten eines Knotens. Spricht man nur vom Grad eines gerichteten Graphen, so ist zumeist der Ausgrad gemeint.
In Abbildung A.2 ist der Graph mit der Kantenmenge $E = \{(1,3), (2,1), (2,3), (2,5), (3,1), (3,2), (3,5), (4,5)\}$ dargestellt.

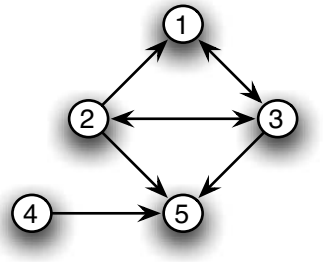


Abb. A.2. Ein gerichteter Graph.

Gerichtete Graphen sind besonders geeignete Zeigerstrukturen zu beschreiben, wie sie zum Beispiel im Internet bei Verbindungen mit UDP vorkommen. Auch Datenstrukturen mit Zeigern lassen sich dadurch darstellen.

- Der *gerichtete Multigraph* verfügt als Kantenmenge über eine Multimenge, in der Kanten mehrfach vorkommen können. Zusätzlich sind Selbstkanten, wie (u, u) (Schleifen) erlaubt. Abbildung A.3 zeigt den Graph mit der Kantenmenge $E = \{(1,3), (1,3), (1,3), (2,1), (2,1), (2,2), (2,2), (2,3), (2,3), (3,2), (3,4)\}$.

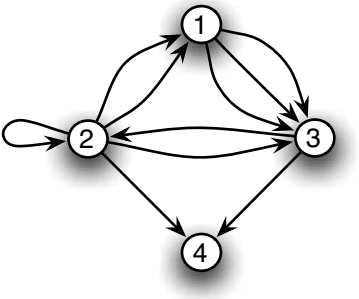


Abb. A.3. Ein gerichteter Multigraph.

Gerichtete Multigraphen entstehen zum Beispiel, wenn man mehrere Verbindungen zu Knoten (z.B. TCP-Verbindungen) unterhält. Die Selbstkanten spielen eine große Rolle bei Zustandsübergangsgraphen.

- *Gewichtete Graphen*: Jeder dieser Graphtypen kann auch als gewichteter Graph betrachtet werden. Hierzu wird jeder Kante e eine Zahl $w(e)$ zugewiesen. In Abbildung A.4 werden die Gewichte gemäß Tabelle A.1 verwendet.

Tabelle A.1. Die Gewichtung eines gerichteten Graphen.

e	(2,1)	(2,3)	(2,5)	(3,2)	(3,5)	(4,5)
$w(e)$	3	1	1	2	2	1

Zumeist beschreibt das Gewicht einer Kante den Abstand zweier Knoten. Es kann sich aber auch um eine Kapazität, das Nachrichtenaufkommen, u.v.a. handeln.

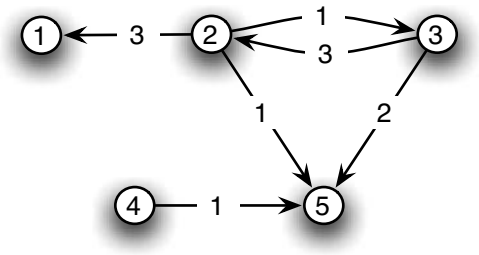


Abb. A.4. Ein gewichteter gerichteter Graph.

Pfade und Zusammenhang

Ein *ungerichteter Pfad* in einem ungerichteten Graphen ist eine Folge von Kanten, in dem jeder Kante eine Richtung so zugewiesen wird, dass der Endknoten und der Anfangsknoten zweier aufeinanderfolgender Kanten gleich sind. Ungerichtete Pfade kommen zum Beispiel in der Paketweiterleitung im Internet vor. Ein ungerichteter Graph ist *zusammenhängend*, wenn es zwischen allen Knoten einen ungerichteten Pfad gibt. Die *Länge eines Pfades* ist die Anzahl der Kanten. Die Länge des minimalen Pfades zweier Knoten ist der *Abstand* der Knoten. Das Gewicht eines Pfades ist das Gewicht der einzelnen Kanten. Der *Durchmesser* eines Graphen ist der maximale Abstand zweier Knoten.

Ein *gerichteter Pfad* in einem gerichteten Graphen oder einem Multigraphen ist eine Folge von Kanten, in der Endknoten einer Kante mit dem Startknoten der darauf folgenden Kante übereinstimmt. Ein gerichteter Graph oder ein Multigraph ist *stark zusammenhängend*, wenn es zwischen allen Knoten einen gerichteten Pfad gibt. Mit

Hilfe der gerichteten Pfade wird der *Abstand* und *Durchmesser* von gerichteten Graphen und Multigraphen analog wie im ungerichteten Graph definiert.

Ein *ungerichteter Pfad* in solchen Graphen ist ein gerichteter Pfad, in dem neben den Kanten des Graphen zusätzlich auch die entsprechenden (evtl. nicht im Graphen vorhandenen) Rückwärtskanten gewählt werden können. Ein Graph ist *schwach zusammenhängend*, wenn es zwischen allen Knoten einen ungerichteten Pfad gibt.

Metrik

Die Metrik wird durch gewichtete gerichtete oder ungerichtete Graphen mit oder ohne Selbstkanten beschrieben. Oft werden hierzu vollständige Graphen betrachtet, in denen Kante und Selbstkante vorhanden sind. Diese unterscheiden sich nur durch die Gewichtung. Folgende vier Eigenschaften müssen für die Gewichtung gelten, damit sie als Metrik bezeichnet werden kann. Zur Vereinfachung schreiben wir statt $w((u,v))$ nur $w(u,v)$:

1. Selbstkanten, soweit vorhanden, haben Gewicht 0: $w(u,u) = 0$ für alle Knoten $u \in V$
2. Für alle Kanten gilt $w(u,v) > 0$ für $u \neq v$
3. Symmetrie: $w(u,v) = w(v,u)$ für alle $u, v \in V$
4. Die Dreiecksungleichung: $w(u,v) \leq w(u,z) + w(z,v)$ für alle $u, v, w \in V$, siehe auch Abbildung A.5

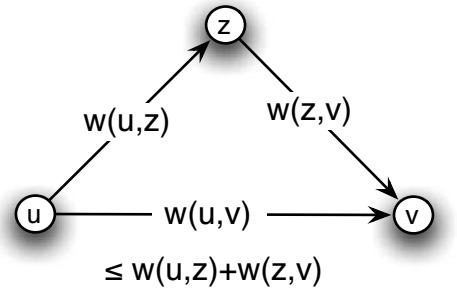


Abb. A.5. Die Dreiecksungleichung.

A.3 Wahrscheinlichkeitstheorie

Ein *Zufallsexperiment* beschreibt einen Vorgang, der ein nicht vorhersagbares Ergebnis zur Folge hat. Ein *Wahrscheinlichkeitsraum* besteht aus einer Ergebnismenge Ω, einer Ereignisalgebra (zum Beispiel der Potenzmenge der Ergebnismenge Ω) und einem Wahrscheinlichkeitsmaß Pr auf Σ. Für das Warscheinlichkeitsmaß müssen die folgenden Eigenschaften gelten:

- $Pr[\Omega] = 1$
- Für paarweise disjunkte Ereignisse $A_1, A_2 \ldots$ gilt:

$$Pr\left[\bigcup_i A_i\right] = \sum_i Pr\left[A_i\right]$$

Die Gesamtwahrscheinlichkeit aller Ereignisse ist also Eins. Die Wahrscheinlichkeit der Vereinigung von disjunkten Ereignissen entspricht der Summe der Einzelwahrscheinlichkeiten.

Eine *Zufallsvariable* ist eine Funktion $X : \Omega \to \Omega'$, wobei hier nur zwei Typen von Variablen betrachtet werden:

1. *Diskrete Zufallsvariablen*: $X : \Omega \to \Omega'$ mit endlicher oder abzählbarer Menge Ω', wie zum Beispiel $\Omega' = \mathbb{N}$
2. *Kontinuierliche Zufallsvariablen* $X : \Omega \to \mathbb{R}$. Hier kann man die Wahrscheinlichkeit durch eine Dichtefunktion $f : \mathbb{R} \to \mathbb{R}$ beschreiben, wobei

$$Pr[X < y] = \int_{-\infty}^{y} f(x)dx \; .$$

Hierbei ist $F_X(y) = Pr[X < y]$ die so genannte *kumulierte Wahrscheinlichkeitsfunktion* oder die *Wahrscheinlichkeitsverteilung* von X.

Der Erwartungswert einer kontinuierlichen Zufallsvariablen, gegeben durch die Dichtefunktion f, ist:

$$E[X] = \int_{-\infty}^{\infty} x \cdot f(x)dx \; .$$

Für diskrete Zufallsvariablen ist der *Erwartungswert* definiert durch

$$E[X] := \sum_{x \in X} x \cdot Pr[X = x] \; ,$$

wobei mit $x \in X$ die Menge aller diskreten Werte der Zufallsvariablen aufgezählt wird. Die *Varianz* ist definiert als

$$V[X] := E[(X - E[X])^2] \; .$$

Es gilt

$$V[X] = E[X^2] - E[X]^2 \; .$$

Markov, Tschebyscheff und Chernoff

Für die Abschätzung von Zufallsvariablen kennt man die folgenden Ungleichungen:

Theorem A.1 (Markov-Ungleichung).

$$Pr[X \geq c\ E[X]] \leq \frac{1}{c}\,.$$

Theorem A.2 (Tschebyscheff-Ungleichung). *Für alle $k > 0$ gilt*

$$Pr[|X - \mu| > k] \leq \frac{V(X)}{k^2}\,.$$

Sind die Variablen unabhängig, so lässt sich folgende schärfere Abschätzung verwenden. Ein Bernoulli-Experiment hat als Ergebnisraum $\{0, 1\}$. Zwei Zufallsexperimente sind unabhängig, wenn

$$Pr[A \cap B] = Pr[A] \cdot Pr[B]\,.$$

Entsprechend gilt für das Ergebnis mehrerer Zufallsvariablen X_i die Unabhängigkeit, wenn

$$Pr\left[\bigcap_i (X_i = y_i)\right] = \prod_i Pr\left[X_i = y_i\right]\,,$$

für alle Möglichkeiten von y_i.

Theorem A.3 (Chernoff-Schranke). *Seien $X_1, \ldots, X_n$ unabhängige Bernoulli-Experimente mit $Pr[X_i = 1] = p$ und $X = \sum_{i=1}^{n} X_i$. Dann gilt für $\delta \geq 0$, dass*

$$Pr[X \geq (1+\delta)pn] \leq e^{-\frac{1}{3}\min\{\delta,\delta^2\}pn}\,.$$

Des Weiteren gilt für $0 \leq \delta \leq 1$, dass

$$Pr[X \leq (1-\delta)pn] \leq e^{-\frac{1}{2}\delta^2 pn}\,.$$

B

Eingetragene Warenzeichen

Sun™, Sun Microsystems™, Java ™, JXTA™, und SunOS™sind Warenzeichen oder eingetragene Warenzeichen von Sun Microsystems, Inc. in den Vereinigten Staaten und in anderen Ländern.

Linux™ist ein eingetragenes Warenzeichen von Linus Torvalds.

Mac™- Operating System software - ist ein eingetragenes Warenzeichen von Apple Computer, Inc.

Microsoft™, Windows ™ sind Warenzeichen oder eingetragene Warenzeichen von Microsoft Corporation in den Vereinigten Staaten und in anderen Ländern.

Google und Google-Mail sind Warenzeichen oder eingetragene Warenzeichen von Google Inc. in den Vereinigten Staaten und in anderen Ländern.

CacheLogic und Streamsight sind eingetragene Warenzeichen von CacheLogic Limited.

BitTorrent ist ein Warenzeichen von BitTorrent Inc.

Napster™ ist ein eingetragenes Warenzeichen von Napster, LLC.

Der Name Kazaa und Kazaa verwandte Produkte sind Warenzeichen von Sharman Networks Limited.

Skype ™ ist ein Warenzeichen von Skype Technologies SA.

Andere genannte Produktmarken sind eingetragene Warenzeichen der jeweiligen Inhaber.

Literaturverzeichnis

1. Sullivan, J.: GNU General Public License. http://www.gnu.org/copyleft/gpl.html (2006)
2. CollabNet: JXTA — company spotlight archive. http://www.jxta.org/companies/companyarchive.html (2006)
3. Baset, S.A., Schulzrinne, H.: An analysis of the skype peer-to-peer internel telephony protocol. http://arxiv.org/abs/cs/0412017 (2004)
4. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. In: International Workshop on Design Issues in Anonymity and Unobservability. (2000) 311–320
5. Orwant, J.: What's on Freenet? http://www.openp2p.com/pub/a/p2p/2000/11/21/freenetcontent.html (2001)
6. Cachelogic: P2P in 2005. http://www.cachelogic.com (2005)
7. Clip2: The Gnutella protocol specification v0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf (2001)
8. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Computer Communication Review. Volume 31., Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley (2001) 161–172
9. Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., Panigrahy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, El Paso, Texas (1997) 654–663
10. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In Guerin, R., ed.: Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01). Volume 31, 4 of Computer Communication Review., New York, ACM Press (2001) 149–160
11. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, In Proc. of the International Conference on Distributed Systems Platforms (IFIP/ACM), **2218** (2001) 329–350
12. Hildrum, K., Kubiatowicz, J.D., Rao, S., Zhao, B.Y.: Distributed object location in a dynamic network. In: SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, ACM Press (2002) 41–52
13. Plaxton, C.G., Rajaraman, R., Richa, A.: Accessing nearby copies of replicated objects in a distributed environment. In: 9th Annual ACM Symposium on Parallel Algorithms

and Architectures (SPAA '97), New York, Association for Computing Machinery (1997) 311–320
14. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In Druschel, P., Kaashoek, M.F., Rowstron, A.I.T., eds.: Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers. Volume 2429 of Lecture Notes in Computer Science., Springer (2002) 53–65
15. Futey, D., Roos, L.: Peer-to-peer working group. http://p2p.internet2.edu/ (2002)
16. Lynch, D.C.: Historical Evolution. In Internet System Handbook,. Daniel C. Lynch and Marshall T. Rose, eds., Addison Wesley, Reading (1993)
17. Stevens, W.R.: TCP/IP Illustrated, Volume 1; The Protocols. Addison Wesley, Reading (1995)
18. Postel, J.: Internet Protocol. Network Information Center RFC 791 (1981) 1–45
19. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959)
20. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press/McGraw-Hill, Cambridge, Massachusetts (1990)
21. Malkin, G.: RFC 2453: RIP version 2. http://www.apps.ietf.org/rfc/rfc2453.html (1998)
22. R. Coltun, D. Ferguson, J.M.: OSPF for IPv6, RFC 2740. http://www.apps.ietf.org/rfc/rfc2740.html (1999)
23. Hedrick, C.L.: An introduction to IGRP. Cisco Systems, Inc, Document ID: 26825, The State University of New Jersey, Rutgers Center for Computers and Information Services, Laboratory for Computer Science Research (1991)
24. Cisco Systems, I.: Enhanced interior gateway routing protocol. White Paper, Document ID: 16406 (2005)
25. Y. Rekhter, T. Li, S.H.: RFC 4271: A Border Gateway Protocol 4 (BGP-4). http://www.apps.ietf.org/rfc/rfc4271.html (2006)
26. Postel, J.: Transmission Control Protocol. Network Information Center RFC 793 (1981) 1–85
27. Jacobson, V.: Congestion Avoidance and Control. In: ACM SIGCOMM '88. Volume 18, 4., ACM SIGCOMM, ACM Press (1988) 314–329 Stanford, CA.
28. Karn, P., Partridge, C.: Improving round-trip time estimates in reliable transport protocols. Computer Communication Review **17** (1987) 2–7
29. Braden, R.T.: RFC 1123: Requirements for Internet hosts — application and support. http://www.apps.ietf.org/rfc/rfc1123.html (1989)
30. Jacobson, V.: Modified TCP congestion avoidance algorithm. end2end-interest mailing list (1990)
31. Chiu, D.M., Jain, R.: Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Computer Networks and ISDN Systems **17** (1989) 1–14
32. Deering, S.E., Hinden, R.M.: Internet protocol, version 6 (IPv6) specification. Internet proposed standard RFC 1883 (1995)
33. Jovanovic, M.A., Annexstein, F.S., Berman, K.A.: Scalability issues in large peer-to-peer networks — a case study of Gnutella. Technical report, Technical Report, http://www.ececs.uc.edu/ mjovanov/Research/paper.html, University of Cincinnati (2001)
34. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making Gnutella-like P2P systems scalable. In Feldmann, A., Zitterbart, M., Crowcroft, J., Wetherall, D.,

eds.: Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 25-29, 2003, Karlsruhe, Germany, ACM (2003) 407–418
35. Cohen, E., Shenker, S.: Replication strategies in unstructured peer-to-peer networks. In: SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM Press (2002) 177–190
36. Rivest, R.L.: The MD5 Message Digest algorithm, RFC 1321 (1992)
37. Eastlake, D.E., Jones, P.E.: US secure hash algorithm 1 (SHA1). Internet informational RFC 3174 (2001)
38. Hansen, T.: US secure hash algorithms (sha and hmac-sha). Internet informational RFC 4634 (2006)
39. Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M.F., Morris, R.: Designing a DHT for low latency and high throughput. In: NSDI, USENIX (2004) 85–98
40. Gummadi, P.K., Gummadi, R., Gribble, S.D., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In Feldmann, A., Zitterbart, M., Crowcroft, J., Wetherall, D., eds.: Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 25-29, 2003, Karlsruhe, Germany, ACM (2003) 381–394
41. Mullender, S.J., Vitányi, P.M.B.: Distributed match-making. Algorithmica **3** (1988) 367–391
42. Guyton, J.D., Schwartz, M.F.: Locating nearby copies of replicated internet servers. In: SIGCOMM. (1995) 288–298
43. van Steen, M., Hauck, F.J., Tanenbaum, A.S.: A model for worldwide tracking of distributed objects. In: TINA '96, Conference, Heidelberg, Germany (1996) 203–212 http://www.cs.vu.nl/˜steen/globe/publications.html.
44. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A scalable and dynamic emulation of the butterfly. In: PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing, New York, NY, USA, ACM Press (2002) 183–192
45. Naor, M., Wieder, U.: Novel architectures for p2p applications: the continuous-discrete approach. In: SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, ACM Press (2003) 50–59
46. Kaashoek, M.F., Karger, D.R.: Koorde: A simple degree-optimal distributed hash table. In: 2nd International Workshop on Peer-to-Peer Systems, Berkeley, California (2003)
47. Aberer, K.: P-Grid: A self-organizing access structure for P2P information systems. In Batini, C., Giunchiglia, F., Giorgini, P., Mecella, M., eds.: Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings. Volume 2172 of Lecture Notes in Computer Science., Springer (2001) 179–194
48. Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-Grid: A self-organizing structured P2P system. SIGMOD Record **32** (2003) 29–33
49. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Sixth Symposium on Principles of Distributed Computing, Vancouver, Canada (1987) 1–12
50. Aspnes, J., Shah, G.: Skip graphs. In: 14th Annual ACM-SIAM Symposium on Discrete Algorithms. (2003) 384–393
51. Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: SkipNet: A scalable overlay network with practical locality properties. In: USENIX Symposium on Internet Technologies and Systems. (2003)

52. Aspnes, J., Wieder, U.: The expansion and mixing time of Skip graphs with applications. In Gibbons, P.B., Spirakis, P.G., eds.: SPAA 2005: Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms, July 18-20, 2005, Las Vegas, Nevada, USA, ACM (2005) 126–134
53. Harvey, N.J.A., Munro, J.I.: Deterministic SkipNet. Inf. Process. Lett. **90** (2004) 205–208
54. Jovanovic, M.A.: Modeling Large-scale Peer-to-Peer Networks and a Case Study of Gnutella. University of Cincinnati / OhioLINK (2001)
55. Ripeanu, M., Foster, I., Iamnitchi, A.: Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Computing Journal **6** (2002)
56. Aiello, W., Chung, F., Lu, L.: A random graph model for massive graphs. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, STOC'2000 (Portland, Oregon, May 21-23, 2000), New York, ACM Press (2000) 171–180
57. Pareto, V.: Cours d'economie politique. Rouge, Lausanne et Paris (1897)
58. Yule, G.: Statistical Study of Literary Vocabulary. Cambridge University Press (1944)
59. Zipf, G.: Human Behavior and the Principle of Least Effort. Addison Wesley (1949)
60. Pumain, D.: Scaling laws and urban systems. Technical report, Technical Report, http://www.santafe.edu/research/publications/workingpapers/04-02-002.pdf, University of Santa Fe (2003)
61. Pumain, D.: Settlement systems in the evolution. Geografiska Annaler **2** (2000) 73–87
62. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: SIGCOMM. (1999) 251–262
63. Barabasi, A.L., Albert, R.: Emergence of scaling in random networks. Science **286** (1999) 509–512
64. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the Web. Computer Networks (Amsterdam, Netherlands: 1999) **33** (2000) 309–320
65. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling the web for emerging cyber-communities. In: Proceedings of the Eighth International World-Wide Web Conference. (1999)
66. Huberman, B.A., Adamic, L.A.: Growth dynamics of the World-Wide Web. Nature **401** (1999) 131
67. Crovella, M.E., Taqqu, M.S., Bestavros, A.: Heavy-tailed probability distributions in the world wide web (1998)
68. Milgram, S.: The small world problem. Psychology Today (1967) 60–67
69. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. Nature **393** (1998) 440–442
70. Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: Proceedings of the 32nd ACM Symposium on Theory of Computing. (2000)
71. Wormald, N.C.: The asymptotic connectivity of labelled regular graphs. Journal of Combinatorial Theory, Series B **31** (1981) 156–167
72. McKay, B.D.: Subgraphs of random graphs with specified degrees. Congressus Numerantium **33** (1981) 213–223
73. Gkantsidis, C., Mihail, M., Saberi, A.: Random walks in peer-to-peer networks. In: INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies. Volume 1. (2004) 120–130
74. McKay, B.D., Wormald, N.C.: Uniform generation of random regular graphs of moderate degree. Journal of Algorithms **11** (1990) 52–67

75. Pandurangan, G., Raghavan, P., Upfal, E.: Building low-diameter P2P networks. In: FOCS '01: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Washington, DC, USA, IEEE Computer Society (2001) 492
76. Mahlmann, P., Schindelhauer, C.: Peer-to-peer networks based on random transformations of connected regular undirected graphs. In Gibbons, P.B., Spirakis, P.G., eds.: SPAA 2005: Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms, July 18-20, 2005, Las Vegas, Nevada, USA, ACM (2005) 155–164
77. Mahlmann, P., Schindelhauer, C.: Distributed random digraph transformations for peer-to-peer networks. In Gibbons, P.B., Vishkin, U., eds.: SPAA 2006: Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architectures, Cambridge, Massachusetts, USA, July 30 - August 2, 2006, ACM (2006) 308–317
78. Montresor, A., Jelasity, M., Babaoglu, O.: Robust aggregation protocols for large-scale overlay networks. In: Proceedings of The 2004 International Conference on Dependable Systems and Networks (DSN), Florence, Italy, IEEE Computer Society (2004) 19–28
79. Jelasity, M., Babaoglu, O.: T-Man: Fast gossip-based construction of large-scale overlay topologies. Technical Report UBLCS-2004-7, University of Bologna, Department of Computer Science, Bologna, Italy (2004) http://www.cs.unibo.it/techreports/2004/2004-07.pdf.
80. Montresor, A., Jelasity, M., Babaoglu, O.: Chord on demand (2005)
81. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM **21** (1978) 120–126
82. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22** (1976) 644–654
83. Garfinkel, S.: PGP: Pretty Good Privacy. O'Reilly & Associates, Inc., Sebastopol, CA, USA (1996)
84. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17** (1988) 281–308
85. Mikle, O.: Practical attacks on digital signatures using MD5 message digest. Report 2004/356, Cryptology ePrint Archive (2004)
86. Douceur, J.R.: The Sybil attack. In Druschel, P., Kaashoek, M.F., Rowstron, A.I.T., eds.: Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers. Volume 2429 of Lecture Notes in Computer Science., Springer (2002) 251–260
87. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems **4** (1982) 382–401
88. Fiat, A., Saia, J.: Censorship resistant peer-to-peer content addressable networks. In: Proceedings of Symposium on Discrete Algorithms. (2002)
89. Dingledine, R.R.: The Free Haven project: Design and deployment of an anonymous secure data haven. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science (2000)
90. Blakley, G.R.: Safeguarding cryptographic keys. In Merwin, R.E., Zanca, J.T., Smith, M., eds.: 1979 National Computer Conference: June 4–7, 1979, New York, New York. Volume 48 of AFIPS Conference proceedings., pub-AFIPS:adr, AFIPS Press (1979) 313–317
91. Shamir, A.: How to share a secret. Communications of the ACM **22** (1979)
92. Goldschlag, D., Reed, M., Syverson, P.: Onion routing. Commun. ACM **42** (1999) 39–41
93. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24** (1981) 84–90

94. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: USENIX Security Symposium, USENIX (2004) 303–320
95. Dingledine, R., Freedman, M.J., Molnar, D.: The Free Haven project: Distributed anonymous storage service. Lecture Notes in Computer Science **2009** (2001) 67–95
96. Clarke, I., Miller, S.G., Hong, T.W., Sandberg, O., Wiley, B.: Protecting free expression online with Freenet. IEEE Internet Computing **6** (2002) 40–49
97. Bennett, K., Grothoff, C.: gap - practical anonymous networking (2002)
98. Aaronson, S.: Guest column: Np-complete problems and physical reality. SIGACT News **36** (2005) 30–52
99. Deering, S., Estrin, D., Farinacci, D., Helmy, A., Thaler, D., , Handley, M., Jacobson, V., Liu, C., Sharma, P., Wei, L.: Protocol independent multicast-sparse mode (PIM-SM): Motivation and architecture. Work in progress (Internet-Draft draft-ietf-idmr-pim-arch-05.txt) (1998)
100. Crowcroft, J., Gemmell, J., Farinacci, D., Lin, S., Leshchiner, D., Luby, M., Fountain, D., Montgomery, T., Corporation, T., Rizzo, L., Tweedly, A., Bhaskar, N., Edmonstone, R., Sumanasekera, R., Vicisano, L.: RFC 3208: PGM reliable transport protocol specification (2001)
101. Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in communications (JSAC) **20** (2002) 1489–1499
102. Ratnasamy, S., Handley, M., Karp, R.M., Shenker, S.: Application-level multicast using content-addressable networks. In Crowcroft, J., Hofmann, M., eds.: Networked Group Communication, Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001, Proceedings. Volume 2233 of Lecture Notes in Computer Science., Springer (2001) 14–29
103. Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J.D.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Proceedings of NOSSDAV. (2001)
104. Jannotti, J., Gifford, D.K., Johnson, K.L., Kaashoek, M.F., O'Toole, Jr., J.W.: Overcast: Reliable multicasting with an overlay network. In USENIX, ed.: Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000), October 23–25, 2000, San Diego, California, USA, pub-USENIX:adr, USENIX (2000) 197–212
105. Chu, Y.H., Rao, S.G., Zhang, H.: A case for end system multicast. In: Measurement and Modeling of Computer Systems. (2000) 1–12
106. Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., Singh, A.: Splitstream: High-bandwidth multicast in cooperative environments (2003)
107. Cohen, B.: Incentives build robustness in BitTorrent (2003)
108. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics **8** (1960) 300–304
109. Weatherspoon, H., Kubiatowicz, J.: Erasure coding vs. replication: A quantitative comparison (2002)
110. Kubiatowicz, J., Bindel, D., Eaton, P., Chen, Y., Geels, D., Gummadi, R., Rhea, S., Weimer, W., Wells, C., Weatherspoon, H., Zhao, B.: OceanStore: An architecture for global-scale persistent storage. ACM SIGPLAN Notices **35** (2000) 190–201
111. Ahlswede, R., Cai, N., Li, S.Y.R., Yeung, R.W.: Network information flow. IEEE Transactions on Information Theory **46** (2000) 1204–1216
112. Gkantsidis, C., Rodriguez, P.: Network coding for large scale content distribution. In: INFOCOM, IEEE (2005) 2235–2245

113. Gnutella-2: Gnutella2 developer network. http://www.gnutella2.com/ (2005)
114. Mochalski, K.: Ipoque-studie. Presseerklärung von Ipoque, http://ipoque.com/dt/pressemitteilung_ipoque_231006.html (2006)
115. Kutzner, K., Cramer, C., Fuhrmann, T.: A self-organizing job scheduling algorithm for a distributed VDR. In Müller, P., Gotzhein, R., Schmitt, J.B., eds.: Kommunikation in Verteilten Systemen (KiVS), Kurzbeiträge und Workshop der 14. GI/ITG-Fachtagung Kommunikation in Verteilten Systemen (KiVS 2005) Kaiserslautern, 28. Februar - 3. März 2005. Volume 61 of LNI., GI (2005) 147–150
116. Waldman, M., Rubin, A.D., Cranor, L.F.: Publius: A robust, tamper-evident, censorship-resistant web publishing system (2000)
117. Mislove, A., Haeberlen, A., Post, A., Druschel, P.: ePOST. In Steinmetz, R., Wehrle, K., eds.: Peer-to-Peer Systems and Applications. Volume 3485 of Lecture Notes in Computer Science., Springer (2005) 171–192
118. Mislove, A., Post, A., Reis, C., Willmann, P., Druschel, P., Wallach, D.S., Bonnaire, X., Sens, P., michel Busca, J.: POST: A secure, resilient, cooperative messaging system (2003)
119. von Lohmann, F.: IAAL*: What peer-to-peer developers need to know about copyright law. http://www.eff.org/IP/P2P/p2p_copyright_wp.php (2006)

Abbildungsverzeichnis

Tabellenverzeichnis

Sachverzeichnis