Erkennung und Vermeidung von unkooperativem Verhalten in Peer-to-Peer-Datenstrukturen

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von: Diplom-Wirtschaftsinformatiker Erik Buchmann

geb. am: 30. Juli 1976 in Magdeburg

Gutachter:

Prof. Dr. Klemens Böhm
Prof. Dr. Gunter Saake
Dr. Manfred Hauswirth

Ort und Datum des Promotionskolloquiums: Magdeburg, den 19. Juni 2006

Erik Buchmann
Erkennung und Vermeidung von unkooperativem Verhalten in Peer-to-Peer-Datenstrukturen
Dissertation, Otto-von-Guericke Universität Magdeburg, 2006

Übersicht

Peer-to-Peer (P2P) - Datenstrukturen (auch bekannt als P2P-Overlays oder Strukturierte P2P-Netzwerke) sind in der Lage, riesige Bestände an (Schlüssel,Wert)-Paaren effizient zu verwalten und dabei viele parallele Abfragen zu unterstützen. Dies wird erreicht, indem Datenbestand und Anfragelast auf alle Teilnehmer des Systems verteilt werden.

Vorschläge für P2P-Datenstrukturen gehen davon aus, dass die Peers ihren Teil der Anfragelast stets protokollgerecht verarbeiten. Für rationale Peers besteht die ökonomisch dominante Verhaltensweise in diesen Systemen jedoch darin, Anfragen anderer Teilnehmer nicht zu verarbeiten, sich also unkooperativ zu verhalten. Existierende Vorschläge zum Umgang mit unkooperativen Teilnehmern skalieren zumeist schlechter als die P2P-Datenstruktur selbst, lassen sich angreifen oder umgehen, oder basieren auf Annahmen, die einen Einsatz in der Praxis nicht zulassen.

Diese Dissertation beschreibt FairNet, ein Protokoll, das unkooperatives Verhalten ökonomisch unattraktiv macht. FairNet beruht darauf, dass topologisch benachbarte Peers Beobachtungen über geleistete oder verweigerte Arbeit austauschen. Jeder Peer prüft dabei, wie verlässlich die Beobachtungen sind, die er von anderen erhält. Peers, über die zu wenige positive Beobachtungen vorliegen, müssen einen Arbeitsbeweis (Proof of Work) erbringen, bevor sie am P2P-Netz partizipieren können. Das Protokoll führt dabei zur Entstehung von logischen transitiven Netzen von kooperativen Peers, die unkooperative Knoten von der Anfrageverarbeitung ausschließen und den Arbeitsbeweis als Eintrittsbarriere verwenden.

Die Dissertation evaluiert das beschriebene Protokoll sowohl analytisch als auch experimentell in einer Implementierung eines Content-Addressable Networks. Es wird gezeigt, dass mit diesem Protokoll kooperatives Verhalten die ökonomisch überlegene Verhaltensweise ist, dass das Protokoll gut skaliert, und dass es robust gegen Angriffe ist. Des weiteren wird analysiert, welcher Zusatzaufwand den kooperativen Teilnehmern beim Einsatz dieses Protokolls entsteht. Eine Erweiterung des Protokolls widmet sich dem Umgang mit gefälschtem Feedback. Die Evaluierung zeigt, dass FairNet gut zwischen kooperativen und unkooperativen Teilnehmern differenziert und in der Praxis anwendbar ist.

Abstract

Peer-to-Peer (P2P) Data Structures (a.k.a. P2P Overlays or Structured P2P Networks) are able to administer huge sets of (key,value)-pairs and cope with many parallel requests. By following the P2P paradigm, P2P Data Structures distribute the data set and the workload among all participants.

In general, approaches for P2P data structures assume that all peers readily follow the protocol and carry out their share of the workload. But the economic dominant strategy for rational peers is to refuse to work off queries issued by other peers, i.e., to behave uncooperative. Most existing proposals designed to deal with uncooperative behavior suffer from scalability in comparison to the P2P Data Structure, are vulnerable or could be bypassed by uncooperative Peers, or base on assumptions that limit the applicability in the real world.

This thesis describes FairNet, a protocol that renders uncooperative behavior unattractive. FairNet is based on feedback that acknowledges performed or refused work. Feedback is exchanged between topologically adjacent peers. Each peer checks the reliability of feedback coming from other peers. Peers stated with little positive feedback are forced to perform a Proof of Work (ProW) before they are able to participate in query processing. Thus, the protocol creates transitive logical networks of peers that rule out uncooperative nodes and use the ProW as entrance fee.

The thesis evaluates the protocol by an algebraic analysis and by means of extensive experiments with an implementation of a Content-Addressable Network. The evaluation confirms that with FairNet cooperative behavior is the dominant strategy. It shows that the protocol scales well and is robust against attacks. In addition, the evaluation specifies the additional cost of the protocol for cooperative participants. An extension of the protocol deals with the challenges that arise from spoof feedback. The evaluation acknowledges that FairNet achieves a good discimination between cooperative and uncooperative participants and is applicable in the real world.

Inhaltsverzeichnis

Kapitel 1

Einführung

Peer-to-Peer (P2P) -Datenstrukturen sind selbstorganisierende, wartungsfreie Strukturen, die riesige Mengen an (Schlüssel,Wert)-Paaren so verwalten können, dass sie von einer sehr großen Zahl von Teilnehmern (den Peers) effizient parallel abgerufen werden können. Dies wird dadurch erreicht, dass die Daten mit dezentralen Algorithmen über alle Peers im System verteilt werden. Jeder Peer, der Anfragen an den Datenbestand stellt, bringt auch eigene Ressourcen (Speicher, CPU-Zeit, Netzwerkbandbreite) zur Anfrageverarbeitung in das System ein. Ein Teilnehmer an der P2P-Datenstruktur startet dazu ein P2P-Programm, das eigene Anfragen an die Peers weiterleitet, die sie beantworten können, und als Hintergrundprozeß die Anfragen von anderen Teilnehmern bearbeitet. Peer-to-Peer-Datenstrukturen haben das Potential, innerhalb der nächsten Jahre zu einer unverzichtbaren Internet-Technologie zu werden. Dies lässt sich an mehreren Entwicklungen erkennen:

- Die Zahl der Teilnehmer im Internet ist enorm. Google wird beispielsweise von 380 Mio. Benutzern pro Monat[1] verwendet. Derart populäre Web-Dienstleistungen erfordern große und im Unterhalt sehr teure Rechenzentren. Damit ist die Markteintrittsbarriere für neue Firmen, die den Massenmarkt adressieren, sehr hoch. Ein skalierendes System, bei dem jeder neue Benutzer eigene Ressourcen beisteuert und das so zentrale Rechenzentren überflüssig macht, erlaubt selbst Open-Source-Projekten ohne gesicherte Finanzierung den Markteintritt.

- Die meisten Dienste im Internet wurden nicht gemäß marktwirtschaftlicher Gesichtspunkte implementiert, sondern anhand der technischen Machbarkeit. Beispielsweise ist das Versenden von E-Mails kostenlos, für den Empfang müssen jedoch Server bereitgestellt werden. Dadurch wird es wirtschaftlich, Reklamesendungen (Spam) an Millionen von Adressen zu versenden, selbst wenn nur wenige Empfänger am Kauf der beworbenen Produkte interessiert sind. Eine andere marktwirtschaftliche Anomalie besteht darin, dass im WWW der Anbieter von Informationen für deren Bereitstellung bezahlen muss (Infrastrukturkosten, Server, Netzwerk-Kapazität). Für den Kunden als Nutznießer ist

[1] http://www.google.com/corporate/facts.html

der Abruf von Webseiten hingegen kostenlos. Das Resultat ist Bannerwerbung auf Webseiten, die von den meisten Teilnehmern als störend empfunden wird. Dienste, die auf der Basis von P2P-Datenstrukturen implementiert sind, verteilen die Arbeitslast (z.B. nach Fairness-Gesichtspunkten) auf alle Teilnehmer, und können derartige Anomalien vermeiden.

- Marktübliche PCs sind so leistungsfähig, dass sie einen Großteil ihrer Zeit untätig im Wartezustand verbringen. Die Verarbeitung von Anfragen in P2P-Datenstrukturen verursacht keinen messbaren Verschleiß, und Ressourcen wie z.B. Netzwerkbandbreite oder CPU-Zeit lassen sich auch nicht ansparen oder aufbrauchen. Mit P2P-Datenstrukturen gelingt es, diese brachliegenden Ressourcen ohne zusätzliche Kosten nutzbar machen, beispielsweise indem ein P2P-Programm als Hintergrundprozeß abläuft.

P2P-Datenstrukturen wie Content-Addressable Networks [RFH+01], P-Grid [Abe01], Chord [SMLN+01] oder Pastry [RD01] haben bereits einen hohen Entwicklungsstand erreicht, was technische Kriterien wie Anfrageverarbeitung, Selbstorganisation oder Ausfallsicherheit anbelangt. Diese Ansätze gehen davon aus, dass jeder Peer *freiwillig* eigene Ressourcen zur Verfügung stellt bzw. einen Teil der Anfragelast übernimmt. Untersuchungen in etablierten P2P-Systemen [AH00] haben jedoch gezeigt, dass diese Annahme nicht realistisch ist: die Teilnehmer versuchen, ihre Infrastrukturkosten zu minimieren. Wenn die Peers aber nicht bereit sind, *kooperativ* an der Anfrageverarbeitung mitzuwirken, sinkt sowohl die Verfügbarkeit des Systems und der Daten als auch die Motivation der anderen Teilnehmer, ihre eigenen Ressourcen einzubringen.

Die Dissertation stellt ein Protokoll namens *FairNet* zum Umgang mit unkooperativen Teilnehmern in Peer-to-Peer-Datenstrukturen vor. FairNet hat die Aufgabe, unkooperatives Verhalten in einer P2P-Datenstruktur unattraktiv zu machen. Peers, die mittels unkooperativem Verhalten versuchen, ihre Infrastrukturkosten zu senken, sollen dabei derart sanktioniert werden, dass die durch Strafmaßnahmen verursachten Kosten die Einsparungen bei weitem übersteigen. Der Umgang mit unkooperativen Teilnehmern in P2P-Datenstrukturen ist aus mehreren Gründen schwierig. So müssen die technischen Eigenschaften der P2P-Systeme erhalten bleiben. Ein Protokoll, das unkooperative Peers diszipliniert, darf daher nicht auf einer zentralen Komponente (Single Point Of Failure, zentrales Angriffsziel, Zensur-Instanz etc.) aufbauen, sondern muss mit den lokalen Beobachtungen und Interaktionen der Peers untereinander auskommen. Des weiteren passen sich unkooperative Peers an jede Gegenmaßnahme an. Das bedeutet, dass unkooperative Peers ihre Absichten verschleiern, also auf Anfragen nicht mit etwa einem 'Connection Refused' antworten. Sie können versuchen, einen Teil der Anfragen kooperativ zu beantworten, um nicht erkannt zu werden. Entdeckte unkooperative Teilnehmer können durch Aus- und Wiedereinsteigen in das P2P-System ihre Identität wechseln und ihr Verhalten mit einem 'frischen' Pseudonym an anderer Stelle fortsetzen. Zuletzt muss es für kooperative Peers rational sein, das Protokoll auch einzusetzen. So müssen die Kosten des Protokolls für kooperative Knoten unbedingt geringer sein als die Mehrkosten, die ihnen durch unkooperative Peers entstehen. Da Menschen im Allgemeinen risikoavers sind [KZ04], muss zudem die

Wahrscheinlichkeit vernachlässigbar gering sein, dass ein kooperativer Teilnehmer fälschlich mit Strafen belegt wird.

FairNet ist in allen P2P-Datenstrukturen einsetzbar, in denen die Peers zahlreiche kleine Anfragen über einen längeren Zeitraum untereinander austauschen und beantworten. Auf der anderen Seite ist FairNet nicht verwendbar, wenn Garantien über die Verfügbarkeit von Daten oder Diensten benötigt werden, wenn der Verlust einzelner Anfragen nicht akzeptabel ist, oder wenn Peers nur für einen kurzen Zeitraum im System verbleiben und zahlreiche angesammelte Anfragen auf einmal stellen können. Des weiteren ist FairNet nicht als Maßnahme gegen unerwünschtes Verhalten geeignet, das nicht der Einsparung von Infrastrukturkosten dient. Gegen Denial-of-Service-Attacken oder andere Angriffe müssen also andere Lösungen gefunden werden.

1.1 Die Beiträge dieser Arbeit

Im Rahmen dieser Dissertation wird ein Protokoll namens *FairNet* aufgestellt, das zwischen kooperativem und unzuverlässigem Verhalten unterscheidet. Als unzuverlässig bzw. unkooperativ wird jeder Teilnehmer angesehen, der nicht *alle* erhaltenen Anfragen zuverlässig bearbeitet. FairNet basiert darauf, dass die Peers positive und negative Rückmeldungen (*Feedback*) über beobachtete Leistungen generieren und untereinander austauschen. Jeder Peer entscheidet individuell, ob er dem Feedback von einem anderen Peer traut, und ob er auf Basis dieses Feedbacks einem Peer zutraut, eine Anfrage kooperativ zu verarbeiten. Um sicherzustellen, dass sich unkooperatives Verhalten nicht auszahlt, verarbeiten die Peers nur Anfragen von Teilnehmern, die sie selbst als kooperativ einschätzen. Teilnehmer mit wenig positivem Feedback (bzw. einer geringen Reputation) müssen mit einem Arbeitsbeweis [BOGKW88, JJ99, ABHL03] nachweisen, dass sie fähig und willens sind, sich kooperativ an der Anfrageevaluierung zu beteiligen. Der Arbeitsbeweis ist erheblich Ressourcen-intensiver als kooperative Mitarbeit, daher ist es für einen rationalen Teilnehmer ökonomisch, sich kooperativ zu verhalten.

Die Dissertation beschreibt ein Kostenmodell für kooperatives und unkooperatives Verhalten. Das Kostenmodell ermöglicht den Nachweis, dass FairNet gut zwischen kooperativen und unkooperativen Peers zu unterscheiden vermag: es ermöglicht, die Parameter von FairNet so zu optimieren, dass die Kosten der unkooperativen Teilnehmer maximal werden, während kooperative Knoten nur mit vorgegebenen Mehrkosten durch das Protokoll beaufschlagt werden. Des weiteren ermöglicht das Kostenmodell die Vorhersage der Kosten für Peers, die neu in das System eintreten, sowie die Bestimmung der Rentabilitätsgrenze für unkooperatives Verhalten bei gegebenen Umweltparametern.

Eine Serie von Experimenten zeigt, dass das vom Kostenmodell vorhergesagte Systemverhalten sowohl unter synthetischen als auch unter realistischen Szenarien eintritt. Dazu wurde ein Prototyp eines Content-Addressable Networks [RFH+01] (eine Variante einer P2P-Datenstruktur) in Java implementiert [BB04c] und unter verschiedenen Bedingungen getestet.

Die Experimente zeigen, dass FairNet auch unter extremen Änderungen in den Umgebungsbedingungen, z.B. wenn die Zahl der Peers im System von 80 auf 160.000 steigt oder die globale Ausfallwahrscheinlichkeit von 0 bis 50% variiert, noch gut zwischen kooperativen und unkooperativen Knoten differenziert.

Es wird eine Protokollerweiterung entwickelt, die es erlaubt, Feedback zwischen Peers *kostenneutral* auszutauschen. Dies wird erreicht, indem Feedback-Objekte an Nachrichten angehängt werden, die miteinander interagierende Peers ohnehin im Rahmen der Anfrageverarbeitung untereinander austauschen: Feedback-Objekte sind nur wenige Bytes groß, der Ressourcenverbrauch durch das Anhängen von Feedback an existierende Nachrichten ist daher vernachlässigbar. Die Erweiterung zum Feedback-Austausch ist somit eine Variante eines Gossip[2]-Protokolls, das sich die Topologie der P2P-Datenstruktur zunutze macht (vgl. [LK00, RHKS01]). Eine experimentelle Evaluierung weist nach, dass der Feedback-Austausch auch unter praxisgerechten Einsatzbedingungen funktioniert.

Die Dissertation legt insbesondere Wert darauf, die Anwendbarkeit des FairNet-Protokolls in der Praxis nachzuweisen. Deswegen wird das FairNet-Protokoll experimentell auch unter extremen Bedingungen (sehr viele unkooperative Peers, hohe Ausfallwahrscheinlichkeit etc.) evaluiert. Des weiteren werden unterschiedliche Varianten von FairNet untersucht, die auf Benachrichtigungen über Transaktionsausgänge verzichten oder verschiedene P2P-Datenstrukturen zugrunde legen. Außerdem werden mögliche Angriffe auf das Protokoll untersucht. Ein Kapitel widmet sich den Herausforderungen, die sich durch gefälschtes Feedback ergeben.

1.2 Populäre Irrtümer

Peer-to-Peer-Systeme haben insbesondere durch zahlreiche, sehr emotional geführten Diskussionen um Dateitauschbörsen, P2P-gestützte Reputationssysteme und freie Meinungsäußerung im Internet einen hohen Bekanntheitsgrad erlangt. Dies hat vor allem dazu geführt, dass sich viele stereotype Vorurteile etablieren konnten. Einige besonders hartnäckige Meinungen werden im folgenden exemplarisch diskutiert.

'Peer-to-Peer' ist gleich Filesharing. Dateitauschbörsen (engl. *Filesharing*, [AH02]) sind das Anwendungsgebiet des P2P-Paradigmas, das in der Öffentlichkeit die meiste Aufmerksamkeit erregt hat. Der Begriff 'Peer-to-Peer' bezieht sich jedoch nicht auf diese Anwendung, sondern auf eine Klasse von Systemen, die durch die Zusammenarbeit einer sehr großen Zahl von gleichberechtigten Teilnehmern charakterisiert ist. Diese Arbeit konzentriert sich auf P2P-Datenstrukturen. Diese unterscheiden sich von Filesharing-Systemen insbesondere dadurch, dass sie eine riesige Zahl von parallelen Operationen mit im Vergleich zu Filesharing winzigem Datenvolumen aufweisen, und Daten strukturiert verwalten. Einen guten Überblick über Anwendungen des P2P-Paradigmas bietet [MKL+02].

[2] to gossip: tratschen, Klatsch verbreiten.

Maßnahmen gegen unkooperatives Verhalten sind überhaupt nicht notwendig. Die Praxis hat gezeigt, dass insbesondere Filesharing-Systeme wie gnutella auch bei einem Anteil von 66% unkooperativen Teilnehmern ([AH00]) noch zufriedenstellend arbeiten. Tatsächlich sind unstrukturierte P2P-Systeme, die jede einzelne Anfrage per Flooding (siehe Abschnitt 2.1) an jeden einzelnen Teilnehmer weiterleiten, sehr robust gegenüber unkooperativem Verhalten.

Flooding stellt jedoch eine starke Einschränkung für die Skalierbarkeit dieser Systeme dar; der Datenverkehr durch viele unnötige Anfragen übersteigt in solchen Systemen zumeist die Netzwerklast, die durch Anfrageergebnisse verursacht wird. P2P-Datenstrukturen verwenden einen uni- oder bidirektionalen Graph als Kommunikationsstruktur, in dem Nachrichten zielgerichtet weitergeleitet werden. Zu jedem Zeitpunkt existiert nur eine Kopie einer Nachricht im System, die von Peer zu Peer weitergeleitet wird. Gibt ein unkooperativer Teilnehmer in der Kette der Weiterleiter die Nachricht nicht protokollgemäß an einen anderen Peer weiter, so kann sie der Initiator nur nach Ablauf einer Zeitspanne wiederholen. Diese Anfälligkeit gegenüber unkooperativem Verhalten ist der Preis, den strukturierte P2P-Netze für ihre ausgezeichnete Skalierbarkeit zahlen. Auf Maßnahmen gegen unkooperatives Verhalten kann daher in diesen Systemen nicht verzichtet werden.

Bestehende Reputationssysteme können in P2P-Datenstrukturen eingesetzt werden. Unkooperatives Verhalten gibt es in allen Systemen, in denen anonyme, unabhängige Teilnehmer in einer offenen Gemeinschaft miteinander interagieren. Daher gibt es auch bereits etliche Ansätze, diesem Problem mittels Vertrauen und Reputation zu begegnen. Bekannt sind insbesondere die Reputationssysteme von eBay[3] und Amazon[4], die jedoch als zentralisierte Systeme nicht auf reine P2P-Netze übertragbar sind. Eine Übertragbarkeit scheint jedoch für fertige Ansätze aus dem Bereich der (mobilen) Ad-Hoc Netzwerke oder Multi-Agenten Systeme gegeben. Eine nähere Untersuchung macht jedoch deutlich, dass die Eigenschaften von P2P-Datenstrukturen eine unmittelbare Übertragung von Ansätzen aus diesen Gebieten ausschließt.

In Ad-Hoc Netzwerken müssen die Knoten wegen der begrenzten Reichweite ihrer Funkübertrager Nachrichten ihrer Nachbarn weiterleiten. Wegen der radialen Ausbreitung der Funkwellen können die Knoten jedoch die Übertragungen ihrer Nachbarn überwachen. Ein klassischer Ansatz dazu ist [MGLB00]. Im Gegensatz dazu kann der Weg einer Nachricht in P2P-Datenstrukturen nicht vom Sender verfolgt werden, jedoch lassen sich die Weiterleiter unabhängig von ihrer geografischen Position wählen. In Multi-Agent-Systemen sind es Unterschiede im Anwendungsszenario, die eine direkte Übertragung von bestehenden Reputationslösungen verhindern. Klassische Beispielszenarien für Multi-Agent-Systeme sind unter anderem Buchungssysteme, in denen Kaufs- mit Verkaufsagenten Preise und Routen für Flüge aushandeln. In diesen Systemen gibt es eine klare Trennung zwischen Anbieter und Abnehmer, und zumindest die Anbieter verbleiben über einen langen Zeitraum im System. Der Handel mit geldwertigen Gütern erfordert zudem Garantien und einen hohen Aufwand bei der Ermittlung von vertrauenswürdigen Agenten.

[3] http://www.ebay.de
[4] http://www.amazon.de

Es existieren bereits zahlreiche einsatzfähige reputationsgestützte P2P-Protokolle Es ist relativ leicht, ein Protokoll zum Umgang mit unkooperativen Teilnehmern zu entwickeln, in dem Beobachtungen aus der Vergangenheit genutzt und zwischen den Peers ausgetauscht werden. Vergleichsweise leicht ist es auch, statistisch nachzuweisen, dass so ein Protokoll das vorab definierte unkooperative Verhalten unattraktiv macht. Es ist jedoch sehr schwer, den Nachweis dafür zu erbringen, dass das Protokoll in der Praxis robust ist gegenüber Fehlern und absichtlichen Manipulationen.

Benutzer sind erfinderisch. Es ist damit zu rechnen, dass sie ihr Verhalten an ein Reputationssystem anpassen. Teilnehmer können versuchen, gezielt einzelne Peers zu diskreditieren. Gruppen von Peers können sich zusammentun und gemeinsam gefälschte Bewertungen ins System einbringen. Sie können versuchen, die Schwellenwerte des Systems so auszureizen, dass sie grade noch nicht als unkooperativ erkannt werden, aber dabei nur ein Minimum an Ressourcen aufwenden müssen. Teilnehmer können auch ihre Identität jederzeit wechseln, um Negativbewertungen zu entgehen. Ebenso können Peers auch versuchen, mit Angriffen gegen einzelne Knoten Vorteile zu gewinnen. Wenn beispielsweise eine P2P-Datenstruktur zur Verwaltung von Reputationsdaten genutzt wird, kann ein unkooperativer Teilnehmer eine Denial-of-Service-Attacke gegen den Knoten fahren, der seine Negativbewertungen speichert. Diese Liste möglicher Reaktionen auf Gegenmaßnahmen lässt sich noch lange fortführen. Eine möglichst umfassende Untersuchung auf Schwachpunkte ist daher von großer Bedeutung. Daneben darf auch die Skalierbarkeit eines Reputationssystems nicht schlechter sein als die der darunterliegenden P2P-Struktur. Simulationen mit wenigen Peers in einer Statistik-Software sind zumeist nicht ausreichend, um diesen Sachverhalt überzeugend zu klären.

Micropayment-Systeme können Peers grundsätzlich am besten disziplinieren. Micropayments[5] sind ein Verfahren zur Bezahlung von Kleinstbeträgen. Im Internet werden Micropayment-Verfahren entweder vollständig über den zentralen Server eines Anbieters abgewickelt, oder als 'elektronische Münze' realisiert. Die Grundidee beim Einsatz von Micropayments in kollaborativen Systemen besteht nun darin, über ein unaufwendiges Verfahren jede einzelne Transaktion mit Beträgen im Sub-Cent Bereich zu vergüten. In einer P2P-Datenstruktur könnte das Stellen eine Anfrage beispielsweise einen 1/100 Cent kosten, und das Beantworten 1/100 Cent einbringen. Auf diese Weise lassen sich Kosten und Nutzen im System unmittelbar als Währung ausdrücken und volkswirtschaftliche Methoden zur Regulierung des Systems anwenden. Es existieren bereits zahlreiche Vorschläge für den Einsatz von Micropayments in P2P-Systemen, z.B. [GLBM01, YGM03, FST04].

Micropayment hat jedoch zahlreiche Nachteile. Zunächst handelt es sich bei elektronischem Geld um eine Schattenwährung, d.h. es bestehen viele Vorbehalte [Eur98], staatlich anerkannte Zahlungsmittel in Micropayment-Währungen zu überführen. Daneben lassen sich Micropayments nur durch die Verwendung einer vertrauenswürdigen Zertifizierungsinstanz sicher implementieren – zentrale Strukturen widersprechen aber dem P2P-Paradigma. In P2P-Datenstrukturen sind selbst Micropayment-Infrastrukturen noch zu ressourcenintensiv. Bei den

[5] http://de.wikipedia.org/wiki/Micropayment

in diesem Szenario üblichen zahlreichen kleinen Operationen würde die Abrechnung einen größeren Aufwand verursachen als die Anfrageevaluierung selbst. Zuletzt gibt es noch nicht-technische Argumente gegen Micropayments. Aktuelle P2P-Systeme leben zum Großteil davon, dass altruistische Benutzer freiwillig mehr Ressourcen einbringen, als sie selbst beanspruchen [AH00]. Ob derart motiviertes Verhalten in einem ökonomisch reglementierten System Bestand hat, ist hingegen fraglich. Des weiteren können Micropayment-Systeme zwar die technischen Kosten für eine Finanz-Transaktion reduzieren, jedoch nicht den Aufwand der Entscheidungsfindung [Odl03]. Der Benutzer muss auch für Transaktionen im Sub-Cent Bereich Entscheidungen fällen wie 'Ist mir diese Anfrage einen Cent wert?' oder 'Werde ich in Zukunft genug Geld haben, um eine wichtigere Anfrage zu stellen?'. Benutzer sind vielfach nicht bereit, diesen Aufwand für zahlreiche kleine Transaktionen zu tragen [Odl03].

1.3 Gliederung der Arbeit

Die Arbeit gliedert sich in 4 Teile. Kapitel 2 führt in die Problemstellung ein. Es beschreibt zunächst die Merkmale von P2P-Systemen und P2P-Datenstrukturen. Danach stellt das Kapitel einige Varianten von P2P-Datenstrukturen (Content-Addressable Networks, Chord, P-Grid) vor und zeigt auf, welche Herausforderungen dabei durch unkooperative Teilnehmer entstehen. Des weiteren wird herausgearbeitet, dass Reputationssysteme geeignet sind, um mit diesen Herausforderungen umzugehen, und welche Anforderungen an diese Systeme zu stellen sind. Das Kapitel 3 beschreibt FairNet, ein Protokoll zum Umgang mit unkooperativen Teilnehmern in P2P-Datenstrukturen. Zunächst stellt das Kapitel Datenstrukturen und Methoden von FairNet vor. Im zweiten Teil des Kapitels wird das FairNet-Protokoll algebraisch analysiert und sowohl numerisch als auch experimentell evaluiert. Der Evaluierung wird eine modifizierte Variante eines Content-Addressable Networks zugrunde gelegt. Bei der Untersuchung der Eigenschaften von FairNet wird insbesondere Wert darauf gelegt, sowohl künstliche Umgebungsbedingungen für den ungünstigsten anzunehmenden Fall zu evaluieren, als auch die Anwendbarkeit unter praxisgerechten Bedingungen nachzuweisen.

Ein wichtiger Bestandteil von FairNet ist eine Komponente zum Austausch von Beobachtungen über andere Teilnehmer zwischen interagierenden Peers. Kapitel 4 beschreibt und evaluiert ein Gossip-Protokoll, das diese Beobachtungen in Form von Feedback-Objekten ausgetauscht, die an ohnehin zwischen den Peers versendete Nachrichten angehängt werden. Weiterhin untersucht Kapitel 4 eine Reihe von Strategien, um die für die Übertragung geeignetsten Feedback-Objekte zu ermitteln. Eine Herausforderung für jedes Reputationssystem besteht darin, dass einzelne Teilnehmer oder ganze Gruppen gefälschtes Feedback über andere generieren können. Kapitel 5 untersucht den Einfluss von gefälschtem Feedback in FairNet, und geht auf Lösungsansätze zum Erkennen von falschem Feedback ein. Im Anschluss stellt es eine Erweiterung von FairNet vor, die mittels Wichtungsfaktoren potentiell gefälschtes Feedback von der Berechnung der Vertrauenskoeffizienten ausschließt und unehrliche Peers abstraft. Die Dissertation schließt mit einer Zusammenfassung und einem Ausblick auf zukünftige Forschungsvorhaben.

Kapitel 2

Einordnung der Thematik

Dieses Kapitel erläutert die Grundlagen, auf die das FairNet-Protokoll zum Umgang mit unkooperativen Teilnehmern in P2P-Datenstrukturen aufsetzt. Dazu werden zunächst die wesentlichen Merkmale von P2P-Systemen und P2P-Datenstrukturen erläutert und das hier zugrundegelegte Content-Addressable Network sowie einige weitere Beispielsysteme vorgestellt. Des weiteren wird herausgearbeitet, welche Auswirkungen die Präsenz von unkooperativen Teilnehmern in einer solchen Datenstruktur auf die Verfügbarkeit des Systems zeigt. Eine Untersuchung bereits existierender Gegenmaßnahmen wird zeigen, dass Reputationssysteme geeignet sind, um unkooperatives Verhalten zu erkennen. Aufbauend auf einer Diskussion bereits existierender Reputationssysteme werden schließlich die Anforderungen ermittelt, die an ein Reputationssystem in einer P2P-Datenstruktur zu stellen sind.

2.1 Peer-to-Peer-Systeme

Mit dem Term 'Peer-to-Peer' (P2P) [MKL+02] wird eine Klasse von dezentralen Systemen beschrieben, die sich dadurch auszeichnet, dass alle Teilnehmer *gleichberechtigt* zusammenarbeiten, um eine gemeinsame Aufgabe zu lösen. Abbildung 2.1 zeigt eine Schemaskizze, die die Architektur von Client-Server-Systemen dem Aufbau von P2P-Systemen gegenüberstellt. Die Pfeile symbolisieren dabei den Informationsfluss vom Dienstleister zum Dienstnutzer.

Peer-to-Peer-Systeme weichen in zahlreichen Eigenschaften von Client-Server-Systemen ab. Dazu gehören sowohl aus technischer oder wirtschaftlicher Sicht wünschenswerte Merkmale, als auch Eigenschaften, die die Realisierung von Anwendungen auf der Basis von P2P-Systemen erheblich herausfordernder gestalten als in klassischen Client-Server-Systemen. Die Eigenschaften von P2P-Systemen wurden bereits ausführlich in [MKL+02, AH02, ATS04] diskutiert. Tabelle 2.1 zeigt eine Zusammenfassung der Möglichkeiten und Herausforderungen von P2P-Systemen.

'Peer-to-Peer' (P2P) bezieht sich im weitesten Sinne auf ein sehr breites Spektrum von Systemen und Applikationen. Ein Beispiel für P2P auf der Ebene zwischenmenschlicher Zusammenarbeit ist das Peer-Review von Konferenzeinreichungen. Schränkt man den P2P-Begriff

Möglichkeiten	**Herausforderungen**
P2P-Systeme erlauben es, **Infrastrukturkosten** zwischen allen Teilnehmern des Systems **aufzuteilen**. Bei klassischen Client-Server-Systemen hingegen muss der Betreiber des Servers allein für dessen Betriebskosten aufkommen.	Die Peers in P2P-Systemen müssen **ohne globales Wissen** auskommen. Die einzige Möglichkeit, globale Informationen zu sammeln, bestünde darin, alle Teilnehmer einzeln abzufragen. Dies ist jedoch nicht effizient möglich.
P2P-Systeme ermöglichen es, die **Ressourcen** vieler Teilnehmer zu **aggregieren**. Moderne Rechner verfügen über viele ungenutzte Ressourcen; die Nutzbarmachung dieser Ressourcen spart u.U. den Zukauf weiterer Rechner.	P2P-Systeme funktionieren nur dann effektiv, wenn alle Teilnehmer **freiwillig kooperativ** zusammenarbeiten. Es gibt keine effiziente Möglichkeit, das Verhalten der Teilnehmer zu überwachen.
Daten oder Dienste werden in P2P-Systemen auf viele Teilnehmer aufgeteilt, deshalb weisen sie eine sehr hohe **Robustheit** auf. Durch klassische Angriffe (z.B. Denial-of-Service o.ä.) lassen sich P2P-Systeme kaum beeinträchtigen.	P2P-Systeme sind **nur für verteilbare Anwendungen** geeignet. Aufgaben, die sich nicht parallelisieren lassen, können nicht in P2P-Systeme implementiert werden.
Da P2P-Systeme ohne zentrale Komponenten auskommen, die einen Flaschenhals bei der Erweiterung darstellen, versprechen P2P-Systeme auch eine Steigerung der **Skalierbarkeit**.	Ohne zentrale Instanz sind viele **bewährte Dienste nicht realisierbar**, z.B. gibt es keine Authentifizierung der Benutzer und kein Trustcenter für die Public-Key Kryptographie.
Im Idealfall bringt jeder neue Teilnehmer so viele eigene Ressourcen in das P2P-System ein, wie er bei anderen konsumiert. Deswegen können P2P-Systeme sehr flexibel mit **dynamisch** wechselnden Teilnehmerzahlen umgehen.	Die im System verfügbaren Ressourcen werden von anonymen Teilnehmern selbst eingebracht. Daher können **keinerlei Garantien** über die Verfügbarkeit von Daten oder Diensten im System gegeben werden.
In P2P-Systemen verbleibt die Kontrolle über die eingestellten Daten bei den Peers, und geht nicht an den Betreiber eines zentralisierten Dienstes über. Damit können P2P-Systeme die **Autonomie** der Daten erhöhen.	Weil die Dienste und Daten im P2P-System keiner Kontrolle unterliegen, bestehen grundsätzliche **Sicherheitsbedenken**. So können Peers gefälschte Informationen anbieten, oder Daten können Viren und Trojaner enthalten.
Die Teilnahme an P2P-Systemen erfordert keine eindeutige Identifizierung, die sich ohne eine zentrale Instanz ohnehin nicht realisieren ließe. Daher können P2P-Systeme die **Anonymität** der Teilnehmer steigern.	Da es in P2P-Systemen keine sichere Authentifizierung der Nutzer gibt, ist die Hemmschwelle für **Fehlverhalten** sehr gering. Peers, die sich nicht im Sinne des Systems verhalten, lassen sich nicht aus dem System ausschließen.
Teilnehmer können sich **Ad Hoc** zu P2P-Systemen zusammenschließen, um eine gemeinsame Aufgabe zu bewältigen.	Einem **Neueinsteiger** muss ein vertrauenswürdiger Peer bekannt sein, der ihn in das System aufnimmt.

Tabelle 2.1: Eigenschaften von Peer-to-Peer-Systemen.

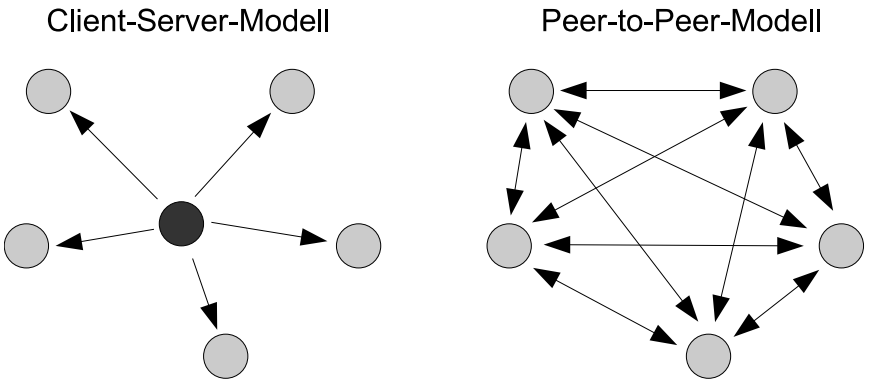

Abbildung 2.1: Client-Server- versus Peer-to-Peer-Paradigma.

auf Computernetzwerke ein, gibt es noch immer zahlreiche Auslegungen des P2P-Paradigmas auf unterschiedlichen technischen Ebenen [MKL+02] . Dazu zählen:

Kollaboration Systeme wie z.B. das Usenet, Wikis[1] oder Instant-Messaging basieren auf der Zusammenarbeit gleichberechtigter Teilnehmer. Es gibt keinen Koordinator, jeder Teilnehmer darf im System schreiben und lesen.

Anwendungen Hier finden sich viele Teilnehmer zusammen, um eine gemeinsame Aufgabe zu erfüllen. Zu den P2P-Anwendungen zählen SetiHome[2] oder FoldingHome[3], welche die Arbeitslast je nach den verfügbaren Ressourcen auf viele Peers verteilen.

Datenverwaltung Beispiele für P2P-Systeme zur Datenverwaltung sind die bekannten Dateitauschbörsen wie gnutella[4] oder Kazaa[5]. Hier hat jeder Benutzer gleichberechtigt die Möglichkeit, eigene Dateien bereitzustellen oder selbst Dateien von anderen Benutzern herunterzuladen.

Metadaten-Verwaltung Die Metadaten-Verwaltung unterscheidet sich von der Datenverwaltung dadurch, dass hier nicht die Daten selbst, sondern Verweise auf die Daten verwaltet werden. Beispiele dafür sind P2P-Datenstrukturen wie Content-Addressable Networks [RFH+01] oder P-Grid [Abe01]. Klassische Anwendungen dafür sind Verzeichnisdienste.

Netzwerke Aus der Netzwerksicht betrachtet ist auch das Internet als P2P-System organisiert: gleichberechtigte Rechner sind zu einem Netz zusammengeschlossen. An einer IP-Adresse lässt sich nicht erkennen, ob es sich dabei um einen Router, einen Web-Server oder eine einfache Workstation handelt.

[1] http://wikipedia.org
[2] http://setiathome.ssl.berkeley.edu (Suche nach extraterrestrischem Leben)
[3] http://folding.stanford.edu (Bestimmung von Protein-Faltungen)
[4] http://www.gnutella.com
[5] http://www.kazaa.com

In Systemen zur Daten- oder Metadaten-Verwaltung verfügt jeder Peer über seinen eigenen, lokalen Datenbestand. Daneben kennt jeder Peer noch einige *Kontakte* – das sind andere Peers im System, mit denen er interagieren kann. Jedoch ist kein Peer in der Lage, sich eine globale, vollständige Übersicht darüber zu verschaffen, welche Peers im System aktiv sind und welcher Peer über welche Daten verfügt. Die Herausforderung besteht nun darin, Anfragen oder andere Operationen *effizient* an die Peers weiterzuleiten, die über die passenden Daten in ihrem lokalen Datenbestand verfügen, um diese Anfragen beantworten zu können. Für dieses Problem gibt es zwei Lösungsansätze:

Flooded Requests Model Der Anfragesteller leitet eine Anfrage an alle seine Kontakte weiter. Ein Peer, der so eine Anfrage erhält, prüft zunächst, ob er sie aus seinem lokalen Datenbestand beantworten kann. Ist das nicht der Fall, sendet er die Anfrage ebenfalls an alle seine Kontakte weiter; ausgenommen werden die Peers, die ihm bereits die gleiche Anfrage zugesendet haben. Das linke Bild in Abbildung 2.2 zeigt schematisch, welchen Weg die Nachrichten in einem P2P-System vom Anfragesteller F zum Beantworter B nehmen. P2P-Dateitauschbörsen verwenden häufig das Flooded Requests Model.

Document Routing Model Einen anderen Ansatz verfolgen Systeme, die das Document Routing Model einsetzen: hier werden alle Daten in einem gemeinsamen Schlüsselraum abgebildet, der nach vorgegeben Regeln zwischen den Peers aufgeteilt wird. Um nun eine Anfrage an den Peer zu senden, der sie beantworten kann, muss sie jeder Peer nur jeweils an den Kontakt weiterleiten, der in diesem Schlüsselraum 'näher' am gesuchten Anfrageergebnis ist. Eine ebenfalls vom System vorgegebene Distanzmetrik bestimmt dabei den Abstand zwischen dem Schlüsselraum eines Peers und dem Schlüssel einer Suchanfrage. Abbildung 2.2 (rechts) zeigt, wie eine Anfrage beim Document Routing Model weitergeleitet wird. Alle P2P-Datenstrukturen arbeiten nach dem *Document-Routing Modell*.

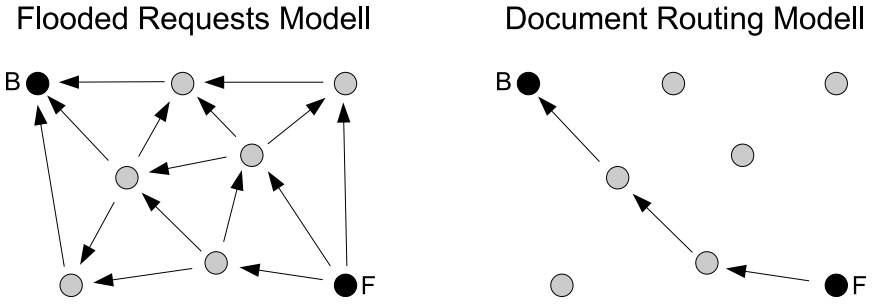

Abbildung 2.2: Flooding versus Document Routing.

Systeme, die auf dem Document Routing Modell aufsetzen, arbeiten erheblich ressourcenschonender als solche, die für jede Anfrage das gesamte System fluten müssen [Cli00, ZYF02]. Sie haben jedoch im Vergleich zum Flooded Requests Model die Nachteile, dass (1) das System die Datenverteilung bestimmt, d.h., die Peers bekommen die zu verwaltenden Inhalte von

anderen zugewiesen. Weiterhin (2) werden Anfragen nun ohne Replikation über eine Kette von Peers weitergeleitet, von denen jeder einzelne die Anfrage korrekt weiterleiten muss. Außerdem (3) unterstützt das Document Routing Modell nur die Operationen, die sich durch Schlüssel im Schlüsselraum adressieren lassen.

2.2 Grundlagen von Peer-to-Peer-Datenstrukturen

P2P-Datenstrukturen sind eine Variante von P2P-Systemen, die auf dem Document Routing Model aufsetzen und zur effizienten Speicherung von *(Schlüssel,Wert)*-Paaren entwickelt wurden. Alle Peers werden gleich behandelt, d.h. jeder Peer darf Anfragen nach allen Daten in der Struktur absenden. Diese Daten werden über eine Programmierschnittstelle verwaltet, die einer Hashtabelle [SS01] entspricht. Es werden Operationen wie `put(Schlüssel,Wert)` oder `Wert = get(Schlüssel)` angeboten. Daher werden P2P-Datenstrukturen auch häufig als *Verteilte Hashtabellen* (engl. *Distributed Hash Tables*, *DHT*) bezeichnet, selbst wenn sie intern auf einer Baumrepräsentation aufbauen [Abe01]. Ebenso wie in klassischen Hashtabellen unterstützen auch DHTs ausschließlich Operationen, die sich über einen Schlüssel adressieren lassen. Das bedeutet, dass beispielsweise Aggregatfunktionen über die gespeicherten Werte nur dann effizient ausgeführt werden können, wenn dem Anfrager alle Schlüssel vorab bekannt sind.

Zur Verbreitung von Informationen nutzen P2P-Datenstrukturen physikalische Netzwerke wie z.B. das Internet: das Document Routing Model macht nur Aussagen dazu, welcher Peer für welche Daten zuständig ist, nicht aber, über welche physikalischen Leitungen eine Nachricht transportiert werden soll. Daher werden P2P-Datenstrukturen ebenfalls mit dem Begriff *P2P-Overlay* benannt. Eine Referenzarchitektur für P2P-Overlay-Netzwerke beschreibt [AAG+05].

Unterscheidungsmerkmale von P2P-Datenstrukturen In den letzten Jahren wurden zahlreiche verschiedene Varianten von P2P-Datenstrukturen vorgestellt, die über unterschiedliche Eigenschaften verfügen. Diese Datenstrukturen lassen sich grob anhand der folgenden Merkmale klassifizieren:

Topologie des Schlüsselraums In Hashtabellen bestimmt eine *Hashfunktion* darüber, wie die verwalteten Daten auf die interne Datenstruktur, den *Schlüsselraum*, abgebildet werden. Bei klassischen Hashtabellen wird der Schlüsselraum dazu in Behälter (*Buckets*) unterteilt, die im Speicher eines Rechners vorliegen. Dagegen werden in P2P-Datenstrukturen die Buckets den einzelnen Peers zugewiesen. Die Topologie des Schlüsselraums bestimmt also zusammen mit der Hashfunktion, welche Daten ein Peer verwaltet. Des weiteren beeinflusst die Organisation des Schlüsselraums die Kontakt- und Pfadwahl. In strukturierten P2P-Systemen sind Schlüsselräume in Form eines Baumes, eines Ringes, eines mehrdimensionalen Torus bzw. Hyperkubus oder einer Graphen-Struktur üblich.

Kontaktwahl Da jeder Peer nur einen kleinen Ausschnitt des Schlüsselraums verwaltet, müssen Peers untereinander kommunizieren, um Operationen auf dem gesamten Datenbestand ausführen zu können. Die Kontaktwahl legt fest, welche Knoten ein Peer kennt,

also mit welchen Peers er kommunizieren kann. Häufig werden die Kontakte eines Peers anhand von Nachbarschaftsbeziehungen im Schlüsselraum bestimmt. Das bedeutet, dass Peers, die für angrenzende Schlüsselräume verantwortlich sind, über IP-Adresse und verwalteten Datenbereich ihres Nachbarn informiert sind. Andere Datenstrukturen bestimmen die Kontakte eines Peers über Entfernungen im Schlüsselraum oder über gemeinsame Präfixe der Peer-IDs. Zusätzlich erlauben einige Systeme eine freie Auswahl für Kontakte, die für für eine bessere Gesamtperformanz nützlich sind.

Pfadauswahl Die Pfadauswahl bestimmt, an welchen seiner Kontakte ein Peer eine Nachricht weitergibt, die nicht für den von ihm verwalteten Schlüsselbereich bestimmt ist. Verteilte Hashtabellen verwenden *Greedy Forwarding*, bei dem sich der Abstand einer Nachricht zum Zielpeer mit jedem Weiterleitungsvorgang verringern muss. Dieser Abstand und somit auch die Pfadwahl wird bestimmt von einer *Distanzmetrik*. Beispiele für solche Distanzmetriken sind der Euklidische Abstand zum Ziel innerhalb des Schlüsselraums oder die Bit-Länge vom gemeinsamen Präfix von Nachrichten-ID und Peer-ID.

Einen ausführlichen Vergleich unterschiedlicher P2P-Datenstrukturen hinsichtlich dieser Merkmale bietet [GGG+03]. Weitere Unterscheidungsmerkmale betreffen die Algorithmen zum Ein- bzw. Austreten von Peers aus der Datenstruktur sowie zur Reparatur von Inkonsistenzen in der Datenstruktur. Solche Inkonsistenzen können entstehen, wenn Peers das Netz verlassen, ohne sich korrekt abzumelden und ihren Datenbereich an einen anderen Peer zu übergeben.

Die im Rahmen dieser Dissertation wichtigsten Effizienzmerkmale von P2P-Datenstrukturen sind die *Pfadlänge* und die *Skalierbarkeit*. Die Pfadlänge gibt an, wie viele Peers protokollgerecht zusammenarbeiten müssen, damit eine Operation erfolgreich durchgeführt wird. Wie oft muss also eine Anfrage in einer P2P-Datenstruktur von einem Peer zum nächsten weitergereicht werden, bis sie den Peer mit dem passenden Schlüsselraum erreicht, der sie beantworten kann? Die minimale Pfadlänge beträgt stets 0, da ein Peer eine Anfrage möglicherweise aus seinem lokalen Datenbestand beantworten kann. Wichtig sind insbesondere die *mittlere Pfadlänge*, die ein Maß für die durchschnittliche Performanz des Systems sind, sowie Garantien zur *maximalen Pfadlänge*.

P2P-Datenstrukturen werden dafür entwickelt, Daten über sehr viele Peers zu verteilen. Dabei sollen die Daten aber effizient abrufbar sein. Bei P2P-Datenstrukturen wird mit Skalierbarkeit im Allgemeinen die Erhöhung der Pfadlänge bei steigender Teilnehmerzahl bezeichnet. Bei den meisten Strukturen wächst die Pfadlänge logarithmisch mit steigender Anzahl von Peers. Ein anderer Aspekt der Skalierbarkeit ist die Zahl der Kontakte, die ein Peer verwalten muss. Eine große Zahl von Kontakten erhöht den Verwaltungsaufwand – in vielen Systemen übermitteln Peers regelmäßig ihren Status an ihre Kontakte – sowie den Aufwand für das Betreten oder Verlassen des Netzes, da hierbei alle Kontakte informiert werden müssen. Optimal sind Systeme, bei denen die Zahl der Kontakte eines Peers nicht von der Gesamtzahl der Peers im Netz abhängt. In dieser Arbeit wird bei Verwendung des Begriffs Skalierbarkeit immer auf die Pfadlänge Bezug genommen.

Anwendungen für P2P-Datenstrukturen Grundsätzlich können P2P-Datenstrukturen mit genügend Teilnehmern eine sehr große Zahl von Datensätzen verwalten und viele Operationen parallel ausführen. Doch drei Merkmale schränken die Anwendungen ein, für die P2P-Datenstrukturen geeignet sind: (1) die Beschränkung auf einfache Anfragen der Art `Wert = get(Schlüssel)`, (2) die von der Datenstruktur vorgegebene Zuordnung der Daten zu Peers, und (3) der unkontrollierbare Zugang zur Datenstruktur. Das bedeutet, dass Anwendungen, die auf einer Hashtabelle aufsetzen, zumeist auch mit einer P2P-Datenstruktur realisiert werden können. Aus der vorgegebenen Verteilung der Daten über die Peers folgt, dass P2P-Datenstrukturen nur für Anwendungen mit sehr zahlreichen, kleinen Datensätzen geeignet sind, in denen die Peers über einen längeren Zeitraum im System verbleiben. Anderenfalls würde der Aufwand, um die verwalteten Daten beim Ein- oder Austritt von Teilnehmern auf andere Peers zu übertragen, die Verwendung einer P2P-Datenstruktur ökonomisch unattraktiv machen. Zuletzt folgt aus dem unkontrollierbaren Zugang zur Datenstruktur, dass keine Garantien über die Verfügbarkeit und Integrität des Systems und der darin gespeicherten Daten gegeben werden können: für sicherheitskritische Anwendungen oder sensible Daten sind P2P-Datenstrukturen nicht geeignet.

Eine Anwendung für P2P-Datenstrukturen sind *verteilte Dateisysteme*. Beispiele dafür sind Past, CFS oder OceanStore. *Past* [DR01] setzt auf die P2P-Datenstruktur Pastry [RD01] auf. Jede Datei wird mit einem 160 Bit langen Schlüssel spezifiziert, der sich aus dem Hash-Wert von Dateiname und einem kryptographischen Schlüssel des Besitzers zusammensetzt. Mittels Replikation wird die Wahrscheinlichkeit für einen Datenverlust verringert. Im Unterschied zu Past arbeitet *CFS* [DKK+01] nicht auf Datei- sondern auf Block-Ebene. Die in Blöcke gleicher Größe zerlegten Dateien werden dabei in der P2P-Datenstruktur Chord [SMLN+01] gespeichert. CFS lässt sich als Dateisystem in einige Unix-Derivate einbinden. *OceanStore* [RWE+01] arbeitet ebenfalls auf Block-Ebene, bietet jedoch durch die Integration von 'm-aus-n'-Datenkodierung, introspektivem Replikat-Management und von 'Byzantine Update Commitment'-Protokollen eine erweiterte Datensicherheit.

Weitere Anwendungen sind *Multicast-Systeme*, wie sie zum Beispiel für Event-Notification-Dienste [RKCD01, CJT01] oder für das Senden von großvolumigen Audio- oder Video-Datenströmen [RHKS01, ZZJ+01, CDK+03] benötigt werden. Das Ziel von P2P-Multicast besteht darin, Nachrichten an Gruppen von Peers auszuliefern, und dabei den Aufwand für die Auslieferung möglichst gleichmäßig auf alle Peers zu verteilen. In *Scribe* [RKCD01] werden die Empfängergruppen über eine numerische ID in Pastry abgelegt. Ein Peer, der so einer Gruppe beitreten möchte, meldet sich bei dem Peer an, der diese ID in Pastry speichert. Nachrichten werden nun unter Ausnutzung der Topologie der Datenstruktur an alle Gruppenmitglieder gesendet. Weitere Multicast-Systeme werden in [Hue03] evaluiert.

Ein Anwendungsgebiet für P2P-Datenstrukturen sind verteilte Suchmaschinen. In [RV03] wird ein Ansatz für die verteilte Suche nach Schlüsselworten in Dokumenten beschrieben, der auf der Basis von Bloom-Filtern [Blo70] – die sich effizient in einer Hashtabelle verarbeiten lassen – Dokumente identifiziert, die vorab in der P2P-Datenstruktur abgelegte Schlüsselwortmengen enthalten. Einen darüber hinausgehenden Ansatz zur Volltextsuche bietet *eSearch* [TD04]. Neben Textsuchmaschinen existieren auch Suchmaschinen für Spezialanwen-

dungen, z.B. für Forschungsarbeiten (*OverCite* [SCL+05]) oder für WWW-Dokumente (*Minerva* [BMT+05], erweitert in [BMT+06]). Eine besondere Herausforderung ist dabei die Realisierung von Anfrageprozessoren für strukturierte Dokumente [SRBB04, RWSB05], da diese nicht direkt auf Hashtabellen abgebildet werden können. Eine Aufstellung weiterer Beispielanwendungen bietet [GWB+01, ATS04].

2.3 Varianten von Peer-to-Peer-Datenstrukturen

Das Content-Addressable Network (CAN) [RFH+01] ist die P2P-Datenstruktur, auf die das in dieser Dissertation entwickelte FairNet-Protokoll aufsetzt. Ein CAN benutzt einen d-dimensionalen kartesischen Schlüsselraum $[0;1]^d$, der als Hyper-Torus organisiert ist. Demzufolge ist ein Schlüssel k eine kartesische Koordinate[6] in diesem Schlüsselraum $k = (k_1, k_2, \cdots, k_d)$, $k \in [0;1]^d$.

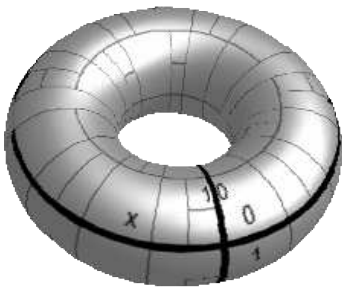

Abbildung 2.3: Beispiel für ein CAN mit zweidimensionalem Schlüsselraum.

Jeder Peer ist für ein bestimmtes Hyper-Rechteck im Schlüsselraum verantwortlich, seine *Zone*. Daneben kennt ein Peer alle Knoten, deren Zone im Schlüsselraum an seine angrenzt, d.h., er speichert Informationen über die Zonen sowie über die Netzwerkadressen seiner Nachbarn in seiner *Kontaktliste*. Aus der Torus-Eigenschaft folgt, dass beispielsweise auch die Peers mit den Zonen (*,1) und (0,*) in einer Dimension benachbart sind. Abbildung 2.3 skizziert die Torus-Eigenschaft für ein CAN mit zweidimensionalem Schlüsselraum.

Beispiel: Der Schlüsselraum des CAN in Abbildung 2.4 (zur Vereinfachung als Ebene dargestellt) ist zweidimensional. Der Knoten P_1 ist verantwortlich für die Zone ([0.5; 0.5), [0.625; 0.75)) im Schlüsselraum. Das bedeutet, er verwaltet alle (Schlüssel,Wert)-Paare mit Schlüsseln im Intervall ([0.5; 0.5), [0.625; 0.75)). Des weiteren speichert der Knoten P_1 in seiner Kontaktliste Informationen über die Zonen und Netzwerkadressen der Peers P_2, P_7, P_4, P_5, P_3 and P_6. Durch die Torus-Eigenschaft ist der Knoten P_2 auch ein Nachbar von P_8.

[6] Die Dimensionalität des Schlüsselraums ist von der verwendeten Hashfunktion abhängig und ist ein Optimierungsparameter des CAN. Sie ist vollständig unabhängig von den Daten, die im CAN gespeichert werden.

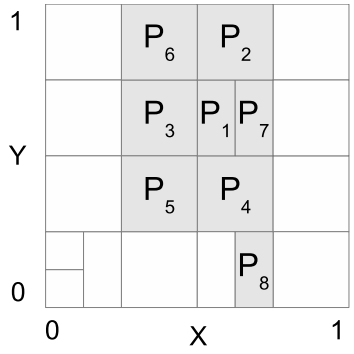

Abbildung 2.4: Nachbarschaft im CAN.

Der Algorithmus, mit dem neue Peers in das CAN aufgenommen werden, bestimmt die Zonenaufteilung: Ein neuer Teilnehmer bestimmt zufällig einen Knoten, der bereits im CAN eingetragen ist. Dieser Knoten übergibt dann eine Hälfte seiner Zone, die in dieser Hälfte enthaltenen (Schlüssel,Wert)-Paare sowie eine Kontaktliste mit den an diese Zone angrenzenden Peers an den Neuankömmling. Des weiteren informiert er alle seine Nachbarn über das veränderte Zonenlayout und die Kontaktdaten des neuen Peers. Da dieser Algorithmus die zu halbierenden Datenbereiche stochastisch bestimmt, haben Peers häufig Zonen mit unterschiedlicher Größe, die Wahrscheinlichkeit für sehr kleine oder sehr große Zonen ist jedoch gering (vgl. [RFH+01]). Abbildung 2.4 zeigt ein auf diese Weise entstandenes Zonenlayout.

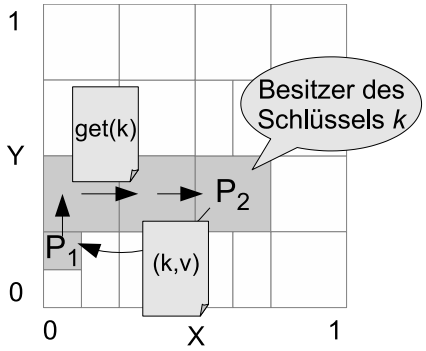

Abbildung 2.5: Nachrichtenweiterleitung im CAN.

Die Anfrageverarbeitung im CAN ist eine Variante des *Greedy Forwarding*. Anfragen werden durch *Punkte* im Schlüsselraum adressiert. Erhält ein Peer eine Anfrage, dessen Schlüssel nicht in seine Zone gehört, so leitet er sie an einen geeigneten Nachbarn weiter. Dazu bestimmt der Peer den euklidischen Abstand zwischen dem Schlüssel der Anfrage und der Zone aller seiner Nachbarn. Ein Nachbar, dessen Zone den geringsten Abstand zum Schlüsselpunkt aufweist, erhält die Anfrage. Auf diese Weise verringert die Nachricht mit jeder Weiterleitung den Abstand zu dem Peer, der sie beantworten kann. Die Weiterleitung ist ein Greedy-Algorithmus, d.h., es kann nicht garantiert werden, dass Nachrichten auf dem kürzesten Pfad vom Sender

zum Empfänger gelangen. Abbildung 2.5 zeigt schematisch, wie die Anfrageverarbeitung im CAN abläuft. Eine Anfrage nach einem Schlüssel k wird von Peer P_1 ausgehend so lange in Richtung des Zielpunkts k weitergeleitet, bis sie den Peer P_2 erreicht, der für die Zone verantwortlich ist, in die k passt. Der Peer P_2 beantwortet die Anfrage nun mit dem (Schlüssel,Wert)-Paar (k, v) und sendet seine Antwort direkt an den Anfragesteller[7].

Zwischen dem ursprünglichen CAN-Entwurf [RFH+01] und der in dieser Arbeit verwendeten Variante besteht ein Unterschied in der Bestimmung der Nachbarschaft. Im *klassischen CAN* gilt nur der Peer als Nachbar eines Knotens, dessen Zone in einer Dimension an den Knoten angrenzt, und in allen anderen Dimensionen identisch ist. Daher hat im klassischen CAN mit d Dimensionen jeder Peer mindestens $2 \cdot d$ Nachbarn[8]. Daraus folgt, dass bei der Nachrichtenübermittlung die Manhattan-Distanz (L_1-Norm) zum Einsatz kommt, und jede Nachricht im Mittel über $d/4 \cdot n^{1/d}$ Peers weitergeleitet wird. Im Unterschied dazu betrachtet das hier verwendete CAN jeden Peer als Nachbarn eines anderen, wenn dessen Zone in einer oder mehr Dimensionen an den anderen angrenzt. Jeder Knoten hat also mindestens $3^d - 1$ Nachbarn, und die Pfadlänge wird durch die Chessboard-Distanz (L_∞-Norm) bestimmt. Dies ist notwendig, weil der Feedback-Transport in FairNet darauf aufbaut, dass zwei Peers mit angrenzenden Zonen eine Teilmenge ihrer Nachbarn gemeinsam haben. Die Feedback-Distribution wird in Kapitel 4 detailliert beschrieben. Die Auswirkungen, die sich aus der Zahl der Nachbarn auf das CAN ergeben, werden in [RFH+01] näher untersucht.

Zu den Vorteilen des CAN zählt die Möglichkeit, Nachrichten um unzuverlässige Knoten herumzuleiten. Durch den mehrdimensionalen Schlüsselraum gibt es stets mehrere Möglichkeiten, einen bestimmten Datenbereich zu erreichen. Des weiteren ist die Skalierbarkeit des CAN hervorzuheben: die Zahl der Kontakte ist nicht von der Zahl der Teilnehmer im System abhängig, und durch die Wahl einer geeigneten Dimensionalität des Schlüsselraums ermöglicht das CAN kurze Pfadlängen auch in Systemen mit sehr vielen Teilnehmern. Einer der größten Nachteile im CAN besteht darin, dass Peers 'Löcher' im Schlüsselraum hinterlassen können, wenn sie (z.B. durch einen Systemfehler) ohne eine protokollgerechte Abmeldung aus dem CAN ausscheiden. Diese Löcher müssen dann durch einen aufwendigen Reparaturmechanismus einem angrenzenden Peer zugewiesen werden. Ein weiterer Nachteil des ursprünglichen Protokolls ist die statische Kontaktwahl: durch Einführung von frei wählbaren zusätzlichen Kontakten lässt sich die mittlere Pfadlänge bei der Weiterleitung erheblich reduzieren [BB03b].

LH* Dies ist der 'Urvater' aller verteilten Hashtabellen [LNS93]. Dabei wird das Konzept des *Linearen Hashens* [Lit80] auf eine verteilte Systemarchitektur übertragen, d.h., die Topologie des Schlüsselraums entspricht einem assoziativen Array. LH* unterscheidet zwischen den Rechnern, die Daten abrufen (Clients) und Maschinen, die die Daten speichern (Server). Jeder

[7] Abschnitt 3.7 untersucht eine alternative Protokollvariante, in der die Antwort auf dem Weg zurückgeschickt wird, auf dem sie den Beantworter erreicht hat.

[8] Da benachbarte Peers Zonen unterschiedlicher Größe haben können, ist $2 \cdot d$ die untere Schranke für die Zahl der Nachbarn eines Peers.

Server verwaltet dabei ein oder mehrere Hash-Buckets. Beim Überlauf wird ein Hash-Bucket aufgesplittet und ggf. auf mehrere Server neu verteilt.

Jeder Client verfügt über eine Liste aller Server (d.h., eine globale Weltsicht) und über eine Hashfunktion, die die Hash-Buckets den Servern zuordnet. Da bei einem Überlauf Splits der Hash-Buckets durchgeführt werden, ohne die Clients darüber zu informieren, ist die Weltsicht der Clients jedoch möglicherweise veraltet. Darum ist die Bestimmung, welches Datenpaket auf welchem Server liegt, beim LH*-Algorithmus ein zweistufiger Prozess. Zunächst ermittelt der Client anhand seiner Hashfunktion den Server, auf dem das gesuchte Datenpaket nach seinem Wissen liegen muss, und sendet ihm eine Anfrage. Hat dieser Server einen Split des Buckets durchgeführt, und ist darum nicht mehr im Besitz des gesuchten Datenpakets, so ist er jedoch im Besitz der Information, welcher Server der neue Verwalter für das gesuchte Datentupel ist. Darum wird im zweiten Schritt die Anfrage an den richtigen Server weitergeleitet und eine Nachricht an den Client versendet, die dessen Weltsicht auf den neuesten Stand bringt. Die globale (wenn auch ggf. veraltete) Weltsicht beschränkt die Einsetzbarkeit von LH* auf größere Rechenzentren: für Internet-weites Datenmanagement mit Millionen von angeschlossenen Rechnern ist LH* nicht geeignet.

Chord Diese P2P-Datenstruktur basiert auf einem ringförmigen Schlüsselraum. Neue Peers treten an einer zufälligen Position auf dem Schlüsselraum in das Chord-Netzwerk [SMLN+01] ein. Jeder Peer ist für einen Ausschnitt des Schlüsselraums zuständig, der von seinem unmittelbaren Vorgänger im Schlüsselraum bis zu seiner eigenen Position reicht. Darüber hinaus kennt jeder Peer seinen Nachfolger, und verwaltet in einer *Finger Table* die Kontaktadressen aller Peers, die für Schlüssel im Abstand $2^0, 2^1, \cdots, 2^x$ zu seiner eigenen Position im Schlüsselraum verantwortlich sind.

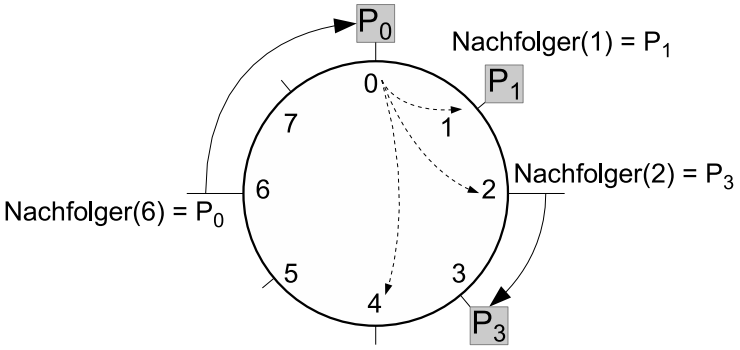

Abbildung 2.6: Aufteilung des Schlüsselraums in Chord.

Beispiel: Abbildung 2.6 zeigt ein Beispiel für einen Chord-Schlüsselraum mit drei Peers P_0, P_1 und P_3, die jeweils an den Positionen 0, 1 und 3 im Schlüsselraum positioniert sind. Der Knoten P_0 ist hier für alle Schlüssel im Intervall $(3, 0]$ zuständig, P_1 verwaltet die Schlüssel im Intervall $(0, 1]$ und P_3 verwaltet $(1, 3]$. Eine Anfrage nach Schlüssel 6 wird also an P_0 weitergegeben. Die gestrichelten Linien in Abbildung 2.6 zeigen auf die Kontakte des Peers

P_0. Der Peer P_0 speichert in seiner Finger Table alle Kontakte, die für die Schlüssel $1, 2, 4$ zuständig sind, das sind in der gegebenen Reihenfolge P_1, P_3 und P_0. In der Finger Table von Peer P_1 befinden sich die für die Schlüssel $2, 3, 5$ zuständigen Peers, d.h. P_3, P_3 und P_0.

Anfragen werden in Chord stets in Richtung aufsteigender Schlüsselwerte weitergeleitet. Gegeben sei eine Anfrage nach dem Schlüssel k. Solange ein Peer in seiner Finger Table einen Knoten findet, der an einer Position $\leq k$ im Schlüsselraum liegt, wird er die Anfrage an diesen weitergeben. Existiert hingegen kein Kontakt mit einer Position im Schlüsselraum, die kleiner oder gleich k ist, so ist der aktuelle Peer für die Anfrage verantwortlich. In Abbildung 2.6 würde P_0 eine Anfrage nach dem Schlüssel 2 also an P_3 weiterleiten, für den Schlüssel 6 wäre er selbst verantwortlich. Bei N Peers in Chord verwaltet jeder Knoten $O(\log N)$ Kontakte. Durch den sich jeweils verdoppelnden Abstand der Kontakte entsteht so ein virtueller binärer Suchbaum: jede Nachricht wird in $O(\log N)$ Schritten vom Sender zum Empfänger geleitet.

Zu den Vorteilen von Chord zählt die Robustheit. Beim Ausfall eines Knotens wird dessen Nachfolger automatisch zum Inhaber des Schlüsselbereichs. Abgesehen von den Informationen über den unmittelbaren Nachfolger des Knotens dürfen sämtliche Einträge in der Finger Table veralten, ohne dass mehr als die Routing-Performanz des Chord-Netzwerks beeinträchtigt wird. Daneben lässt sich Chord durch seine einfache Struktur leicht implementieren. Nachteilig an Chord sind der eindimensionale Schlüsselraum und die statische Kontakt- und Pfadwahl, die keine Alternativrouten um unzuverlässige Knoten herum zulassen. Des weiteren steigt nicht nur die Pfadlänge, sondern auch die Zahl der Kontakte eines Peers mit zunehmender Teilnehmerzahl.

P-Grid Die Topologie von P-Grid [Abe01] entspricht einem Präfix-Baum mit einem binären Alphabet. Jeder Peer ist verantwortlich für einen Blattknoten, d.h. für einen Schlüssel-Präfix. In einem P-Grid mit zwei Peers ist der eine Peer beispielsweise für alle Schlüssel beginnend mit '0' zuständig, der andere verwaltet alle Schlüssel mit dem Präfix '1'. Wenn weitere Peers dem P-Grid beitreten, können diese Präfixe spezialisiert werden. Zum Beispiel kann ein Datenbereich mit dem Präfix '110' in zwei Datenbereiche mit den Präfixen '1100' und '1101' aufgeteilt und an zwei unterschiedliche Peers vergeben werden. Weiterentwicklungen von P-Grid [ADHS05] spezialisieren die Präfixe so, dass jeder Teilbaum nahezu die gleiche Datenmenge verwaltet; auf diese Weise wird eine ausgeglichene Lastverteilung im Suchbaum bei realistischen Schlüsselverteilungen erreicht.

Die Kontaktwahl in P-Grid geschieht wie folgt: ein Peer verwaltet für jeden 'Buchstaben' in seinem Präfix eine Referenz auf einen anderen Peer, der für den gleichen Teilpräfix, aber einen unterschiedlichen Wert an der betrachteten Stelle zuständig ist. So speichert ein Peer mit dem Präfix '0000' Kontakte zu Peers mit den Präfixen '1', '01', '001' und '0001'. In der Baum-Topologie verfügt also ein Peer in jeder Tiefe über eine Referenz zu einem Peer im anderen Teilbaum.

Anfragen, die ein Peer nicht selbst beantworten kann, übergibt er an den ihm bekannten Knoten, der den längsten Präfix mit dem Anfrageschlüssel gemein hat. In P-Grid muss jeder Peer eine Anzahl von Kontakten verwalten, die der Länge seines Präfix bzw. der Tiefe des Baumes

entspricht; üblicherweise sind dies $O(\log N)$ Kontakten in einem System mit N Peers. Die Suche im Baum erfordert dann ebenfalls $O(\log N)$ Schritte.

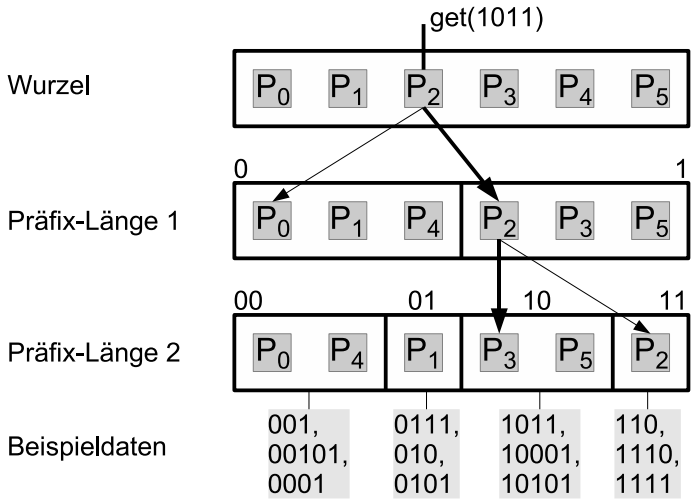

Abbildung 2.7: Der Schlüsselraum von P-Grid.

Beispiel: Abbildung 2.7 zeigt ein Beispiel für ein P-Grid mit 6 Peers. Die Peers P_0 und P_4 sind (gemeinsam als Replikate) für alle Schlüssel zuständig, die mit '00' beginnen. Der Peer P_2 ist verantwortlich für den Präfix '11'. Er speichert Kontakte zu den Peers P_0 und P_3 (in der Abbildung als Pfeile dargestellt). Eine Anfrage nach einem Schlüssel '1010' übergibt P_2 an den Peer P_3, da der Anfrageschlüssel erst ab eine Präfix-Länge von 2 von dem Präfix abweicht, für den P_2 verantwortlich ist (fett gedruckte Pfeile).

Ein großer Vorteil von P-Grid ist die flexible Kontaktwahl. Für die erste Stelle im Präfix kann ein Peer einen Kontakt aus der Hälfte aller Peers in P-Grid wählen, bei einem Präfix der Länge 2 aus einem Viertel etc. Des weiteren lässt sich diese Entscheidung im laufenden Betrieb verfeinern. Unzuverlässige Peers können so beim Weiterleiten von Anfragen umgangen werden. Darüber hinaus kann ein Peer durchaus auch mehr als einen Kontakt je Präfix-Länge verwalten. Ein weiterer Vorteil besteht darin, dass der Suchbaum in P-Grid flexibel wachsen und schrumpfen kann, da Spezialisierungen auf längere Präfixe leicht rückgängig gemacht werden können.

Pastry, Tapestry Die beiden P2P-Datenstrukturen Pastry [RD01] und Tapestry [ZKJ01] sind ähnlich aufgebaut. Sie basieren beide auf dem Plaxton-Mesh [PRR97] und verwenden das Landmark-Routing [Tsu88] zur Nachrichtenweiterleitung. Beide Ansätze unterscheiden sich hauptsächlich darin, wie Replikation und Adress-Lokalität realisiert werden. Im folgenden wird daher nur Pastry näher beschrieben.

Jeder Knoten in Pastry wird über eine 128 Bit lange ID adressiert, die auf einen ringförmig organisierten Schlüsselraum von 0 bis $2^{128} - 1$ abgebildet wird. Die ID wird jedem Knoten

beim Eintritt in Pastry zufällig so zugeordnet, dass eine Gleichverteilung der Knoten über den Schlüsselraum erreicht wird. Ebenso wie in Chord ist jeder Knoten für alle Schlüssel verantwortlich, die in das Intervall zwischen seiner ID bis zur ID des nächsten Knotens im Schlüsselraum passen.

Die Kontaktwahl in Pastry ist komplex: jeder Knoten verwaltet eine *Routing-Tabelle*, ein *Neighborhood Set* und ein *Leaf Set*. Die Routing-Tabelle ist organisiert in $\lceil \log_{2^b} N \rceil$ Zeilen mit $(2^b - 1)$ Spalten. In Zeile l verwaltet der Peer dabei Kontakte zu Knoten, die auf den ersten l Stellen über die gleiche ID verfügen, d.h. deren IDs einen gemeinsamen Präfix der Länge l haben. Der Parameter b dient dazu, einen Kompromiss zwischen einer kurzen Pfadlänge und dem Verwaltungsaufwand einer großen Routing-Tabelle zu bilden. Das Neighborhood Set enthält Kontakte, die in der geografischen Nähe[9] des aktuellen Peers liegen. Auf diese Weise kann Pastry Informationen über die Lokalität zwischen den Peers ausnutzen, die zwar über eine kurze bzw. schnelle Netzwerkschnittstelle verbunden sind, im Schlüsselraum jedoch weit entfernt angesiedelt sind. Im Leaf Set verwaltet jeder Peer Kontakte zu einer Anzahl von Knoten, deren IDs numerisch an die ID des Peers angrenzen. Das bedeutet, das Leaf Set speichert Kontaktinformationen zu den nächsten Nachbarn im Schlüsselraum. Mit Hilfe dieser Kontaktinformationen werden Nachrichten in Pastry von jedem Knoten jeweils an den Kontakt weitergeleitet, der dem Schlüssel der Nachricht numerisch am nächsten ist. Im Gegensatz zu Chord werden also Nachrichten in beide Richtungen im Schlüsselraum weitergegeben. Die Routing-Tabelle ermöglicht als untere Schranke, dass der gemeinsame Präfix der Knoten-ID und des Nachrichtenschlüssels mit jeder Weiterleitung um eine Ziffer wächst. Damit beträgt die zu erwartende Zahl der Weiterleitungen jeweils $\lceil \log_{2^b} N \rceil$ Schritte in einem Netzwerk mit N Peers.

Zu den Vorteilen von Pastry und Tapestry gehört die Kontaktwahl, die ebenso wie bei P-Grid sehr flexible Vorgaben darüber macht, zu welchen Knoten ein Peer Kontakte speichern muss. Eine Auswahl von Kontakten nach Merkmalen wie Zuverlässigkeit, Vertrauenswürdigkeit oder Performanz ist damit gegeben. Ein Alleinstellungsmerkmal ist die Einbindung von Informationen über die Lokalität, also die Berücksichtigung der Kosten, mit denen ein Knoten zu erreichen ist. Nachteilig an den Ansätzen ist jedoch der komplizierte Algorithmus zur Kontaktwahl, der Informationen über im Schlüsselraum entfernte Kontakte zusammenträgt.

Viceroy Dies [MNR02] ist die erste P2P-Datenstruktur, die bei einer konstanten Zahl von Kontakten zwischen den Peers einen logarithmischen Suchaufwand $O(\log N)$ erzielt. Zum Vergleich: im CAN ist die Zahl der Kontakte eines Knotens von der Zonenaufteilung und der Dimensionalität des Schlüsselraums abhängig, und in Chord von der Zahl der Peers im System. Viceroy erzielt die konstante Zahl von Kontakten durch die Übertragung des Konzepts der Butterfly-Networks [DT04] auf eine P2P-Datenstruktur. Der Schlüsselraum von Viceroy ist dabei als ein Ring mit dem Intervall von 0 bis 1 organisiert, und ist in mehrere logische Ebenen ('Level') unterteilt. In einem Viceroy-Netzwerk mit N Peers gibt es dabei $\log_2 N$ Level. Neue

[9] Die Metrik zur Berechnung der Nachbarschaft zweier Knoten lässt sich auch auf die Qualität der Netzwerkverbindung oder auf andere Maße der Lokalität erweitern.

Peers wählen sich zufällig einen Level l und eine Position x auf dem Schlüsselraum aus, und verwalten die folgenden Kontakte:

- einen *Up-Link* auf Level $l - 1$ nahe bei x
- *Level-Ring-Links* zum unmittelbaren Vorgänger und Nachfolger auf Level l
- auf Level $l + 1$ zwei *Down-Links* zu Peers an den Positionen $x + (1/2)^l$ und x

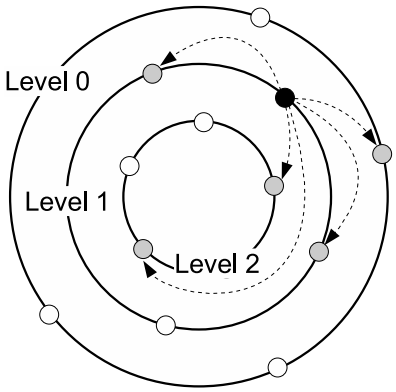

Abbildung 2.8: Der Schlüsselraum von Viceroy.

Beispiel: Abbildung 2.8 zeigt ein ideales Viceroy-Netzwerk mit drei Leveln und 12 Knoten. Die Knoten sind als kleine Kreise dargestellt. Die Pfeile deuten auf alle Peers (grau abgebildet), deren Kontaktdaten der schwarz dargestellte Peer verwalten muss.

Ebenso wie bei Chord ist jeder Peer in Viceroy für die Daten zuständig, die in das Intervall zwischen seiner Position im Schlüsselraum und dem nächsten Peer passen. Die Aufteilung des Schlüsselraums ist unabhängig von den Leveln der Peers: die Level bestimmen ausschließlich die Kontaktwahl. Anfragen werden in Viceroy in einem dreistufigen Prozess an den Peer weitergeleitet, der sie beantworten kann: Zuerst wird die Anfrage von jedem Peer jeweils an seinen Up-Link weitergeleitet, bis Level 0 erreicht ist. Im zweiten Schritt wird die Anfrage nun – beginnend bei dem Peer auf Level 0 – jeweils an denjenigen der beiden Down-Links weitergegeben, der näher am gesuchten Anfrageschlüssel ist. Wenn beide Down-Links weiter vom Anfrageschlüssel entfernt sind als der Vorgänger oder Nachfolger auf dem aktuellen Level, so wird die Anfrage im dritten Schritt über die Level-Ring-Links bis zum Zielpeer weitergeleitet.

Da die Zahl der Level logarithmisch von der Zahl der Peers im System N abhängt und jeder Peer über eine konstante Zahl von Kontakten verfügt, realisiert Viceroy auf diese Weise einen ebenfalls logarithmischen Suchaufwand $O(\log N)$. Die konstante Zahl der Kontakte ist der wesentliche Vorteil dieser Struktur: in Szenarien, in denen die Teilnehmer das System häufig betreten oder verlassen, ist der Aufwand zur Pflege der Kontaktinformationen in Viceroy geringer als in allen anderen hier beschriebenen Datenstrukturen. Problematisch sind jedoch die statische Kontaktwahl und der ebenfalls statische Routing-Prozess, welche keine Alternativrouten um unzuverlässige Peers herum zulassen.

2.4 Unkooperatives Verhalten in P2P-Datenstrukturen

In P2P-Datenstrukturen werden die Datentupel strukturiert auf die einzelnen Knoten aufgeteilt. Operationen wie z.B. Suchanfragen werden von Peer zu Peer *zielgerichtet* zu dem Knoten weitergeleitet, der für den Datenbereich verantwortlich ist. Jeder Teilnehmer muss also eigene Ressourcen einbringen, um (1) die Daten zu verwalten, deren Schlüssel in seine Zone passen, und (2) die Anfragen von anderen Peers zu beantworten oder weiterzuleiten. Wenn einer der Weiterleiter versucht, diese Ressourcen zu sparen, indem er eine Anfrage nicht weiterleitet oder nicht beantwortet, bleibt die Anfrage unbeantwortet. Der Anfragesteller kann dann die Anfrage nur wiederholen und hoffen, dass sie über andere Peers bzw. an andere Replikate der Daten weitergeleitet wird. Im Rahmen dieser Dissertation wird ein *unkooperativer Peer* oder *Free Rider* [RL03] wie folgt definiert:

> *Ein unkooperativer Peer ist ein rationaler Teilnehmer einer P2P-Datenstruktur, der Ressourceneinsparungen zu realisieren versucht, indem er das Protokoll der Datenstruktur (zeitweise) nicht befolgt.*

Die starke Fokussierung auf Ressourceneinsparungen und auf wirtschaftlich rationales Verhalten grenzt dabei das unkooperative Verhalten von anderen Arten unerwünschten Verhaltens in P2P-Datenstrukturen ab. Ein rationaler Peer wird sich sofort kooperativ verhalten, wenn er feststellt, dass er durch unkooperatives Betragen mit einem im Vergleich höheren Ressourcenbedarf beaufschlagt wird. Auf Angreifer, welche die Datenintegrität beeinträchtigen oder einzelne Peers aus dem System herausdrängen wollen, trifft diese Definition beispielsweise nicht zu.

Unkooperative Peers führen dazu, dass die Verfügbarkeit der Daten und des Systems sinkt. Die Antwortzeit verlängert sich durch zu wiederholende Anfragen, und die Motivation der anderen Teilnehmer, selbst Ressourcen einzubringen, sinkt. Auf der anderen Seite führen Maßnahmen gegen unkooperative Teilnehmer in P2P-Datenstrukturen neben den avisierten Effekten jedoch auch stets zu erhöhten Gemeinkosten für alle anderen Teilnehmer. Angesichts der Tatsache, dass etliche populäre P2P-Systeme (z.B. Kazaa[10], gnutella[11] etc.) seit Jahren trotz zahlreicher unkooperativer Peers offenbar erfolgreich betrieben werden, muss sich ein Ansatz gegen unkooperatives Verhalten zuerst der Frage stellen, ob er tatsächlich erforderlich ist. Dazu muss zunächst die Ausfallwahrscheinlichkeit für Anfrageergebnisse ermittelt werden.

Beispiel: In einem CAN mit $N = 10.000$ Knoten und $d = 4$ Dimensionen wird eine Anfrage durchschnittlich $l = d/4 \cdot N^{1/d} = 10$ mal weitergeleitet. Angenommen, in diesem CAN befinden sich $u = 500$ unkooperative Teilnehmer, die eingehende Nachrichten verwerfen, ohne sie zu bearbeiten. Dann beträgt die Wahrscheinlichkeit, ein Anfrageergebnis zu erhalten, nur noch $(1 - u/N)^l \approx 60\%$.

Die Wahrscheinlichkeit für eine erfolgreiche Anfrage sinkt exponentiell mit steigender Pfad-

[10] http://www.kazaa.com
[11] http://www.gnutella.com

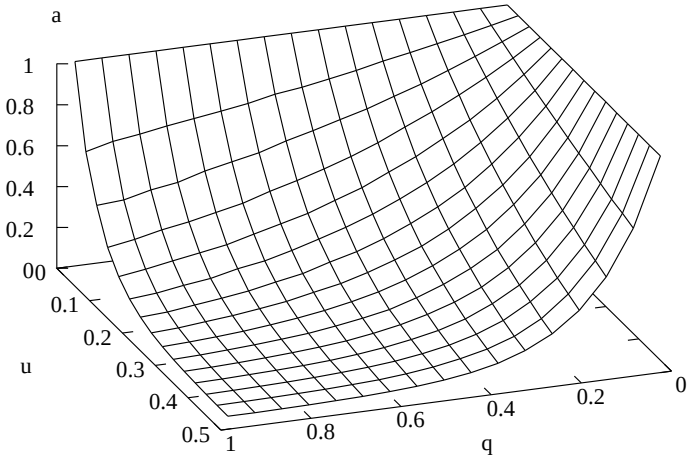

Abbildung 2.9: Wahrscheinlichkeit für eine erfolgreiche Anfrage in Anwesenheit von unkooperativen Teilnehmern.

länge und steigender Ausfallwahrscheinlichkeit. Ein Experiment soll diesen Zusammenhang illustrieren. Dazu wird ein CAN mit $N = 100.000$ Peers und einem vierdimensionalen Schlüsselraum aufgesetzt, und die Zahl der unkooperativen Teilnehmer u von 0% bis 50% variiert. Des weiteren wurde mit einem Anteil der von den unkooperativen Peers verworfenen Nachrichten q zwischen 0% und 100% experimentiert. Abbildung 2.9 zeigt als Ergebnis des Experiments den Anteil der beantworteten Anfragen a auf der z-Achse. Das Experiment zeigt, dass in einem CAN ohne Maßnahmen gegen unkooperatives Verhalten bereits eine kleine Anzahl unkooperativer Teilnehmer und eine geringe Ausfallwahrscheinlichkeit genügen, um die Wahrscheinlichkeit für ein Anfrageergebnis erheblich zu reduzieren. Dieser Herausforderung müssen sich nicht nur CAN, sondern sämtliche P2P-Datenstrukturen stellen, die Nachrichten zielgerichtet weiterleiten. Einzig die von Ansatz zu Ansatz verschiedene mittlere Pfadlänge sorgt dafür, dass das Problem unkooperativer Teilnehmer bei einigen P2P-Datenstrukturen eine höhere Evidenz aufweist als bei anderen.

Kosten im CAN in Anwesenheit unkooperativer Teilnehmer Die einzig mögliche Reaktion auf nicht beantwortete Anfragen besteht darin, diese nach Ablauf einer bestimmten Zeitspanne zu wiederholen. Angenommen, das CAN verfügt über einen sehr hohen Replikationsgrad, viele alternative Realitäten [RFH+01] oder m-aus-n Datenkodierung (s. [RWE+01]), sodass unkooperative Teilnehmer keinen Datenverlust, sondern nur eine schlechtere Antwortzeit und erhöhte Netzwerklast durch wiederholte Anfragen verursachen. Wann ist es in diesem Szenario rentabel[12], zusätzlichen Aufwand in ein Protokoll zur Erkennung und Vermeidung von unkooperativen Teilnehmern zu investieren? Beziehungsweise präziser formuliert: Wann sind Latenz und Netzwerklast bei einem Szenario mit wiederholten Anfragen insgesamt größer

[12] Hier sollen die Kosten, die durch Replikation entstehen, nicht berücksichtigt werden. In Systemen mit häufigen Änderungsoperationen auf dem Datenbestand kann bereits der Aufwand zur Synchronisation [LMR02] einen Anreiz für Maßnahmen gegen unkooperative Teilnehmer schaffen.

als bei einem Protokoll, das neben dem Nachrichtentransport selbst auch Kosten für Erkennung und Vermeidung von unkooperativen Peers verursacht?

Zuerst soll die durchschnittliche Zahl der Weiterleitungen einer Nachricht $\overline{f}$ in Anwesenheit von unkooperativen Teilnehmern ermittelt werden. Wenn jede Nachricht ihr Ziel erreicht, beträgt die mittlere Pfadlänge im CAN $l = d/4 \cdot N^{1/d}$. Nun sei $p = u \cdot q/N$ die globale Ausfallwahrscheinlichkeit in einem CAN mit u unkooperativen Teilnehmern, die jeweils einen Anteil von q Nachrichten nicht beantworten. In einem CAN ohne unkooperative Peers oder ohne verworfene Nachrichten ($u = 0$, $q = 0$) gilt $\overline{f} = l$. Der andere Extremfall ist ein CAN, das nur aus unkooperativen Teilnehmern besteht, die alle Nachrichten verwerfen ($u = N$, $q = 1$). In diesem Fall wird jede Anfrage genau einmal weitergeleitet und dann verworfen, also gilt[13] $\overline{f} = 1$. Im allgemeinen Fall ist die durchschnittliche Zahl der Weiterleitungen die Summe der Wahrscheinlichkeiten, dass eine Nachricht nach $1 \cdots l$ Weiterleitungen verworfen oder nach l Weiterleitungen beantwortet wird, jeweils multipliziert mit der entsprechenden Zahl der Weiterleiter:

$$\overline{f} = \Big(\sum_{i=1}^{l} (1-p)^{i-1} \cdot p \cdot i \Big) + (1-p)^l \cdot l \tag{2.1}$$

Nun soll eine Anfrage so lange wiederholt werden, bis entweder das Anfrageergebnis eintrifft, oder t Wiederholungen überschritten werden. Dann ergeben sich die durchschnittlichen Kosten $\overline{c_n}$ für eine Anfrage aus den Kosten c_m für das Weiterleiten einer Nachricht von einem Peer zum nächsten, multipliziert mit der durchschnittlichen Zahl der Weiterleitungen je Versuch und der Summe der Wahrscheinlichkeiten für $1 \cdots t$ Versuche:

$$\overline{c_n} = c_m \cdot \overline{f} \cdot \sum_{j=1}^{t} \left(1 - (1-p)^l\right)^{j-1} \cdot (1-p)^l \tag{2.2}$$

Wie sind nun die Kosten beschaffen bei einem Protokoll, dass unkooperative Teilnehmer ausschließt, indem an jede ausgehende Nachricht Feedback mit den Kosten c_a angehängt wird? Unter den Annahmen, dass Strafmaßnahmen (z.B. Arbeitsbeweise) nur unkooperative Peers betreffen, kein Datenverlust auftritt und die Wege im CAN durch das Umgehen von unkooperativen Peers nicht verlängert werden, betragen diese Kosten durchschnittlich:

$$\overline{c_r} = l \cdot (c_m + c_a) \tag{2.3}$$

Realistische Angaben für den Anteil der unkooperativen Teilnehmer im System und für die Kosten der Datenhaltung Erfahrungen mit P2P-Tauschbörsen, die bereits über einen langen Zeitraum betrieben werden und zahlreiche Teilnehmer aufweisen, zeigen einen Anteil

[13] Bei $N >> 1$ kann der Anteil der Anfragen vernachlässigt werden, die ein Peer aus seinem eigenen Datenbestand beantworten kann.

von bis zu 70% Free Ridern (gnutella, [AH00]). Allerdings lassen sich diese Erkenntnisse nicht unmittelbar auf P2P-Datenstrukturen übertragen; so unterscheiden sich die Anwendungsprofile von Tauschbörsen und P2P-Datenstrukturen erheblich, und das hier vorweggenommene FairNet-Protokoll motiviert unkooperative Teilnehmer dazu, das Netz zu verlassen oder ihr Verhalten zu ändern. Daher soll eine Rate von 10% vollständig unkooperativen Teilnehmern angenommen werden – bei einem höheren Anteil unkooperativer Peers wird es noch schneller rentabel, Maßnahmen gegen diese Peers einzusetzen.

Um realistische Kosten zu bestimmen, soll ein P2P-Webcrawler als Beispielanwendung herangezogen werden. Hier speichert jeder Peer URLs in der P2P-Datenstruktur. Die Kosten werden bei dieser Anwendung durch die Länge (in Bytes) der zu verwaltenden URLs bestimmt. Um dafür realistische Werte zu erhalten, wurde ein experimenteller Webcrawler eingesetzt, der in einem Partnerprojekt entwickelt wird. Der Webcrawler beginnt bei einer vorgegebenen Startseite im WWW und folgt den darin enthaltenen URLs zu anderen Seiten mittels eines modifizierten[14] Random Walk Algorithmus. Tabelle 2.2 zeigt einige ausgewählte Statistiken aus einem Lauf des Crawlers.

Gesamtzahl der besuchten WWW-Seiten	7.646.238
Minimum-Länge der besuchten URLs	12,0
Maximum-Länge der besuchten URLs	255,0
Durchschnittliche Länge der URLs	61,1
Standardabweichung der URL-Längen	18,7

Tabelle 2.2: Statistiken des experimentellen Webcrawlers.

Neben der URL selbst enthält jede Anfrage in einer P2P-Datenstruktur zusätzliche Protokollinformationen. Dazu zählen Daten über den Absender wie ID, Port oder IP-Adresse, eventuell Informationen über seinen Datenbereich und dergleichen. Dazu kommt noch der Protokolloverhead einer IP-Verbindung, also SYN-, ACK- und FIN-Pakete zum Verbindungsauf- und abbau und der TCP/IP-Paketrahmen. Eine konservative Schätzung für die Kosten der Übertragung einer Nachricht von einem Peer zu seinem Nachbarn ist daher $c_m = 600$ Bytes.

In Abschnitt 3.5 wird gezeigt, dass 10 Feedback-Objekte ausreichen, um nahezu sämtliche unkooperativen Peers zu identifizieren. Feedback-Objekte bestehen aus einer Peer-ID, einem Zeitstempel (jeweils long-Datentypen) und einigen Flags (s. Abschnitt 3.1). Daher gilt die Abschätzung $c_a = 30 \cdot 10$ Bytes.

[14] Der Algorithmus merkt sich besuchte URLs in einem Zwischenspeicher. Wenn ein Webserver einen Fehler meldet oder eine URL ins Leere führt, beginnt der Algorithmus bei einer zufällig ausgewählten Seite aus diesem Zwischenspeicher von neuem.

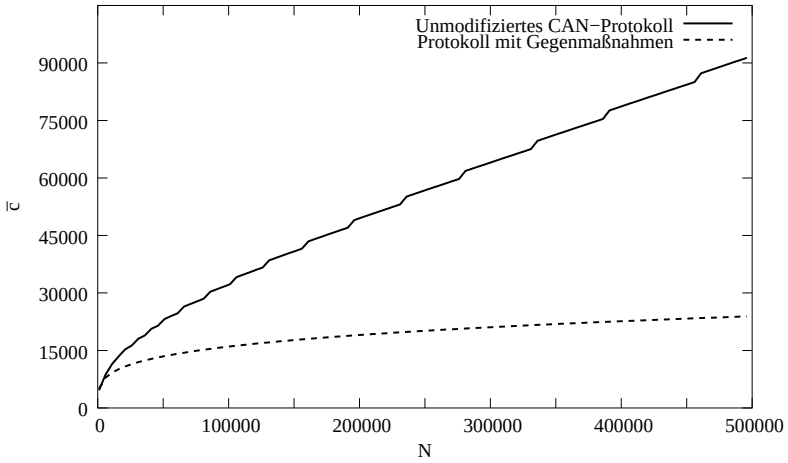

Abbildung 2.10: Netzwerklast mit und ohne Maßnahmen gegen unkooperative Peers.

Bestimmung der Gewinnschwelle für Maßnahmen gegen unkooperative Peers Abbildung 2.10 zeigt nun die mit diesen Werten ermittelten Gesamtkosten $\overline{c}$ für ein unmodifiziertes CAN-Protokoll und eines mit Maßnahmen gegen unkooperative Teilnehmer. Dabei wurde die Gesamtzahl der Peers in einem CAN mit $d = 4$ von $N = 1.000$ bis $N = 500.000$ variiert, und jeweils ein Anteil von 10% vollständig unkooperativen Teilnehmern beibehalten. Die Zahl der Wiederholungen für fehlgeschlagene Anfragen im unmodifizierten CAN wurde nicht limitiert, d.h., es wurde davon ausgegangen, dass eine Anfrage so oft wiederholt wird, bis das Anfrageergebnis vorliegt. Wie erwartet erhöht eine steigende Zahl von Peers durch steigende Pfadlängen und damit einhergehender erhöhter absoluter Ausfallwahrscheinlichkeiten die Kosten beim ursprünglichen CAN-Protokoll beträchtlich. Der Anstieg bei den Kosten des erweiterten Protokolls resultiert hingegen nur aus dem Anstieg der Pfadlänge bei steigender Teilnehmerzahl. Selbst bei einem geringen Anteil von nur 10% unkooperativen Peers unterbietet das erweiterte CAN-Protokoll so die Kosten des Standardprotokolls bereits bei einer geringen Zahl von Teilnehmern.

Selbstverständlich ist es möglich, Szenarien zu konstruieren, in denen Peers mit einem Protokoll ohne Maßnahmen gegen unkooperative Teilnehmer mit geringeren Kosten auskommen als beim erweiterten Protokoll. Aber diese Szenarien sind zumeist unrealistisch oder unpraktikabel. Im hier zugrundegelegten Beispiel befindet sich der Schnittpunkt der Kostenkurven bei einem Anteil von 2,9% unkooperativen Teilnehmern von insgesamt 500.000 Peers im System, oder bei weniger als 2.500 Peers in einem Szenario mit 10% unkooperativen Peers.

Eine Analyse der Latenz bei der Anfrageverarbeitung führt zu vergleichbaren Resultaten. Auch hier würde eine steigende Zahl von Wiederholungen zu einer hohen Verzögerung bis zum Erhalt des Anfrageergebnisses beim unmodifizierten Protokoll führen. Im Unterschied zur Kostenanalyse führt das Anhängen von zusätzlichen Feedback-Objekten an ausgehende Nachrichten jedoch nicht zu einer wahrnehmbar steigenden Latenz. Die Zeit zum Übertragen von einigen zusätzlichen Bytes bzw. zum Berechnen des Feedback-Paketes ist um Größenordnungen kleiner als die Wartezeit, nach der eine Anfrage wiederholt werden muss. Wird also einzig die

Latenz bei der Anfrageverarbeitung betrachtet, sind Gegenmaßnahmen gegen unkooperatives Verhalten stets vorteilhaft.

2.5 Der Umgang mit unkooperativen Teilnehmern

Nachdem motiviert wurde, welche Probleme unkooperative Teilnehmer in P2P-Datenstrukturen verursachen können, sollen in diesem Abschnitt bestehende Lösungen für diese Probleme diskutiert werden. Die Herausforderungen bei der Kooperation zahlreicher Teilnehmer in einer großen Gemeinschaft wird von einem sehr breiten Spektrum unterschiedlicher Forschungsdisziplinen thematisiert:

- Die *Makro- und Mikroökonomie* ermittelt, unter welchen Umständen (Kosten, Nutzen, verfügbare Informationen) ein einzelner rationaler Teilnehmer oder eine Gesellschaft welche Ergebnisse erzielt [FNSY96].
- *Rechts- und Politikwissenschaften* versuchen, Regeln für das soziale Zusammenleben aufzustellen.
- Die *Entscheidungstheorie* evaluiert die Konsequenzen der von den Teilnehmern getroffenen Entscheidungen und sucht nach Kriterien für rationale Entscheidungen.
- Beim *Risikomanagement* wird versucht, (Ausfall-) Risiken in den Prozess der Entscheidungsfindung einfließen zu lassen [CBN05].
- Die evolutionäre *Spieltheorie* erforscht anhand stark vereinfachter Modelle die Anwendbarkeit kooperativer Strategien [Axe84].
- Die *Evolutionsbiologie* ermittelt, wie sich kooperative Beziehungen zwischen unabhängigen Individuen (z.B. in Bienenschwärmen) auf das Überleben der Population auswirken.
- Die *Anthropologie* untersucht, wie sich natürlich entstandene kooperative menschliche Gemeinschaften bewähren.
- Die *Soziologie* untersucht die Verhaltensdynamik in kleinen und großen Gruppen.
- Die *Sozialpsychologie* erforscht Entscheidungsprozesse in Gruppen sowie Gruppendenken, Gruppenkonformität, Intergruppenbeziehungen, etc. [TP05].
- In der *Soziophysik* wird untersucht, wie physikalische Methoden (z.B. Ising-Modelle zur Beschreibung des Ferromagnetismus, Energieerhaltungssatz der Thermodynamik) zur Beschreibung kollektiver Entscheidungen in sozialen Systemen anwendbar sind [Sch03].
- Die *Biotechnologie* überträgt in der Natur beobachtete Verhaltensweisen auf Computernetzwerke (Organic Computing, Schwarm-Intelligenz) [AB05].
- Die *Informatik* versucht, technisch realisierbare Algorithmen zu finden, die Kooperation in großen Computernetzwerken ermöglichen.

Diese Liste ist inhärent unvollständig. Des weiteren sind Grenzen zwischen den Forschungsdisziplinen fließend, bzw. lassen sich die Disziplinen weiter untergliedern. In der Informatik interessieren sich unter anderem Forscher der Fachrichtungen *Netzwerke, Agenten-Systeme, Peer-to-Peer-Systeme, Wissensmanagement, Computerspiele, Kollaborative Systeme, Elektro-*

nische Bezahlsysteme und *Elektronische Märkte* für diese Problemstellung. Dementsprechend vielfältig ist die Zahl der Ansätze, um mit unkooperativem Verhalten in Computernetzwerken umzugehen.

In elektronischen Märkten [ZMM99] kommen häufig zentrale Lösungen zur Authentifizierung und zum Monitoring der Benutzer zum Einsatz, z.B. bei eBay[15] oder Amazon[16]. Für mobile Agentensysteme wurden Ansätze zum Austausch von zertifiziertem Programmcode [Nec97] entwickelt, die Manipulationen auf Code-Ebene ausschließen und nur vertrauenswürdigen Agenten den Zugang zum System gestatten. Lösungen zum Umgang mit unkooperativen Teilnehmern in mobilen Ad-Hoc Netzwerken basieren häufig darauf, dass die Funkschnittstellen geografisch benachbarter Teilnehmer abgehört werden können [MGLB00]. In Sensornetzen werden vertrauenswürdige Kommunikationspartner häufig per Vorkonfiguration [MPS+02] festgelegt, bevor das Netz in Betrieb genommen wird. Jedoch ist keiner dieser Ansätze in der Lage, mit den in Tabelle 2.1 aufgeführten Eigenschaften von P2P-Systemen umzugehen. Explizit für den Einsatz in P2P-Systeme wurden folgende Ansätze vorgeschlagen:

Black- und Whitelists Eine Möglichkeit, unkooperative Teilnehmer aus einem System auszuschließen, besteht in der Einführung von weißen oder schwarzen Listen mit im System erlaubten bzw. ausgeschlossenen Teilnehmern. Ein Ansatz zur verteilten Bestimmung von Black- und Whitelists ist [NT04]. Grundsätzlich problematisch ist an diesen Verfahren, dass sie nur mit unveränderlichen Identitäten funktionieren [FR98]. In allen P2P-Datenstrukturen ist es jedoch einem Peer leicht möglich, sich ab- und unter einer neuen Identität wieder anzumelden, wodurch sämtliche Blacklist-Einträge über ihn obsolet werden.

Micropayment-Systeme Diese Systeme wurden bereits in Abschnitt 1.2 angesprochen. Beim Einsatz von Micropayment werden alle Transaktionen, die zwischen den Peers ablaufen, entweder mit Kleinstbeträgen im Sub-Cent-Bereich oder mit virtuellen Guthabenpunkten ohne Bezug zu einem staatlichen Zahlungsmittel vergütet. Es existieren zahlreiche Forschungsarbeiten zum Einsatz von Micropayments in P2P-Systemen [GLBM01, YGM03, FST04]. Auch einige P2P-Dateitauschbörsen setzen Micropayment-Systeme auf der Basis von Guthabenpunkten ein, z.B. Applejuice[17].

Der Vorteil von Micropayment-Systemen besteht darin, dass sie die Interaktion im P2P-System auf eine wirtschaftliche Basis stellen. Das bedeutet, dass mit existierenden mikro- und makroökonomischen Methoden Kosten und Nutzen so festgelegt werden können, dass ein global optimales Verhalten (z.B. ein möglichst großes Angebot von Daten oder Diensten im System, oder möglichst viele Transaktionen zwischen den Peers) der Anwender erreicht werden kann. Allerdings sprechen viele Argumente gegen den Einsatz von Micropayments in P2P-Datenstrukturen: die Transaktionskosten von

[15] http://www.ebay.de
[16] http://www.amazon.de
[17] http://applejuicenet.org

Micropayments sind meist höher als die Anfrageverarbeitung in der P2P-Datenstruktur selbst, es existiert kein sicheres Micropayment-Verfahren ohne eine vertrauenswürdige zentrale Instanz, gewünschtes Verhalten wie z.B. Altruismus [NT04] wird ebenfalls unterdrückt, und bei geldwerten Transaktionen ist der Aufwand der Entscheidungsfindung [Odl03] bei den Benutzern hoch.

Ressourcentausch Im Unterschied zu Micropayment-Verfahren mit virtuellen Münzen verwenden Ressourcentauschprotokolle Transaktionsbestätigungen als Obligationen auf Ressourcen. Ein Peer, der eine Dienstleistung in Anspruch nimmt, stellt dem Leistungserbringer eine 'Schuldverschreibung' aus, die den Inhaber dazu berechtigt, vom Aussteller eine Dienstleistung einzufordern. Diese Schuldverschreibungen sind handelbar, d.h., können von Peer zu Peer weitergegeben werden. Beispiele für Ressourcentauschprotokolle in P2P-Datenstrukturen sind NICE (eine Kollaborationsplattform im Internet, [SLB06]) und Samsara (ein P2P-Backup-System [CN03]).

Im Vergleich zu Micropayment bietet der direkte Ressourcenaustausch den Vorteil, dass er ohne eine zentrale Überwachungsinstanz auskommt: jeder Peer garantiert eigenverantwortlich, das die von ihm ausgegebenen Obligationen auch eingelöst werden können. Nachteilig am Ressourcenaustausch ist jedoch, dass die Obligationen ihre Gültigkeit verlieren, wenn der Aussteller aus dem System austritt oder seine Identität wechselt. Des weiteren funktioniert der wechselseitige Austausch nur in symmetrischen Anwendungen, in denen jeder Peer die gleiche Menge an Ressourcen von anderen verbraucht, wie er auch selbst für andere bereitstellt.

Web-of-Trust Die Public-Key-Kryptographie[18] bietet die Möglichkeit, den Peers verifizierbare Identitäten zuzuordnen. Ist ein Teilnehmer eindeutig identifizierbar, werden auch einfache Ansätze zur Zugangskontrolle (wie beispielsweise Black- oder Whitelists) zum Umgang mit unkooperativen Teilnehmern praktikabel. Beim Web-of-Trust [KR97] unterzeichnet jeder Peer den öffentlichen Schlüssel derjenigen Teilnehmer mit einer Signatur, denen er vertraut. Dieses Vertrauen kann beispielsweise dadurch entstehen, dass ein Peer viele Anfragen für den anderen beantwortet hat. Die Gesamtheit aller Signaturen bildet einen dezentral verwalteten Trust-Graphen über alle Teilnehmer. Ein Peer, der den öffentlichen Schlüssel eines anderen verifizieren möchte, sucht nun im Trust-Graphen nach mindestens einer ununterbrochenen Kette von Signaturen einander vertrauender Peers zwischen sich selbst und dem Zielpeer.

In der Praxis weist der Web-of-Trust-Ansatz eine Reihe von Problemen auf: in Systemen mit sehr vielen Teilnehmern ist es sehr aufwendig, zusammenhängende Ketten von vertrauenswürdigen Teilnehmern zu finden. Wenn Unberechtigte in den Besitz eines privaten Schlüssels gelangt sind, fehlen effiziente Möglichkeiten, korrumpierte Schlüssel und damit erstellte Signaturen zurückzuziehen. Die Kette der einander vertrauenden Knoten ist nur so stark wie ihr schwächstes Glied – bereits ein Knoten kann die Vertrauensbeziehung zerstören, wenn er dem falschen Teilnehmer vertraut. Eine Erweiterung

[18] http://de.wikipedia.org/wiki/Asymmetrisches_Kryptosystem

des Web-of-Trust besteht darin, die öffentlichen Schlüssel in einer P2P-Datenstruktur zu speichern [DHA03], und diese Datenstruktur als Trust-Graph zu nutzen. Die Schlüssel werden mehrfach repliziert in der Struktur abgelegt, und bei der Anfrage wird ein Quorum aus allen zurückgelieferten Schlüsseln gebildet. Diese Vorgehensweise ermöglicht ein Zurückziehen von korrumpierten Schlüsseln und eine effiziente Suche nach dem Schlüssel eines Peers, bringt jedoch neue Gefahren mit sich. Kann beispielsweise ein Angreifer in Erfahrung bringen, welche Peers für einen Schlüssel verantwortlich sind, könnte er Denial-of-Service Angriffe darauf ausführen, oder versuchen, sich selbst an dieser Stelle im Schlüsselraum zu positionieren.

Trust- und Reputationssysteme In diesen Systemen dienen Beobachtungen aus dem Verhalten eines Peers in der Vergangenheit dazu, die Wahrscheinlichkeit für den Erfolg einer Transaktion mit diesem Peer zu ermitteln. Zu den Vorteilen von diesen Systemen zählt, dass sie sich im Vergleich zu den anderen vorgestellten Verfahren einfach dezentral realisieren lassen und wenig Ressourcen benötigen. Nachteilig ist, dass sie keine absoluten Garantien darüber bieten können, dass nur kooperative Teilnehmer vom System profitieren können. Das in dieser Arbeit vorgestellte FairNet-Protokoll ist eine Implementierung eines Reputationssystems. Im folgenden Abschnitt sollen daher Vertrauen und Reputation im Kontext von P2P-Datenstrukturen näher erläutert werden.

2.6 Vertrauen und Reputation in P2P-Datenstrukturen

Die Begriffe *Vertrauen* (engl. *Trust*) und *Reputation* werden ebenfalls in zahlreichen Forschungsdisziplinen sowie im umgangssprachlich Kontext verwendet. Dementsprechend häufig sind alternative Definitionen dieser Begriffe. Eine Übersicht bietet [Mar94]. In Anlehnung daran wird Vertrauen in dieser Arbeit so definiert:

> *Vertrauen ist die subjektive Überzeugung eines Peers, dass ein anderer Peer Willens ist und auch über die nötigen Fähigkeiten und Ressourcen verfügt, um eine bestimmte Transaktion korrekt zu bearbeiten.*

Die Definition impliziert, dass Vertrauen unidirektional ist, d.h., ein Peer, der einem anderen vertraut, wird nicht automatisch auch von diesem als vertrauenswürdig betrachtet. Weiterhin ist Vertrauen subjektiv. Der gleiche Peer kann von unterschiedlichen anderen Peers auch unterschiedlich eingeschätzt werden. Außerdem ist Vertrauen an das Aufgabengebiet gebunden. Ein Teilnehmer, der beispielsweise gute Empfehlungen zu Kinos abgibt, muss nicht automatisch auch gute Restaurant-Empfehlungen abgeben.

Die wesentliche Eigenschaft von Vertrauen besteht darin, dass es nicht auf eine vollständige Sammlung von Fakten angewiesen ist. Eine umfassende Faktensammlung ist in koordinatorfreien Systemen nur sehr schwer bzw. teuer zu beschaffen, beispielsweise indem ein Teilnehmer alle anderen abfragt. Fakten über das Verhalten eines Peers in der Zukunft stehen prinzi-

piell nicht zur Verfügung. Das Konzept 'Vertrauen' ermöglicht es nun, auf Basis von wenigen unvollständigen Daten und allgemeinen Beobachtungen praxisrelevante Entscheidungen zu treffen. Dabei besteht jedoch immer das Risiko, dass das gegebene Vertrauen enttäuscht wird, bzw. das ein scheinbar vertrauenswürdiger Peer Anfragen nicht korrekt verarbeitet. Mathematisch lässt sich Vertrauen als eine Wahrscheinlichkeit dafür abbilden, dass eine Transaktion erfolgreich ausgeführt wird.

In P2P-Systemen basiert Vertrauen zumeist[19] auf den Beobachtungen über das Verhalten eines Peers in der Vergangenheit. Dies ist problematisch: in P2P-Systemen kann es häufig passieren, dass Peers neu hinzukommen oder das System verlassen. Die Peers können ihr Verhalten rasch ändern, wodurch alte Beobachtungen ihre Aussagekraft verlieren. Des weiteren ist die Zahl der Peers im System sehr groß. Aus diesen Gründen ist es für einen einzelnen Peer nur schwer möglich, eine für eine zuverlässige Abschätzung der Vertrauenswürdigkeit ausreichende Anzahl von Beobachtungen über alle seine Transaktionspartner zu sammeln. Das Konzept der Reputation (bzw. der 'soziale Status') ermöglicht nun reputationsbasierte Vertrauenssysteme (bzw. kurz 'Reputationssysteme' genannt), die Beobachtungen über das Verhalten eines Peers vielen angeschlossenen Teilnehmern rasch zugänglich machen. Eine formale Definition von Reputation und eine Abgrenzung zu Vertrauen aus der Perspektive von Multi-Agenten-Systemen bietet [CBG02]. Gemäß [CBG02] wird Reputation im Rahmen dieser Arbeit wie folgt definiert:

> *Reputation ist die gemeinsame Meinung einer Gemeinschaft von Peers über die Fähigkeiten und die Bereitschaft eines Peers, eine bestimmte Transaktion korrekt zu bearbeiten.*

Die Definition enthält, dass die Reputation das Aggregat aus vielen Beobachtungen einer Gemeinschaft von Peers ist. Eine Reputationsinformation ist dabei allen Mitgliedern der Gemeinschaft zugänglich, d.h., jeder Peer verfügt über die gleichen Reputationsinformationen. Diese Tatsache impliziert, dass Reputation in einer generalisierten und normalisierten Art und Weise zu verwalten ist, d.h., die Peers müssen sich auf einen gemeinsamen Maßstab für die Reputation verständigen. Ebenso wie Vertrauen sind auch Reputationsinformationen stets an ein bestimmtes Aufgabengebiet gebunden.

2.7 P2P-Reputationssysteme

Eine allgemeine Einführung zu Reputationssystemen gibt [RKZF00]. Reputationssysteme in P2P-Systemen nehmen häufig Anleihen bei Verfahren der Spieltheorie, bei Bayes-Netzen oder bei Methoden aus dem Bereich des Collaborative Filtering.

[19] Hier sollen einfache, statische Verfahren nicht berücksichtigt werden, in denen ein Peer grundsätzlich allen anderen vertraut oder misstraut.

Spieltheorie Die Interaktion in P2P-Systemen kann aus Sicht der Spieltheorie als ein Anwendungsfall des Gefangenendilemmas beschrieben werden, welches als eines der grundlegenden Modelle der Spieltheorie seit langem gut bekannt und untersucht ist [Axe84]. Im Gefangenendilemma versuchen zwei rationale Spieler (Untersuchungshäftlinge), ihren Profit ohne Rücksicht auf den anderen zu maximieren. Jeder Spieler hat die Wahl, den anderen entweder durch ein Geständnis zu verraten, oder mit ihm durch Verweigern der Aussage zu kooperieren. Aus der Perspektive einer P2P-Datenstruktur bedeutet 'kooperieren', dass ein Peer eine Anfrage korrekt verarbeitet; 'gestehen' bedeutet das Gegenteil. Tabelle 2.3 zeigt ein Beispiel[20] für die Auszahlungen, die die Spieler bei den vier möglichen Ausgängen des Spiels erhalten. Das globale Optimum (der in der Summe höchste Auszahlungsbetrag) wird erreicht, wenn beide Teilnehmer kooperieren. Für den einzelnen Teilnehmer ist es jedoch rational, den anderen durch ein Geständnis zu verraten.

	Spieler A kooperiert	Spieler A gesteht
Spieler B kooperiert	A: 3, B: 3	A: 5, B: 0
Spieler B gesteht	A: 0, B: 5	A: 1, B: 1

Tabelle 2.3: Beispiel für eine Auszahlungsmatrix beim Gefangenendilemma.

Das ursprüngliche Gefangenendilemma sieht nur ein einziges Spiel vor. Die wiederholten Interaktionen zwischen gleichen Teilnehmern in einem P2P-System bildet das Iterierte Gefangenendilemma (engl. *Iterated Prisoner's Dilemma* [Axe84]) nach. Hier können Strategien eruiert werden, bei denen ein Spieler auf das Verhalten des anderen in der Vergangenheit reagiert. Als wirksam hat sich dabei die Strategie 'Tit-for-Tat' [Axe84] herausgestellt, bei der ein Spieler in der ersten Runde kooperiert und in den Folgerunden das selbe tut wie sein Gegenspieler in der vorangegangenen Runde: hat der Gegenspieler in der Runde zuvor kooperiert, so kooperiert der Spieler in der aktuellen Runde. Wenn sein Gegenspieler gestanden hatte, gesteht der Spieler nun ebenso. Im Bezug auf P2P-Datenstrukturen kommt es nun darauf an, Strategien und Umgebungsbedingungen zu schaffen, in denen das einzige Nash-Gleichgewicht [Nas51] im Spiel darin besteht, dass alle Spieler in jeder Runde miteinander kooperieren [BAS03].

In P2P-Datenstrukturen interagieren jedoch nicht zwei Spieler miteinander, sondern ein Teilnehmer kann sich seinen Interaktionspartner aus einer Menge seiner Kontakte wählen. Das 'offene Gefangenendilemma' (engl. *Disclosed Prisoner's Dilemma* [PS05]) bildet diesen Fall nach. Das 'räumliche Gefangenendilemma' (engl. *Spatial Prisoner's Dilemma* [SBM02]) berücksichtigt, dass die Teilnehmerwahl in P2P-Datenstrukturen räumlich (zum Beispiel durch Nachbarschaftsbeziehungen im Schlüsselraum) eingeschränkt sein kann. Spieltheoretische Modelle sind meist sehr stark vereinfacht. Zu den Annahmen der Spieltheorie, die sich nur schwer mit P2P-Datenstrukturen vereinbaren lassen, zählen wiederholte Interaktionen zwischen den gleichen Peers und für alle Peers identische und zeitlich konstante Kosten und

[20] Die Auszahlungen sind hier als positive Geldbeträge zu verstehen.

Nutzen. Die Spieltheorie eignet sich jedoch gut zum Evaluieren neuer Strategien, z.B. mit genetischen Algorithmen [DY95].

Bayes'sche Netze Diese Grundlage für Reputationssysteme kommt aus der Graphentheorie bzw. stammt vom maschinellem Lernen ab. Ein Bayes'sches Netz [WV03] ist ein Graph, in dem die Knoten die Dichtefunktion für die Ausprägungen eines bestimmten Merkmals enthalten, und die Kanten die bedingten Abhängigkeiten zwischen diesen Merkmalen repräsentieren. In einem Reputationssystem für P2P-Netze würde jeder Teilnehmer ein eigenes Bayes-Netzwerk für jeden seiner Kontakte verwalten. Der Wurzelknoten des Bayes-Netzes enthält dabei die Merkmale $V = 1$ (vertrauenswürdig) und $V = 0$ (nicht vertrauenswürdig). Die Wahrscheinlichkeit für $T = 1$ entspricht dabei dem Anteil der Interaktionen, die zufriedenstellend verlaufen sind, d.h., bei denen sich der Peer als vertrauenswürdig erwiesen hat. Die Wahrscheinlichkeit ist somit ein Maß für das Vertrauen, das diesem Peer *insgesamt* entgegengebracht wird. Für einen Peer Π wird die Wahrscheinlichkeit wie folgt bestimmt:

$$\begin{aligned} p_\Pi(V=1) &= \frac{\text{Zahl der zufriedenstellenden Interaktionen mit } \Pi}{\text{Zahl aller Interaktionen mit } \Pi} \\ &= 1 - p_\Pi(V=0) \end{aligned} \tag{2.4}$$

Die Knoten unterhalb der Wurzel des Bayes'sche Netzes enthalten Merkmale wie die Art der Interaktion oder die Qualität der zurückgelieferten Ergebnisse, und repräsentieren die Vertrauenswürdigkeit des Peers in Abhängigkeit unterschiedlicher Merkmalsausprägungen. Diese werden dabei in jedem Knoten in einer Tabelle mit bedingten Wahrscheinlichkeiten ('Conditional Probability Table', CPT) für Ausprägungen dieses Merkmals verwaltet.

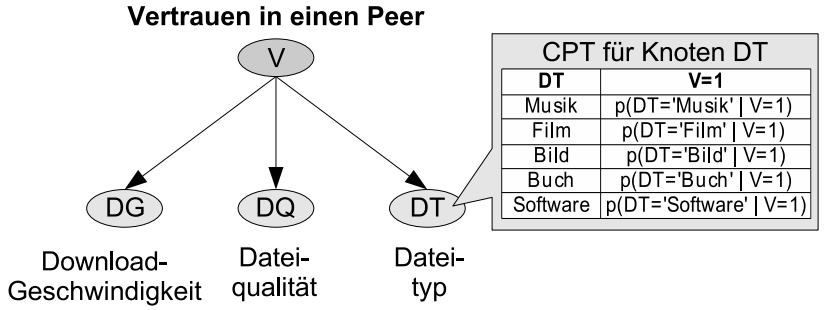

Abbildung 2.11: Beispiel für ein Bayes-Netzwerk.

Beispiel: Abbildung 2.11 zeigt ein Bayes-Netzwerk, mit dem ein Peer in einer P2P-Dateitauschbörse das Maß an Vertrauen in einen anderen bestimmt. Der Peer differenziert dabei zwischen der Download-Geschwindigkeit, der Dateiqualität und dem Dateityp. Beispielhaft ist die Tabelle mit den bedingten Wahrscheinlichkeiten für die unterschiedlichen Dateitypen dargestellt. Eine Wahrscheinlichkeit $p(DT{=}\text{'Musik'}|V{=}1) = 0{,}2$ würde beispielsweise bedeuten,

dass jede fünfte von diesem Peer heruntergeladene Datei ein zufriedenstellendes Musikstück enthält.

Mit Hilfe dieser Bayes-Netze kann ein Peer nun Vertrauenswürdigkeit eines anderen in Abhängigkeit verschiedenster Merkmalsausprägungen bestimmen, z.B.:

$$p_{\Pi}(DT = \text{'Musik'}, V = 1) = \frac{\text{Musik von } \Pi \text{ heruntergeladen und zufrieden gewesen}}{\text{Zahl aller Interaktionen mit } \Pi} \quad (2.5)$$

Auch komplexere Wahrscheinlichkeiten lassen sich auf diesem Wege bestimmen. So gibt der Term $p(DT = \text{'Musik'}, DG = \text{'100kBit'}, DQ = \text{gut}|V = 1)$ die Wahrscheinlichkeit dafür an, dass von einem Peer eine Musikdatei mit 100kBit in guter Qualität heruntergeladen werden kann. Einen Einstieg zum Einsatz von Bayes-Netzwerken in P2P-Vertrauenssystemen bietet [WV03].

Collaborative Filtering Damit nicht jeder Peer die gleichen schlechten Erfahrungen mit den selben unkooperativen Teilnehmern machen muss, können Methoden des Collaborative Filtering eingesetzt werden. Diese basieren darauf, die Reputation eines Peers auf der Grundlage einer Vielzahl von Meinungen anderer Peers zu bestimmen. Die Berücksichtigung der Meinungen von Dritten ist schwierig, da unehrliche Teilnehmer gefälschte Informationen abgeben können. Des weiteren müssen sich die Teilnehmer auf ein gemeinsames Maß einigen: ein Peer könnte beispielsweise mit der Leistung eines anderen noch zufrieden sein, während andere Peers bereits negative Bewertungen verfassen.

Das klassische Collaborative Filtering ist eine Methode, um gemeinsame Präferenzen aus den Meinungen einer großen Zahl von unterschiedlichen Teilnehmern zu extrahieren. Ein prominentes Beispiel für angewandtes Collaborative Filtering ist Amazon[21]. Hier werden die Buchkäufe eines Teilnehmers genutzt, um andere Käufer zu finden, welche die gleichen Bücher gekauft haben. Anhand der Bücher, die die anderen Käufer darüber hinaus erworben haben, werden nun Vorschläge generiert, welche Bücher dem Teilnehmer ebenfalls gefallen könnten.

In [ZMM99] wird ein Ansatz für das Reputationsmanagement beschrieben, der Anleihen beim Collaborative Filtering macht. Der Ansatz geht davon aus, dass Peers mit einer hohen Reputation auch Bewertungen über andere Peers mit einer guten Reputation machen können, die zu einem zuverlässigen Vertrauenswert zusammengefasst werden können. Im Gegensatz zu klassischen Einsatzgebieten des Collaborative Filtering ist in P2P-Datenstrukturen zu berücksichtigen, dass es keine zentrale Datenbasis für die Bewertungen der Teilnehmer gibt. Die Bewertungen sind stattdessen über alle Teilnehmer im System verteilt gespeichert. Deswegen wird wie beim Web-of-Trust Ansatz [KR97] das Reputationssystem als ein gerichteter Graph aufgefasst, dessen Knoten die Teilnehmer der Datenstruktur bilden. Die Reputationsinformationen stellen die Kanten zwischen diesen Knoten dar, und sind mit dem Vertrauenswert gewichtet. Jede Transaktion zwischen zwei Knoten bildet eine neue Kante, bzw. ersetzt eine eventuell

[21] http://www.amazon.de

bereits vorhandene Kante zwischen den Teilnehmern. Will nun ein Peer den Vertrauenswert eines Knotens bestimmen, so sucht er mittels Breitensuche alle Ketten von Bewertungen zwischen dem fraglichen Knoten und sich selbst bis zu einer gegebenen Maximallänge. Aus jeder Kette berechnet der Peer nun einen Hilfswert für das Maß an Vertrauen zwischen sich und dem gesuchten Knoten, und bestimmt aus allen Hilfswerten zusammen einen personalisierten Vertrauenswert für diesen Knoten. Die konkrete Ausgestaltung dieser Berechnungen ist vom Anwendungsszenario abhängig und kann beispielsweise die Länge der Vertrauensketten oder das Alter der Kanten berücksichtigen.

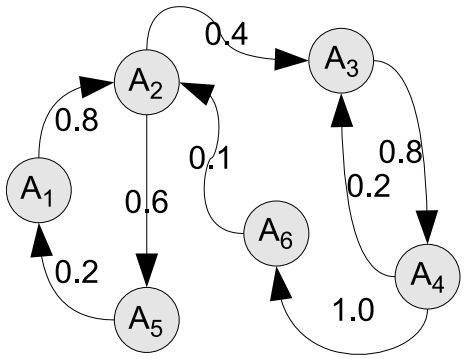

Abbildung 2.12: Beispiel für ein verteiltes Reputationsnetzwerk.

Beispiel: Abbildung 2.12 zeigt ein Vertrauensnetzwerk mit 6 Knoten. Wenn der Peer A_3 den Vertrauenswert für A_4 abruft, so sucht das Reputationssystem mittels Breitensuche nach allen Pfaden von A_4 nach A_3, und findet $A_4 \rightarrow A_3$ und $A_4 \rightarrow A_6 \rightarrow A_2 \rightarrow A_3$.

Beispiele für Reputationssysteme in P2P-Datenstrukturen Jede der aufgeführten Ansätze hat ihre Stärken und Schwächen. Um ein in der Praxis taugliches Reputationssystem zu entwickeln, ist darum eine Kombination dieser Techniken erforderlich.

[AD01] ist einer der ersten Ansätze für ein Reputationssystem in P2P-Datenstrukturen. Der Ansatz basiert auf Beschwerden (d.h., negativem Feedback), das die Peers bei nicht zufriedenstellenden Transaktionsausgängen generieren. Die Menge des negativen Feedbacks ist somit ein Maß für die Unzuverlässigkeit eines Peers. Das Feedback wird in einem P-Grid gespeichert, das ein globales Repository für alle Bewertungen über jeden einzelnen Knoten zur Verfügung stellt. Jeder Peer darf aus diesem Repository zu lesen, und jeder vertrauenswürdige Peer hat das Recht, neue Beschwerden einzutragen. Problematisch an diesem Ansatz ist, dass er von unveränderlichen Identitäten ausgeht. Ein Peer, der seine Identität wechseln kann, entfernt auf diesem Wege die über ihn gespeicherten Beschwerden. Des weiteren erschweren einmal im Repository gespeicherte Beschwerden die Rehabilitation vormals nicht vertrauenswürdiger Knoten schwierig. Darüber hinaus gibt es etliche Möglichkeiten, das Repository anzugreifen: Teilnehmer können beispielsweise Denial-of-Service Angriffe auf den Knoten durchführen,

der für die Beschwerden über sie verantwortlich ist, oder ihn mit einer großen Zahl gefälschter Beschwerden überlasten.

PeerTrust [XL04] verfolgt einen anderen Ansatz: Hier generiert ein Peer numerisches Feedback über jeden anderen Teilnehmer. Dieses Feedback wird ebenfalls in einer P2P-Datenstruktur gespeichert, und setzt sich aus folgenden Einzelwerten zusammen:

- die Zahl aller Transaktionen mit diesem Peer
- wie zufriedenstellend jede Transaktion mit diesem Peer verlaufen ist (*Amount of Satisfaction*)
- die Glaubwürdigkeit des von diesem Peers abgegebenen Feedbacks (*Credibility*)
- der Wert bzw. die Wichtigkeit der Transaktion (*Transaction Context*)
- ein Faktor, der sich aus der Meinung anderer Peers herleitet (*Community Context*)

Zu den Vorzügen dieses Ansatzes zählt die feingranulare Bestimmung der Vertrauenswerte, die die Begleitumstände berücksichtigt, unter denen das Feedback erzeugt wurde. Allerdings ist das Speichern einer vollständigen Transaktionshistorie über viele Peers sehr ressourcenintensiv, und die bei dem vorangegangenen Ansatz genannten Angriffspunkte gegen ein zentrales Repository bestehen weiterhin.

Ein spieltheoretisch fundiertes Reputationssystem bietet [FLSC04]. Dabei wird ein für alle Peers zugreifbares Log (*Shared History*) verwendet, das als P2P-Datenstruktur realisiert ist, und in dem sämtliche Transaktionen aller Peers aufgezeichnet sind. Eine Strategie im spieltheoretischen Sinne besteht in diesem Ansatz aus einer Entscheidungsfunktion, den Log-Einträgen, einer Regel zum Auswahl geeigneter Transaktionspartner und einer Regel zum Umgang mit unbekannten Teilnehmern. Die Entscheidungsfunktion bestimmt anhand eines Reputationswertes, ob ein Peer mit einem anderen kooperiert. Der Reputationswert wird dabei als Summe der angebotenen Dienste geteilt durch die Summe der konsumierten Dienste bestimmt. Es kann diskutiert werden, ob ein spieltheoretischer Ansatz wirklich als Reputationssystem bezeichnet werden kann – die Entscheidungsfunktion in [FLSC04] ist jedoch mit Reputationssystemen vergleichbar.

In [BB04a] wird ein Bayes-Ansatz verfolgt, um ein Reputationssystem zu realisieren, bei dem die Knoten bei jeder Interaktion untereinander auch Beobachtungen über das Verhalten von anderen austauschen. Dabei unterscheidet [BB04a] zwischen der Reputation eines Peers in Bezug auf seine Fähigkeiten (*Reputation Rating*), und der Glaubwürdigkeit der von ihm weitergegebenen Beobachtungen (*Trust Rating*). Der Reputationswert wird aus den lokalen Beobachtungen des Peers selbst, sowie aus den mit der Glaubwürdigkeit gewichteten Beobachtungen anderer Peers gemäß dem Bayes-Theorem ermittelt. Die Glaubwürdigkeit eines Peers wird bestimmt, indem die eigenen Beobachtungen mit den vom anderen Peer übermittelten Beobachtungen verglichen werden. Zu den größten Vorteilen dieses Ansatzes zählt die Integration von Beobachtungen aus zweiter Hand, ohne dass dabei auf ein zentrales Repository zurückgegriffen wird. Weiterhin gelingt es dem Ansatz, durch die Trennung zwischen Reputation und Glaubwürdigkeit mit gefälschten Informationen von anderen Peers umzugehen. Problematisch ist hingegen, dass sich der Bayes-Ansatz nicht gut an dynamische Verhaltensänderungen der Teilnehmer anpassen lässt – [BB04a] löst dieses Problem nur unzureichend über periodisches Verringern von alten Reputationswerten nach vorgegebenen Zeitspannen.

Anforderungen an P2P-Reputationssysteme Nachdem die Grundlagen von Reputationssystemen sowie die Vor- und Nachteile einiger Ansätze beispielhaft diskutiert wurden, ist es nun möglich, eine systematische Aufstellung aller Anforderungen an ein Reputationssystem für P2P-Datenstrukturen anzugeben. Einige der Anforderungen sind allgemeiner Natur, es sind jedoch auch Aspekte zu beachten, die für diese Datenstrukturen spezifisch sind.

Nash-Gleichgewicht bei global optimalem Verhalten. Das Nash-Gleichgewicht [Nas51] ist ein Begriff aus der Spieltheorie, der einen Systemzustand beschreibt, in dem sich kein Teilnehmer verbessern kann, wenn er allein sein Verhalten ändert. In herkömmlichen P2P-Systemen existiert nur ein Nash-Gleichgewicht, und zwar genau dann, wenn sich alle Teilnehmer unkooperativ verhalten. Jeder einzelne Teilnehmer, der in einem solchen Szenario auf kooperatives Verhalten umstellt, wird nur seine Kosten erhöhen, aber nicht seinen Nutzen. Das Ziel eines Protokolls zum Umgang mit unkooperativen Peers muss also darin bestehen, das P2P-System so zu verändern, dass das *einzige* Nash-Gleichgewicht für den Fall existiert, dass sämtliche Peers vollständig kooperativ sind.

Gute Differenzierung zwischen kooperativen und unkooperativen Peers. Eine für jedes Reputationssystem essentielle Eigenschaft ist eine möglichst gute Unterscheidung zwischen kooperativen und unkooperativen Teilnehmern. Dabei ist zu berücksichtigen, dass Peers ihr Verhalten an das Protokoll anpassen, wenn sie bei erkanntem unkooperativem Verhalten mit Nachteilen beaufschlagt werden. Beispielsweise könnte in einer P2P-Datenstruktur ein unkooperativer Peer einen kleinen Teil der eingehenden Nachrichten korrekt verarbeiten, wenn er dafür von anderen Peers nicht mehr als unkooperativ erkannt würde. Eine andere Art der Anpassung könnte darin besteht, nur die Nachrichten von Peers zu beantworten, die über eine sehr gute Bewertung verfügen, und dabei darauf zu spekulieren, dass negative Aussagen von Peers mit einem mäßigen 'Ruf' nur gering gewichtet werden. Im Kontext dieser Arbeit kommt der Differenzierung zwischen Peers mit unterschiedlichem Verhalten eine besonders große Bedeutung zu, da das hier vorgestellte Protokoll eine Bestrafung unkooperativer Peers vorsieht.

Einhaltung des Peer-to-Peer-Paradigmas. Eine P2P-spezifische Anforderung an das Protokoll besteht darin, dass die gewünschten Eigenschaften der P2P-Datenstruktur erhalten bleiben müssen (siehe Tabelle 2.1). Das Protokoll muss auf vollständig dezentralen Algorithmen basieren und ohne eine zentrale Instanz auskommen, die das Netz kontrollieren könnte und einen zentralen Angriffspunkt bietet. Es darf darüber hinaus nicht auf globalen, unveränderlichen Identitäten basieren oder globales Wissen erfordern. Das bedeutet, dass das Protokoll ausschließlich auf den lokalen Beobachtungen und den Interaktionen zwischen im Schlüsselraum benachbarten Peers aufsetzen darf.

Geringe zusätzliche Transaktionskosten. Eine P2P-Datenstruktur wird üblicherweise dazu genutzt, sehr viele kleine Anfragen zu verarbeiten. Ein reputationsbasiertes Protokoll, das bei jeder weiterzuleitenden Anfrage zur Bestimmung des geeignetsten Empfängers herangezogen wird, muss daher mit einem geringen Ressourcenbedarf hinsichtlich Netzwerkbandbreite, Speicherkapazität und CPU-Zeit auskommen. Dabei sind zwei Aspekte zu beachten:

- Das Reputationssystem muss so gut skalieren wie die darauf aufbauende P2P-Datenstruktur. Anderenfalls wird das Reputationssystem bei steigender Nutzerzahl, Transaktionsvolumen oder Transaktionshäufigkeit zum Flaschenhals.
- Das Reputationssystem muss im Vergleich zur Transaktion vernachlässigbar geringe Kosten verursachen. Ansonsten könnten sich einzelne kooperative Peers besser stellen, wenn sie das Reputationsmanagement anderen Peers überlassen, bzw. wäre eine weitere Komponente erforderlich, die die Teilnahme aller Peers am Reputationssystem überwacht.

Weil die Peers in P2P-Systemen zur Datenhaltung autonom operieren und von unterschiedlichsten Parteien betrieben werden, können ohnehin höchstens stochastische 'Garantien' für Antwortzeit und Verfügbarkeit gegeben werden. Daher ist es sinnvoll, einen (geringen) Anteil von durch unkooperative Teilnehmer fehlgeschlagene Transaktionen in Kauf zu nehmen, wenn im Gegenzug die Kosten des Protokolls auf ein angemessenes Maß[22] reduziert werden können.

Automatische Verarbeitung von Reputationsdaten. In einer P2P-Datenstruktur mit sehr vielen kleinen Transaktionen ist es sehr wichtig, dass das Reputationssystem ohne Eingriffe des Benutzers auskommt. Das Reputationssystem muss also auf jedem Peer automatisch Beobachtungen vornehmen, diese mit den Kontakten des Peers austauschen, daraus das Vertrauen in die einzelnen Kontakte errechnen und das Verhalten des Peers entsprechend adaptieren.

Robustheit gegenüber Systemfehlern und Angriffen. Wesentlich für die Funktionsfähigkeit eines Reputationssystems für Peer-to-Peer-Datenstrukturen ist nicht nur, dass es auf jedem Peer möglichst fehlerfrei und unangreifbar arbeitet. Wichtiger noch ist es, dass beim Auftreten eines Fehlers oder bei einem erfolgreichen Angriff nur jeweils ein Peer korrumpiert wird, und nicht das gesamte Reputationssystem. Dies impliziert, dass jeder Peer eingehende Informationen auf Plausibilität überprüft, und stets überwacht, ob die Reputationsinformationen, die von einem Peer übermittelt werden, zu richtigen oder falschen Entscheidungen führen.

Jedes Reputationssystem, das auf Daten aus externen Quellen angewiesen ist, lässt sich mit genügend Aufwand erfolgreich angreifen, indem eine ausreichende Anzahl von Knoten falsche (aber in sich konsistente) Informationen verbreiten. Beispielsweise könnte eine Gruppe von zusammenarbeitenden Knoten einem anderen Peer fortlaufend erfolgreiche Transaktionen attestieren. Ein in der Praxis anwendbares Reputationssystem muss darum so entworfen sein, dass der Aufwand, um es zu überlisten, höher ist als der Aufwand, um die Anfragelast kooperativ und protokollgerecht zu verarbeiten.

Umgang mit dynamischem Verhalten. Peers können ihr Verhalten jederzeit und beliebig oft ändern. Es ist Bestandteil der Aufgaben des Reputationssystems, auf diese Änderungen möglichst schnell und adäquat zu reagieren. Insbesondere muss das Reputationssystem

[22] Angemessen sind beispielsweise 10% Mehrkosten durch das FairNet-Protokoll; vgl. Abschnitt 3.5.

verhindern, dass unkooperative Peers über einen langen Zeitraum von einer vormals guten Reputation profitieren. Die Abbildung dynamischen Verhaltens erfordert also, dass sämtliche Reputationsinformationen möglichst rasch durch neuere Informationen verdrängt werden.

Niedrige Kosten für Neueinsteiger. Die Integration von Neueinsteigern in Reputationssysteme ist problematisch. Einerseits ist eine möglichst niedrige Einstiegshürde für Neueinsteiger wünschenswert. Andererseits müssen Neueinsteiger mit den Reputationswerten für vollkommen unkooperative Peers beginnen [FR98]. Anderenfalls könnten sich unkooperative Knoten verbessern, indem sie sich ab- und unter einer anderen Identität wieder im System anmelden.

Anwendbarkeit in der Praxis. Es ist unvermeidbar, bei der Analyse und der Evaluierung des Protokolls einige Vereinfachungen vorzunehmen, da die Abbildung sämtlicher Parameter zu schwer handhabbaren Modellen führt. Dennoch ist sicherzustellen, dass das beschriebene Protokoll unter praxisgerechten Bedingungen wie gewünscht arbeitet. Die wesentlichen Bedingungen sind:

- Peers verwalten unterschiedlich große Datenmengen und Schlüsselräume.
- Einige Peers haben mehr Last als andere.
- Die verfügbaren Ressourcen der Peers differieren.
- Die Anfragen sind über den Schlüsselraum ungleich verteilt, und werden von den Peers mit unterschiedlicher Rate abgesetzt.

Die Umsetzung der hier vorgestellten Anforderungen an ein Reputationssystem in Peer-to-Peer-Datenstrukturen ist eine Herausforderung. Die meisten Anforderungen sind untereinander stark verwoben, bzw. stehen im Konflikt zueinander. Beispielsweise erschwert die Forderung nach geringem Overhead und automatischer Verarbeitung die Anforderung nach Robustheit gegenüber Angriffen und Fehlern. Weiterhin hängen die Anforderungen vom Profil der auf die P2P-Datenstruktur zugreifenden Anwendung ab. Ein praktisch nutzbares Protokoll auf Basis eines Reputationssystems kann also nur ein Kompromiss zwischen den verschiedenen gewünschten Anforderungen sein. Daraus folgt, dass das Protokoll auf einem Kostenmodell aufsetzen muss, dass das Profil der eingesetzten Anwendungen reflektiert. Darauf aufbauend muss die Möglichkeit bestehen, das Verhalten des Gesamtsystems durch Parametrisierung an die Einsatzumgebung maßzuschneidern.

Kapitel 3

FairNet: Diskriminierung unkooperativer Peers

Dieses Kapitel beschreibt *FairNet*, ein Protokoll zum Umgang mit unkooperativen Peers. Das Kapitel ist in drei große Abschnitte untergliedert. Im ersten Teil wird das FairNet-Protokoll mit seinen Datenstrukturen und Methoden beschrieben. Im zweiten Teil des Kapitels folgt eine analytische und experimentelle Evaluierung des Protokolls. Im dritten Teil wird ermittelt, wie robust FairNet gegenüber Angriffen ist, und ob es sich auch in abgewandelten Implementierungen oder in unterschiedlichen P2P-Datenstrukturen effektiv einsetzen lässt. Die ersten beiden Teile basieren dabei auf [BB04b, BB06].

FairNet soll das kooperative Verhalten der Peers für alle Arten von Operationen (Anfrage, Einfügen, Löschen etc.) sicherstellen. Für eine vereinfachte Präsentation werden im folgenden ausschließlich *Anfragen* betrachtet. FairNet kann aber für alle Arten von DHT-Operationen eingesetzt werden, bei denen der Absender der Operation das Ergebnis verifizieren kann. Der Anfrageverarbeitung in FairNet wird der folgende Ablauf zugrunde gelegt:

1. Der Anfragesteller generiert eine Anfrage, die mit einem Schlüssel adressiert ist.
2. Kann ein Peer eine Anfrage nicht aus seiner Zone beantworten, sendet er sie an einen *vertrauenswürdigen* Kontakt weiter, dessen Zone dem Anfrageschlüssel am nächsten ist.
3. Der Peer, in dessen Zone der Anfrageschlüssel passt, sendet die Antwort auf direktem Weg zum Anfragesteller zurück.
4. Erhält der Anfragesteller eine korrekte Antwort, so generiert er positives Feedback über den Peer, an den er die Anfrage zuvor weitergeleitet hatte, und sendet eine positive Feedback-Benachrichtigung an ihn. Erhält der Anfragesteller keine Antwort, erzeugt er negatives Feedback und sendet eine negative Feedback-Benachrichtigung.
5. Jeder Peer, der eine Feedback-Benachrichtigung erhält, erzeugt Feedback über den Peer, an den er vorher die Anfrage weitergeleitet hatte, und sendet die Feedback-Benachrichtigung an diesen weiter.

Das FairNet-Protokoll lässt sich wie folgt zusammenfassen: Die Peers leiten Anfragen nur an vertrauenswürdige Knoten weiter. Erhält ein Peer eine Information darüber, dass der Knoten,

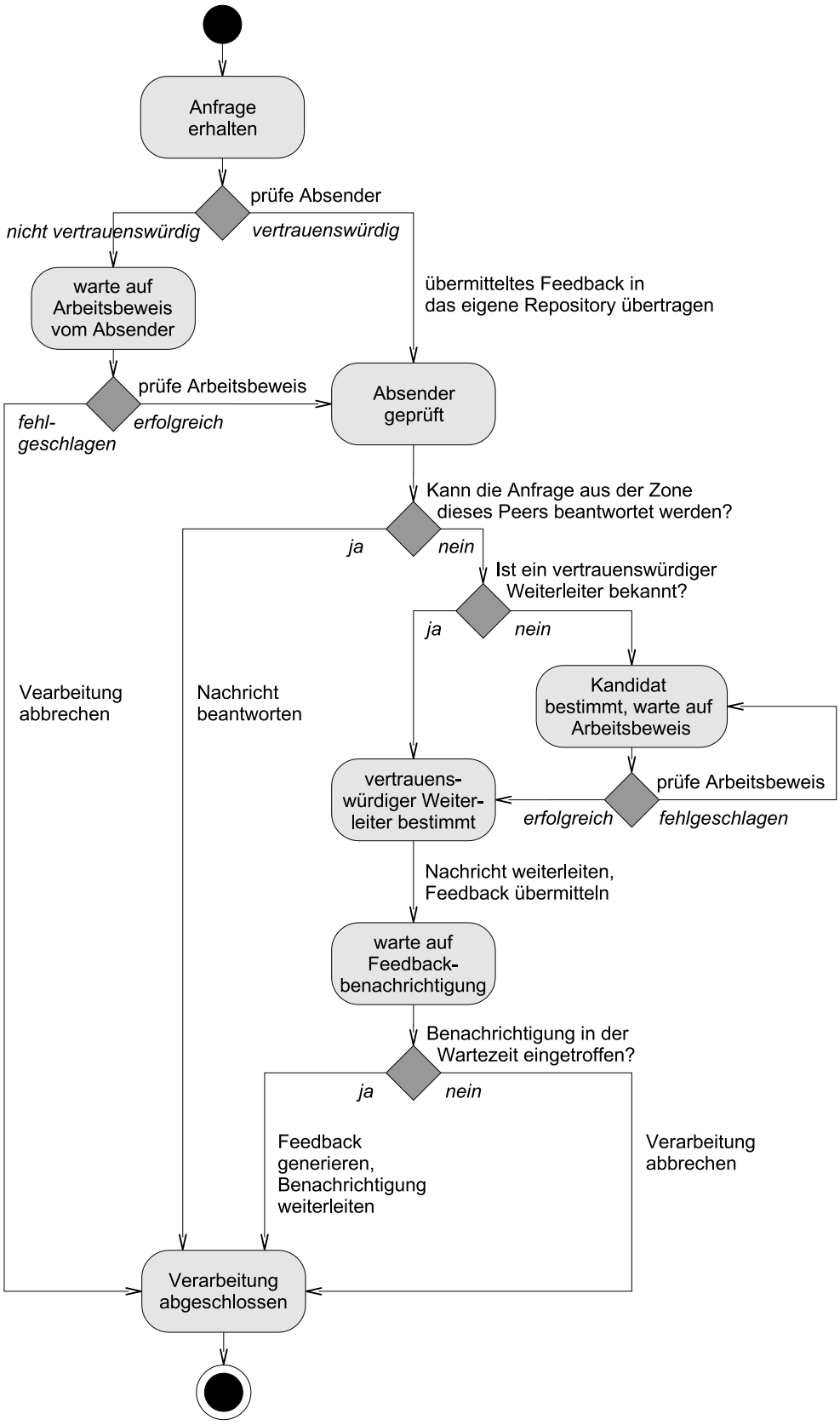

Abbildung 3.1: Zustandsdiagramm der Anfrageverarbeitung in FairNet.

an den er selbst weitergeleitet hat, ebenfalls ordentlich gearbeitet hat, generiert er positives Feedback und gibt es an die Nachbarn dieses Peers weiter. Bei einer Information über eine nicht korrekte Bearbeitung einer Anfrage wird negatives Feedback generiert und weitergegeben. Abbildung 3.1 zeigt die Abläufe in FairNet anhand der Zustände[1], die ein Peer bei der Verarbeitung einer Anfrage annehmen kann. Mit welchen Datenstrukturen und Methoden lässt sich dieses Protokoll nun als verteilte Implementierung realisieren, und wie effektiv ist es gegenüber unkooperativen Peers?

3.1 Die Datenstrukturen von FairNet

Das Protokoll zum Umgang mit unkooperativen Teilnehmern erfordert drei neue Datenstrukturen: *Feedback-Repositories*, *Feedback-Objekte* und *Anfrage-Logs*.

Feedback-Objekt Diese Objekte repräsentieren die Beobachtungen der Peers über das Verhalten ihrer Nachbarn. Es existieren sowohl *positive* als auch *negative Feedback-Objekte*. Die Feedback-Objekte enthalten einen Verweis auf den Peer, den sie bewerten (das *Feedback-Subjekt*) und sind mit einem Zeitstempel markiert, der den Zeitpunkt[2] der Erstellung angibt. Alle Feedback-Objekte haben das gleiche Gewicht; es gibt also keine Feedback-Objekte, die wichtiger oder aussagekräftiger sind als andere. Feedback-Objekte können global eindeutig über die ID des Erzeugers und eine vom Erzeuger vergebene fortlaufende Nummer identifiziert werden.

Feedback-Repository Jeder Peer besitzt ein lokales Feedback-Repository zur Verwaltung der von ihm selbst generierten bzw. von anderen Peers erhaltenen Feedback-Objekte. Die Zahl der positiven Objekte bestimmt das Vertrauen, dass der Repository-Inhaber dem Feedback-Subjekt entgegenbringt. Die Kapazität des Repositories ist beschränkt: Für jedes Feedback-Subjekt sind maximal s Objekte im Repository enthalten. Enthält das Repository nach einer Einfügeoperation mehr als s Feedback-Objekte über ein Feedback-Subjekt, so wird jeweils das älteste dieser Objekte verdrängt.

Anfrage-Log Neben dem Repository verfügt jeder Peer über ein lokal gespeichertes Anfrage-Log. In diesem Log vermerkt ein Peer, wann und an welchen Peer er eine bestimmte Anfrage weitergeleitet hat. Anfragen lassen dabei sich global eindeutig über die ID des Absenders und eine vom Absender vergebene fortlaufende Anfrage-ID identifizieren. Das Anfrage-Log dient zum Einen dazu, bei Erhalt einer Benachrichtigung über einen erfolgreichen (oder fehlgeschlagenen) Ausgang einer Transaktion festzustellen, welcher Peer eine Anfrage (nicht) protokollgerecht bearbeitet hat. Zum Anderen ermöglicht das Anfrage-Log, beim Wiederholen einer fehlgeschlagenen Anfrage diejenigen Peers zu

[1] Das Zustandsdiagramm wurde auf das Wesentliche reduziert; es fehlen Abbruchzustände, wenn kein vertrauenswürdiger Weiterleiter ermittelt werden kann oder keine Feedback-Benachrichtigung eintrifft.

[2] Hier wird eine globale Zeit benötigt, z.B. die mitteleuropäische Zeit (engl. Central European Time, CET). Die Zeit dient nur dazu, Feedback-Objekte ihrer Erstellreihenfolge nach zu sortieren, besondere Anforderungen an die Synchronität der Zeit bei unterschiedlichen Peers bestehen nicht.

meiden, die diese bereits beim beim letzten Versuch nicht erfolgreich beantwortet oder weitergeleitet haben.

Neben diesen neuen Datenstrukturen wird im folgenden auch auf die bereits in der ursprünglichen Beschreibung des CAN (s. [RFH+01]) eingeführten Datenstrukturen Bezug genommen. Dies sind insbesondere die *Schlüsselwerte* im Schlüsselraum, die *(Schlüssel,Wert)-Paare*, sowie die *Kontaktliste*, in der jeder Peer die Netzwerkadresse der ihm bekannten Peers sowie deren Zone im Schlüsselraum speichert.

3.2 Überblick über das Protokoll

Das Prinzip des Protokolls besteht darin, dass zwei Knoten nur dann miteinander kooperieren, wenn sie sich *gegenseitig* als kooperativ einschätzen. So wird ein Knoten P_1 eine Anfrage nur dann an einen Nachbarknoten P_2 weiterleiten, wenn P_1 Peer P_2 als kooperativ bewertet. Auf der anderen Seite wird P_2 diese Anfrage auch nur dann beantworten oder weiterleiten, wenn P_2 den Knoten P_1 für kooperativ hält.

Ein Knoten P_1 wird einen anderen Knoten P_2 genau dann als kooperativ einschätzen, wenn das Repository von P_1 mindestens eine Anzahl von t *positiven* Feedback-Objekten mit Subjekt P_2 enthält. Es liegt somit im Interesse eines Knoten, dass seine Nachbarn genügend positives Feedback über ihn besitzen. Das hier vorgestellte Protokoll bietet drei Alternativen dafür, dass ein Knoten P_2 positives Feedback über einen Nachbarn P_1 erhält:

R1 P_1 beantwortet eine Anfrage, die P_2 an ihn übermittelt hat, und P_2 erhält eine Benachrichtigung darüber, dass die Anfrage beantwortet wurde.

R2 P_1 erbringt einen Arbeitsbeweis, der von P_2 verlangt wurde. Hier kann P_2 unmittelbar beobachten, dass P_1 Arbeit geleistet hat.

R3 Die Nachbarn von P_1 gleichen ihr Feedback untereinander ab, und P_2 erfährt so davon, dass P_1 kooperative Arbeit für andere Peers geleistet hat.

Ebenso existieren auch drei Möglichkeiten dafür, dass P_2 negatives Feedback über P_1 erhält:

R4 P_2 übermittelt eine Anfrage an P_1, die P_1 nicht beantwortet oder weiterleitet, und P_2 erhält eine Benachrichtigung darüber.

R5 P_1 leitet eine Anfrage von P_2 weiter, die einer der folgenden Weiterleiter nicht protokollgemäß verarbeitet, und P_2 erhält eine Benachrichtigung über eine unbeantwortete Anfrage.

R6 Nachbarn von P_1 tauschen ihr Feedback untereinander aus, und P_2 erhält auf diesem Wege negatives Feedback über P_1.

Generierung von Feedback Um eine unkomplizierte Verwaltung der Feedback-Objekte in den Repositories zu ermöglichen, gehen sämtliche Feedback-Objekte mit der gleichen Gewichtung in die Berechnung der Vertrauenswürdigkeit des Feedback-Subjekts ein. Allerdings sind die mit Feedback bedachten Operationen unterschiedlich 'teuer'. Beispielsweise benötigt

das Weiterleiten verglichen mit einem Arbeitsbeweis viel weniger Ressourcen. Des weiteren ist es möglicherweise für eine effektive Disziplinierung der Peers wünschenswert, unkooperatives Verhalten mit mehr negativem Feedback zu bestrafen als kooperatives Verhalten zu belohnen.

Um diesen Sachverhalt abzubilden, wird nicht bei jedem Auftreten von **R1**, **R2**, **R4** oder **R5** ein Feedback-Objekt generiert. Stattdessen wird ein Feedback-Objekt mit einer Wahrscheinlichkeit erzeugt, die von der beobachteten Aktion abhängt. Die Wahrscheinlichkeiten repräsentieren also indirekt unterschiedliche Feedback-Gewichte. Wenn beispielsweise **R4** als doppelt so wichtig angesehen würde wie **R1**, bekäme **R4** auch eine doppelt so hohe Wahrscheinlichkeit für das Erzeugen von Feedback zugewiesen; folglich wäre auch doppelt so viel Feedback im Umlauf, welches **R4** zur Ursache hätte. Im hier vorgestelltem Protokoll erfolgt die Zuweisung der Gewichte für die einzelnen Feedback-Ursachen für alle Peers identisch zur Startzeit[3]. Die Ermittlung der Gewichte in Abhängigkeit der Umgebungsparameter der P2P-Datenstruktur ist ein Optimierungsproblem, das in Abschnitt 3.4 beschrieben und evaluiert wird.

Feedback-Benachrichtigungen Peer P_2 kann ausschließlich **R2** direkt beobachten. Bei **R1**, **R4** und **R5** löst eine *Feedback-Benachrichtigung* über eine erfolgreich oder nicht erfolgreich durchgeführte Anfrage die Erzeugung von Feedback aus. Feedback-Benachrichtigungen werden wie folgt abgearbeitet: Jeder Peer verwaltet ein lokales Anfrage-Log. Wenn ein Peer eine Anfrage stellt oder weiterleitet, wird der Knoten im Log vermerkt, der die Anfrage als nächstes erhält. Wenn der Knoten, der die Anfrage stellte, ein korrektes Anfrageergebnis erhält, so liest er zunächst den Peer aus seinem Anfrage-Log, an den er die Nachricht weitergeleitet hatte. Über diesen Peer schreibt der Knoten nun mit der dafür vorgesehenen Wahrscheinlichkeit positives Feedback in sein lokales Repository. Des weiteren erzeugt er eine positive Feedback-Benachrichtigung, und leitet sie an diesen Peer weiter. Bekommt ein Peer so eine Feedback-Benachrichtigung, liest er ebenfalls den nächsten Empfänger aus seinem Anfrage-Log, generiert positives Feedback über ihn und leitet die Benachrichtigung weiter, bis der Beantworter der Anfrage erreicht ist. Mit Hilfe der Anfrage-Logs der Weiterleiter einer Nachricht vollzieht die Benachrichtigung also den Weg der Anfrage exakt nach.

Eine negative Feedback-Benachrichtigung wird versendet, wenn (1) nach Ablauf einer Wartezeit noch kein Anfrageergebnis eingetroffen ist, oder (2) das übermittelte Anfrageergebnis ungültig ist. Hier wird jeweils über den nächsten Peer in der Kette der Weiterleiter negatives Feedback generiert; ansonsten wird die Feedback-Benachrichtigung wie beschrieben gehandhabt.

Zunächst erscheint **R5** dabei ungerecht: hier wird nicht nur der Verursacher einer fehlgeschlagenen Anfrage mit negativem Feedback bestraft, sondern auch alle Peers, die die Anfrage zuvor an ihn in kooperativer Weise weitergeleitet haben. Die Begründung für **R5** liegt darin, dass ein einzelner Peer nicht bestimmen kann, welcher Teilnehmer sich unkooperativ verhalten hat. So kann ein Teilnehmer unter mehreren Identitäten im Netz registriert sein [Dou02],

[3]Die Herausforderungen, die sich durch Peers mit individuellen Kostenstrukturen und adaptivem Verhalten (Self-Tuning) ergeben, werden derzeit im Rahmen einer Kooperation mit der Universität Karlsruhe eruiert.

oder Gruppen von Teilnehmern können koordiniert Angriffe durchführen [ZGG+04]. Nur **R5** stellt sicher, dass – neben allen anderen Weiterleitern einer Nachricht – auch stets der Verursacher einer fehlgeschlagenen Anfrage negatives Feedback erhält. Über Peers, die mehrfach unkooperativ handeln, ist mehr negatives Feedback im Umlauf als über vollständig kooperative Knoten, die nur gelegentlich an diesen Peer weitergeleitet haben. Abschnitt 3.4 wird zeigen, wie die Wahrscheinlichkeiten für die Erzeugung von Feedback, die Repository-Größe und der Schwellenwert beschaffen sein müssen, um mit diesem Mechanismus zwischen kooperativen und unkooperativen Peers unterscheiden zu können. Zuletzt motiviert **R5** die Peers auch dazu, sich nicht nur selbst kooperativ zu verhalten, sondern auch Nachrichten nur an erwiesen kooperative Teilnehmer weiterzuleiten.

Arbeitsbeweise Wenn P_2 weniger als t positive Feedback-Objekte mit Subjekt P_1 im Repository hat, schätzt dieser Peer P_1 als unkooperativ ein. Wenn P_2 nun eine Anfrage von P_1 erhält – sei es, um sie zu beantworten oder um sie weiterzuleiten – so verlangt P_2 von P_1 einen *Arbeitsbeweis*. Andererseits kann auch der Peer P_1 von P_2 einen Arbeitsbeweis verlangen, falls er eine Nachricht an P_2 weiterleiten muss und diesen Peer für unkooperativ hält.

Ein Arbeitsbeweis ist eine Aufgabe, die leicht zu formulieren ist, und dessen Lösung leicht überprüft werden kann. Die Lösung selbst ist jedoch nur schwer bzw. unter hohem Ressourceneinsatz zu finden [JJ99, Bac02, ABHL03]. Hat P_1 einen Arbeitsbeweis für P_2 erfolgreich erbracht, so schreibt P_2 (wiederum mit einer bestimmten Wahrscheinlichkeit) eine Anzahl von positivem Feedback in sein lokales Repository. Für einen Peer ist nun es weniger ressourcenintensiv, sich über einen längeren Zeitraum kooperativ und protokollgerecht zu verhalten, als auch nur einen einzigen Arbeitsbeweis zu erbringen.

Um die Zahl der Arbeitsbeweise zu verringern, welche die Peers erbringen müssen, tauschen benachbarte Knoten ihr Feedback untereinander aus (**R3**, **R6**). In FairNet wird dazu Feedback an Nachrichten angehängt, die die Peers ohnehin versenden (siehe Abschnitt 4). Da Feedback-Objekte nur wenige Bytes groß sind, können die Kosten für den Feedback-Austausch daher vernachlässigt werden.

Grundsätzlich tragen Arbeitsbeweise wie z.B. die Primzahlzerlegung großer Zahlen nichts zur Anfrageverarbeitung in der P2P-Datenstruktur bei. Doch es gibt mehrere Gründe, warum sich ein Knoten nicht durch reguläre Arbeit eine positive Reputation erarbeiten kann, also indem er Nachrichten weiterleitet oder Anfragen beantwortet. Peers werden durch **R5** mit negativem Feedback bedacht, wenn sie an unkooperative Knoten weiterleiten. Es somit ist nicht im Interesse der Peers, Nachrichten 'auf Verdacht hin' an einen unbekannten oder gar unkooperativen Knoten zu übermitteln. Stattdessen muss ein Peer vorab prüfen können, ob ein Knoten kooperativ ist. Des weiteren muss diese Überprüfung schnell vonstatten gehen, um die Antwortzeiten bei der Anfrageverarbeitung nicht unnötig zu verlängern. Ein Peer kann einen Arbeitsbeweis in einer erheblich kürzeren Zeitspanne erbringen, als kooperative Mitarbeit bei vergleichbarem Ressourceneinsatz erfordert. Möchte ein von seinen Nachbarn als unkooperativ eingeschätzter Peer eine Anfrage stellen, so ist es für ihn in den meisten Anwendungsszenarien nicht sinnvoll, über einen längeren Zeitraum eine positive Reputation durch reguläre Arbeit zu erwerben.

Abschnitt 3.4 wird sich mit dem Zeitbedarfen von Arbeitsbeweisen und regulärer Arbeit auseinandersetzen.

Das hier vorgestellte Protokoll macht nur Aussagen zu den Kosten eines Arbeitsbeweises. Annahmen zur Implementierung werden explizit nicht gemacht. Es ist somit möglich, anstelle eines künstlichen Problems, das nichts mit den Zielsetzungen der P2P-Anwendung gemein hat, applikationsspezifische Probleme als Arbeitsbeweis zu verwenden. Angenommen, eine verteilte WWW-Suchmaschine wurde auf Basis einer P2P-Datenstruktur realisiert. Dann könnte beispielsweise ein Peer als Arbeitsbeweis fordern, dass ein Knoten eine bestimmte WWW-Domain indexiert. Diese Aufgabe ist schnell zu formulieren, anhand stichprobenhafter Tests auf zufällig ausgesuchte URLs innerhalb der Zieldomain leicht zu prüfen, und stellt für den Bearbeiter des Arbeitsbeweises eine ressourcenintensive Belastung dar.

Es verbleibt zu diskutieren, warum ein Peer einen Arbeitsbeweis erbringen sollte, nur um anschließend vielleicht nur eine Anfrage für einen fremden Knoten zu verarbeiten. Ein Neueinsteiger oder ein von anderen als unkooperativ eingeschätzter Peer, der Anfrageergebnisse erhalten will, kann das Beantworten eines Arbeitsbeweises in FairNet nicht vermeiden, sondern nur hinauszögern. Arbeitsbeweise kosten jedoch Zeit. Je eher ein Peer seine Arbeitsbeweise erbringt, desto weniger werden seine eigenen Anfragen durch zuvor zu leistende Beweise verzögert. Deswegen ist es für einen Peer anzuraten, auch Arbeitsbeweise im Kontext fremder Anfragen zu bearbeiten.

Die Bootstrap-Problematik Die Wahrscheinlichkeiten zur Erzeugung von Feedback so zu optimieren, dass unkooperative Knoten viele Arbeitsbeweise erbringen müssen, während von kooperativen Knoten kein solcher Beweis verlangt wird, ist ein lösbares Optimierungsproblem. Neueinsteiger, über die noch kein Feedback in Umlauf ist, werden jedoch mit Sicherheit einen Arbeitsbeweis erbringen müssen. Zunächst scheinen also Arbeitsbeweise ein Hindernis für neue Knoten zu sein, die in die P2P-Datenstruktur eintreten wollen (*Markteintrittsbarriere*, s. Abschnitt 3.4 für eine analytische Bestimmung).

Zunächst ist dabei zu berücksichtigen, dass P2P-Datenstrukturen nicht für Anwendungen geeignet sind, in denen die Peers sich häufig ab- und anmelden. Dies liegt daran, dass ein sich ab- oder anmeldender Peer stets seinen gesamten Datenbestand übertragen und alle seine Kontakte informieren muss. Des weiteren haben Untersuchungen ergeben, dass in offenen Systemen ohne überprüfbare Identitäten 'Eintrittsgebühren' zwingend erforderlich sind. In [FR98] wird dies formal an einem spieltheoretischen Modell auf Basis des Gefangenendilemmas nachgewiesen. Dabei wird bewiesen, dass die Strategie mit der höchsten Auszahlung diese Eintrittsgebühren als Kosten für den Wechsel der Identität beinhalten muss. In P2P-Systemen gibt es keine Instanz, die die Identität der Teilnehmer authentifizieren und ihr Verhalten überwachen kann. Folglich kann ein Teilnehmer jederzeit seine Reputation löschen, indem er aus dem System ausscheidet und unter einem neuen, unbekannten Pseudonym wieder eintritt. Unter diesen Umständen ist es unvermeidbar, Neuankömmlinge ebenso wie unkooperative Peers zu behandeln. Neuen Teilnehmern Vorteile zukommen zu lassen ermöglicht unkooperativen Peers, zunächst diese Vorteile zu konsumieren, sich dann unkooperativ zu verhalten bis das genutzte

Pseudonym durch negatives Feedback 'verbrannt' ist, und sich anschließend unter einer neuen Identität wieder am System anzumelden, um erneut als Neuankömmling aufzutreten.

Es gibt einige Möglichkeiten, den Eintritt neuer Peers im Netz zu unterstützen. In Abschnitt 2.2 wird beschrieben, wie ein neuer Peer von einem bereits im Netz präsenten Knoten eine Zone zugewiesen bekommt und sich bei Peers mit benachbarten Zonen in die Kontaktliste einträgt. Eine nahe liegende Erweiterung dieses Prozesses besteht darin, dem Neuankömmling nicht nur die zu verwaltende Zone und die Kontaktdaten seiner Nachbarn zu übertragen, sondern auch das Feedback über seine Nachbarn. Dies ist möglich unter der Annahme, dass ein Peer einen mit einem Arbeitsbeweis vergleichbaren Aufwand betreibt, wenn er einen Neueinsteiger in das P2P-System einbindet. Des weiteren ist es denkbar, die Identitäten der Peers mittels Public-Key Signaturen abgesichert in der P2P-Datenstruktur abzulegen. Da der Besitzer des privaten Schlüssels beweisen kann, dass eine Identität ihm gehört, könnte ein Peer dadurch seine Identität und damit auch seine Reputation behalten, wenn er das Netz verlässt und später erneut betritt. Weil dieser Algorithmus nur beweisen muss, dass einem Teilnehmer ein bestimmtes Pseudonym gehört, kann auf eine zentrale Zertifizierungsinstanz verzichtet werden.

3.3 Die Implementierung des Protokolls

Nun werden die Methoden beschrieben, die das Protokoll zum Umgang mit unkooperativen Teilnehmern realisieren. Die Methoden in Pseudocode dargestellt. Dabei wird mit *this* der Peer bezeichnet, der aktuell mit der Verarbeitung des Codes beschäftigt ist. Des weiteren gibt es keine nebenläufigen Anfragen, und Schlüsselwerte werden nur einmal abgefragt. Diese Einschränkungen sind jedoch nicht Bestandteil der für die Experimente in Abschnitt 3.5 verwendeten Java-Implementierung.

Methode query() Wenn ein Peer den Wert in Erfahrung bringen möchte, der einem bestimmten Schlüssel zugeordnet ist, ruft er die in Abbildung 3.2 dargestellte Methode *query()* auf, und übergibt ihr als Parameter den Punkt im Schlüsselraum. *Query* wiederum ruft die noch zu erläuternde Methode *handleQuery()* auf, und wartet auf eine Antwort. Wenn rechtzeitig vor Ablauf der Zeitspanne *timeout* eine korrekte Antwort eintrifft, veranlasst *query()* den Aufruf der Methode *handleFeedbackNotification()* mit einem positiven Argument, um das Erzeugen von positivem Feedback und das Absenden einer positiven Benachrichtigung zu veranlassen. Des weiteren reicht *query()* das Anfrageergebnis an die aufrufende Applikation weiter. Wird innerhalb der Zeitspanne *timeout* keine korrekte Antwort auf die Anfrage geliefert, so wird *handleFeedbackNotification()* mit negativem Argument aufgerufen und führt zur Erzeugung von negativem Feedback und einer negativen Feedback-Benachrichtigung.

Methode handleFeedbackNotification() Die in Abbildung 3.3 gezeigte Methode wird nach Abschluss einer Anfrageoperation aufgerufen, um Feedback zu generieren und die an der Anfrageverarbeitung beteiligten Peers zu informieren. Die Methode erhält als Parameter den An-

```
1  query(Point x) {
2      // sende die Anfrage ab
3      handleQuery(x, this, this);
4
5      // warte auf Anfrageergebnis
6      Result r := waitForAnswer(timeout);
7
8      if (korrektes Anfrageergebnis rechtzeitig erhalten) {
9          handleFeedbackNotification(x, Antwort erhalten, this);
10         return r;
11     } else {
12         handleFeedbackNotification(x, keine Antwort erhalten, this);
13         return ∅;
14     }
15 }
```

Abbildung 3.2: Methode *query*.

fragepunkt, den Typ der Benachrichtigung (je nach Transaktionsausgang positiv oder negativ) sowie den Peer, der die Benachrichtigung an den aktuellen Peer (*this*) verschickt hat.

```
1  handleFeedbackNotification(Point x, NotificationType n, LastForwarder f) {
2
3      // ist der letzte Weiterleiter der Benachrichtigung nicht vertrauenswürdig?
4      if (f ≠ this ∧ this.repository.getNumOfPosFb(f) < t) {
5
6          // beende die Bearbeitung der Benachrichtigung
7          return;
8      }
9      // hole den Peer aus dem Anfragelog, an den die Anfrage übermittelt wurde
10     Addressee a := this.queryLog.get(x);
11
12     if (a ≠ ∅) {
13         // erzeuge Feedback und sende die Benachrichtigung weiter
14         generateFeedback(a, n);
15         send(a, NotificationMessage(x, n));
16     }
17 }
```

Abbildung 3.3: Methode *handleFeedbackNotification*.

Zunächst überprüft *handleFeedbackNotification()* die Vertrauenswürdigkeit des Absenders der Feedback-Benachrichtigung. Benachrichtigungen von nicht vertrauenswürdigen Peers werden aus Sicherheitsgründen verworfen. Bearbeitet werden dagegen Benachrichtigungen, die der aktuelle Peer *this* selbst erstellt hat. Dieser Fall tritt ein, wenn *this* der Absender der ursprünglichen Anfrage war. Ebenfalls vertrauenswürdig sind Peers, über die der aktuelle Peer mindestens t positive Feedback-Objekte im lokalen Repository verzeichnet.

Nach erfolgreicher Prüfung des Absenders wird der Peer aus dem Anfrage-Log geholt, an den *this* die Anfrage weitergeleitet hatte. *handleFeedbackNotification()* veranlasst nun die Erzeugung von Feedback passend zum Typ der Benachrichtigung, und leitet die Benachrichtigung an den nächsten Peer weiter. Dieser wiederum führt ebenfalls *handleFeedbackNotification()* aus, und der Prozess wird fortgesetzt, bis ein Peer keinen nächsten Adressaten in seinem Anfrage-Log vorfindet. Dieser Fall tritt ein, wenn der Peer erreicht ist, der die Anfrage beantwortet hat oder sich unkooperativ verhielt.

Methode generateFeedback() Diese Methode generiert Feedback-Objekte und speichert sie im lokalen Repository des Peers *this*. Die in Abbildung 3.4 dargestellte Methode erhält dazu als Übergabeparameter das Feedback-Subjekt und die Ursache für das Erzeugen von Feedback. Zunächst bestimmt *generateFeedback()* aus dieser Ursache die Wahrscheinlichkeit, mit der das Feedback-Objekt erzeugt werden soll, sowie den Typ des Feedbacks (positiv oder negativ). Im nächsten Schritt wird daraus das Feedback-Objekt generiert, mit dem aktuellen Zeitstempel *currentTime* versehen, und in das lokale Repository geschrieben. Als letzte Handlung prüft *generateFeedback()* die Zahl der Feedback-Objekte im Repository, die den übergebenen Peer als Feedback-Subjekt haben. Befinden sich mehr als die Maximalzahl s solcher Objekte im Repository, so wird das älteste dieser Objekte entfernt.

Methode handleQuery() Diese Methode (Abbildung 3.5) ist dafür verantwortlich, eine Anfrage ausschließlich über vertrauenswürdige Peers vom Absender bis zum Beantworter weiterzuleiten, die Antwort zurückzusenden und Feedback zwischen den Peers auszutauschen. Als Parameter werden dabei der Anfragepunkt, der Peer von dem *this* die Anfrage erhalten hat, sowie der Absender der Anfrage an *handleQuery()* übergeben. Zunächst überprüft *handleQuery()* die Vertrauenswürdigkeit des Peers, der die Anfrage an den aktuellen Knoten übermittelt hat. Ist der aktuelle Peer *this* selbst der Anfragesteller, oder wurde die Anfrage von einem Peer weitergeleitet, über den mindestens t positive Feedback-Objekte im lokalen Repository vorhanden sind, so wird die Anfrage sofort bearbeitet. Anderenfalls muss der Weiterleiter einen Arbeitsbeweis erbringen. Ohne diesen Arbeitsbeweis wird die Anfrage verworfen.

Nach festgestellter Vertrauenswürdigkeit des Absenders prüft *handleQuery()*, ob der aktuelle Peer *this* die Anfrage selbst beantworten kann. Es wird also geprüft, ob der Anfragepunkt innerhalb der Zone liegt, für die *this* verantwortlich ist. In diesem Fall wird das Anfrageergebnis unmittelbar an den Absender der Anfrage gesendet, und die Anfrageverarbeitung ist abgeschlossen. Ist der aktuelle Peer nicht für die passende Zone zuständig, so ermittelt *handleQuery()* zunächst eine Menge von Kandidaten, an die die Anfrage gemäß *Greedy Forwarding* weitergeleitet werden darf, d.h., deren Zonen näher am Anfragepunkt liegen als die Zone von *this*. Dann wird aus der Menge der Kandidaten der *vertrauenswürdige* Knoten bestimmt, dessen Zone den geringsten Abstand zum Anfragepunkt aufweist. Dabei kann es vorkommen, dass kein Knoten existiert, der zugleich vertrauenswürdig ist und als Weiterleiter infrage kommt. Tritt dieser Fall ein, sortiert *handleQuery()* die Kandidaten nach dem Abstand ihrer Zonen zum Anfrageschlüssel. Beginnend mit dem Peer mit der kleinsten Distanz werden nun

```
1  generateFeedback(Peer a, NotificationType n) {
2      // bestimme die Wahrscheinlichkeiten
3      switch (n) {
4          case Arbeitsbeweis erhalten
5              FeedbackType t := true;
6              Wahrscheinlichkeit p := q_ProW;
7          case Anfrageergebnis erhalten
8              FeedbackType t := true;
9              Wahrscheinlichkeit p := q_answer;
10         case kein Anfrageergebnis erhalten
11             FeedbackType t := false;
12             Wahrscheinlichkeit p := q_answer · q_pn;
13         case Nachricht weitergeleitet
14             FeedbackType t := true;
15             Wahrscheinlichkeit p := q_forward;
16         case Nachricht nicht weitergeleitet
17             FeedbackType t := false;
18             Wahrscheinlichkeit p := q_forward · q_pn;
19     }
20     // erzeuge neues Feedback−Objekt mit Subjekt a und Typ t
21     Feedback f = new Feedback(a, t, currentTime);
22
23     // schreibe Feedback mit Wahrscheinlichkeit p ins lokale Repository;
24     this.repository.write(p, f);
25     while (this.repository.size > s) {
26
27         // entferne überzähliges Feedback über Subjekt a
28         this.repository.removeOldestFeedback(a);
29     }
30 }
```

Abbildung 3.4: Methode *generateFeedback*.

nacheinander Arbeitsbeweise von allen Kandidaten verlangt, bis einer der Peers den Arbeitsbeweis erbringt und als nächster kooperativer Empfänger der Anfrage bestimmt werden kann.

Im nächsten Schritt bestimmt *handleQuery()* die Menge der Feedback-Objekte, die als Attachment zusammen mit der Anfrage an den nächsten Peer übermittelt werden. Im Pseudocode ist eine sehr simple Strategie abgebildet, bei der ein Peer alle Feedback-Objekte über Knoten erhält, die mit ihm benachbart sind. Ausgefeiltere Strategien sind leicht realisierbar; beispielsweise könnte ein Peer stets die k aktuellsten Objekte übertragen und dabei sicherstellen, dass er seinem Nachbarn Feedback-Objekte nicht mehrfach übermittelt. Kapitel 4 geht auf diesen Punkt näher ein. Im letzten Schritt wird die Anfrage zusammen mit dem Feedback-Anhang an den designierten Peer übertragen und diese Weiterleitung im Anfragelog vermerkt.

Nachdem es textuell sowie mit Hilfe von Pseudocode spezifiziert wurde, soll das FairNet-Protokoll in den folgenden Abschnitten analytisch und experimentell untersucht werden. FairNet basiert ausschließlich auf Operationen, die die Peers auf ihrem lokalen Repository aus-

```
1  handleQuery(Point x, LastForwarder f, Issuer i) {
2      // ist der Absender der Anfrage kooperativ?
3      if (f ≠ this ∧ this.repository.getNumOfPosFb(f) < t) {
4
5          // nein; also verlange Arbeitsbeweis
6          requestProW(f);
7          waitForProWAnswer(timeout);
8          if (korrekter Arbeitsbeweis rechtzeitig erbracht) {
9              generateFeedback(f, Arbeitsbeweis erhalten);
10         } else {
11             // kein Arbeitsbeweis erbracht; beende Anfrageverarbeitung
12             return;
13         }
14     }
15     // kann dieser Peer die Anfrage beantworten?
16     if ( dist(this,x) = 0) {
17         send(i, new AnswerMessage(x,value(x)));
18         return;
19     }
20     // Peers ermitteln, an die die Anfrage gesendet werden könnte
21     CandidatePeers C := {p | dist(p,x) < dist(this,x) ∧ p ∈ this.contactList};
22
23     // bestimme kooperativen Peer, um Anfrage weiterzuleiten
24     ReliablePeers R := {p | this.repository.getNumOfPosFb(p) ≥ t ∧ p ∈ C};
25     if (R ≠ ∅) {
26         sortiere R gemäß dist(p ∈ R, x);
27         Addressee a := getFirstElement(C);
28     } else {
29         sortiere C gemäß dist(p ∈ C, x);
30         forall (p ∈ C) {
31             requestProW(p);
32             waitForProWAnswer(timeout);
33             if (korrekter Arbeitsbeweis rechtzeitig erbracht) {
34                 generateFeedback(p, Arbeitsbeweis erhalten);
35                 Addressee a := p;
36                 break;
37             }
38         }
39     }
40     // erzeuge Feedback−Anhang für die zu sendende Nachricht
41     FbAttachment F := {f | isNeighbor(a, f.subject) ∧ f ∈ this.repository };
42
43     // leite Nachricht einschließlich Feedback−Anhang weiter
44     send(a, new QueryMessage(x, i, F));
45     this.queryLog.add(x, a);
46 }
```

Abbildung 3.5: Methode *handleQuery*.

führen können, d.h., Peers konkurrieren nicht um gemeinsame Ressourcen. Des weiteren ist die Reihenfolge ohne Bedeutung, mit der Feedback-Objekte, Anfragen oder Antworten bei einem Peer eintreffen. Zudem warten die Peers auf Arbeitsbeweise oder auf Antworten stets nur eine begrenzte Zeitspanne. Deswegen kann auf eine Untersuchung von Problemen wie Deadlock-Freiheit, Nebenläufigkeit, oder 'fairer' Lastausgleich aus dem Forschungsbereich der verteilten Systeme verzichtet werden. Stattdessen soll der Schwerpunkt der Untersuchungen darauf gelegt werden, den Nachweis dafür zu erbringen, dass das Protokoll auch unter realistischen Umständen mit unkooperativen Teilnehmern umgehen kann.

3.4 Analyse des FairNet-Protokolls

Das FairNet-Protokoll zum Umgang mit unkooperativen Peers beinhaltet ein komplexes Optimierungsproblem. Das Ziel dieser Optimierung besteht darin, die zahlreichen Parameter des Protokolls so zu bestimmen, dass

- kooperativen Peers nur ein geringer, vorab definierter Zusatzaufwand entsteht,
- unkooperative Peers einen hohen Aufwand betreiben müssen, um an der P2P-Datenstruktur partizipieren zu können,
- das Protokoll schnell auf Änderungen im Verhalten der Peers reagiert, also unkooperativen Peers eine rasche Rehabilitation ermöglicht und kooperative Teilnehmer bei Fehlverhalten schnell als unkooperativ erkennt, und dass
- diese Eigenschaften auch unter wechselhaften Umgebungsbedingungen erhalten bleiben, beispielsweise bei einem stark schwankenden Anteil unkooperativer Peers oder einer veränderlichen Anzahl von Teilnehmern im System.

Um die Analyse zu vereinfachen, sollen zunächst einige Annahmen über die Eigenschaften der P2P-Datenstruktur gemacht werden:

Reguläre Zonenaufteilung Die Zonen aller Peers im CAN sind gleich groß, haben die gleiche Seitenlänge im Schlüsselraum und den gleichen Replikationsgrad r. Der Schlüsselraum eines CAN wird als ein Hyperwürfel mit der Seitenlänge 1 und d Dimensionen betrachtet, der n^d unterschiedliche Zonen enthält. Das bedeutet, die Kantenlänge einer Zone beträgt $1/n$ in jeder Dimension. Insgesamt gibt es $N = r \cdot n^d$ Peers im CAN. Abbildung 3.6 ist ein Beispiel für einen Schlüsselraum mit einer regulären Zonenaufteilung. In Abbildung 2.4 ist ein unregelmäßiges CAN dargestellt.

Anfrageverarbeitung in Runden Die *Runde* ist die in der Analyse verwendete Zeiteinheit. Im Durchschnitt stellt ein Peer in jeder Runde eine Anfrage, und beantwortet eine Anfrage eines anderen Peers. Anfragen, die wegen unkooperativer Peers nicht beantwortet wurden, gelten als verloren und werden nicht wiederholt. Neu generiertes Feedback erreicht alle Nachbarn des Feedback-Subjekts bis zum Beginn der nächsten Runde.

Gleichverteilte Anfragen Die abgefragten Schlüssel (Anfragepunkte) und die Peers, die diese Anfragen abgesendet haben, sind unabhängig und gleichverteilt über den Schlüssel-

raum. Es gibt also keine 'Hot Spots' in Form von beliebteren Schlüsselbereichen oder aktiveren Knoten.

Greedy Routing ist stets möglich Die Dimensionalität d und der Replikationsgrad r des CAN sind so groß, dass jeder Peer stets an einen kooperativen anderen Knoten weiterleiten kann. Dazu wird angenommen, dass die unkooperativen Peers unabhängig und mit Gleichverteilung über den Schlüsselraum verteilt sind. Es gibt also keine 'Inseln' von unkooperativen Knoten im Schlüsselraum, die die Weiterleitung von Nachrichten in eine bestimmte Richtung vollständig unterbinden können.

Unveränderliche Weglängen In der Praxis erhöht sich durch das Umgehen von unkooperativen Teilnehmern die Zahl der Peers, die eine Nachricht im Durchschnitt weiterleitet. In der Analyse soll diese Veränderung der Pfadlängen nicht berücksichtigt werden. Dies ist realistisch für P2P-Systeme mit einer geringen globalen Ausfallwahrscheinlichkeit.

Eingeschwungener Zustand Diese Annahme legt fest, dass sich das betrachtete System in einem stabilen Zustand befindet. Jeder Peer hat bereits die erforderlichen Arbeitsbeweise erbracht, um sich ins System zu integrieren, und die Repositories der Peers sind mit dem Feedback aus den vergangenen Runden gefüllt. In [GMSK03] wird beschrieben, wie dieser Zustand erreicht werden kann, und warum er keine Restriktion für die Analyse darstellt.

Rationale Peers Alle Peers im System sind risikoneutral. Es gilt die Relation *Kosten = Nutzen*, und die Kosten für den Arbeitsbeweis (c_{ProW}) sowie für das Beantworten (c_{answer}) oder Weiterleiten ($c_{forward}$) von Nachrichten sind für alle Peers identisch. Die Analyse wird zeigen, dass diese Annahme keine Einschränkung in der Praxis bedeutet, solange gilt $c_{forward} << c_{ProW}$ und $c_{answer} << c_{ProW}$.

Diese Vereinfachungen gelten nur für die Analyse. Die experimentelle Evaluierung in Abschnitt 3.5 wird nachweisen, dass die Annahmen der reguläre Zonenaufteilung, der gleichverteilten Anfragen und der Anfrageverarbeitung in Runden keine Einschränkungen in der Praxis nach sich ziehen. Das erste Experiment in Abschnitt 3.5 untersucht die Auswirkungen veränderlicher Pfadlängen. Kapitel 4 erklärt, auf welche Weise neu generiertes Feedback innerhalb einer Runde an die Nachbarn des Feedback-Subjekts verteilt werden kann. Der Abschnitt 3.6 erläutert, wie die (Schlüssel,Wert)-Paare so über den Schlüsselraum verteilt werden können, dass unkooperative Teilnehmer keine Datenverluste verursachen können. Des weiteren wird in Abschnitt 5.5 nachgewiesen, dass auch Kollaborationsangriffe die Wirksamkeit von FairNet nur wenig beeinträchtigen: es kann also auch in der Realität vorausgesetzt werden, dass Greedy Routing stets möglich ist. Am Ende dieses Abschnitts wird untersucht, wie teuer es für einen Neuankömmling ist, den in der vorletzten Annahme vorausgesetzten eingeschwungenen Zustand zu erreichen. Weiterhin wird die Forderung nach Rationalität der Peers in den Sektionen 3.6 und 5.1 explizit aufgehoben.

Protokollparameter Tabelle 3.1 zeigt die exogenen Parameter des Protokolls. Die angegebenen Vorgabewerte werden für Beispielrechnungen im Rahmen der Analyse sowie für die Experimente in Abschnitt 3.5 verwendet. Zu den exogenen Parametern zählen alle Angaben, welche die P2P-Datenstruktur extern vorgibt. Dazu gehörten Umgebungsparameter, die die Pfadlänge und die Zahl der Peers im System festlegen, sowie die globale und die lokale Ausfallwahrscheinlichkeit. Die *lokale Ausfallwahrscheinlichkeit* q gibt an, mit welcher Wahrscheinlichkeit ein unkooperativer Peer eine Nachricht nicht beantwortet bzw. nicht weiterleitet. Ein Peer mit $q = 0{,}2$ bearbeitet 80% der eingehenden Anfragen protokollgerecht, und verhält sich bei 20% der Nachrichten unkooperativ. Die *globale Ausfallwahrscheinlichkeit* p sagt aus, mit welcher Wahrscheinlichkeit eine Nachricht von *einem beliebigen* Knoten verworfen wird. Sie ist somit der Mittelwert über alle Ausfallwahrscheinlichkeiten im System. In der Analyse sollen Systemfehler nicht berücksichtigt werden. Die Ausfallwahrscheinlichkeit wird daher allein von der Anzahl der unkooperativen Teilnehmer und ihrer lokalen Ausfallwahrscheinlichkeit bestimmt.

Beispiel: Gegeben sei ein CAN mit 10.000 Teilnehmern. Ein Anteil von 500 Peers verhält sich unkooperativ und bearbeitet $q = 0{,}5$ aller Anfragen nicht. Dann ergibt sich eine globale Ausfallwahrscheinlichkeit von $p = 0{,}5 \cdot 500/10000 = 0{,}025$.

Parameter	**Symbol**	**Vorgabewert**
Dimensionalität des Schlüsselraums.	d	4
Zahl der Knoten pro Dimension.	n	10
Replikationsgrad der P2P-Datenstruktur.	r	1
Gesamtanzahl der Knoten.	N	$N = r \cdot n^d$
Globale Ausfallwahrscheinlichkeit für alle Knoten in der P2P-Datenstruktur.	p	$0 \cdots 1$
Lokale Ausfallwahrscheinlichkeit, mit der ein unkooperativer Knoten Nachrichten verwirft.	q	$0 \cdots 1$

Tabelle 3.1: Die exogenen Parameter der P2P-Datenstruktur.

Um geeignete Protokollparameter festzulegen, die es für die Peers ökonomisch unattraktiv machen, sich unkooperativ zu verhalten, ist ein Kostenmodell erforderlich. Tabelle 3.2 zeigt die Kosten des Modells, das Kostenmodell selbst wird in Gleichung 3.26 spezifiziert. Die in der Tabelle angegebenen Vorgabewerte für die Kosten dienen hier nur zur Veranschaulichung und werden in Abschnitt 3.5 näher erläutert.

Tabelle 3.3 enthält sämtliche Parameter, mit denen sich das Verhalten von FairNet einstellen lässt. Die höchstmögliche Anzahl der Feedback-Objekte im Repository über jeweils ein

Parameter	**Symbol**	**Vorgabewert**
Kosten zum Weiterleiten einer Nachricht.	$c_{forward}$	2
Kosten zum Beantworten einer Anfrage.	c_{answer}	5
Kosten für einen Arbeitsbeweis.	c_{ProW}	100

Tabelle 3.2: Die Parameter des Kostenmodells.

Subjekt legt der Parameter s fest. Ein großes s ermöglicht eine differenziertere Einschätzung der Kooperationsbereitschaft der Peers. Ein Repository der Größe $s = 1$ lässt beispielsweise nur zwei mögliche Aussagen zu: es enthält entweder ein positives Feedback-Objekt oder eben nicht. Auf der anderen Seite führt ein Repository mit hoher Kapazität zu einem ebenfalls hohen Verwaltungsaufwand und zu einem schlechteren dynamischen Verhalten durch später aus dem Repository verdrängte alte Feedback-Objekte. Der Schwellenwert t dient zur Differenzierung kooperativer und unkooperativer Peers anhand der Anzahl positiven Feedbacks im Repository.

Ebenfalls variabel sind die Wahrscheinlichkeiten, mit denen die Peers als Reaktion auf erbrachte Leistungen Feedback generieren. Um die Zahl der Freiheitsgrade des Protokolls zu reduzieren, wurde darauf verzichtet, separate Parameter für positive und negative Beobachtungen desselben Sachverhalts zu definieren. Stattdessen legt der Parameter q_{pn} fest, um welchen Faktor sich die Wahrscheinlichkeiten für positives Feedback von denen für negatives unterscheiden. Für eine nicht beantwortete Anfrage wird also $q_{answer} \cdot q_{pn}$ negatives Feedback generiert, für eine nicht weitergeleitete Nachricht $q_{forward} \cdot q_{pn}$.

Eine Einschränkung besteht für die Wahrscheinlichkeiten für Feedback über das Weiterleiten oder Beantworten von Anfragen: das Verhältnis dieser Wahrscheinlichkeiten muss dem Verhältnis der Kosten der Aktionen entsprechen. Es gilt also $c_{forward}/c_{answer} = q_{forward}/q_{answer}$. Anderenfalls kann sich ein unkooperativer Peer verbessern, indem er beispielsweise Nachrichten weiterleitet, aber nicht beantwortet, wenn das Weiterleiten ihn im Verhältnis weniger kostet bzw. mehr Feedback erbringt.

In FairNet wird jedem Peer zu Beginn ein fester Parametersatz gemäß Tabelle 3.3 zugewiesen. Das bedeutet, jeder Peer verfügt über die selben Schwellenwerte, Repository-Kapazitäten und Wahrscheinlichkeiten für die Feedback-Erzeugung. Experimente in Abschnitt 3.5 werden zeigen, dass diese Vorgehensweise keine Einschränkung darstellt. Erweiterungen zur Selbstoptimierung, in denen jeder Peer seine Parameter individuell auf Basis von Beobachtungen über seine Umwelt anpasst, sollen in einem Nachfolgeprojekt thematisiert werden.

Distanzen im CAN Im folgenden wird mit P_i ein Peer in einem Content-Addressable Network und mit $x_i = (x_{i1}, x_{i2}, \cdots, x_{id})$ ein beliebiger Punkt in der Zone von P_i bezeichnet. Gesucht wird nun die Zahl der Peers, die eine Nachricht in einem CAN mit d Dimensionen weiterleiten müssen, damit sie die Entfernung zwischen zwei Punkten $x_1 = (x_{11}, \cdots, x_{1d})$

Parameter	**Symbol**	**Vorgabewert**
Anzahl der Feedback-Objekte, die pro Feedback-Subjekt im Repository gehalten werden.	s	10
Schwellenwert zur Erkennung unkooperativer Teilnehmer. Ein Peer mit t oder mehr positiven Feedback-Objekten wird als kooperativ betrachtet.	t	6
Wahrscheinlichkeit, dass ein Feedback-Objekt wegen einer Nachrichtenweiterleitung generiert wird.	$q_{forward}$	0,2
Wahrscheinlichkeit, dass ein Feedback-Objekt für eine Antwort auf eine Anfrage erzeugt wird.	q_{answer}	0,5
Relation zwischen positivem und negativem Feedback.	q_{pn}	3
Anzahl der für einen Arbeitsbeweis erzeugten Feedback-Objekte.	q_{ProW}	1

Tabelle 3.3: Die Optimierungsparameter von FairNet.

und $x_2 = (x_{21}, \cdots, x_{2d})$; $x_1, x_2 \in [0;1]^d$ überbrückt. Gleichung 3.1 bestimmt dafür zunächst die *Entfernung in Zonen* zwischen x_1, x_2 in einer Dimension i.

$$celldist(i, x_1, x_2) = \min\big(\begin{array}{l} |\lfloor x_{1i} \cdot n \rfloor - \lfloor x_{2i} \cdot n \rfloor|, \\ 1 + \lfloor \min(x_{1i}, x_{2i}) \cdot n \rfloor + \lfloor n - \max(x_{1i}, x_{2i}) \cdot n \rfloor \big) \end{array} \tag{3.1}$$

Beim Greedy Forwarding im CAN ist der Schlüsselraum als Torus realisiert, und Nachrichten können auch über den 'Umbruch' im Schlüsselraum von 1 zu 0 hinweg transportiert werden. Weitergeleitet wird dabei stets über die kürzeste Entfernung. Gleichung 3.1 bestimmt daher das Minimum aus der direkten (euklidischen) Entfernung zwischen den zwei Punkten und aus der Entfernung über den 0,1-Umbruch im Schlüsselraum hinweg.

Beispiel: Abbildung 3.6 zeigt einen Schlüsselraum mit $d = 2$ und $n = 8$. Die Entfernung in Zonen zwischen beliebigen Punkten in den Zonen von P_1 und P_5 beträgt 4 in Dimension x und 3 in Dimension y.

Die absolute Zahl der Weiterleiter einer Nachricht zwischen x_1 und x_2 ist nach unserem Protokoll die L_∞-Entfernung (*Chessboard Distance*), d.h., das Maximum der Entfernungen in allen Dimensionen wie in Gleichung 3.2 beschrieben.

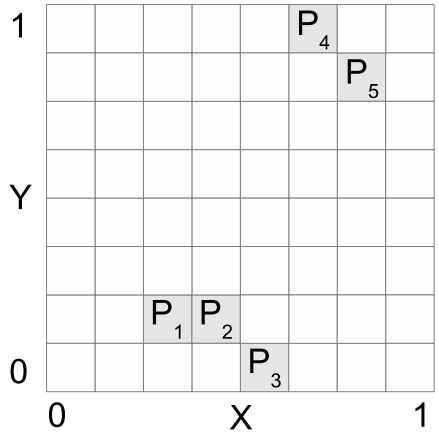

Abbildung 3.6: Entfernung im CAN bei einer regelmäßigen Aufteilung des Schlüsselraums.

$$dist(x_1, x_2) = \max_{i=1}^{d}(celldist(i, x_1, x_2)) \tag{3.2}$$

Diese Ermittlung der Entfernungen steht im Gegensatz zum ursprünglichen CAN-Entwurf: hier wird Manhattan-Entfernung verwendet, d.h., $dist_m(x_1, x_2) = \sum_{i=1}^{d} celldist(i, x_1, x_2)$. FairNet ist darauf angewiesen, dass in aufeinander folgenden Dimensionen zwei Nachbarn eines Peers ebenfalls benachbart sind, um untereinander Feedback über diesen Knoten auszutauschen. So sind in FairNet die Peers P_2 und P_3 in Abbildung 2.4 benachbart, und können direkt Feedback über P_1 austauschen. Im CAN-Entwurf in [RFH+01] wäre dies nicht möglich; Feedback müsste über einen vierten Peer P_6 übertragen werden. Als weiteren Vorteil reduziert die erhöhte Zahl der Nachbarn die mittlere Pfadlänge, weil Nachrichten nun orthogonal und diagonal (wie die Dame im Schachspiel) im CAN weitergeleitet werden können.

Unter Berücksichtigung der vorab festgelegten Gleichverteilung von Anfragepunkten und den Anfragestellern im Schlüsselraum ergibt sich die mittlere Pfadlänge δ aus dem arithmetischen Mittel der Distanzen zwischen allen Knoten im Schlüsselraum. Gleichung 3.3 bestimmt diese Entfernung mit Hilfe eines bestimmten Doppelintegrals über alle im Schlüsselraum enthaltene Punkte $x_1, x_2 \in [0; 1]^d$.

$$\delta = \iint_{x_1, x_2 \in [0;1]^d} dist(x_1, x_2) dx_1 dx_2 \tag{3.3}$$

Wenn die Zahl der Peers steigt, erhöht sich die Zahl der Zellen n pro Dimension. Dadurch steigt die ermittelte Pfadlänge δ, obwohl sich die Position der Punkte im Schlüsselraum nicht verändert hat. Weil die Zonen aller Peers die gleiche Größe haben, ist die mittlere Pfadlänge δ zugleich die Zahl der Anfragen, die ein Peer pro Runde weiterleiten oder beantworten muss.

Unbeantwortete Anfragen Gleichung 3.3 bestimmte die mittlere Pfadlänge in einem idealen CAN, in dem jede Anfrage beantwortet wird. Gesucht wird nun die mittlere Pfadlänge in einem *realistischen* Szenario. Sei p die globale Ausfallwahrscheinlichkeit über alle Knoten unter Einbeziehung aller Peers, die Anfragen nicht beantworten oder nicht weiterleiten. Aufgrund der Annahme, dass das CAN sich in einem eingeschwungenen Zustand befindet, wird p als konstant und zeitlich unveränderlich betrachtet. Im folgenden wird ein Peer als *kooperativ* betrachtet, wenn er eine lokale Ausfallwahrscheinlichkeit kleiner oder gleich p hat. Jeder Peer ist verpflichtet, Anfragen von anderen Knoten zu beantworten, wenn sie in an seine eigene Zone adressiert sind, und sie anderenfalls weiterzuleiten. Ein Peer, der eine Anfrage stellt, leitet diese mit Sicherheit weiter. Ein Weiterleiter reicht die Anfrage nun jeweils mit einer Wahrscheinlichkeit von $1-p$ an den nächsten durch. Erhält der für die mit dem Schlüssel adressierte Zone verantwortliche Peer die Anfrage, so beantwortet er sie wiederum mit $1-p$ Wahrscheinlichkeit.

In realistischen P2P-Datenstrukturen gilt $N >> \delta$, d.h., die Zahl der Peers im System ist erheblich größer als die Pfadlänge. Weiterhin besagt das Greedy Forwarding, dass Nachrichten stets in Richtung des Ziels und niemals zurück gesendet werden. Deswegen wird davon ausgegangen, dass die Wahrscheinlichkeiten für das Weiterleiten und Beantworten einer Anfrage unabhängig voneinander sind (im Urnenmodell: ziehen mit zurücklegen). Eine Nachricht wird daher über eine Entfernung i weitergeleitet und anschließend beantwortet mit Wahrscheinlichkeit $(1-p)^{i-1}\cdot(1-p)$. Die Zahl der Weiterleitungen $h_{forward}$ einer Nachricht in einem CAN mit einer globalen Ausfallwahrscheinlichkeit $p > 0$ ist nun genau die Summe der Wahrscheinlichkeiten, dass eine Nachricht von x_1 nach x_2 über $1, 2, \cdots, dist(x_1, x_2) - 1$ Peers erfolgreich weitergeleitet wurde. Die zu erwartende durchschnittliche Zahl der Weiterleitungen, die ein Peer pro Runde leisten muss, wird nun von Gleichung 3.4 ermittelt. Für ein CAN mit einer Ausfallwahrscheinlichkeit von $p > 0$ gilt also $h_{forward} < \delta$.

$$h_{forward} = \iint\limits_{x_1,x_2\in[0;1]^d} \sum_{i=1}^{dist(x_1,x_2)-1} (1-p)^{i-1} dx_1 dx_2 \tag{3.4}$$

Nach der Zahl der Weiterleitungen pro Runde soll nun die Zahl der von einem Peer zu beantwortenden Anfragen berechnet werden. Stellt ein Peer eine Anfrage, die er aus seinem eigenen Datenbereich beantworten kann ($dist(x_1, x_2) = 0$), so erhält er das Ergebnis mit einer Wahrscheinlichkeit von 1. Ebenfalls mit Sicherheit wird eine Anfrage übertragen, die an die Zone des unmittelbaren Nachbars des Anfragestellers adressiert ist. Peers, die i Weiterleiter vom Anfragesteller entfernt sind, erhalten die Anfrage mit einer Wahrscheinlichkeit von $(1-p)^{i-1}$. Dieser Sachverhalt wird von der Hilfsfunktion 3.5 abgebildet. Gleichung 3.6 ermittelt daraus

die durchschnittliche Zahl h_{answer} der Anfragen, die ein Peer pro Runde beantworten muss.

$$f_{\mathfrak{a}}(l) = \begin{cases} 1 & \text{wenn } l = 0 \\ (1-p)^{l-1} & \text{wenn } l > 0 \end{cases} \tag{3.5}$$

$$h_{answer} = \iint\limits_{x_1,x_2 \in [0;1]^d} f_{\mathfrak{a}}(dist(x_1,x_2))dx_1dx_2 \tag{3.6}$$

Feedback für das Beantworten von Anfragen In einem CAN mit einer globalen Ausfallwahrscheinlichkeit von $p > 0$ erhalten kooperative Peers sowohl positives als auch negatives Feedback. Die Hilfsfunktion 3.7 bestimmt die Wahrscheinlichkeit, mit der über einen Peer positives Feedback für das Beantworten von Nachrichten generiert wird, der l Zonen vom Schlüsselraum des Anfragestellers entfernt ist. Damit der Peer Feedback erhält, muss folgendes gelten:

1. Die Anfrage muss vom Anfragesteller bis zum passenden Peer weitergeleitet werden. Dies geschieht mit der Wahrscheinlichkeit $(1-p)^{l-1}$.
2. Die Anfrage muss beantwortet werden. Die Wahrscheinlichkeit dafür beträgt $1-p$.
3. Die Feedback-Benachrichtigung muss auf dem gleichen Pfad bis zum Vorgänger des Beantworters gelangen, also von einem Peer weniger weitergeleitet werden als die ursprüngliche Anfrage. Daher ist die Wahrscheinlichkeit dafür $(1-p)^{l-2}$.

Der unterste Term in Hilfsfunktion 3.7 bildet diesen Sachverhalt ab. Die beiden anderen Fallunterscheidungen behandeln zwei Sonderfälle: Kann ein Peer eine Anfrage direkt aus seinem lokalen Schlüsselraum beantworten ($dist(x_1,x_2) = 0$), wird kein Feedback generiert. Und wird eine Anfrage vom direkten Nachbarn des Anfragestellers beantwortet ($dist(x_1,x_2) = 1$), so wird sie mit Sicherheit übertragen, mit $1-p$ beantwortet, und der Anfragesteller generiert Feedback wieder mit Sicherheit; eine Benachrichtigung wird hingegen nicht versandt.

$$f_{\mathfrak{b}}(l) = \begin{cases} 0 & \text{wenn } l = 0 \\ 1 \cdot (1-p) & \text{wenn } l = 1 \\ (1-p)^{l-1} \cdot (1-p) \cdot (1-p)^{l-2} & \text{wenn } l > 1 \end{cases} \tag{3.7}$$

$$h_{answer,pos} = q_{answer} \cdot \iint\limits_{x_1,x_2 \in [0;1]^d} f_{\mathfrak{b}}(dist(x_1,x_2))dx_1dx_2 \tag{3.8}$$

Die Gleichung 3.8 bestimmt nun, wie viel positives Feedback ein kooperativer Peer im Durchschnitt für das Beantworten von Anfragen erhält. Wieder wird dazu ein bestimmtes Doppelintegral über alle möglichen Start- und Zielpunkte im Schlüsselraum benutzt. Um unterschiedliche Ursachen für das Erzeugen von Feedback auch unterschiedlich zu gewichten, sollen

Feedback-Objekte nur mit einer vorab bestimmten Wahrscheinlichkeit erzeugt werden (vgl. Abschnitt 3.2). Dazu wird die vom Doppelintegral ermittelte durchschnittliche Wahrscheinlichkeit, dass eine Anfrage erfolgreich beantwortet und die positive Benachrichtigung bis zum Vorgänger des Beantworters gelangt ist, mit q_{answer} multipliziert.

Die Gleichungen 3.9 und 3.10 basieren auf dem gleichen Prinzip. Die Hilfsfunktion 3.9 bestimmt die Wahrscheinlichkeit dafür, dass eine Nachricht bis zu einem Peer weitergeleitet wird, der sie diesmal *nicht* beantwortet, woraufhin eine negative Feedback-Benachrichtigung bis zum Vorgänger dieses Peers gelangt. Im Vergleich zu 3.7 ist dazu nur die Wahrscheinlichkeit $1-p$ für das Beantworten durch p für das Nicht-Beantworten zu ersetzen. Gleichung 3.10 bestimmt analog zu 3.8, wie viel negatives Feedback ein kooperativer Peer für unbeantwortete Anfragen erhält. Diese Anzahl setzt sich wieder zusammen aus den Wahrscheinlichkeiten für eine Weiterleitung der Anfrage an den Beantworter, das Nichtbeantworten, und die Wahrscheinlichkeit dafür, dass die Benachrichtigung den Vorgänger des Beantworters erreicht. Die Gewichtung für das Generieren von negativem Feedback für das Nicht-Beantworten beträgt $q_{answer} \cdot q_{pn}$.

$$f_{\mathfrak{c}}(l) = \begin{cases} 0 & \text{wenn } l = 0 \\ 1 \cdot p & \text{wenn } l = 1 \\ (1-p)^{l-1} \cdot p \cdot (1-p)^{l-2} & \text{wenn } l > 1 \end{cases} \tag{3.9}$$

$$h_{answer,neg} = q_{answer} \cdot q_{pn} \cdot \iint\limits_{x_1,x_2 \in [0;1]^d} f_{\mathfrak{c}}(dist(x_1,x_2))dx_1dx_2 \tag{3.10}$$

Feedback für das Weiterleiten von Anfragen Je mehr Peers an der Verarbeitung einer einzelnen Anfrage involviert sind, desto wichtiger ist es, eine geringe globale Ausfallwahrscheinlichkeit im CAN zu erreichen. Des weiteren verringert sich die Wahrscheinlichkeit exponentiell mit steigender Pfadlänge, positives Feedback für das Beantworten von Anfragen zu erhalten. Um es für die Peers ökonomisch rational zu machen, Anfragen von anderen Peers nicht nur zu beantworten, sondern auch korrekt weiterzuleiten, beinhaltet FairNet neben Feedback für das Beantworten auch Feedback für das Weiterleiten von Anfragen.

Eine Hilfsfunktion (3.11) ist dafür zuständig, die Zahl an Feedback-Benachrichtigungen zu ermitteln, die ein Peer in Entfernung l pro Runde erhält. Wird eine Anfrage von einem Peer in einer Entfernung $dist(x_1,x_2) < 2$ beantwortet, wird Feedback für das Antworten (Gleichung 3.8), aber nicht für das Weiterleiten generiert. Dies wird vom oberen Term in Gleichung 3.11 realisiert. Werden Anfragen an weiter entfernte Knoten gestellt, müssen für positives Feedback drei Bedingungen erfüllt sein:

1. Die Anfrage muss den Beantworter erreichen. Die Wahrscheinlichkeit dafür ist $(1-p)^{l-1}$ für eine Entfernung l.
2. Mit Wahrscheinlichkeit $1-p$ wurde die Anfrage beantwortet.
3. Die Feedback-Benachrichtigung hat den Vorgänger des Weiterleiters erreicht.

Ebenso wie die Anfrage selbst kann auch die Benachrichtigung von jedem Weiterleiter mit Wahrscheinlichkeit p verworfen werden. Eine Benachrichtigung erhält ein Peer mit Sicherheit, nachdem die Bedingungen (1) und (2) erfüllt wurden. Dies hat seine Ursache darin, dass ein Peer im Durchschnitt eine Anfrage pro Runde selbst absetzt und anhand des Anfrageergebnisses die Kooperationsbereitschaft des ersten Weiterleiters direkt beobachten kann. Dazu kommen so viele Benachrichtigungen, wie der der Summe der Wahrscheinlichkeiten entspricht, dass die Benachrichtigung über $1, 2, \cdots, (l-2)$ Schritte bis zum Vorgänger des Beantworters gelangt.

$$f_{\mathfrak{d}}(l) = \begin{cases} 0 & \text{wenn } l < 2 \\ (1-p)^{l-1} \cdot (1-p) \cdot \left(1 + \sum_{j=1}^{l-2} (1-p)^{j-1}\right) & \text{wenn } l \geq 2 \end{cases} \tag{3.11}$$

$$h_{forward,pos} = q_{forward} \cdot \iint\limits_{x_1,x_2 \in [0;1]^d} f_{\mathfrak{d}}(dist(x_1,x_2)) dx_1 dx_2 \tag{3.12}$$

Gleichung 3.12 integriert nun über alle Start- und Zielpunkte des Schlüsselraums, um die durchschnittliche Anzahl von Feedback-Benachrichtigungen pro Runde zu ermitteln. Diese Anzahl wird wiederum mit dem Gewicht $q_{forward}$ für das Generieren von positivem Feedback multipliziert, um aus der Zahl der Benachrichtigungen die Zahl der positiven Feedback-Objekte für das Weiterleiten $h_{forward,pos}$ zu ermitteln.

Die Anzahl negativen Feedbacks lässt sich ähnlich bestimmen. Hier muss die Funktion 3.13 nur berücksichtigen, dass eine der Weiterleitungen oder das Beantworten mit p fehlschlägt und nicht mit $1-p$ erfolgreich durchgeführt wird.

$$f_{\mathfrak{e}}(l) = \begin{cases} 0 & \text{wenn } l < 2 \\ (1-p)^{l-1} \cdot p \cdot \left(1 + \sum_{j=1}^{l-2} (1-p)^{j-1}\right) & \text{wenn } l \geq 2 \end{cases} \tag{3.13}$$

$$h_{forward,neg} = q_{forward} \cdot q_{pn} \cdot \iint\limits_{x_1,x_2 \in [0;1]^d} \sum_{i=1}^{dist(x_1,x_2)} f_{\mathfrak{e}}(i) \; dx_1 dx_2 \tag{3.14}$$

Ein wesentlicher Faktor kommt jedoch bei der Gleichung 3.14 für die durchschnittliche Anzahl negativen Feedbacks pro Runde für das Weiterleiten $h_{forward,neg}$ hinzu: hier ist zu berücksichtigen, dass eine Nachricht möglicherweise 'unterwegs' von einem beliebigen Weiterleiter in der Kette der involvierten Peers vom Anfrager zum Beantworter nicht verarbeitet wurde.

Beispiel: Eine Nachricht muss über eine Entfernung von $l = 8$ Peers weitergeleitet werden, um den Peer zu erreichen, der sie beantworten kann. Der erste Peer in der Kette der Weiterleiter erhält nun immer dann negatives Feedback, wenn er selbst, einer der 6 folgenden

Weiterleiter oder der Beantworter unkooperativ waren. Der letzte Weiterleiter erhält hingegen nur dann negatives Feedback, wenn er selbst oder der Beantworter unkooperativ waren, aber alle vorhergehenden Weiterleiter sowohl die Anfrage als auch die Feedback-Benachrichtigung korrekt verarbeitet haben.

Gleichung 3.14 trägt dieser Problematik Rechnung, indem sie für jeden möglichen Start- und Zielpunkt im Schlüsselraum jeweils die Summe der Wahrscheinlichkeiten für unkooperatives Verhalten an der 1., 2., ... l. Position in der Kette der Weiterleiter ermittelt. Bisher wurde nur das Feedback für das Beantworten oder Weiterleiten von Anfragen ermittelt. Da im folgenden nur auf die Summen des generierten positiven oder negativen Feedbacks Bezug genommen wird, kann Feedback für andere Operationen ohne Schwierigkeiten an dieser Stelle in das Modell integriert werden.

Häufigkeit von Arbeitsbeweisen Peers, über die zu wenig positives Feedback in Umlauf ist, müssen zum Nachweis ihrer Kooperationsbereitschaft Arbeitsbeweise erbringen. Nun soll ermittelt werden, wie viele Arbeitsbeweise ein Peer pro Runde im Durchschnitt erbringen muss, und wie viel Feedback $h_{ProW,pos}$ ein Peer dafür erhält. Die Gleichungen 3.15 und 3.16 beschreiben dafür zunächst, wie viel positives und negatives Feedback über einen kooperativen Peer pro Runde von seinen Nachbarn generiert wird.

$$\sigma_{pos} = \sum_{x \in \{forward, answer, ProW\}} h_{x,pos} \tag{3.15}$$

$$\sigma_{neg} = \sum_{x \in \{forward, answer, ProW\}} h_{x,neg} \tag{3.16}$$

Je nach Pfadlänge, Anfragen und Repository-Kapazität kann selbst während einer Runde mehr Feedback generiert werden, als die Repositories seiner Nachbarn aufnehmen kann. Dabei wird stets das älteste Feedback verdrängt. Das Verhältnis zwischen positivem und negativem Feedback bleibt also gewahrt. Die Wahrscheinlichkeit dafür, dass ein Feedback-Objekt im Repository eines Peers positiv ist, wird von Gleichung 3.17 ermittelt. Gleichung 3.18 bestimmt die Wahrscheinlichkeit für negatives Feedback.

$$p_{pos} = \frac{\sigma_{pos}}{\sigma_{pos} + \sigma_{neg}} \tag{3.17}$$

$$p_{neg} = \frac{\sigma_{neg}}{\sigma_{pos} + \sigma_{neg}} \tag{3.18}$$

Mit den Wahrscheinlichkeiten p_{pos} und p_{neg} lässt sich nun der Erwartungswert für die Zahl der positiven und negativen Feedback-Objekte in einem Repository der Größe s eines Nachbarn des Feedback-Subjekts ermitteln:

$$E(\text{positives Feedback im Repository}) = s \cdot p_{pos} \tag{3.19}$$
$$E(\text{negatives Feedback im Repository}) = s \cdot p_{neg} = s \cdot (1 - p_{pos}) \tag{3.20}$$

Weiterhin kann nun bestimmt werden, wie viele Arbeitsbeweise ein kooperativer Peer pro Runde erbringen muss. Diese Zahl hängt davon ab, wie wahrscheinlich es für einen kooperativen Peer ist, dass seine Nachbarn weniger als t positive Feedback-Elemente über ihn im Repository haben. Diese Wahrscheinlichkeit lässt sich mit Hilfe der Binominalverteilung $B(i, s, p_{pos}); \forall i : i < t, i \in \mathbb{N}_0$ ermitteln, da die Reihenfolge der Elemente im Repository keine Rolle spielt und jedes einzelne Feedback-Objekt mit p_{pos} positiv und mit $1 - p_{pos}$ negativ ist. Die Wahrscheinlichkeit für weniger als t positive Feedback-Objekte ist die Summe der Einzelwahrscheinlichkeiten für $i = 0, 1, \cdots, t-1$ positive Objekte:

$$\begin{aligned} p_r = & \; P(\text{Anzahl positives Feedback im Repository} < t) \\ = & \sum_{i=0}^{t-1} \binom{s}{i} \cdot (p_{pos})^i \cdot (1 - p_{pos})^{s-i} \end{aligned} \tag{3.21}$$

Ebenso wie beim Feedback für das Beantworten oder Weiterleiten wird auch beim Arbeitsbeweis Feedback in einer vorab festgelegten Anzahl q_{ProW} generiert. Peers müssen mehrere Arbeitsbeweise erbringen, wenn die Zahl des generierten Feedbacks zu niedrig bzw. die Zahl des positiven Feedbacks im Repository zu weit unterhalb des Schwellenwertes liegt.

Beispiel: Für einen erbrachten Arbeitsbeweis werden $q_{ProW} = 2$ positive Feedback-Objekte generiert, und der Schwellenwert t wurde auf $t = 6$ festgelegt. Ein Peer, dessen Nachbarn nur ein einziges positives Feedback-Element über ihn im Repository haben, muss also drei Arbeitsbeweise erbringen, bis er den Schwellenwert erreicht. Ein Peer mit 4 positiven Feedback-Objekten braucht hingegen nur einen Arbeitsbeweis leisten.

Um die zu erbringenden Arbeitsbeweise zu ermitteln, sind also jeweils die Wahrscheinlichkeiten für $0, 1, \cdots, t-1$ positive Feedback-Objekte im Repository mit der Zahl der Arbeitsbeweise zu multiplizieren, die nötig sind, um den Schwellenwert zu erreichen. Gleichung 3.22 zeigt nun die Zahl der Arbeitsbeweise, die ein kooperativer Peer pro Runde leisten muss:

$$h_{ProW} = \sum_{i=0}^{t-1} \binom{s}{i} \cdot (p_{pos})^i \cdot (1 - p_{pos})^{s-i} \cdot \left\lceil \frac{t-i}{q_{ProW}} \right\rceil \tag{3.22}$$

Unkooperative Peers Im folgenden wird davon ausgegangen, dass sich nur ein unkooperativer Teilnehmer in der Kette der Weiterleiter befindet. Dies ist realistisch für eine Anzahl u von unkooperativen Teilnehmern, für die $\delta << N/u$ gilt. Befindet sich mehr als ein unkooperativer Peer in der Kette der Weiterleiter, so sinkt die Wahrscheinlichkeit für eine erfolgreich beantwortete Anfrage, es wird noch weniger positives Feedback generiert, und die Kosten für unkooperative Teilnehmer steigen noch weiter.

Eine Nachricht, die an kooperative Knoten übermittelt wurde, wird mit der globalen Ausfallwahrscheinlichkeit p nicht korrekt bearbeitet. Unkooperative Knoten zeichnen sich in dieser Analyse hingegen dadurch aus, dass sie Nachrichten mit einer lokalen Ausfallwahrscheinlichkeit q unter der Nebenbedingung $q > p$ nicht bearbeiten, wobei p und q voneinander unabhängig sind. Ein unkooperativer Knoten bearbeitet also nur einen Anteil von $(1 - q)$ Nachrichten, während kooperative $(1 - p)$ Nachrichten beantworten. Die Anzahl der von unkooperativen Peers bearbeiteten Weiterleitungen $\tilde{h}_{forward}$ und Anfragen $\tilde{h}_{answer}$ pro Runde lässt sich wie folgt aus den für kooperative Knoten bestimmten Gleichungen ableiten:

$$\begin{aligned} \tilde{h}_{forward} &= h_{forward} \cdot \tfrac{1-q}{1-p} \\ \tilde{h}_{answer} &= h_{answer} \cdot \tfrac{1-q}{1-p} \end{aligned} \tag{3.23}$$

Nun soll das generierte positive Feedback $\tilde{h}_{forward,pos}$, $\tilde{h}_{answer,pos}$ ermittelt werden. Befindet sich ein unkooperativer Peer an einer beliebigen Stelle in der Kette der Knoten, die eine Nachricht weiterleiten und beantworten müssen, so ist in den Gleichungen für die Anzahl positiven Feedbacks ebenfalls ein Faktor $(1 - p)$ durch $(1 - q)$ zu ersetzen:

$$\begin{aligned} \tilde{h}_{forward,pos} &= h_{forward,pos} \cdot \tfrac{1-q}{1-p} \\ \tilde{h}_{answer,pos} &= h_{answer,pos} \cdot \tfrac{1-q}{1-p} \end{aligned} \tag{3.24}$$

Unkooperative Knoten leiten nicht weiter oder beantworten nicht mit Wahrscheinlichkeit q anstelle von p. Die Formeln für die Anzahl des generierten negativen Feedbacks $\tilde{h}_{forward,neg}$, $\tilde{h}_{answer,neg}$ müssen also mit q/p angepasst werden:

$$\begin{aligned} \tilde{h}_{forward,neg} &= h_{forward,neg} \cdot \tfrac{q}{p} \\ \tilde{h}_{answer,neg} &= h_{answer,neg} \cdot \tfrac{q}{p} \end{aligned} \tag{3.25}$$

Basierend auf diesen Formeln können nun wiederum die Wahrscheinlichkeiten für positives oder negatives Feedback im Repository, sowie die Wahrscheinlichkeit und die Anzahl von Arbeitsbeweisen ermittelt werden, die unkooperative Peers erbringen müssen.

Kosten des Protokolls Das Ziel von FairNet besteht darin, unkooperatives Verhalten ökonomisch unattraktiv zu machen, und kooperative Peers nur mit definierten (geringen) Overhead-Kosten für das Protokoll zu belasten. Der Erwartungswert für die Kosten von kooperativen und unkooperativen Knoten pro Runde besteht nun aus der Summe der Häufigkeiten der verschiedenen Aufgaben multipliziert mit ihren spezifischen Kosten:

$$
\begin{aligned}
E(\text{Kosten kooperativer Knoten}) = \quad & h_{answer} \cdot c_{answer} \\
+ \quad & h_{forward} \cdot c_{forward} \\
+ \quad & h_{ProW} \cdot c_{ProW} \\
E(\text{Kosten unkooperativer Knoten}) = \quad & \tilde{h}_{answer} \cdot c_{answer} \\
+ \quad & \tilde{h}_{forward} \cdot c_{forward} \\
+ \quad & \tilde{h}_{ProW} \cdot c_{ProW}
\end{aligned}
\tag{3.26}
$$

Nun sollen die Kosten für den Beitritt eines Peers zum FairNet (die *Markteintrittsbarriere*) quantifiziert werden. Ein neuer Knoten hat noch kein einziges positives Feedback-Objekt, daher muss er am Anfang Arbeitsbeweise erbringen, bis über ihn mindestens t positive Feedback-Objekte generiert wurden. Diese Anzahl ist für kooperative und unkooperative Peers identisch. Gleichung 3.27 ermittelt den Erwartungswert für die Beitrittskosten:

$$
E(\text{Kosten für den Beitritt}) = \left\lceil \frac{t}{q_{ProW}} \right\rceil \cdot c_{ProW} \tag{3.27}
$$

Dynamik des Protokolls Zum Abschluss der Analyse soll bestimmt werden, wie dynamisch FairNet auf Veränderungen im Verhalten der Knoten reagiert. Damit ein *unkooperativer* Knoten mit $\mathfrak{t}_u = s \cdot p_{pos}$ positiven Feedback-Objekten ($0 \leq \mathfrak{t}_u < t$) sich rehabilitiert ($\mathfrak{t}_u \geq t$), muss er mindestens $t - \mathfrak{t}_u$ weitere positive Feedback-Objekte erwerben. In FairNet werden Nachrichten nicht an unkooperative Peers weitergegeben. Möchte ein unkooperativer Peer selbst eine Anfrage stellen, so muss er also einen Arbeitsbeweis erbringen. Deswegen wird die Zeit für die Rehabilitation eines unkooperativen Peers nur von der Schwelle t, den bereits verfügbaren positiven Feedback-Objekten $\mathfrak{t}_u$ sowie von der Anzahl q_{ProW} des pro Arbeitsbeweis generierten positiven Feedbacks bestimmt. Im Durchschnitt stellt jeder Peer pro Runde eine Anfrage. Darum beträgt der Erwartungswert für die 'Rehabilitationszeit' eines unkooperativen Peers in FairNet:

$$
E(\text{Zeit bis } \mathfrak{t}_u \geq t) = \frac{t - \mathfrak{t}_u}{q_{ProW}} \tag{3.28}
$$

Anders verhält es sich, wenn ein als kooperativ angesehener Peer mit t_c positiven Feedback-Elementen plötzlich unkooperativ wird, d.h., seine lokale Ausfallwahrscheinlichkeit von q auf q' steigert ($q \leq p$ und $q' \geq p$). Damit fällt seine Wahrscheinlichkeit für positives Feedback zugleich von p_{pos} auf p'_{pos} (vgl. Gleichung 3.17 und vorhergehende). Wenn der Peer kooperativ war und nun unkooperativ geworden ist gilt $s \cdot p'_{pos} < t \leq s \cdot p_{pos}$.

In jeder Runde muss ein Peer im Durchschnitt eine Anfrage beantworten und δ Weiterleitungen leisten. Dafür werden über einen kooperativen Peer $\sigma_{pos} + \sigma_{neg}$ Feedback-Objekte (Gleichungen 3.15 und 3.16) generiert, und über einen unkooperativen $\tilde{\sigma}_{pos} + \tilde{\sigma}_{neg}$ Objekte. Daher dauert es $s/(\tilde{\sigma}_{pos} + \tilde{\sigma}_{neg})$ Runden, bis das Feedback in den Repositories einmal vollständig ersetzt wurde. Von größerer Bedeutung ist jedoch die Zeitspanne, die vergeht, bis die Zahl des

positiven Feedbacks t_c eines unkooperativ gewordenen Peers unter den Schwellenwert t fällt. Gleichung 3.29 bestimmt den Erwartungswert für diese Zeitspanne:

$$E(\text{Zeit bis } \mathfrak{t}_c < t) = \frac{t - s \cdot p_{pos}}{(\tilde{\sigma}_{pos} + \tilde{\sigma}_{neg}) \cdot (p'_{pos} - p_{pos})} \tag{3.29}$$

Beispiel: In einem CAN mit 10.000 Peers und einer geringen globalen Ausfallwahrscheinlichkeit beträgt die Wahrscheinlichkeit für positives Feedback $p_{pos} = 0{,}85$ für kooperative $(q = 0)$ Knoten, und $p_{pos} = 0{,}13$ für unkooperative Knoten mit $q = 0{,}5$. Der Erwartungswert für die Zeit, bis sich das positive Feedback unter den Schwellenwert $t = 6$ verringert, wenn ein kooperativer Peer unkooperativ wird, beträgt hier $\approx 1{,}2$ Runden.

3.5 Analytische und experimentelle Evaluierung

Mit dem algebraischen Modell ist es nun möglich, den Einfluss der Parameter des Protokolls sowie die Abhängigkeiten zwischen ihnen zu untersuchen. Darüber hinaus erlaubt es das erstellte Kostenmodell, die Parametrisierung von FairNet als ökonomisches Optimierungsproblem aufzufassen. Da sich das aufgestellte Gleichungssystem nur schwer algebraisch auflösen ließ, wurden numerische Methoden eingesetzt, um die Formeln zu interpretieren. Soweit nicht anders angegeben, werden dabei für die Parameter die in den Tabellen 3.1 bis 3.3 angegebenen Vorgabewerte verwendet, d.h., es wird ein CAN mit einem vierdimensionalen Schlüsselraum und 10.000 Peers zugrunde gelegt, die Kosten betragen 2 für das Weiterleiten, 5 für das Beantworten und 100 für den Arbeitsbeweis, und die Repositories der Peers haben eine Kapazität von 10 Objekten pro Feedback-Subjekt.

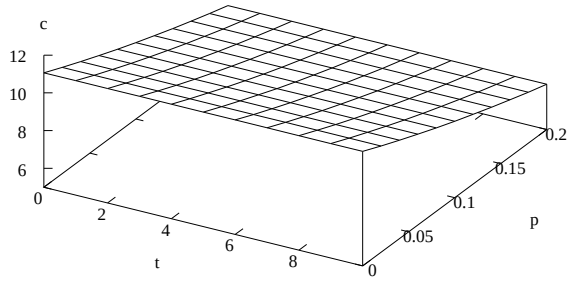

Abbildung 3.7: Kosten für reguläre Arbeit.

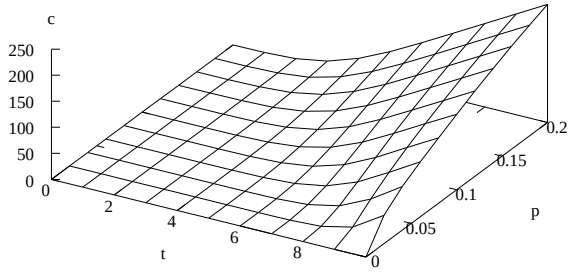

Abbildung 3.8: Kosten für Arbeitsbeweise.

Kosten für kooperative Knoten Zuerst sollen kooperative Knoten betrachtet werden, die mit der globalen Ausfallwahrscheinlichkeit p Nachrichten nicht korrekt verarbeiten. Deren Kosten hängen wesentlich vom Schwellenwert t und von der Ausfallwahrscheinlichkeit p ab.

Abbildung 3.7 zeigt die Kosten für das Weiterleiten und Beantworten von Nachrichten, Abbildung 3.8 stellt die Kosten für das Erbringen von Arbeitsbeweisen dar. Die globale Ausfallwahrscheinlichkeit wird von 0 bis 20% variiert. Der Fall $p = 0$ steht für ein CAN, in dem jede Anfrage beantwortet wird. Eine globale Ausfallwahrscheinlichkeit von $p = 0{,}2$ bedeutet hingegen, dass jeder Peer im Durchschnitt jede 5. Nachricht verwirft, oder dass jeder 5. Peer keine einzige Nachricht korrekt verarbeitet. Im folgenden wird nachgewiesen, dass unkooperative Teilnehmer im Vergleich mit kooperativen vielfach höheren Kosten durch Arbeitsbeweise tragen müssen. Das bedeutet, dass unkooperative Teilnehmer entweder ihr Verhalten rasch ändern oder das System verlassen. Knoten, die keine Arbeitsbeweise erbringen, werden beim Weiterleiten umgangen. Eine globale Ausfallwahrscheinlichkeit von 20% ist deshalb eine sehr vorsichtige Annahme für den ungünstigsten Fall.

Abbildung 3.7 zeigt, dass eine hohe globale Ausfallwahrscheinlichkeit die Kosten für das Beantworten und Weiterleiten für kooperative Knoten reduziert. Bei einer hohen Ausfallwahrscheinlichkeit werden viele Nachrichten bereits von den ersten Weiterleitern nicht verarbeitet, und erreichen somit ihr Ziel nicht. Auf der anderen Seite erhöht sich dadurch die Zahl der Arbeitsbeweise, die sowohl kooperative als auch unkooperative Knoten erbringen müssen (Abbildung 3.8). Arbeitsbeweise sind in unserem Kostenmodell 20 mal so teuer wie das Beantworten von Anfragen. Sofern der Schwellenwert t nicht geeignet bestimmt wurde, übersteigen deswegen die Kosten für Arbeitsbeweise, die kooperative Peers erbringen müssen, bei weitem die Einsparungen durch verloren gegangene Nachrichten.

Abbildung 3.8 zeigt, dass für $t = 0$ keine Kosten durch Arbeitsbeweise entstehen. Hier wird jeder Peer stets als kooperativ betrachtet. Das andere Extrem ist ein Schwellenwert $t = 10$, der der Repository-Kapazität entspricht. Das bedeutet, *jedes einzelne* negative Feedback-Objekt verursacht so lange Arbeitsbeweise, bis das negative Objekt aus dem Repository verdrängt ist. Selbst bei einer sehr geringen globalen Ausfallwahrscheinlichkeit müssen hier kooperative Peers hohe Kosten durch Arbeitsbeweise in Kauf nehmen.

Kosten für unkooperative Knoten Die Kosten unkooperativer Knoten hängen vom Schwellenwert t und von der lokalen Ausfallwahrscheinlichkeit q ab. Um die Kosten unkooperativer Knoten mit denen von kooperativen Teilnehmern ins Verhältnis zu setzen, wird die lokale Ausfallwahrscheinlichkeit (willkürlich) auf auf $q = p \cdot 2$ festgelegt. Ein unkooperativer Knoten verarbeitet eine Nachricht also doppelt so häufig nicht korrekt, wie beliebige andere (kooperative) Knoten. Auch hierbei handelt es sich wieder um eine sehr konservative Annahme, die nachweisen soll, dass auch geringe Unterschiede in den Ausfallraten vom Protokoll erkannt und abgestraft werden. Bei einer kleinen globalen Ausfallwahrscheinlichkeit von $p = 5\%$ würde ein unkooperativer Peer immer noch 90% aller Nachrichten korrekt verarbeiten! Bei geringen Ausfallwahrscheinlichkeiten können Peers nur geringe Einsparungen erzielen (vgl. Abb. 3.7). In der Realität würden unkooperative Teilnehmer wahrscheinlich häufiger als nur mit dem Doppelten der globalen Ausfallwahrscheinlichkeit Nachrichten verwerfen, und damit weit höhere Kosten tragen müssen.

Abbildung 3.9 zeigt nun für unkooperative Peers die Summe der Kosten für Arbeitsbeweise, für das Weiterleiten und das Beantworten in Abhängigkeit vom Schwellenwert t und der globa-

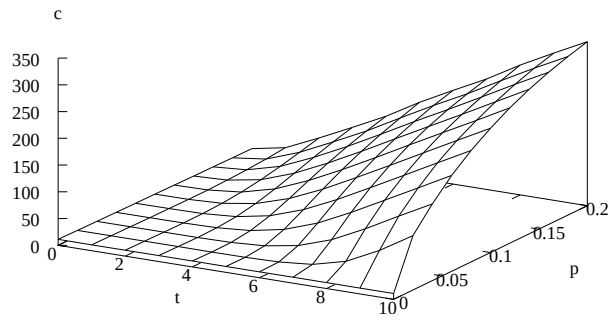

Abbildung 3.9: Gesamtkosten für unkooperative Peers.

len Ausfallwahrscheinlichkeit p. Beim Schwellenwert $t = 0$ wird jeder Knoten für kooperativ gehalten, und es werden keine Arbeitsbeweise verlangt. Abbildung 3.9 weist für diesen Fall erwartungsgemäß niedrige Gesamtkosten aus. Gemäß Abbildung 3.7 sinken die Gesamtkosten geringfügig mit steigender Ausfallwahrscheinlichkeit. Die dadurch erzielbaren Einsparungen sind jedoch zu vernachlässigen, wenn sinnvolle Schwellenwerte gewählt werden: mit steigendem Schwellenwert t nimmt die Zahl der Arbeitsbeweise rapide zu. Selbst bei so geringen Ausfallwahrscheinlichkeiten wie $p = 0{,}05$ und $q = 0{,}1$ werden ab $t \geq 5$ Arbeitsbeweise verlangt.

Diskriminierung unkooperativer Peers Nun soll untersucht werden, unter welchen Parametereinstellungen unkooperative Knoten höhere Gesamtkosten tragen müssen als kooperative Peers. Dazu wird die *Diskriminierung* c' definiert als der Quotient aus den Gesamtkosten der unkooperativen Peers dividiert durch die Gesamtkosten der kooperativen Peers:

$$c' = \frac{\tilde{h}_{answer} \cdot c_{answer} + \tilde{h}_{forward} \cdot c_{forward} + \tilde{h}_{ProW} \cdot c_{ProW}}{h_{answer} \cdot c_{answer} + h_{forward} \cdot c_{forward} + h_{ProW} \cdot c_{ProW}} \tag{3.30}$$

Eine Diskriminierung von $c' = 2$ bedeutet, dass ein unkooperativer Peer in FairNet doppelt so hohe Gesamtkosten aufbringen muss wie ein kooperativer. Damit das Protokoll unkooperatives Verhalten unattraktiv macht, muss die Diskriminierung stets deutlich größer sein als 1. Auf keinen Fall darf die Diskriminierung kleiner sein als 1 – dies würde bedeuten, dass sich unkooperatives Verhalten auszahlt.

Abbildung 3.10 zeigt die Diskriminierung in Abhängigkeit vom Schwellenwert t und der Ausfallwahrscheinlichkeit q. Wieder gilt $q = p \cdot 2$. Bei kleinen Schwellenwerten t und geringen Ausfallwahrscheinlichkeiten werden sowohl von kooperativen als auch unkooperativen Knoten keine Arbeitsbeweise verlangt; die Kosten für beide Verhaltensweisen sind gleich und $c' = 1$. Bei mittleren Ausfallwahrscheinlichkeiten und Schwellenwerten um den Wert $t = 5$ kann eine Diskriminierung bis zu $4{,}5$ erreicht werden. Wird der Schwellenwert weiter erhöht, sinkt die Diskriminierung wieder, weil nun sowohl von kooperativen als auch von unkooperativen Knoten sehr viele Arbeitsbeweise verlangt werden. Die Abbildung weist nach, dass es selbst unter Extrembedingungen wie $t = 0$, $t = s$ oder einer hohen Ausfallwahrscheinlichkeit

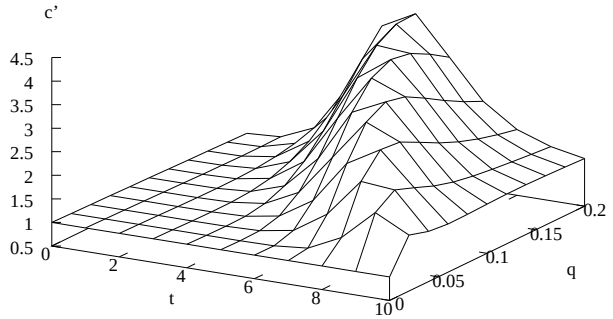

Abbildung 3.10: Diskriminierung zwischen kooperativen und unkooperativen Peers.

niemals vorkommt, dass die Diskriminierung kleiner 1 wird. Zum Anderen zeigt sie, dass es einen sehr großen Bereich gibt, in dem die Kosten für kooperative Peers moderat sind (vgl. Abb. 3.8), in dem die Diskriminierung aber so groß ist, dass unkooperative Knoten mindestens doppelt so viel leisten müssen wie kooperative.

Mehrkosten für kooperative Peers Bis hierher wurde festgestellt, dass sich unkooperatives Verhalten nur dann auszahlt, wenn der Schwellenwert zur Differenzierung zwischen kooperativen und unkooperativer Knoten zu niedrig ist. Auf der anderen Seite zeigt Abbildung 3.8, dass bei hohem t auch kooperativen Knoten mit zahlreichen Arbeitsbeweisen bestraft werden. Die Bestimmung geeigneter Parameter in FairNet ist also ein Optimierungsproblem, das den Zielkonflikt zwischen der Bestrafung unkooperativer Knoten und begrenztem Protokolloverhead für kooperative Knoten auflösen muss. Der Protokolloverhead kooperativer Knoten c_o wird hier definiert als die Kosten für Arbeitsbeweise, geteilt durch die Gesamtkosten:

$$c_o = \frac{h_{ProW} \cdot c_{ProW}}{h_{answer} \cdot c_{answer} + h_{forward} \cdot c_{forward} + h_{ProW} \cdot c_{ProW}} \tag{3.31}$$

Die Maximierung der Diskriminierung bei gleichzeitiger Minimierung der Overheadkosten kooperativer Knoten ist nicht möglich: die Overheadkosten betragen 0 nur bei $t = 0$. Deswegen wird für die folgenden Untersuchungen ein Grenzwert von $c_o = 10\%$ für den Protokolloverhead kooperativer Knoten eingeführt. Das bedeutet, dass ein kooperativer Knoten bereit ist, höchstens 10% seiner Gesamtkosten für das Erbringen von Arbeitsbeweisen aufzuwenden. Der Wert 10% wurde gewählt, um nachzuweisen, dass FairNet auch mit geringen zusätzlichen Kosten für kooperative Knoten effektiv ist. Mit Hilfe der Restriktion für den Protokolloverhead lässt sich nun der Parametersatz (s. Tabelle 3.3) numerisch bestimmen, bei dem die Diskriminierung maximal wird. Optimiert werden die Parameter t, $q_{forward}$, q_{answer}, q_{ProW} und q_{pn}.

Beispiel: Gegeben ist eine P2P-Datenstruktur, in der die exogenen Parameter für die Anzahl der Knoten, die Dimensionalität und die Pfadlänge dazu führen, dass ein kooperativer Knoten pro Runde Kosten von 20 für das Weiterleiten und Beantworten von Anfragen aufbringen muss. Der untersuchte Parametersatz von FairNet führt zu Kosten von 5 für das Bearbeiten von Arbeitsbeweisen; die Gesamtkosten betragen also 25 pro Runde. Es gilt $5/25 = 0{,}2$, somit

übersteigen die Overhead-Kosten den erlaubten 10%-Anteil an den Gesamtkosten. Deswegen wird der untersuchte Parametersatz verworfen und nach einem besseren gesucht.

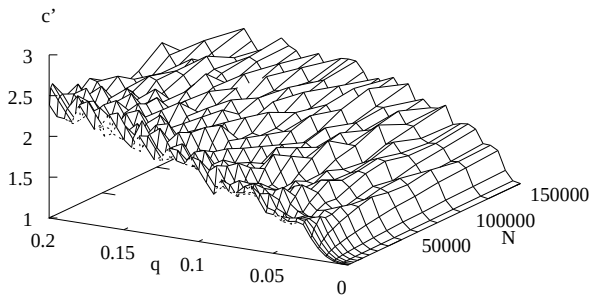

Abbildung 3.11: Diskriminierung bei begrenztem Mehraufwand für kooperative Knoten.

Abbildung 3.11 gibt nun die erreichbare Diskriminierung unter Berücksichtigung dieser 10%-Schwelle an. Die x-Achse im Diagramm 3.11 zeigt die Ausfallwahrscheinlichkeit, wobei wieder der Zusammenhang $q = p \cdot 2$ zwischen der lokalen Ausfallwahrscheinlichkeit q der unkooperativen Knoten und der globalen Ausfallwahrscheinlichkeit p aller Peers gilt. Auf der y-Achse ist die Zahl der Peers in der Datenstruktur abgetragen. Da die Dimensionalität des Schlüsselraums $d = 4$ beibehalten wurde, reichen die Pfadlängen von $\delta \approx 2{,}6$ bei 1.000 Peers bis $\delta \approx 6{,}9$ bei 100.000 Peers.

Abbildung 3.11 zeigt, dass sich auch unter stark variablen Umgebungsbedingungen eine gute Diskriminierung unkooperativer Peers erreichen lässt. Je höher der Anteil der von unkooperativen Peers verworfenen Nachrichten, desto besser ist die bei 10% Overheadkosten erreichbare Diskriminierung. Die Zahl der Peers im System zeigt hingegen keine Auswirkungen auf die Diskriminierung – auch bei sehr wenigen oder vielen Teilnehmern lassen sich Parametersätze finden, die zu einer zuverlässigen Unterscheidung zwischen kooperativen und unkooperativen Knoten führen. Weiterhin sind die Annahmen sehr konservativ, dass kooperative Teilnehmer nur 10% Mehrkosten tragen und unkooperative Peers nur mit dem doppelten der globalen Ausfallwahrscheinlichkeit arbeiten. Unter diesen Gesichtspunkten zeigt Diagramm 3.11 ein sehr positives Ergebnis.

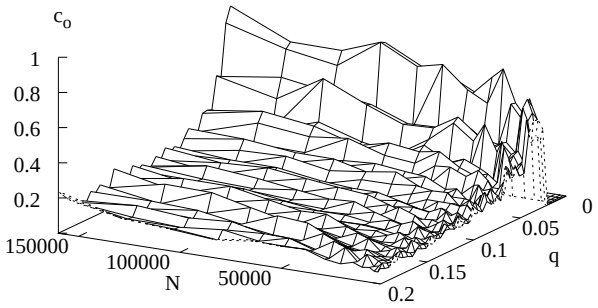

Abbildung 3.12: Mehrkosten für kooperative Knoten bei gegebener Diskriminierung.

Das Diagramm 3.12 zeigt nun die inverse Sichtweise: hier wurde als Bedingung die Diskriminierung auf mindestens $c' = 1{,}2$ festgesetzt, d.h., ein unkooperativer Peer mit einer lokalen

Ausfallwahrscheinlichkeit $q = p \cdot 2$ muss um 20% höhere Gesamtkosten tragen als ein kooperativer Knoten. Mit dieser Nebenbedingung wurde nach dem Parametersatz gesucht, bei dem die Mehrkosten für kooperative Teilnehmer möglichst gering sind. Die z-Achse zeigt nun die Overheadkosten c_o für kooperative Teilnehmer. Das Diagramm zeigt, dass – abgesehen von $q < 0{,}05$ – kooperative Peers nur geringe Mehrkosten aufwenden müssen, um unkooperative Peers zu disziplinieren. Soll jedoch auch sehr geringfügig unkooperatives Verhalten mit lokalen Ausfallraten von unter 5% so mit Arbeitsbeweisen belegt werden, dass eine Diskriminierung von mindestens $1{,}2$ erreicht wird, müssen auch kooperative Knoten erhebliche Mehrkosten tragen.

Auswirkungen der Repository-Kapazität Der Einfluss der Repository-Kapazität s scheint offensichtlich: je mehr Feedback-Objekte im Repository zur Verfügung stehen, desto besser müsste das Protokoll zwischen kooperativen und unkooperativen Peers differenzieren können, und desto höher wird die Diskriminierung. Auf der anderen Seite erhöht ein großes Repository den Verwaltungsaufwand für die einzelnen Peers.

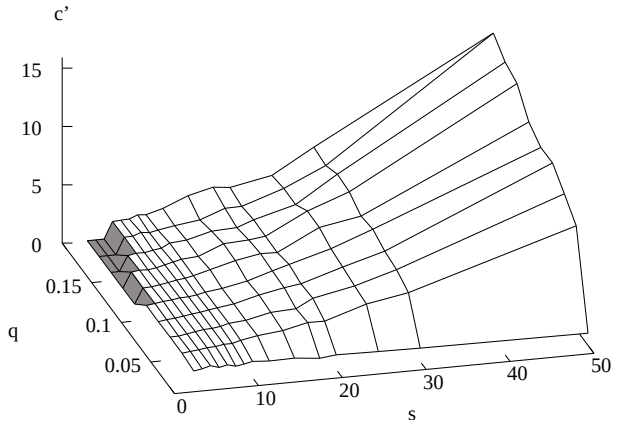

Abbildung 3.13: Zusammenhang zwischen Repository-Kapazität und der Diskriminierung.

Abbildung 3.13 zeigt die erreichbare Diskriminierung bei variierter Repository-Größe. Es gilt wieder $q = p \cdot 2$ und ein maximaler Protokolloverhead von $c_o = 10\%$ für kooperative Knoten. Die grau markierten Bereiche von s und q heben die Wertebereiche hervor, in denen die Beschränkung des Overheads nicht eingehalten werden konnte. Das Diagramm 3.13 zeigt ein positives Resultat: abgesehen von sehr kleinen Repository-Kapazitäten gibt es stets Parametersätze, bei denen FairNet gut zwischen kooperativen und unkooperativen Knoten unterscheidet. Weiterhin zeigt das Diagramm, dass sich die Diskriminierung tatsächlich durch eine Erhöhung der Repository-Größe steigern lässt.

Aus Gleichung 3.29 folgt, dass große Repositories die Reaktionsgeschwindigkeit des Protokolls auf Änderungen im Verhalten der Peers beeinträchtigen. Abbildung 3.13 zeigt jedoch, dass Repositories mit einer Kapazität von 10 bis 20 Elementen für eine Implementierung bereits ausreichend dimensioniert sind.

Kosten für den Arbeitsbeweis Der Parameter c_{ProW} spezifiziert die Menge an Arbeit, die ein Peer leisten muss, um seine Kooperationsbereitschaft zu beweisen. Die Kosten, die dieser Arbeitsbeweis bei den Peers verursacht, sind eine kritische Größe bei der Übertragung des Protokolls in die Praxis. Bis hierher wurde angenommen, dass alle Peers über identische Ressourcen verfügen, d.h., jede Operation war für jeden Peer mit den gleichen Kosten verbunden. In der Realität sind jedoch einige Teilnehmer mit mehr Ressourcen ausgestattet als andere, zum Beispiel weil sie über leistungsstärkere Rechner oder eine bessere Internet-Verbindung verfügen. Es muss nun der Nachweis dafür erbracht werden, dass der Arbeitsbeweis auch in heterogenen Umgebungen wirksam bleibt und unkooperatives Verhalten unattraktiv macht. Es existieren zwar Ansätze, die einen Arbeitsbeweis von Rechnerressourcen unabhängig machen. Eine Möglichkeit dafür sind Captchas [ABHL03]; dabei handelt es sich um eine Klasse von Problemen, die ein Computer selbständig nicht lösen kann. Beispiele für Captchas sind einfache Intelligenztests, nach denen ein Anwender eine Anzahl Fotos einem gemeinsamen Oberbegriff zuordnen muss.

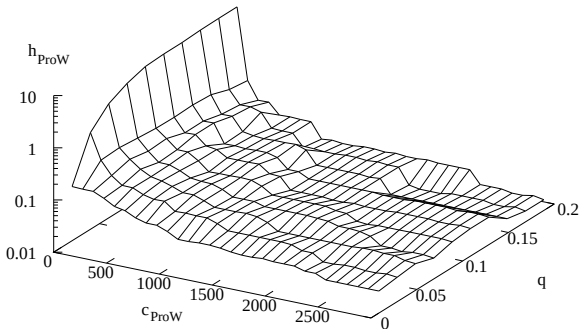

Abbildung 3.14: Zusammenhang zwischen den Kosten für den Arbeitsbeweis und der Zahl der von unkooperativen Peers verlangten Arbeitsbeweise.

Das Diagramm 3.14 zeigt nun den Zusammenhang zwischen der Anzahl der von unkooperativen Peers zu verlangenden Arbeitsbeweise h_{ProW}, den Kosten eines einzelnen Beweises und der Ausfallwahrscheinlichkeit. Erneut gelten die Nebenbedingungen, dass kooperative Knoten nur 10% Mehrkosten für den Arbeitsbeweis tragen dürfen, dass die Ausfallwahrscheinlichkeiten mit $q = p \cdot 2$ zusammenhängen, und dass die Diskriminierung $c' = 1{,}2$ beträgt. Die Abbildung zeigt ein überraschendes Ergebnis: die Zahl der unter den gegebenen Bedingungen erforderlichen Arbeitsbeweise fällt exponentiell mit steigenden Kosten für den einzelnen Beweis. Der Wendepunkt liegt bei $c_{ProW} \approx 200$. Sind die Arbeitsbeweise nur teuer genug, haben selbst große Änderungen in den Kosten eines Beweises kaum mehr Auswirkungen auf die Anzahl der Beweise, die von unkooperativen Peers verlangt werden müssen. Dies gilt auch bei variablen Ausfallwahrscheinlichkeiten. Im Umkehrschluss können Peers mit unterschiedlichen Ressourcenkapazitäten also auch erfolgreich mit dem gleichen Arbeitsbeweis diszipliniert werden.

Beispiel: Angenommen, ein Arbeitsbeweis kostet 2000 auf einem Rechner mit einer 1 GHz-CPU. Nun brauchen von Peers mit einer 2 GHz-Maschine nicht etwa zwei Arbeitsbeweise

verlangt werden, um eine identische Disziplinierungswirkung zu erzielen. Stattdessen erreicht auch bei einem Rechner mit einer doppelt so hohen CPU-Kapazität der eine Arbeitsbeweis eine nahezu vergleichbare Disziplinierung.

Experimentelle Evaluierung Bislang wurde numerisch gezeigt, dass das FairNet-Protokoll unkooperatives Verhalten zuverlässig unökonomisch machen kann. Dies gilt selbst unter bewusst ungünstig gestalteten Randbedingungen, beispielsweise bei unkooperativen Peers, deren Ausfallwahrscheinlichkeit nur um einen kleinen Betrag vom globalen Durchschnitt abweicht. Das Protokoll lässt sich so parametrisieren, dass kooperative Teilnehmern nur geringe Mehrkosten entstehen. Der Verwaltungsaufwand und der Speicherbedarf von FairNet ist ebenfalls begrenzt, da Feedback-Repositories mit einer geringen Kapazität für eine gute Differenzierung bereits ausreichen. Weiterhin wurde gezeigt, dass FairNet auch dann wirksam bleibt, wenn exogene Einflussfaktoren wie die Teilnehmerzahl oder die globale Ausfallwahrscheinlichkeit über einen weiten Bereich variiert werden. Nun soll gezeigt werden, dass das FairNet-Protokoll auch in der Praxis anwendbar ist. Anhand einer prototypischen Implementierung in Java sollen dazu die folgenden Punkte untersucht werden:

- Weist die Implementierung des Protokolls die in der numerischen Analyse berechneten Eigenschaften auf? Ist das algebraische Modell also korrekt und in die Praxis übertragbar?
- Wie restriktiv sind die Modellannahmen aus Abschnitt 3.4? Was passiert, wenn Annahmen wie eine regelmäßige Zonenaufteilung oder gleichverteilte Anfragen aufgegeben werden?
- Wie anfällig ist das System gegenüber Änderungen in der Umwelt? In der Realität sind exogene Parameter wie die Ausfallwahrscheinlichkeit oder die Zahl der Peers im System selten exakt bekannt und fluktuieren über die Zeit. Deswegen muss nachgewiesen werden, dass das Protokoll auch dann noch funktionsfähig bleibt, wenn die Umgebungsbedingungen von den Werten abweichen, auf die die Parameter von FairNet optimiert wurden.

Alle Experimente wurden auf einem Linux-Cluster mit 32 Knoten durchgeführt. Jeder Knoten ist mit einer 2 GHz-CPU, 2 GB RAM und 100 MBit ausgestattet. Auf dem Cluster wurden Experimente mit zwei verschiedene Implementierungen in Java durchgeführt, einem *CAN-Prototypen* und einem *CAN-Simulator*.

Der *CAN-Prototyp* [BB04c] ist eine fast vollständige Implementierung eines Content-Addressable Networks, bei der nur auf die Teilkomponenten verzichtet wurde, die zur Durchführung der Experimente ohne Bedeutung waren. Dazu gehören insbesondere eine persistente Datenhaltung und Reparaturmechanismen, welche die durch den Ausfall von Peers entstandenen 'Löcher' im Schlüsselraum anderen Peers zuweisen. Abbildung 3.15 zeigt schematisch die Architektur des Prototypen. Um mehrere tausend Peers auf einem Rechner arbeiten zu lassen, wurden die Peers jeweils als Threads innerhalb einer einzigen virtuellen Java-Maschine (*Peer-Manager*) realisiert. Nachrichten zwischen Peers innerhalb der gleichen virtuellen Maschine können so mittels Interprozesskommunikation effizient versendet werden; nur Nachrichten an

Knoten auf anderen Maschinen müssen über die Netzwerk-Schnittstelle geschickt werden. Zentrale Server-Komponenten (*ExperimentManager*, *LogConsole*) steuern die Experimente. Sie teilen jedem Peer seinen Datenbereich zu, bestimmen die abzusendenden Anfragen der Peers, stellen eine zentrale Uhr bereit und ermöglichen eine synchrone Erfassung der Messwerte aller Peers. Der Prototyp ist in der Lage, Experimente mit > 100.000 Peers durchzuführen. Die Zahl der Peers wird nur durch die Zahl der verfügbaren Rechner begrenzt, auf denen jeweils mehrere tausend Peers parallel gestartet werden können. Der Prototyp wird in [BB04c] ausführlich beschrieben.

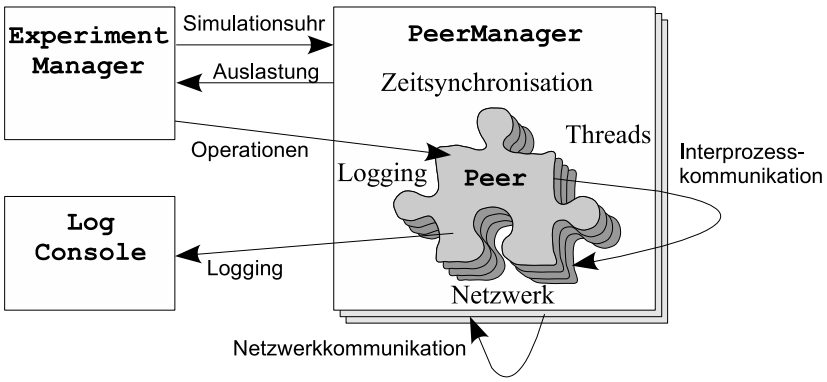

Abbildung 3.15: Die Architektur des CAN-Prototypen.

Der *CAN-Simulator* verfolgt einen anderen Ansatz als der Prototyp. Er wurde daraufhin optimiert, ein minimales, laufzeiteffizientes und leicht änderbares Framework zur Durchführung von Experimenten auf einem *einzelnen* Rechner bereitzustellen. Dazu wurde auf eine Kommunikation per Netzwerk verzichtet; sämtliche Informationen werden über eine Shared Memory-Schnittstelle übertragen. Es werden nur Nachrichten zwischen den Peers weitergeleitet, eine Datenhaltung findet nicht statt. Weiterhin konnte auf Mechanismen zur Synchronisierung verzichtet werden. Stattdessen werden sämtliche Operationen sequentiell von einem einzigen Programmthread abgearbeitet. Wegen der Beschränkung auf einen Rechner limitiert die Größe des verfügbaren Hauptspeichers die Zahl der an einer Simulation beteiligten Peers auf ca. 50.000 Knoten auf einer Maschine mit 1 GB RAM. Mit dem Simulator ist es möglich, ein Experiment mit jeweils leicht abweichenden Umgebungsparametern parallel auf jedem der 32 Maschinen im Cluster zu starten.

Mit Hilfe der beiden Implementierungen wurden nun Experimente in einem vierdimensionalen CAN mit bis zu 160.000 Peers durchgeführt. Dabei wurden jeweils 5.000.000 Anfragen evaluiert. Um nur den eingeschwungenen Zustand zu berücksichtigen, wurden die ersten 500.000 Anfragen nicht in die Messergebnisse mit einbezogen. Sofern nicht explizit anders angegeben, wurden alle Parameter von FairNet (s. Tabelle 3.3) ihre analytisch ermittelten optimalen Werte gesetzt.

Effektivität im Experiment Die zunächst wichtigste Frage lautet: lassen sich die aus der numerischen Analyse gewonnenen Erkenntnisse auf eine reale Implementierung übertragen? Das Diagramm 3.10 zeigte die Diskriminierung zwischen kooperativen und unkooperativen Peers. Es ist eines der komplexesten Diagramme der numerischen Analyse: Auf Basis sämtlicher exogenen Parameter (vgl. Tabelle 3.1) wurden die Wahrscheinlichkeiten für Feedback unterschiedlicher Art bestimmt. Daraus wiederum wurden die Wahrscheinlichkeiten für Arbeitsbeweise abgeleitet, und aus allen Faktoren gemeinsam die Gesamtkosten kooperativer und unkooperativer Peers ermittelt. Stimmt das vom Modell vorhergesagte Verhalten von FairNet mit der Realität überein, so sollte sich bei gleichen exogenen Parametern im Experiment ein identischer Kurvenverlauf zeigen.

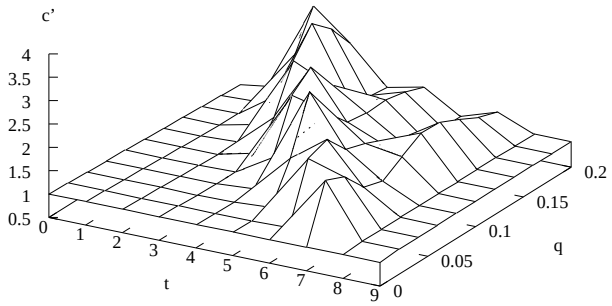

Abbildung 3.16: Diskriminierung unkooperativer Knoten im Experiment.

Die Abbildung 3.16 zeigt nun das Ergebnis des Experiments: abgesehen von einem nicht so 'glatten' Kurvenverlauf ist das Diagramm deckungsgleich mit Abbildung 3.10! Die geringen Differenzen lassen sich damit erklären, dass das algebraischen Modell davon ausgeht, dass der Feedback-Transport keine Zeit in Anspruch nimmt und Pfadlängen durch das Umgehen von unkooperativen Peers steigen. Diese Eigenschaften sind im Experiment nicht zu realisieren.

Auswirkungen der Modellannahmen Neben dem unmittelbaren Feedback-Transport wurden in Abschnitt 3.4 zur Vereinfachung der Analyse noch zahlreiche weitere Annahmen getroffen, die sich in der Realität nicht halten lassen. Es ist daher unbedingt erforderlich, einen Nachweis dafür zu erbringen, dass diese Annahmen die Anwendbarkeit des Modells nicht beeinträchtigen. Für diesen Nachweis wurde eine Serie von Experimenten durchgeführt. Als Basis dient ein CAN mit 10.000 Peers, von denen sich 10% unkooperativ verhalten, indem sie mit einer Wahrscheinlichkeit von $q = 0{,}2$ Nachrichten nicht verarbeiten. Das erste Experiment reproduziert die Annahmen aus Abschnitt 3.4, d.h., eine regelmäßige Zonenaufteilung, über den Schlüsselraum gleichverteilte Anfragen und Peers, die mit der gleichen Rate Anfragen absenden. In den weiteren Experimenten wird dann jede dieser Annahmen durch realistischere Szenarien ersetzt:

- Die vollständig reguläre Zonenaufteilung des Schlüsselraums wird ersetzt durch ein Zonenlayout, wie es durch das verteilte Split-Protokoll des CAN entsteht (vgl. Abschnitt 2.3). Die Peers sind nun für Zonen mit unterschiedlicher Größe verantwortlich,

und haben unterschiedlich viele Nachbarn. Das Greedy Routing bevorzugt beim Weiterleiten Peers mit größeren Zonen, da diese eine Nachricht über einen weiteren Weg im Schlüsselraum transportieren können. Weiterhin muss bei gleichverteilten Anfragen ein Peer mit größerer Zone auch mehr Anfragen bearbeiten. Deswegen führt das realistische Zonenlayout zu unterschiedlich belasteten Knoten.

- Die Annahme, nach der alle Peers genau eine Anfrage pro Runde absetzen, wird durch eine Wahrscheinlichkeitsverteilung ersetzt. Jedem Peer wird ein normalverteilter Wert im Intervall $(0; 2)$ mit $\mu = 1{,}0$ und $\sigma = 0{,}5$ zugewiesen. Der Wert bestimmt die Zahl der abgesendeten Anfragen pro Runde. Der Mittelwert stellt sicher, dass das algebraische Modell anwendbar bleibt, weil im Durchschnitt weiterhin eine Anfrage pro Peer und Runde gestellt wird. Die Normalverteilung ist in der Realität vielfach zu beobachten; ohne weitere Annahmen über eine spezielle P2P-Anwendung lassen sich keine realistischeren Verhaltensweisen modellieren.

- Eine weitere Annahme besagt, dass die Anfragepunkte über den Schlüsselraum gleichverteilt sind. Gute Hashfunktionen sollten tatsächlich dazu in der Lage sein, die gespeicherten Daten gleichmäßig über den Schlüsselraum zu verteilen. Dies jedoch nicht immer sinnvoll, beispielsweise wenn strukturierte Daten in der P2P-Datenstruktur abgelegt werden [SRBB04] oder die Lokalität der abgelegten Daten für effizientere Anfragen erhalten bleiben soll [BB03b]. Als Alternative werden normalverteilte Anfragepunkte evaluiert. Der Mittelwert μ ist dabei das Zentrum des vierdimensionalen Schlüsselraums, die Standardabweichung beträgt $\sigma = 0{,}5$. Es werden also mehr Anfragen in die Mitte des Schlüsselraums gesendet als zu den Randbereichen[4]. Solche Anfragemuster sind in Anwendungen häufig anzufinden, z.B. im WWW [TG97].

In den Experimenten wurde nun jede der Vereinfachungen einzeln durch die realitätsnähere Annahme ersetzt. Des weiteren wurden in einem letzten Experiment alle Vereinfachungen zusammen entfernt, um eventuell auftretende Synergieeffekte zu untersuchen. Der Nachweis, dass die Annahmen nicht zu einer praxisfernen Vereinfachung des algebraische Modells geführt haben, ist erbracht, wenn sich nur geringfügige Abweichungen in den Messwerten ergeben.

Abbildung 3.17 zeigt die Gesamtkosten für kooperative und unkooperative Knoten. Jeweils der linke Balken bildet das Experiment mit alle Vereinfachungen ab. Der rechte Balken zeigt das Experiment, bei dem alle Vereinfachungen gleichzeitig deaktiviert waren. Die Abbildung weist nach, dass die größte Abweichung durch die Einführung der realistischen Zonenaufteilung verursacht wird. Dies lässt sich durch die Veränderung in den Pfadlängen erklären. Peers mit größeren Zonen können Nachrichten über eine weitere Entfernung im Schlüsselraum weiterleiten. Somit sinkt die Pfadlänge und die Kosten für das Weiterleiten. Weil weniger Peers an der Weiterleitung einer Nachricht beteiligt sind, obwohl die Zahl der unkooperativen Peers konstant

[4] Der Schlüsselraum ist ein d-Torus. Mit 'Randbereich' sind die Bereiche im Schlüsselraum gemeint, die in der Nähe des 0-1 - Umbruchs einer Dimension liegen.

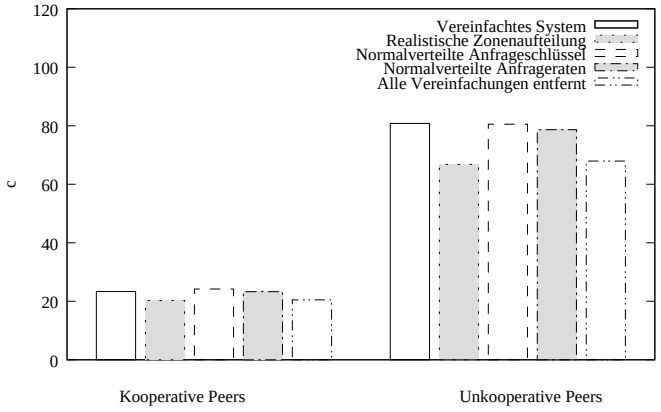

Abbildung 3.17: Auswirkungen der Modellannahmen auf die Gesamtkosten.

bleibt, verringert sich die Ausfallwahrscheinlichkeit für eine Anfrage. Deswegen sinkt auch die Zahl der Arbeitsbeweise, die einen wesentlichen Kostenfaktor ausmacht.

Im Gegensatz dazu ändern sich die Kosten nur sehr wenig bei einer veränderten Anfrageverteilung im Schlüsselraum und bei Peers mit unterschiedlichen Anfrageraten. Zwar unterscheidet sich nun die absolute Anzahl der pro Runde generierten Feedback-Objekte von Peer zu Peer, und die Geschwindigkeit variiert, mit der neue Objekte hinzukommen bzw. alte verdrängt werden. Die Wahrscheinlichkeit für positives oder negatives Feedback bleibt aber davon unbeeinflusst.

Abbildung 3.18 zeigt die Diskriminierung unter den gleichen Simulationsparametern. Realistischere Annahmen betreffen kooperative und unkooperative Knoten gemeinsam. Die Abbildung zeigt, dass die Diskriminierung dadurch noch weniger beeinflusst wird als die Gesamtkosten. Insgesamt lässt sich festhalten, dass FairNet auch in der Realität anwendbar bleibt, und dass sich das algebraische Modell sehr gut zur Bestimmung optimaler Parameterwerte in der Praxis heranziehen lässt.

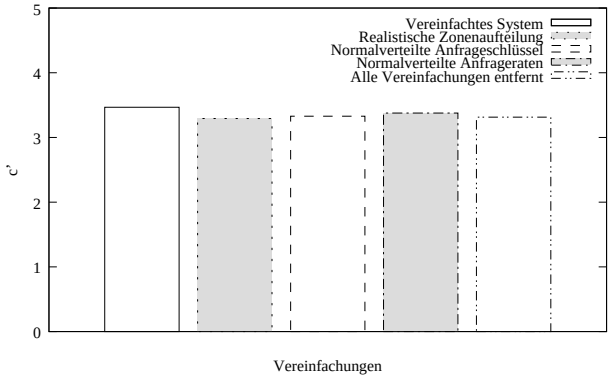

Abbildung 3.18: Auswirkungen der Modellannahmen auf die Diskriminierung.

Benachteiligt FairNet die Nachbarn unkooperativer Knoten? Das Arbeitsprinzip von FairNet besteht darin, dass bei einer fehlgeschlagenen Anfrage *sämtliche* Weiterleiter negatives Feedback erhalten. Bei einem unkooperativen Peer sammelt sich dadurch negatives Feedback an. Dadurch besteht aber auch die Gefahr, dass Peers in der unmittelbaren Nähe häufig an unkooperativen Knoten weiterleiten, und deswegen ebenfalls viel negatives Feedback erhalten und Arbeitsbeweise erbringen müssen.

Diese Gefahr ist allerdings gering. Während beim unkooperativen Peer jede unbeantwortete Anfrage zu negativem Feedback führt, erhalten seine Nachbarn nur dann derartiges Feedback, wenn sie auch an ihn weitergeleitet haben. Das heißt, bei gleichverteilten Anfragen nimmt die Dichte des negativen Feedbacks, das von einem unkooperativen Knoten verursacht wird, mit 1 / Zahl der Nachbarn pro Weiterleitungsschritt ab. Des weiteren werden erkannte unkooperative Peers bei der Weiterleitung gemieden. Ein Experiment soll nachweisen, dass dieses Problem in der Praxis tatsächlich nicht relevant wird. Dazu wurde ein CAN mit 9.500 kooperativen und 500 unkooperativen Peers ($q = 0{,}2$) mit einem vierdimensionalen Schlüsselraum aufgesetzt.

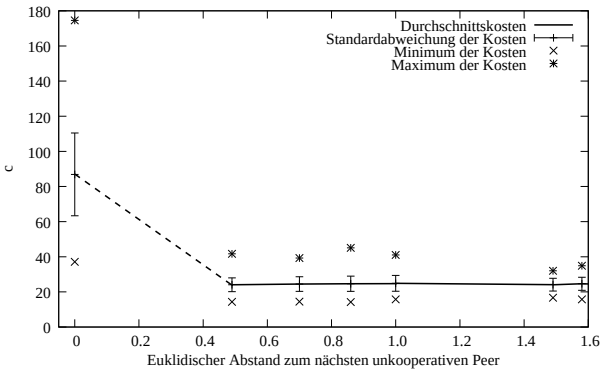

Abbildung 3.19: Zusammenhang zwischen den Gesamtkosten und dem Abstand zum nächsten unkooperativen Peer.

Abbildung 3.19 zeigt nun die Kosten der kooperativen Knoten in Abhängigkeit zur Entfernung zum nächstgelegenen unkooperativen Peer im Schlüsselraum des CAN. Als Entfernungsmaß wurde der euklidische *Abstand in Zellen* verwendet, jeweils gemessen vom Zentrum der Zone des unkooperativen Peers. Der Schlüsselraum wurde regelmäßig in Zonen aufgeteilt. Das bedeutet, ein Peer mit Abstand 0 ist unkooperativ, ein Knoten in Abstand $0{,}5$ ist verantwortlich für eine Zone unmittelbar neben dem unkooperativen Peer, und ein Knoten in Abstand $0{,}7$ hat mit der Zone des unkooperativen Peers eine Ecke gemeinsam. Die Durchschnittskosten werden als durchgezogene schwarze Linie dargestellt. Weiterhin werden die Standardabweichung als Errorbars sowie die absoluten Minimum- und Maximumwerte als Kreuz und Stern dargestellt. Das Diagramm bestätigt, dass einem Knoten durch Nachbarschaft mit einem unkooperativen Peer tatsächlich keine Nachteile entstehen. Dies gilt nicht nur im Mittel aller Fälle, auch die Standardabweichung der Kosten wird durch die Entfernung zu unkooperativen Knoten nicht beeinflusst.

Natürlich ist es leicht möglich, Fälle zu konstruieren, in denen kooperative Peers tatsächlich durch unkooperative Nachbarn in Mitleidenschaft gezogen werden. Das ist beispielsweise der Fall, wenn ein Knoten in einer 'Richtung' im Schlüsselraum vollständig von unkooperativen Peers umgeben ist. Angriffe, die so etwas leisten können, werden in Abschnitt 3.6 untersucht. Durch den regulären Beitritt von unkooperativen Peers zum CAN allein können solche 'Muster' in der Zonenaufteilung kaum entstehen. Die Wahrscheinlichkeit dafür beträgt $> 0{,}5\%$ für ein CAN mit 4 Dimensionen und einem Anteil von 50% unkooperativen Peers. Durch Replikation lässt sich diese Wahrscheinlichkeit beliebig weiter senken.

Weiterhin erlaubt das Diagramm eine weitere sehr wesentliche Beobachtung: unkooperatives Verhalten zahlt sich auch für *risikobereite* Peers nicht aus! Im Experiment waren die Minimum-Kosten der unkooperativen Peers etwa vergleichbar mit den Maximum-Kosten der kooperativen Peers. Das bedeutet für dieses Experiment: der Peer, der nur 80% der Anfragen verarbeitet und dann das 'Glück' hatte, die niedrigsten Kosten von allen unkooperativen Peers tragen zu müssen, leistet immer noch etwa ebenso viel wie der kooperative Peer, der das 'Pech' hatte, die höchsten Kosten aller kooperativen Peers erbringen zu müssen. Betrachtet man die Standardabweichungen vom Mittelwert der Kosten kooperativer und unkooperativer Peers, so wird diese Differenz noch größer.

Rate der Feedback-Erzeugung Dieses Experiment soll der Frage nachgehen, wie die Zahl der erzeugten Feedback-Objekte die Zahl der Arbeitsbeweise beeinflusst. Die Zahl des Feedbacks für reguläre Arbeit hängt von den Parametern $q_{forward}$ und q_{answer} ab. Damit es für die Peers nicht ökonomisch sinnvoll wird, zwischen Weiterleiten und Beantworten zu optimieren, muss die Menge des dafür generierten Feedbacks an die Kosten gebunden werden: $c_{forward}/c_{answer} = q_{forward}/q_{answer}$. Frei gewählt werden kann hingegen die Zahl des Feedbacks q_{ProW}, das für einen Arbeitsbeweis verlangt wird. Ein $q_{ProW} = 3$ bedeutet, dass für einen erbrachten Arbeitsbeweis drei positive Feedback-Objekte erzeugt werden. Weiterhin kann frei festgelegt werden, 'wie schlimm' das Nichtbearbeiten von Anfragen ist. Der Parameter q_{pn} legt fest, um welchen Faktor sich die Wahrscheinlichkeiten für positives Feedback von denen für negatives unterscheiden. Für eine nicht beantwortete Anfrage wird also $q_{answer} \cdot q_{pn}$ negatives Feedback generiert, für eine nicht weitergeleitete Nachricht $q_{forward} \cdot q_{pn}$.

Beispiel: Sei $q_{forward} = 1$ und $q_{pn} = 3$. Über einen Peer, der die Anfrage weiterleitet, wird nun ein positives Feedback-Objekt generiert. Entscheidet er sich, die Anfrage nicht korrekt zu verarbeiten, so werden über ihn hingegen drei negative Feedback-Objekte in Umlauf gebracht.

Aufgrund der Komplexität von FairNet ist es schwierig, die Auswirkungen von q_{ProW} und q_{pn} intuitiv abzuschätzen. Zu erwarten ist lediglich, dass sehr kleine Werte für q_{ProW} zu sehr vielen Arbeitsbeweisen führt, und dass ein $q_{pn} < 1$ dazu führt, dass unkooperatives Verhalten dominiert. Alles weitere soll ein Experiment ermitteln. Dazu wird ein CAN mit 10.000 Peers aufgesetzt, von denen sich 50% unkooperativ ($q = 0{,}2$) verhalten. Diese hohe Zahl unkooperativer Peers wurde gewählt, um möglichst aussagekräftige Kurven zu erhalten – vorab

durchgeführte Experimente haben gezeigt, dass die Kurven unter geringeren Anteilen unkooperativer Peers erheblich flacher verlaufen.

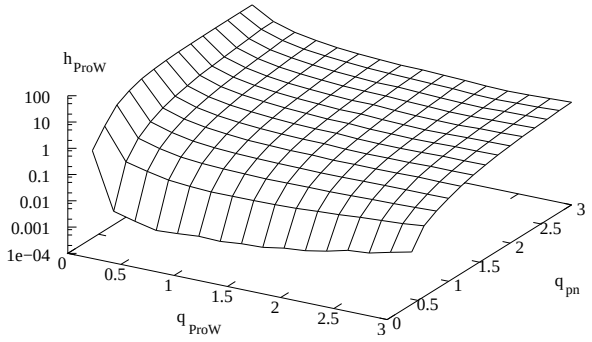

Abbildung 3.20: Einfluss von q_{pn} und q_{ProW}.

Die Abbildung 3.20 zeigt, dass mit steigendem q_{ProW} die Zahl der abgefragten Arbeitsbeweise rapide sinkt – die z-Achse ist logarithmisch skaliert. Weiterhin zeigt das Diagramm, dass die Zahl der Arbeitsbeweise nahezu konstant bleibt bei $q_{pn} > 1$. Insgesamt weist das Diagramm nach, dass der Effekt von q_{pn} und q_{ProW} über einen weiten Wertebereich vernachlässigbar klein ist, insbesondere im direkten Vergleich mit den Auswirkungen von Parametern wie z.B. dem Schwellenwert t. Dies erlaubt es, die Komplexität von FairNet zu reduzieren: es ist ohne Einschränkungen möglich, bei der Bestimmung der optimalen Parameterwerte q_{pn} und q_{ProW} mit sinnvollen Vorgabewerten statisch zu belegen.

Skalierbarkeit Bis hierher wurde gezeigt, wie sich das Protokoll bei Anwendung optimaler Parameterwerte verhält. Diese Werte wurden mit dem algebraischen Modell bestimmt. Dafür ist jedoch die Kenntnis sämtlicher exogenen Parameter des P2P-Systems erforderlich. Außerdem müsste die Optimierung jedes mal erneut durchgeführt werden, wenn sich diese Parameter ändern, beispielsweise weil neue Peers dem System beigetreten sind. Das Anliegen des folgenden Experiments ist es, nachzuweisen, dass das Protokoll auch stabil ist gegenüber Änderungen in den exogenen Parametern. Dafür wurde der optimale Parametersatz für ein CAN mit 10.000 Knoten bestimmt, von dem sich 10% mit $q = 0{,}2$ unkooperativ verhalten. Nun wurde die Zahl der Peers im System von 81 bis 160.000 variiert. Ändert sich die Zahl der Peers, ändern sich die Pfadlängen, d.h., die Zahl der an der Verarbeitung einer Anfrage beteiligten Peers. Dadurch verändert sich auch die globale Ausfallwahrscheinlichkeit. Von Pfadlänge und Ausfallwahrscheinlichkeit hängen jedoch alle anderen Parameter von FairNet ab. Zu erwarten sind mit der Pfadlänge steigende Kosten für alle Peers. Je länger die Pfade, desto mehr Weiterleitungen muss der einzelne Peer pro Runde erbringen. Ebenfalls kann erwartet werden, dass sich die optimale Diskriminierung genau bei den 10.000 Peers einstellt, für die die Parameter optimiert wurden.

Abbildung 3.21 setzt die Zahl der Peers ins Verhältnis zu den Kosten der kooperativen und unkooperativen Peers (rechte y-Achse, gestrichelte Linien) sowie zu der Diskriminierung (linke y-Achse, durchgezogene Linie). Tatsächlich stellt sich die optimale Diskriminierung (nahezu)

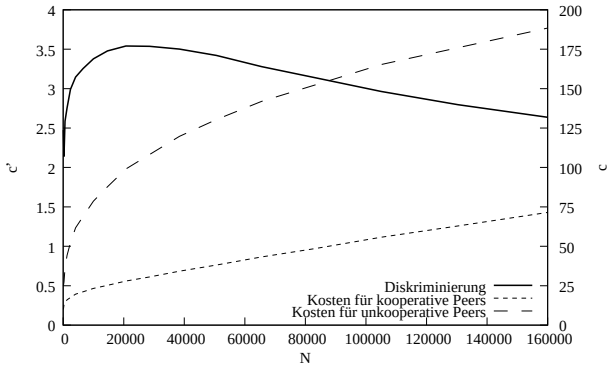

Abbildung 3.21: Skalierbarkeit von FairNet.

an der erwarteten Stelle im Diagramm ein. Die mit der Pfadlänge steigenden Kosten können ebenfalls beobachtet werden.

Darüber hinaus zeigt die Abbildung ein außerordentlich positives Ergebnis: selbst wenn die Zahl der Knoten beinahe um den Faktor 2000 von der kleinsten zur größten variiert, bleiben die Kosten der kooperativen Knoten erheblich kleiner als die der unkooperativen, und die Diskriminierung liegt stets über einem Wert von $2{,}5$! Das bedeutet, dass FairNet in der Realität – in der globales Wissen über exogene Parameter nicht zur Verfügung steht – auch mit Hilfe von Schätzwerten sehr wirksam gegen unkooperatives Verhalten vorgehen kann. Solche Schätzwerte können sehr leicht gewonnen werden; beispielsweise kann ein Peer aus dem Verhältnis seiner beantworteten und nicht beantworteten Anfragen die globale Ausfallwahrscheinlichkeit ermitteln oder aus der beobachteten Pfadlänge von eingehenden Nachrichten auf die Zahl der Peers im System schließen.

Das Verbreiten von Feedback FairNet basiert darauf, dass Peers ihr Feedback anderen Knoten freiwillig und korrekt zur Verfügung stellen. Mit dem Problem des gefälschten Feedbacks beschäftigt sich Abschnitt 5; hier wird nun diskutiert, *warum* Peers ihr Feedback untereinander austauschen sollten.

Zunächst kann argumentiert werden, dass das Austauschen von Feedback nahezu nichts kostet. Feedback wird an Nachrichten angehängt, die ohnehin zwischen den Peers ausgetauscht werden. Und die Repositories werden von jedem Peer schon deswegen gepflegt, um selbst unkooperative Peers erkennen zu können. Teile des Repositories auszuwählen und an Nachrichten anzuhängen ist also unaufwendig.

Einem anderen Peer das eigene Feedback zur Verfügung zu stellen bringt zunächst keinen unmittelbaren Vorteil. Es ist jedoch leicht, eine einfache 'Tit-For-Tat'-Erweiterung in das Protokoll aufzunehmen und experimentell zu evaluieren. Hier erhält jeder Peer nur dann Feedback von seinem Nachbarn, wenn er ihm selbst Feedback übermittelt. Hat ein Peer nicht ausreichend positives Feedback über einen anderen, so muss er dessen Kooperationswilligkeit mittels Arbeitsbeweis testen. Es ist zu erwarten, dass ein Peer, der nur auf eigene Beobachtungen angewiesen ist und kein Feedback von seinen Nachbarn 'aus zweiter Hand' bekommt, erheblich

mehr Arbeitsbeweise verlangen muss als ein Peer, der sein Feedback mit anderen abgleichen kann. Ein Peer, der sehr viel mehr Arbeitsbeweise verlangt als alle seine Nachbarn wird jedoch bald gemieden und von der Anfrageverarbeitung ausgeschlossen (vgl. Abschnitt 3.6).

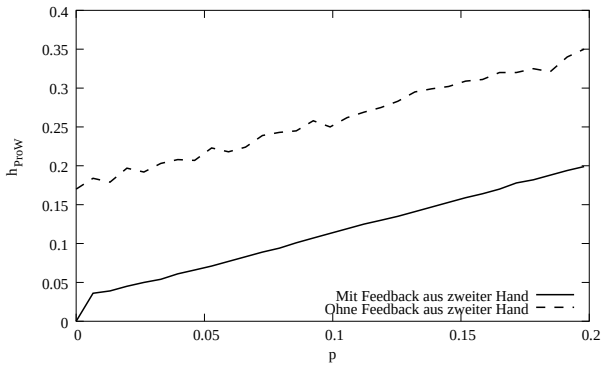

Abbildung 3.22: Zahl der Arbeitsbeweise mit und ohne Feedback-Austausch.

Eine geringe Zahl der verlangten Arbeitsbeweise ist also ein Indikator für den Vorteil, den ein Peer aus dem Feedback seiner Nachbarn zieht. Ein Experiment soll klären, wie groß dieser Vorteil in der Realität ist. Dazu wurde die oben genannte Tit-For-Tat Strategie implementiert. Im Experiment tauschen 10% von 10.000 Peers kein Feedback mit ihren Nachbarn aus. Dabei wurde die globale Ausfallwahrscheinlichkeit variiert.

Abbildung 3.22 stellt nun die globale Ausfallwahrscheinlichkeit p der Zahl der verlangten Arbeitsbeweise gegenüber. Die Abbildung zeigt, dass Peers, die kein Feedback von ihren Nachbarn erhalten (gestrichelte Linie) erheblich mehr Arbeitsbeweise verlangen müssen als Peers, die ihr Feedback mit anderen austauschen. In Szenarios mit einer sehr geringen globalen Ausfallwahrscheinlichkeit müssen Peers, die kein Feedback von anderen bekommen, bis zu vier mal mehr Arbeitsbeweise anfordern. Bei einer sehr hohen Ausfallrate von 20% sind es immer noch doppelt so viele. Ein rationaler Peer wird stets solche Knoten bevorzugen, die weniger Arbeitsbeweise von ihm verlangen. Nachbarn, die doppelt so viele Arbeitsbeweise verlangen wie der Durchschnitt, können sehr leicht erkannt und gemieden werden; ihnen entsteht durch den verweigerten Feedback-Austausch also ein erheblicher Nachteil.

Die experimentelle Evaluierung hat nachgewiesen, dass das FairNet-Protokoll nicht nur analytisch, sondern auch in einer prototypischen Implementierung und unter realitätsnahen Umgebungsbedingungen unkooperatives Verhalten zuverlässig unattraktiv machen kann. Insbesondere wurde gezeigt, dass die Modellannahmen der Analyse nur geringe Auswirkungen auf die Anwendbarkeit von FairNet in der Praxis verursachen, und dass das Protokoll auch bei großen Veränderungen in den Umgebungsparametern funktionsfähig bleibt.

3.6 Angriffe gegen FairNet

Dieser Abschnitt soll die Anfälligkeit von FairNet gegenüber Angriffen untersucht werden. Dabei soll noch einmal betont werden, dass FairNet *nicht* dazu dient, Angriffe auf P2P-Datenstrukturen zu verhindern. Vielmehr wurde es entwickelt, um kooperatives Verhalten für rationale Peers attraktiv zu machen. Dennoch ist die Robustheit von FairNet gegenüber verschiedensten Angriffen eine kritische Voraussetzung für den Betrieb in der Praxis. Angriffe auf das Protokoll einer P2P-Datenstruktur können sowohl durchgeführt werden, um Ressourcen zu sparen, als auch um andere Peers oder das System als ganzes zu beeinträchtigen. Die Annahme, dass alle Peers rational und risikoneutral sind, wird hier explizit aufgehoben: Böswillige Angreifer können erheblich mehr Ressourcen aufwenden, als kooperative Mitarbeit an der P2P-Datenstruktur kostet. Untersucht werden sollen sowohl altbekannte Attacken wie zum Beispiel Denial-of-Service-Angriffe, als auch Angriffstypen, die für das FairNet-Protokoll spezifisch sind. Dabei wird jeweils diskutiert, wie leicht oder schwer sich der jeweilige Angriff durchführen lässt, und welche Auswirkungen er auf einzelne Peers und das Gesamtsystem hat. Leicht durchführbare Angriffe können dabei von einem einzelnen Peer realisiert werden. Schwer durchführbare Angriffe lassen sich nur als Kollaborationsattacke ausführen, d.h., es werden die vereinigten Ressourcen mehrerer Peers benötigt. Detailliertere Analysen einiger Angriffe können in [RBBW05] eingesehen werden.

Bis jetzt existieren keine effektiven Ansätze für ein verteiltes Benutzer- und Rechtemanagement in P2P-Datenstrukturen. Zudem verwalten Peers in ihrer eigenen Zone die Daten, die andere in die Struktur eingespeist haben, und können diese jederzeit beliebig verändern. Ein Datenverlust wird in dieser Umgebung dadurch verhindert, dass Peers die Integrität der von ihnen selbst eingefügten Daten überwachen (z.B. [CN03]), durch 'm-aus-n'-Kodierung (*Erasure Codes*, z.B. [LEB+03]) oder durch verschiedene Arten der Datenreplikation (z.B. *Multiple Realities*, [RFH+01]). Die Wirksamkeit diese Ansätze wird vom FairNet-Protokoll nicht beeinträchtigt. Deswegen sollen im folgenden nur Angriffe auf das FairNet-Protokoll berücksichtigt werden, nicht aber Angriffe auf die Verfügbarkeit des Systems oder die Integrität der gespeicherten Daten. Ebenfalls nicht beachtet werden Angriffsmöglichkeiten, die durch fehlerhafte Implementierungen entstehen können.

Man-in-the-Middle-Angriffe Diese Art von Angriffen ist in Computer-Netzen altbekannt. Der Angriff wird ausgeführt, indem sich bei einem Kommunikationsvorgang zwischen zwei Teilnehmern A und B ein dritter Peer C dazwischenschaltet. C gibt nun gegenüber A vor, B zu sein, und behauptet B gegenüber, er wäre A. Dadurch kann C falsche Daten in den Kommunikationsvorgang einbringen. Derartige Man-in-the-Middle-Angriffe lassen sich durch Verschlüsselung der Kommunikation zwischen beiden A und B verhindern, zum Beispiel über IPSec [KS05].

Damit kann jedoch nur die direkte Kommunikation zwischen zwei benachbarten Peers geschützt werden. In P2P-Datenstrukturen werden Nachrichten hingegen über Ketten von Weiterleitern vom Urheber zum Zielpeer transportiert, und jeder dieser Weiterleiter ist ebenfalls in der Lage, die Nachricht zu verändern. Eine Verschlüsselung ist hier nicht möglich, weil

sich Urheber und Zielpeer nicht kennen und sichere Protokolle zum verteilten Generieren von sicheren Identitäten zu teuer bzw. nicht für P2P-Systeme mit Millionen von Teilnehmern geeignet sind [Dou02]. Gegen die Verfälschung von Nachrichten während des Weiterleitens existieren jedoch etliche Maßnahmen: so ist es möglich, Nachrichten zu replizieren und über unterschiedliche Wege zum Ziel zu schicken [CDG+02, RBBW05]. Protokolle wie BST [CL99] garantieren dabei die Unversehrtheit der Daten, solange eine Mindestanzahl von replizierten Nachrichten unversehrt beim Empfänger eintrifft.

Eine andere Möglichkeit besteht darin, dass jeder Weiterleiter die Nachricht kryptographisch signiert, bevor er sie an den nächsten Peer weitergibt. Jeder Weiterleiter kann mit der angehängten Liste der Signaturen stichprobenhaft die Gültigkeit einer Signatur und damit die Unversehrtheit der Nachricht zu einem bestimmten Zeitpunkt bei der Weiterleitung prüfen (s. Zero-Knowledge-Proof [BOGKW88]), ohne dazu a priori-Wissen über den Aussteller der Signatur zu benötigen.

Kann der Empfänger einer Nachricht deren Unversehrtheit prüfen, so kann auch FairNet für die Bestrafung der Angreifer herangezogen werden. Generieren die Peers für verfälschte Nachrichten negatives Feedback, wird der Angreifer gezwungen, zahlreiche teure Arbeitsbeweise zu erbringen. Das Modell in Abschnitt 3.4 ist dazu nur um eine zusätzliche Ursache für das Generieren von Feedback zu ergänzen. Damit ist es möglich, aus einem ressourcenschonenden Angriff (im Vergleich zu Denial-of-Service-Attacken) einen sehr teuren zu machen, bzw. eventuellen ökonomischen Vorteilen durch das Verfälschen von Nachrichten entgegenzuwirken.

Gefälschte Anfrageergebnisse Um die Kosten für Datenhaltung und Anfrageverarbeitung zu sparen, könnte ein Knoten auf Anfragen mit gefälschten Ergebnissen anstelle korrekter Daten antworten.

Gefälschten Antworten kann mit Replikation begegnet werden. Der Anfragesteller sendet seine Anfrage an mehrere Replikate parallel, und bildet ein Quorum aus den zurückgegebenen unterschiedlichen Antworten. Je nach Anwendung lassen sich gefälschte Antworten auch ohne diese Maßnahme entdecken. Bei einer verteilten WWW-Suchmaschine beispielsweise kann der Anfrager unmittelbar nach Erhalt der Antwort prüfen, ob die gelieferte URL zu seiner Anfrage passt (vgl. [GWB+01] für eine Aufzählung relevanter Anwendungen). Ebenso wie im Zusammenhang mit Man-in-the-Middle-Angriffen kann FairNet auch hier so erweitert werden, dass der Aussteller von gefälschten Anfrageergebnissen mit Arbeitsbeweisen bestraft wird.

(Distributed) Denial-of-Service-Angriffe Diese Angriffe werden ausgeführt, indem ein oder mehrere Angreifer so viele Anfragen an einen einzelnen Knoten stellen, dass dieser zu überlastet ist, um reguläre Anfragen anderer Teilnehmer zu beantworten. Beim Einsatz des FairNet-Protokolls können statt einfacher Anfragen die vielfach teureren Arbeitsbeweise für eine Denial-of-Service (DoS) Attacke verwendet werden.

Es ist schwer, DoS-Angriffe von regulären Anfragen zu unterscheiden. Es gibt jedoch statistische Methoden, die eine Abschätzung leisten. Beispielsweise kann ein Knoten die Zwischenankunftszeit von Anfragen eines bestimmten Knotens messen und diese mit einem Tiefpass filtern. Ein Knoten wird als Angreifer markiert, wenn die Zwischenankunftszeit seiner Anfragen den Durchschnitt über alle Knoten signifikant unterschreitet. Weitere Ansätze basieren auf der lokalen, globalen oder verteilten Clusteranalyse (eine Übersicht bietet [Per04]). Die einfachste Maßnahme gegen solche Angriffe besteht darin, den Datenverkehr von Angreifern einfach nicht zu beantworten bzw. nicht weiterzuleiten. Die Wirksamkeit dieser Gegenmaßnahmen ist unabhängig davon, ob reguläre Anfragen oder die Arbeitsbeweise von FairNet als DoS-Attacke verwendet werden.

Des weiteren sind die Auswirkungen von DoS-Angriffen auf einzelne Peers begrenzt. Es wurde bereits am Anfang dieses Abschnitts motiviert, wie Datenverluste durch den Ausfall einzelner Knoten wirksam verhindert werden können. Das FairNet-Protokoll selbst basiert nur auf lokalen Informationen, die ein Peer entweder direkt beobachten kann oder von seinen unmittelbaren Nachbarn erhält. Von DoS-Angriffen ist daher stets nur ein einzelner Peer betroffen, das System als ganzes bleibt unbeeinträchtigt.

Stellvertreter-Angriffe Ein viel versprechender Weg zur Reduktion der Kosten mehrerer Teilnehmer besteht darin, dass sich mehrere Teilnehmer zusammentun und gemeinsam einen Peer als 'Proxy' betreiben. Der Proxy arbeitet kooperativ in der P2P-Datenstruktur mit, sendet aber Anfragen im Auftrag aller angeschlossenen Teilnehmer ab. Von allen Teilnehmern ist also nur ein Rechner im P2P-System sichtbar.

Es lässt sich diskutieren, ob es sich bei der Verwendung eines Peers als Proxy um einen Angriff oder um ein legitimes Verhalten handelt. Auf der einen Seite ermöglicht ein Proxy einer Vielzahl von Knoten die Teilnahme an der P2P-Datenstruktur zu den Kosten eines einzigen Knotens. Andererseits verhält sich der eine im System präsente Knoten kooperativ, und schadet der Datenstruktur nicht. Weiterhin ist es aus Sicherheitsgründen häufig erforderlich, lokale Netzwerke hinter einer Firewall zu platzieren. Ein Proxy auf der Firewall (oder einem Rechner in der 'demilitarisierten Zone') ist in solchen Fällen die einzige Möglichkeit für eine Gruppe von Teilnehmern, an der Datenstruktur zu partizipieren. Eine Lösung dieses Problems könnte in der Anwendung 'fairer' Lastausgleichsmechanismen bestehen, die einem Knoten, der viele Anfragen stellt, einen hohen Anteil der Arbeitslast zuteilt. Dazu existieren jedoch erst einleitende Vorarbeiten [Knö03].

Unregelmäßige oder angesparte Anfragen Bisher galt die Annahme, dass die Peers regelmäßig Anfragen absetzen (eine pro Runde). Diese Annahme wurde damit begründet, dass die Hauptanwendungsgebiete von P2P-Datenstrukturen (vgl. [GWB+01]) häufige und regelmäßige Operationen erfordern. Doch wie ändert sich die Effektivität von FairNet, wenn das Anwendungsszenario ein Ansammeln von Anfragen erlaubt? Ein Peer kann nun seine Kosten reduzieren, indem er sich unkooperativ verhält seine Anfragen über einen längeren Zeitraum sammelt. Von Zeit zu Zeit erbringt er nun genügend Arbeitsbeweise, um als kooperativ zu

gelten, und sendet die angesammelten Anfragen auf einmal ab. Nachdem er seine Ergebnisse erhalten hat, verhält er sich wieder unkooperativ und sammelt erneut Anfragen an.

Ein vollständig unkooperativer Peer mit ausschließlich negativem Feedback muss $\lceil t/q_{ProW} \rceil$ Arbeitsbeweise erbringen, bis er als kooperativ anerkannt wird. Die Kosten dafür betragen $c_u = c_{ProW} \cdot \lceil t/q_{ProW} \rceil$. Auf der anderen Seite muss ein kooperativer Knoten jede Runde Kosten von $c_c = h_{answer} \cdot c_{answer} + h_{forward} \cdot c_{forward} + h_{ProW} \cdot c_{ProW}$ tragen. Damit sich ein unkooperativer Peer verbessern kann, muss er seine Anfragen daher mindestens $\lceil c_u/c_c \rceil$ Runden lang ansammeln. Mit den Parametern aus den Tabellen 3.1 bis 3.3 ist diese Gewinnschwelle nach 24 Runden erreicht. Ein Knoten, der mehr als 24 mal so lange auf seine Anfrageergebnisse warten kann wie ein kooperativer Peer, kann sich also durch das Ansammeln von Anfragen und das Erbringen von Arbeitsbeweisen verbessern. In den meisten Anwendungen von P2P-Datenstrukturen ist so eine lange Wartezeit jedoch nicht praktikabel. Weiterhin lässt sich diese Schwelle durch das Verteuern der Arbeitsbeweise noch weiter erhöhen. Und zuletzt ist fraglich, warum ein Knoten, der über einen langen Zeitraum nicht kooperativ mitarbeitet, überhaupt im P2P-System verbleiben sollte – beim Anmelden muss ein Neuankömmling genauso viele Arbeitsbeweise erbringen wie ein vollständig unkooperativer Peer (vgl. Abschnitt 3.2).

Es bleibt als Fazit dieses Abschnitts festzuhalten, dass (1) abgesehen von extremen Szenarien, beispielsweise Angreifern mit unbegrenzten Ressourcen, FairNet von Angriffen nahezu unbeeinträchtigt funktionsfähig bleibt. (2) Des weiteren wurde gezeigt, dass einzelne Peers auch von Angriffen gegen die FairNet-Infrastruktur nicht profitieren können. (3) Reparaturmechanismen der Datenstruktur, Replikation oder Erasure-Codes lassen sich parallel zum FairNet-Protokoll betreiben, um Datenverluste zu verhindern.

3.7 Alternative Protokollvarianten

In der Einleitung dieses Kapitels wurde der grundsätzliche Ablauf bei der Verarbeitung einer Anfrage vorgegeben. Dieser Ablauf entstand aus der Überlegung heraus, FairNet möglichst wenig anfällig gegen Manipulationen zu gestalten. Aus dem Anforderungsprofil einer Anwendung kann sich jedoch die Anforderung ergeben, eine abweichende Protokollvariante zu implementieren. Die Realisierbarkeit von zwei möglichen Alternativvarianten soll im folgenden diskutiert werden.

Es wird nur positives Feedback vergeben In FairNet wird sowohl positives als auch negatives Feedback verwendet. Es existieren jedoch Ansätze, die ausschließlich mit positivem [XL02] oder negativem [AD01, BB02] Feedback auskommen. Ausschließlich negatives Feedback (also Beschwerden) zu verwenden ist problematisch in Systemen, in denen die Teilnehmer ihre Identität sehr leicht wechseln können [FR98]. Ein Peer, über den sich viele Beschwerden angesammelt haben, kann das System verlassen und mit einem neuen Pseudonym wieder betreten. Gegen Systeme, die nur positives Feedback verwenden, bestehen diese Vorbehalte nicht.

Wird auf negatives Feedback verzichtet, gibt es keine Möglichkeit mehr, dass ein Peer einen anderen mittels einer Beschwerde über das Fehlverhalten eines Knotens informiert. Stattdessen ist das Ausbleiben von neuem, positivem Feedback ein Indiz dafür, dass ein Peer nicht kooperativ mitarbeitet. Damit Peers kein positives Feedback 'ansparen' können, um nach einer initialen Phase der Kooperation jede Mitarbeit ungestraft zu verweigern, muss das gegebene Feedback in diesem Szenario über die Zeit veralten [BB04a].

Wie ist nun diese Protokollvariante auszugestalten? Alle Gleichungen mit einem Bezug auf die Anzahl positiven Feedbacks in Abschnitt 3.4 können weiterhin angewendet werden. Das bedeutet, dass positives Feedback mit den in den Gleichungen 3.8 und 3.12 angegebenen Zahlen pro Runde generiert wird. Negatives Feedback entfällt ersatzlos. Des weiteren verliert alles positive Feedback seine Gültigkeit, das älter ist als m Runden. Wieder gilt ein Peer als unkooperativ, wenn über ihn weniger als t positive Feedback-Objekte in einem Repository der Kapazität s vorhanden sind. Dann muss die Gültigkeitsdauer m so groß sein, dass das in m Runden generierte Feedback abzüglich dem in der aktuellen Runde verfallenen Feedback mindestens so groß ist wie t. Gleichung 3.32 zeigt diesen Zusammenhang:

$$m \geq \frac{t + h_{forward,pos} + h_{answer,pos}}{h_{forward,pos} + h_{answer,pos}} \tag{3.32}$$

Entsprechend muss auch das Feedback-Repository der einzelnen Peers genügend Kapazität aufweisen, um $s = m \cdot (h_{forward,pos} + h_{answer,pos})$ Feedback-Objekte aufzunehmen.

Der größte Nachteil dieser Protokollvariante besteht darin, dass alle Peers mit der gleichen Rate positives Feedback erwerben müssen, d.h. es muss ein perfekter Lastausgleich zwischen allen Knoten im System erfolgen. Knoten, die wenig Arbeit von anderen erhalten, können auch nur wenig positives Feedback durch ihre Mitarbeit erwerben. Das bedeutet, dass ein Peer, der beispielsweise für wenig nachgefragte Datentupel im Schlüsselraum verantwortlich ist, leicht als unkooperativ eingeschätzt werden kann, weil er positives Feedback in größeren Zeitabständen erwirbt als andere Peers. Alternativ kann auch das Veralten von Feedback über einen längeren Zeitraum erfolgen; in diesem Falle können aber Knoten, die für 'beliebte' Datentupel zuständig sind, positives Feedback ansparen und einen Teil der Anfragen pro Zeiteinheit ungestraft unbeantwortet lassen. Vergleichbare Probleme entstehen durch über die Zeit stochastisch schwankende Anfrageraten oder vergleichbare Phänomene.

Antworten werden auf dem gleichen Weg zurückgeschickt Im vorgestellten FairNet-Protokoll werden die Antworten auf Anfragen direkt vom Beantworter an den Anfrager zurückgeschickt. Eine separate Feedback-Benachrichtigung informiert jeden an der Bearbeitung involvierten Peer darüber, ob die Anfrage beantwortet oder nicht beantwortet wurde. Unter Umständen ergeben sich daraus Probleme: so ist die Feedback-Benachrichtigung vom Anfrageergebnis entkoppelt. Ein Peer, der ein korrektes Anfrageergebnis erhält, könnte trotzdem eine negative Feedback-Benachrichtigung absenden[5]. Des weiteren können ungünstig platzierte Firewalls oder schlecht konfigurierte Router zwar die Weiterleitung der Anfrage von Peer zu

[5]Dieses Problem wird in Abschnitt 5.6 diskutiert.

Peer zulassen, eine direkte Kommunikation zwischen Anfragesteller und -beantworter jedoch möglicherweise verhindern.

Eine Lösung für diese Probleme kann darin bestehen, die Antworten auf dem gleichen Weg zurückzuschicken, auf dem die Anfrage zum Beantworter weitergeleitet wurde. Jeder Peer, der eine Antwortnachricht weiterleitet, generiert positives Feedback über den Weiterleiter. Der Ablauf der Anfrageverarbeitung ist nun also wie folgt:

1. Der Anfragesteller generiert eine Anfrage, die mit einem Schlüssel adressiert ist.
2. Kann ein Peer eine Anfrage nicht aus seiner Zone beantworten, sendet er sie an einen vertrauenswürdigen Kontakt weiter, dessen Zone dem Anfrageschlüssel am nächsten liegt. Daneben merkt sich der Peer, von welchem Knoten er die Anfrage erhalten hat.
3. Der Peer, in dessen Zone der Anfrageschlüssel passt, sendet die Antwort auf die Anfrage zu dem Knoten zurück, von dem er sie unmittelbar erhalten hat.
4. Erhält ein Peer eine Antwortnachricht, leitet er sie an den Knoten weiter, von dem er die zuvor die zugehörige Anfrage erhalten hatte. Daneben generiert er positives Feedback über den Peer, der ihm die Antwort überstellt hat.
5. Erhält ein Peer nach einer vorgegebenen Zeitspanne keine Antwortnachricht zum Zurücksenden, so generiert er negatives Feedback über den Peer, an den er zuvor die Anfrage weitergeleitet hatte.

Weil die Antwort auf die Anfrage nun nicht nur vom Anfragesteller zum Beantworter, sondern auch wieder zurück weitergeleitet wird, halbiert sich Wahrscheinlichkeit, dass eine Transaktion korrekt verarbeitet wird. Gleichung 3.33 bestimmt die Wahrscheinlichkeit dafür, dass ein Peer eine Antwort auf eine Anfrage erhält, wenn der Beantworter l Peers vom Anfragesteller entfernt ist. Die durchschnittliche Wahrscheinlichkeit für die gesamte Datenstruktur wird von Gleichung 3.34 ermittelt (vgl. Gleichungen 3.5 und 3.6).

$$f_{\mathfrak{f}}(l) = \begin{cases} 1 & \text{wenn } l = 0 \\ (1-p)^{2 \cdot l - 1} & \text{wenn } l > 0 \end{cases} \tag{3.33}$$

$$h'_{answer} = \iint\limits_{x_1, x_2 \in [0;1]^d} f_{\mathfrak{f}}(dist(x_1, x_2)) dx_1 dx_2 \tag{3.34}$$

Für positives Feedback sind auch weiterhin zwei Durchläufe notwendig: die Aufgabe der Feedback-Benachrichtigung im FairNet-Protokoll übernimmt hier die zurückgesendete Antwortnachricht. Die Zahl des zu erwartenden positiven Feedbacks bleibt also wie in den Gleichungen 3.8 und 3.12 berechnet. Dagegen ändert sich die Zahl des generierten negativen Feedbacks, weil nun jeder Peer selbständig negatives Feedback vergibt, wenn nach einiger Zeit noch keine Antwort-Nachricht eingetroffen ist. Damit ein unkooperativer Peer negatives Feedback für das Nichtbeantworten von Anfragen erhält, muss also nicht zuerst eine Feedback-Benachrichtigung vom Anfragesteller den Vorgänger des unkooperativen Peers erreichen, sondern nur die Antwort innerhalb einer vorgegebenen Zeitspanne ausbleiben. Somit erhöht sich

die Zahl des negativen Feedbacks $h'_{answer,neg}$, das für das Nichtbeantworten vergeben wird, von Gleichung 3.10 auf den in Gleichung 3.36 bestimmten Wert. Die Hilfsfunktion 3.35 dient wieder dazu, die Einzelwahrscheinlichkeit für negatives Feedback über eine gegebene Zahl von l Weiterleitern hinweg zu bestimmen (vgl. Gleichung 3.9).

$$f_{\mathfrak{g}}(l) = \begin{cases} 0 & \text{wenn } l = 0 \\ (1-p)^{l-1} \cdot p & \text{wenn } l > 0 \end{cases} \tag{3.35}$$

$$h'_{answer,neg} = q_{answer} \cdot q_{pn} \cdot \iint\limits_{x_1,x_2 \in [0;1]^d} f_{\mathfrak{g}}(dist(x_1,x_2)) dx_1 dx_2 \tag{3.36}$$

Auch die Wahrscheinlichkeit, negatives Feedback für ein verweigertes Weiterleiten zu erhalten, wird in dieser Protokollvariante erhöht. Im bisher beschriebenen FairNet-Protokoll musste nach einer verweigerten Weiterleitung eine Feedback-Benachrichtigung bis zum Vorgänger des Peers gelangen, über den negatives Feedback generiert werden sollte (Gleichung 3.14). Nun genügt es, dass eine Antwort-Nachricht nicht bis zum Vorgänger dieses Peers gelangt. Die Zahl des dabei generierten Feedbacks $h'_{forward,neg}$ wird von den Gleichungen 3.37 und 3.38 bestimmt.

$$f_{\mathfrak{h}}(l) = \begin{cases} 0 & \text{wenn } l < 2 \\ p \cdot \left((l-1)(1-p)^{l-1} + \sum\limits_{i=1}^{l-1} i \cdot \left((1-p)^{i-1} + (1-p)^{2 \cdot l - i} \right) \right) & \text{wenn } l \geq 2 \end{cases} \tag{3.37}$$

$$h'_{forward,neg} = q_{forward} \cdot q_{pn} \cdot \iint\limits_{x_1,x_2 \in [0;1]^d} f_{\mathfrak{h}}(dist(x_1,x_2)) \; dx_1 dx_2 \tag{3.38}$$

Nun soll untersucht werden, wie sich FairNet ohne Feedback-Nachrichten im Vergleich zu der in Kapitel 3 vorgestellten Variante verhält. Es ist zu erwarten, dass die Protokollvariante ohne Feedback-Benachrichtigung ein sehr ähnliches Verhalten zum bisher analysierten FairNet-Protokoll aufweist: der Feedback-Mechanismus selbst bleibt unverändert, und die erhöhte Wahrscheinlichkeit für negatives Feedback wird bei der in Abschnitt 3.5 beschriebenen numerischen Optimierung der Parameterwerte ausgeglichen. Um diese Hypothese zu bewerten, soll die der Abbildung 3.10 zugrundeliegende Analyse für beide Protokollvarianten wiederholt werden. Es wird also von den Parametern in Tabelle 3.3 ausgegangen, und die lokale Ausfallwahrscheinlichkeit q ist wieder nur doppelt so groß wie die globale Ausfallwahrscheinlichkeit $p = q/2$.

Die verwendeten Parameter (s. Tabelle 3.3) werden darauf optimiert, dass kooperativen Knoten durch das Protokoll Mehrkosten von 10% entstehen und die Kosten unkooperativer Knoten maximiert werden. Als Ergebnis der Optimierung verwendet das FairNet mit Feedback-Benachrichtigungen einen Schwellenwert von $t = 6$ und eine Relation von $q_{pn} = 2$ zwischen

der generierten Menge positiven und negativen Feedbacks, wohingegen die Variante ohne Benachrichtigung die Parameter $t = 7$ und $q_{pn} = 1$ verwendet. Die Parameter, die die Menge des generierten positiven Feedbacks bestimmen, sind identisch. Gemessen wird nun die Diskriminierung c' bei unterschiedlichen Ausfallwahrscheinlichkeiten.

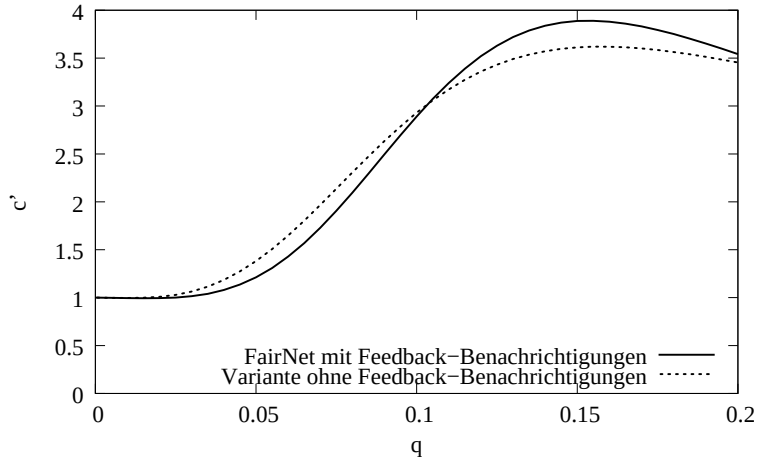

Abbildung 3.23: Diskriminierung mit und ohne Feedback-Benachrichtigungen.

Die Abbildung 3.23 zeigt, dass die Erwartung zutrifft: beide Kurven im Diagramm sind sehr ähnlich. Die Diskriminierung des Protokolls mit Feedback-Benachrichtigungen ist geringfügig höher, d.h., unkooperative Peers haben etwas größere Kosten zu tragen. Der Abstand zwischen den Kurvenverläufen ist jedoch so gering, dass schon eine Veränderung des Schwellenwertes um den Betrag 1 bei einer der beiden Kurven eine deutlichere Abweichung verursachen würde. Es gibt also aus der Sicht der Wirksamkeit keine praxisgerechten Argumente dafür, die eine oder die andere Protokollvariante zu bevorzugen. Anders sieht es bei der Anfälligkeit gegen Angriffe aus: hier hat die Protokollvariante ohne Feedback-Benachrichtigungen erhebliche Schwächen aufzuweisen. Zunächst werden bei dieser Variante die kostbarsten Informationen – die Antworten auf Anfragen – über einen doppelt so langen Weg weitergeleitet, und somit einer erhöhten Ausfallwahrscheinlichkeit preisgegeben. Ein noch stärkeres Argument gegen die Protokollalternative hat ihre Ursache darin, dass es in den meisten Anwendungsszenarien sehr aufwendig ist, die Korrektheit der übermittelten Antwort zu prüfen. Wenn es aber nicht realisierbar ist, dass *jeder* Weiterleiter die Korrektheit der Antwort prüft, generieren die Weiterleiter Feedback für ein nicht verifizierbares Ergebnis. Das bedeutet, ein Peer bekommt auch dann positives Feedback, wenn er eine Anfrage nicht korrekt weiterleitet, sondern sie stattdessen direkt mit einer gefälschten Antwort selbst beantwortet. Erst der Anfragesteller kann die Fälschung aufdecken, aber kein negatives Feedback mehr für den Verursacher generieren.

3.8 FairNet in anderen P2P-Datenstrukturen

Dieser Abschnitt soll zeigen, dass das FairNet-Protokoll auch für andere P2P-Datenstrukturen neben Content-Addressable Networks geeignet ist. In [AAG+05] wird beschrieben, wie ei-

ne Abstraktion von einer konkreten Implementierung einer P2P-Datenstruktur erreicht werden kann. Grundsätzlich kann FairNet in allen Datenstrukturen angewendet werden, in denen zahlreiche 'billige' Anfragen von Peer zu Peer weitergeleitet werden, und in denen die Kontakte eines Peers über einen längeren Zeitraum stabil bleiben. Zahlreiche Anfragen werden benötigt, damit ein Peer genug Gelegenheit hat, das Verhalten seiner Kontakte zu beobachten. Mit 'billig' ist gemeint, dass der Verlust einiger Anfragen aus Anwendungssicht akzeptabel ist – FairNet kann nicht verhindern, dass vereinzelt Anfragen an unkooperative Teilnehmer weitergeleitet werden und verloren gehen. Die Anforderung nach stabilen Kontakten leitet sich aus den Tatsachen ab, dass Neueinsteiger stets ressourcenintensive Arbeitsbeweise erbringen müssen, und dass das Protokoll von Beobachtungen über das Verhalten der Peers in der Vergangenheit abhängt (vgl. Gleichung 3.29 zur Dynamik des Protokolls). Um FairNet an eine andere P2P-Datenstruktur anzupassen, sind (neben der Implementierung der Software) nur zwei Schritte erforderlich:

1. Entwicklung einer Distanzfunktion und Bestimmung der optimalen Protokollparameter.
2. Entwurf einer Methode zur Feedback-Verbreitung, die auf die neue Datenstruktur abgestimmt ist.

Die Gleichung 3.2 zur Bestimmung der Zahl der Weiterleiter zwischen zwei Punkten x_1 und x_2 im Schlüsselraum ist der einzige Teil der analytischen Untersuchungen von FairNet, der für Content-Addressable Networks spezifisch ist. Alle anderen Gleichungen, die die Ausfallwahrscheinlichkeiten, die Rate der Feedback-Erzeugung oder die Wahrscheinlichkeiten für Repositories mit einer bestimmten Zahl positiven Feedbacks ermitteln, sind nur über diese Distanzfunktion von der darunterliegenden Datenstruktur abhängig. Für die Anpassung an eine neue Datenstruktur genügt es somit, die Distanzfunktion auszutauschen, um mit Hilfe der aufgestellten Gleichungssysteme optimale Parameterwerte für FairNet (s. Tabelle 3.3) zu ermitteln. Verschiedene Algorithmen zur Feedback-Verteilung in CAN und anderen P2P-Datenstrukturen werden in Kapitel 4 beschrieben.

Der Einsatz von FairNet in Chord Nun soll die Anpassung von FairNet an das Protokoll von Chord [SMLN+01] untersucht werden. Chord wurde gewählt, da es beispielhaft für eine ganze Klasse von P2P-Datenstrukturen steht, die sich durch eine Pfadlänge von $O(logN)$ auszeichnet. Zu dieser Klasse gehören auch P-Grid [Abe01], Pastry [RD01] und Tapestry [ZKJ01]. Des weiteren ist Chord wegen seiner schlanken Protokollstruktur leicht analytisch zu beschreiben, und ist aus dem selben Grund in einigen Anwendungen zu finden, z.B. im Dateisystem *CFS* [DKK+01] oder in einem *Replica Location Service* [CCF04].

Chord wurde in Abschnitt 2.3 bereits kurz beschrieben. Die Struktur basiert auf einem ringförmigen Schlüsselraum, auf dem Nachrichten in Richtung aufsteigender Schlüsselwerte weitergeleitet werden. Jeder Knoten kennt – ausgehend von seiner eigenen Position auf dem Schlüsselraum – weitere Kontakte im Schlüsselraum, die für Zonen in jeweils verdoppeltem Abstand $(2^0, 2^1, \cdots, 2^x)$ verantwortlich sind. Hier wird der Schlüsselraum von Chord als Ring mit einem Wertebereich im Intervall $[0; 1]$ modelliert. In Anlehnung an Abschnitt 3.4 wird von einer regelmäßigen Aufteilung des Schlüsselraums zwischen allen N Peers ausgegangen. Das bedeutet, jeder Peer ist verantwortlich für eine Zone mit der Breite $1/N$.

Da der Schlüsselraum von Chord als Ring organisiert ist, werden Nachrichten, die die Schlüsslraumgrenze 1 erreichen, an den Peer weitergegeben, der für das mit 0 beginnende Intervall im Schlüsselraum verantwortlich ist. Gleichung 3.39 berechnet zunächst den Abstand l, den eine Nachricht unter Berücksichtigung dieses 'Umbruchs' im Schlüsselraum zurücklegen muss. Aus der Zonenbreite und der Kontaktwahl folgt, dass die Nachricht mit jedem Weiterleitungsschritt mindestens die Hälfte der Entfernung zum Zielpunkt überwindet. Der Zielpeer ist erreicht, wenn der Abstand zum Ziel kleiner ist als die Zonenbreite $1/N$. Die Hilfsfunktion 3.40 bestimmt auf diese Weise die Zahl der Peers, die in eine Weiterleitung über eine Distanz l involviert sind.

$$dist(x_1, x_2) = \begin{cases} f_{\text{i}}(x_2 - x_1) & \text{wenn } x_1 \leq x2 \\ f_{\text{i}}(1 + x_2 - x_1) & \text{sonst} \end{cases} \tag{3.39}$$

$$f_{\text{i}}(l) = \begin{cases} 0 & \text{wenn } l \leq \frac{1}{N} \\ 1 + f_{\text{i}}\Big(l - \frac{1}{N} \cdot 2^{\lfloor \log_2(l \cdot N) \rfloor}\Big) & \text{sonst} \end{cases} \tag{3.40}$$

Nun soll untersucht werden, wie sich FairNet als Maßnahme gegen unkooperative Peers in Chord verhält. Um Vergleichbarkeit zu den Experimenten im CAN herzustellen, wird von einem Chord mit 10.000 Teilnehmern ausgegangen, wobei wieder die lokale Ausfallwahrscheinlichkeit der unkooperativen Teilnehmer q doppelt so groß ist wie die globale Ausfallwahrscheinlichkeit $p = q/2$. Ebenfalls wie in den vorangegangenen Experimenten werden die Parameter von FairNet numerisch so bestimmt, dass bei einer Repository-Größe von $s = 10$ die Kosten für unkooperative Teilnehmer mit $q = 10\%$ maximal werden, während kooperative Knoten nur 10% Mehrkosten durch Arbeitsbeweise tragen müssen. Dafür wurde nun ein Schwellenwert von $t = 7$ ermittelt, sowie die Wahrscheinlichkeiten $q_{pn} = 1$ und $q_{ProW} = 1$ für das Generieren von Feedback.

Abbildung 3.24 zeigt nun das Ergebnis eines Experiments, bei dem die lokale Ausfallwahrscheinlichkeit q der unkooperativen Teilnehmer variiert und die Diskriminierung c' gemessen wurde. Zum Vergleich ist im Diagramm die Diskriminierung aufgetragen, die in einem CAN mit Parametern aus Tabelle 3.3 und unter den gleichen Bedingungen gemessen wurde.

Die Abbildung zeigt, dass die Effektivität bei der Diskriminierung unkooperativer Teilnehmer wenig von der darunterliegenden Datenstruktur abhängt. Die Unterschiede zwischen CAN und Chord sind geringer als die Differenz der Messwerte zwischen dem algebraischen Modell und der experimentellen Evaluierung im CAN. Zusammenfassend kann also festgehalten werden, dass das FairNet-Protokoll so flexibel parametrisiert werden kann, dass es auch bei grundlegend veränderten Umgebungsbedingungen wirksam gegen unkooperatives Verhalten bleibt.

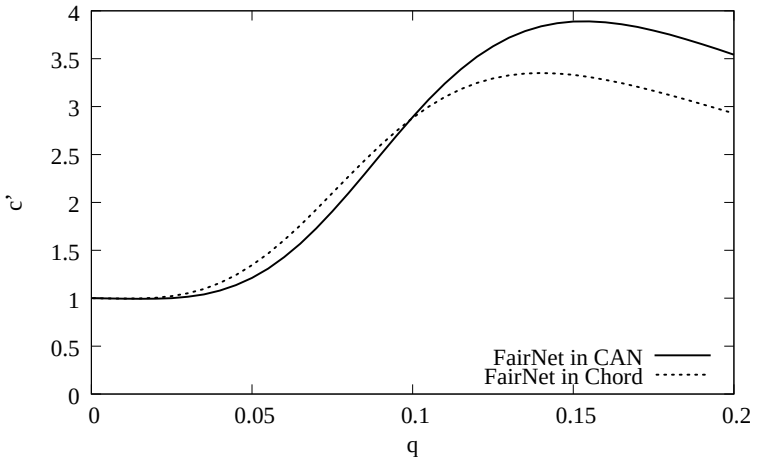

Abbildung 3.24: Diskriminierung in CAN und in Chord.

3.9 Zusammenfassung

Im ersten Teil dieses Kapitels wurde FairNet vorgestellt, ein Protokoll zum Umgang mit unkooperativen Teilnehmern in P2P-Datenstrukturen. Dabei wurde darauf eingegangen, wie die Anfrageverarbeitung in FairNet abläuft, und mit welchen Datenstrukturen und Methoden die Algorithmen von FairNet in eine bestehende P2P-Datenstruktur integriert werden können.

Im zweiten Teil des Kapitels wurde ein Kostenmodell sowie ein Gleichungssystem aufgestellt, mit dessen Hilfe die zahlreichen Parameter des Protokolls so bestimmt werden können, dass kooperativen Teilnehmern definierte Mehrkosten durch das Protokoll entstehen, während unkooperative Peers mit den höchstmöglichen Kosten beaufschlagt werden. Es wurde sowohl analytisch als auch experimentell an einem Content-Addressable Network nachgewiesen, dass FairNet mit den so ermittelten Parametersätzen in der Realität einsetzbar ist. Insbesondere wurde dabei das Verhalten des Protokolls in Grenzsituationen ermittelt. Dazu zählen unter anderem P2P-Datenstrukturen mit 50% unkooperativen Peers, die jeweils 20% der eingehenden Anfragen verwerfen; oder Knoten, die ihr Verhalten dynamisch ändern. Ebenso problematisch sind Szenarien, in denen sich die Zahl der Knoten um den Faktor 2.000 ändert, in denen einige Daten im Schlüsselraum häufiger angefragt werden, oder in denen manche Peers aktiver sind als andere. Die Evaluierung hat gezeigt, dass FairNet diesen Herausforderungen gewachsen ist: so kann eine gute Diskriminierung unkooperativer Peers selbst dann erreicht werden, wenn den kooperativen Teilnehmern nur 10% Mehrkosten entstehen, und Verhaltensänderungen werden auch unter realistischen Bedingungen in weniger als 2 Runden erkannt und abgestraft.

Der dritte Teil des Kapitels untersuchte Fragestellungen, die die Einsetzbarkeit von FairNet in offenen Systemen betreffen. Dazu wurde die Robustheit gegen Angriffe auf das Protokoll diskutiert. Des weiteren wurde gezeigt, wie FairNet an unterschiedliche Implementierungen bzw. unterschiedliche P2P-Datenstrukturen angepasst werden kann.

Kapitel 4

FairNet: Transport von Feedback

Die Verbreitung von Feedback ist ein Grundpfeiler von FairNet. In den Abschnitten 3.2 und 3.3 wurde dazu nur beschrieben, dass Feedback als Attachment an ohnehin versandte Nachrichten angehängt wird, und an welcher Stelle im Protokollablauf dies geschieht. In diesem Kapitel soll präzisiert werden, welche Alternativen bei der Verbreitung von Feedback zur Verfügung stehen, welche Design-Entscheidungen bei der Implementierung getroffen werden müssen und mit welcher Kapazität und Geschwindigkeit der Feedback-Transport abläuft [BA05]. Des weiteren soll diskutiert werden, wie sich der Feedback-Transport in anderen P2P-Datenstrukturen als dem hier zugrundegelegten Content-Addressable Network realisieren lässt.

Die Feedback-Weiterleitung stellt folgende Anforderungen:

- Feedback muss alle Kontakte des Feedback-Subjekts erreichen.
- Die Feedback-Übertragung muss schnell geschehen. Veraltetes Feedback ist nutzlos.
- Die Übertragung von Feedback darf keinen hohen Ressourcenbedarf haben[1].

Beispiel: Abbildung 4.1 zeigt ein CAN mit einem zweidimensionalen Schlüsselraum. Hier hat Peer P_1 ein Feedback-Objekt über P_2 generiert, das alle schraffiert dargestellten Kontakte von P_2 erhalten sollen. Pfeile kennzeichnen die Peers, an die P_1 das Feedback-Objekt direkt übertragen kann; alle anderen Nachbarn von P_2 sind diesem Peer unbekannt und können dessen Feedback nur mittelbar erhalten.

4.1 Ansätze zum Feedback-Transport

Grundsätzlich gibt es sehr viele Möglichkeiten, das von einem Peer generierte Feedback an alle Kontakte des Feedback-Subjekts weiterzuleiten. Die einfachste Variante besteht darin, dass

[1] Aus Abschnitt 3.4 folgt, dass die Peers anderenfalls auf die Rentabilitätsschwelle für den Feedback-Transport optimieren würden. Das heißt, sie würden die Kosten für den Feedback-Transport gleichsetzen mit dem Nutzen, der ihnen aus dem Feedback von anderen Peers entsteht. Dies würde jedoch die Diskriminierung unkooperativer Peers verringern.

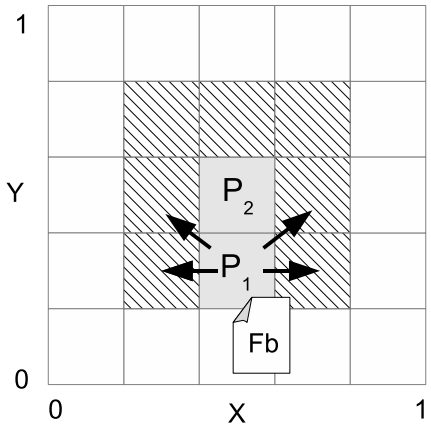

Abbildung 4.1: Feedback-Übermittlung im CAN.

der Feedback-Erzeuger mittels *Flooding* Nachrichten an alle Kontakte schickt, die wiederum das gleiche Verfahren anwenden. Ein Zähler, der mit jedem Weiterleitungsvorgang verringert wird, sorgt dann dafür, dass Nachrichten nicht weiter als bis zum entferntesten Kontakt des Feedback-Subjekts im Netz verbreitet werden. Diese Art der Feedback-Weiterleitung ist zwar sehr schnell, jedoch zugleich sehr ressourcenintensiv. Dies liegt zum Einen daran, dass beim Flooding die selben Nachrichten mehrfach den gleichen Empfänger erreichen. Zum Anderen sind nur die Kontakte des Feedback-Subjekts an Feedback über diesen Peer interessiert. Beim Flooding werden Nachrichten jedoch an alle erreichbaren Kontakte versendet, bis ein mit der Nachricht übertragener Zähler abgelaufen ist, der mit jeder Weiterleitung um 1 verringert wird ('Hop-Counter').

Unter Ausnutzung der Topologie der P2P-Datenstruktur lassen sich effektivere Mechanismen zum Feedback-Transport realisieren. In [RHKS01] wird beschrieben, wie ein (fast) duplikatfreier *Multicast* in einem CAN zu realisieren ist. Dazu sendet der Initiator seine Nachricht an alle seine Kontakte. Ein Peer, der die Nachricht aus Dimension i erhält, leitet sie dagegen nur an den in Dimension i gegenüberliegenden Peer und an Kontakte in den Dimensionen $1 \cdots (i-1)$ weiter. Vergleichbare Multicast-Ansätze existieren auch für andere P2P-Datenstrukturen, z.B. *Bayeux* ([ZZJ+01], basiert auf Tapestry), *daMulticast* ([BEG03], P-Grid) oder *Scribe* ([CDKR03], Pastry).

Eine weitere Verbesserung bietet die Einführung von *Push-Pull-Diensten*. Der in [KRD03] vorgestellte Ansatz beschreibt einen dreistufigen dezentralen Algorithmus, bei dem zunächst einige Peers ('Seers') bestimmt werden, die über viele Kontakte verfügen (Discovery-Phase). Ein Peer, der eine Nachricht an viele Teilnehmer übermitteln will, leitet diese nun an die Seers weiter (Push-Phase). Alle anderen Teilnehmer, die am Erhalt dieser Nachricht interessiert sind, fragen periodisch einen Seer in ihrer Nähe ab (Pull-Phase). Soll Feedback nach Art der Push-Pull-Dienste übertragen werden, müssen in der Discovery-Phase die Seers so bestimmt werden, dass (1) die Verbundmenge der Kontakte aller Seers auch alle Kontakte des Feedback-Subjekts enthält, und dass (2) die Mengen der Kontakte der Seers möglichst disjunkt sind.

Der Feedback-Transport kann ressourcenschonender gestaltet werden, wenn *Gossip-Protokolle* [CAPMN03] zum Feedback-Transport genutzt werden. Gossip-Protokolle nutzen das Konzept der epidemischen Algorithmen [DGH+87] zum Update von Replikaten in verteilten Datenbanken. Ein prominentes Beispiel für ein P2P-Gossip-Protokoll ist *PlanetP* [CAPMN03]. PlanetP basiert darauf, dass die Peers periodisch Nachrichten mit zufällig bestimmten anderen Peers austauschen, und auf diesem Wege Informationen untereinander abgleichen. In den meisten P2P-Datenstrukturen übermitteln die Peers zyklisch Informationen über ihren Status (Zonen, Netzwerkadressen etc.) an Nachbarknoten und leiten Anfragen von anderen Peers weiter; Gossip-Protokolle können daher unmittelbar zum Feedback-Transport in diesen Datenstrukturen eingesetzt werden.

Des weiteren ist es möglich, Techniken aus dem Bereich der mobilen Ad-Hoc-Netzwerke zu adaptieren. In mobilen Ad-Hoc-Netzwerken bewegen sich die Teilnehmer geografisch und kommunizieren über Funk-Schnittstellen mit begrenzter Reichweite. Die Knoten sind also darauf angewiesen, gemäß einer *Store-and-Forward* Policy Nachrichten anzusammeln, bis ein geeigneter Nachbarknoten in Kommunikationsreichweite ist. In [LK00] wird beschrieben, wie sich so ein effizienter Multicast unter Ausnutzung lokaler Strukturen realisieren lässt. Beim Feedback-Transport lässt sich dieses Protokoll dazu nutzen, Feedback-Pakete an ohnehin versandte Nachrichten anzuhängen: Feedback-Objekte werden so lange zwischengespeichert, bis Nachrichten 'in der richtigen Richtung' versendet werden müssen.

4.2 Der Feedback-Transport in FairNet

Für den Feedback-Transport im CAN ideal ist eine Kombination aus den oben genannten Ansätzen: ein Protokoll, das (1) die Struktur des Netzwerks ausnutzt, um möglichst ressourcensparend und duplikatfrei zu arbeiten, das (2) Feedback nur an Peers weiterleitet, die daran auch interessiert sind, und das (3) mittels Store-and-Forward Feedback ansammelt und an Nachrichten anhängt, die im Rahmen der normalen Arbeit der P2P-Datenstruktur versendet werden.

Es wird davon ausgegangen, dass an jede Nachricht bis zu β Objekte angehängt[2] werden können, ohne dass sich dadurch die Kosten für das Absenden einer Nachricht merklich erhöhen. Für kleine Datenmengen ist das realistisch. Feedback-Objekte bestehen gemäß Abschnitt 3.1 aus wenigen Bytes: den numerischen IDs von Feedback-Erzeuger und Feedback-Subjekt, einem Zeitstempel und einigen Flags. Auf der anderen Seite müssen allein für das Übertragen einer einzigen Nachricht der TCP/IP-Frame, der Handshake mittels SYN- und ACK-Datagrammen sowie darunter liegende Ethernet- oder DSL-Protokollinformationen übertragen werden. Das macht je nach Art des Netzwerkanschlusses bereits einige hundert Bytes aus; hinzu kommt noch die Nachricht selbst.

Überblick Das in FairNet verwendete Protokoll zum Feedback-Transport [BA05] benötigt neben den in Abschnitt 3.1 beschriebenen Datenstrukturen ein *Übertragungs-Log*. In diesem

[2] Die Kapazität β des Attachments ist ein applikationsabhängiger Optimierungsparameter.

Log wird festgehalten, an welchen Peer ein Feedback-Objekt bereits übertragen wurde. Weitere Strukturen sind nicht erforderlich, da Feedback nur an Peers versendet wird, die diese auch gebrauchen können und daher in ihr Repository aufnehmen.

```
1  attachFeedback(Message m, NextPeer p) {
2      // bestimme Kandidaten für das Feedback−Attachment
3      CandidateFeedback F :=
4          {f | isNeighbor(p, f.subject) ∧ f ∈ this.Repository ∧ (f,p) ∉ transmissionLog};
5
6      // falls erforderlich, erstelle eine Rangfolge der wichtigsten Feedback−Objekte
7      if ( |F| > β) {
8          sortiere F gemäß utility(f ∈ F, p);
9      }
10     // hänge bis zu β Feedback−Objekte an die Nachricht an
11     i := 0;
12     while (F ≠ ∅ ∧ i < β) {
13         Feedback f := getFirstElement(F);
14         m.attachObject(f);
15         this.transmissionLog.add(f,p);
16         removeFirstElement(F);
17         i := i + 1;
18     }
19 }
```

Abbildung 4.2: Methode *attachFeedback*.

Das Übertragungsprotokoll für Feedback arbeitet wie folgt: jedes Mal, wenn ein Peer eine beliebige Nachricht absendet, ruft er die Methode *attachFeedback()* auf, die in Abbildung 4.2 gezeigt wird. Der Methode werden der Peer, an den die Nachricht geschickt wird, sowie die Nachricht selbst übergeben. In der Methode werden zunächst die Feedback-Objekte bestimmt, die überhaupt für eine Übertragung an den Zielpeer der Nachricht geeignet sind. Das sind alle Objekte, (1) deren Feedback-Subjekt ein Kontakt des Peers ist, und (2) die der Peer noch nicht vorher erhalten hat. Steht mehr Feedback zur Verfügung, als die Kapazitätsbeschränkung auf β Objekte je Nachricht erlaubt, wird in Zeile 8 die Reihenfolge der für den Empfänger nützlichsten Objekte erstellt. Verschiedene Algorithmen für die Bestimmung des 'nützlichsten' Feedbacks werden in Abschnitt 4.3 diskutiert. Zuletzt werden die wichtigsten β Objekte an die Nachricht angehängt. Weiterhin wird im Übertragungs-Log vermerkt, welches Feedback der Empfänger erhalten wird, um Mehrfachübertragungen auszuschließen.

Kapazität des Protokolls Um die Bestimmung der Kapazität allgemeingültig zu gestalten, sollen im folgenden ausschließlich Anfragenachrichten betrachtet werden; Kontrollnachrichten wie beispielsweise periodische Status-Updates oder Feedback-Benachrichtigungen bleiben unberücksichtigt. An jede Nachricht können nur maximal β Objekte angehängt werden. Die Zahl der übertragenen Nachrichten wurde bereits in Abschnitt 3.4 bestimmt. Wie groß ist nun die Übertragungskapazität bei der Feedback-Übermittlung? Tabelle 4.1 zeigt die neuen

Parameter des Gossip-Protokolls. Daneben gelten die gleichen Annahmen wie in Abschnitt Abschnitt 3.4, d.h. ein System in eingeschwungenem Zustand, gleichverteilte Anfragen und eine Anfrageverarbeitung in Runden.

Parameter	**Symbol**	**Vorgabewert**
Kapazität des Attachments, d.h. Maximalzahl der angehängten Feedback-Objekte.	β	10
Zahl der pro Runde zu übertragenden Feedback-Objekte.	ρ	$\rho = \sigma_{pos} + \sigma_{neg}$
Zahl der Kontakte pro Nachbar, d.h. Zahl der Peers, die ein Feedback-Objekt erhalten sollen.	κ	$\kappa = 3^d - 1$

Tabelle 4.1: Die Parameter des Gossip-Übertragungsprotokolls.

In jeder Runde generiert jeder der N Peers im System ρ neue Feedback-Objekte (Gleichungen 3.15 und 3.16), stellt und beantwortet eine Anfrage, und leitet δ Nachrichten an andere Peers weiter (Gleichung 3.3). Dann beträgt die Transportkapazität θ für die Zahl der insgesamt pro Runde übertragbaren Feedback-Objekte:

$$\theta = N \cdot \delta \cdot \beta \tag{4.1}$$

Wenn jeder Peer sein Feedback an eine Zahl von κ Kontakten versenden möchte, gilt darüber hinaus für jede Runde:

$$\delta \cdot \beta = \kappa \cdot \rho \tag{4.2}$$

Wird eine konkrete Datenstruktur zugrunde gelegt, lassen sich diese Angaben weiter konkretisieren. In einem CAN mit einem $d = 4$-dimensionalen Schlüsselraum besitzt jeder Nachbar mindestens $\kappa = 3^d - 1 = 80$ Kontakte[3]. Die Zahl der Weiterleitungen pro Runde wurde in Abschnitt 3.4 bestimmt und beträgt für ein CAN mit $N = 10.000$ Peers etwa $\delta = 4$. Daraus folgt, dass in diesem Szenario jeder Peer pro Runde im Durchschnitt $\rho = 4 \cdot \beta/80$ Feedback-Objekte generieren und an seine Nachbarn verteilen kann, ohne dabei die Transportkapazität zu überschreiten, und ohne auf andere Nachrichtentypen wie z.B. die in [RFH+01] vorgesehenen Status-Nachrichten auszuweichen. Zum FairNet-Protokollentwurf gehört, dass das Ergebnis jeder Transaktion mit einer Feedback-Benachrichtigung an alle Weiterleiter bekannt gegeben wird; hier verdoppelt sich die Transportkapazität noch einmal.

[3] Im hier eingesetzten CAN werden alle mit einem Punkt im Schlüsselraum angrenzenden Peers als Nachbar aufgefasst (vgl. Abschnitt 3.4). Daher unterscheidet sich die Zahl der Nachbarn und die Berechnung der Entfernungen vom ursprünglichen CAN-Entwurf.

4.3 Auswahlstrategien für Feedback

Wenn die Kapazität β des pro Nachricht angehängten Feedbacks und die Zahl der Nachrichten zu klein sind, um sämtliches generierte Feedback sofort zu übermitteln, muss eine Auswahl der zu übertragenden Objekte getroffen werden. Diese Auswahl ist nicht trivial: Welches Feedback ist für den Empfänger jetzt grade am nützlichsten? Und wie lässt sich Feedback so transportieren, dass die Kapazität des Übertragungsweges optimal ausgenutzt wird? Es ist wahrscheinlich besser, drei Objekte zu direkten Nachbarn zu befördern als eines über eine Entfernung von drei Peers zu versenden. Des weiteren sollte die Zahl der Objekte minimal sein, die von unterschiedlichen Knoten aus mehrfach an einen Peer gesendeten werden. Im CAN bieten sich dazu sehr unterschiedliche Auswahlstrategien an, unter anderem:

Random Die Objekte werden zufällig mit Gleichverteilung bestimmt. Diese Strategie ist unaufwendig und hat sich bereits bei der Auswahl von Kontakten [BB03b] als überraschend wirksam erwiesen.

LIFO 'Last In First Out' ist eine klassische Strategie, die neue Objekte bevorzugt. Alte Objekte werden dabei jedoch möglicherweise überhaupt nicht übertragen, bevor sie nach einiger Zeit aus dem Repository verdrängt werden.

FIFO 'First In Last Out' befördert jedes Feedback-Objekt in der Reihenfolge, in der es in das Repository aufgenommen wurde. Nachteilig an dieser Strategie ist, dass die neuesten Feedback-Objekte stets zuletzt übertragen werden.

Gerichtete Weiterleitung Die gerichtete Weiterleitung basiert analog zum Greedy Forwarding darauf, dass der Abstand zum Feedback-Erzeuger mit jedem Weiterleitungsschritt größer werden muss. Die Strategie verhindert dadurch 'Rückwärts'-Übertragungen. Abbildung 4.3 (links) zeigt die Wege, welche die Feedback-Objekte bei dieser Strategie nehmen können.

CAN-Multicast Hier merkt sich ein Peer, aus welcher Richtung er ein Feedback-Objekt erhalten hat (Abb. 4.3 rechts). Ein Objekt aus Dimension i wird nur in den Dimensionen $1 \cdot (i-1)$ sowie in der Empfangsdimension weitergeleitet. Die Strategie wird in [RHKS01] ausführlich beschrieben.

Jeder Peer verfügt über ein Übertragungs-Log, in dem festgehalten wird, welches Feedback der Peer bereits an einen anderen Knoten übermittelt hat. Ein Peer sendet also niemals zweimal das gleiche Objekt an einen anderen Knoten. Weil zwei Peers mit aneinander angrenzenden Zonen jedoch etwa die Hälfte der Nachbarn gemein haben, kommen Mehrfachübertragungen vor: ein Peer erhält das gleiche Feedback-Objekt von mehreren Nachbarn. Random, FIFO und LIFO berücksichtigen diese Problematik nicht. Bei der gerichteten Weiterleitung werden bereits alle Peers ausgeschlossen, die ein Feedback-Objekt von einem Peer erhalten können, der näher an der Feedback-Quelle positioniert ist. Abbildung 4.3 zeigt, dass so im zweidimensionalen CAN nur maximal 4 Duplikate zu erwarten sind. Einen Schritt weiter geht der CAN-Multicast, der

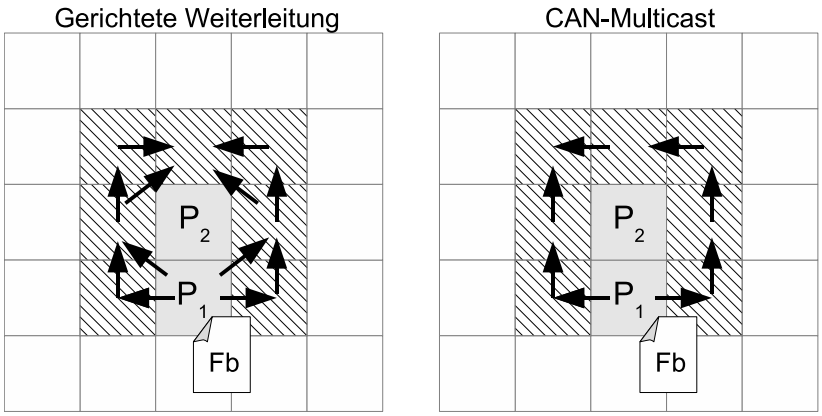

Abbildung 4.3: Feedback-Ausbreitung bei gerichteter Weiterleitung und CAN-Multicast.

sehr genaue Angaben darüber macht, in welche Richtungen ein Peer Feedback weiterleiten darf. Es ist aber insbesondere in höherdimensionalen Schlüsselräumen zu erwarten, dass beim Multicast Feedback lange im Repository 'warten' muss, bis eine Nachricht exakt in die richtige Richtung abgesendet wird. Es ist darum zu erwarten, dass die Ausbreitungsgeschwindigkeit des Feedbacks bei den anderen Auswahlstrategien auf Kosten der Ressourceneffizienz höher ist.

4.4 Evaluierung des Übertragungsprotokolls

Die folgenden Experimente sollen das Verhalten des Gossip-Protokolls in der Praxis untersuchen. Dazu wird die in Abschnitt 3.5 beschriebene Experimentierumgebung genutzt. Wieder werden ausschließlich Anfragenachrichten zum Anhängen von Feedback-Objekten genutzt, um allgemeingültige Aussagen zu erhalten. Periodische Status-Updates oder Feedback-Benachrichtigungen werden nicht genutzt.

Anwendbarkeit in der Praxis Im ersten Experiment soll die grundsätzliche Anwendbarkeit der Feedback-Verteilung in der Realität demonstriert werden. Dazu wird ein CAN mit einem zweidimensionalen Schlüsselraum und 10.000 Peers aufgesetzt. Jeder Peer generiert ein Feedback-Objekt über einen zufällig ausgewählten Nachbarn. Das Feedback-Objekt muss nun alle Nachbarn des Feedback-Subjekts erreichen, wie in Abb. 4.1 dargestellt. Jede Nachricht kann $\beta = 10$ Feedback-Objekte transportieren. In diesem Szenario werden die Kapazitätsgrenzen (vgl. Gleichung 4.1 und 4.2) bei weitem unterschritten: bei $d = 2$ Dimensionen des Schlüsselraums hat jeder Peer nur $\kappa = 8$ Nachbarn, muss aber $\delta \approx 33$ Weiterleitungen pro Runde erbringen.

Abbildung 4.4 zeigt das Ergebnis des Experiments. Gemessen wurde, nach welcher Zeit Feedback wie viele der 8 Nachbarn erreicht hat (dicke Linie), sowie die Standardabweichung dieser Werte (als Errorbars eingezeichnet). Am Anfang ist jedes Feedback-Objekt genau einem Peer

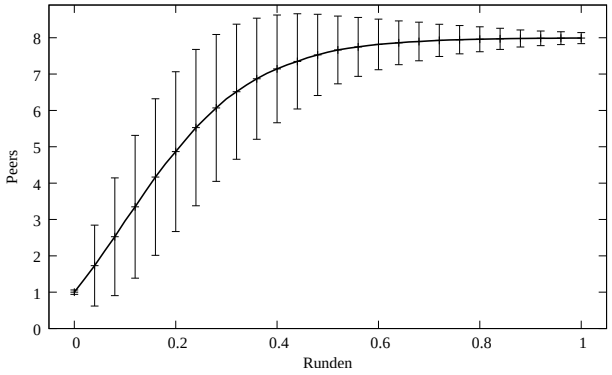

Abbildung 4.4: Zeit bis zum Erreichen aller Nachbarn des Feedback-Subjekts.

– dem Erzeuger – bekannt. Bereits nach $0{,}15$ Runden hat das Feedback-Objekt die Hälfte aller Nachbarn des Subjekts erreicht, und am Ende einer Runde ist es allen Nachbarn bekannt. Die Standardabweichung zeigt, dass die Streuung der Werte bis zum Ende der Runde stark abnimmt, d.h., nicht nur der Durchschnitt, sondern (nahezu) alle Objekte haben am Rundenende ihre acht Zielpeers erreicht. Damit erfüllt das Gossip-Protokoll die Erwartungen: in einem Szenario mit vielen Nachrichten zwischen den Peers und ohne Limitierungen der Attachment-Größe wird Feedback im Durchschnitt sehr rasch vom Feedback-Erzeuger zu jedem Nachbarn des Feedback-Subjekts transportiert.

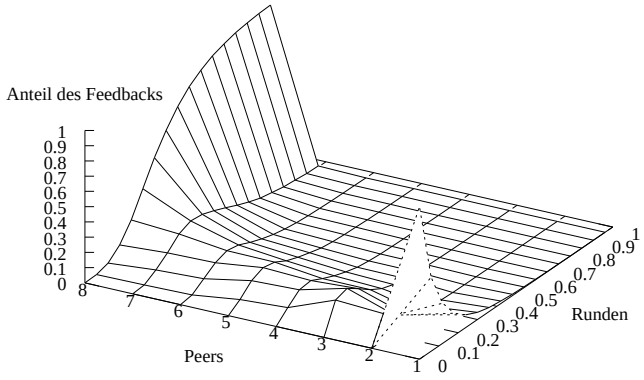

Abbildung 4.5: Ausbreitung des Feedbacks über die Zeit.

Eine detailliertere Darstellung der Ergebnisse des Experiments bietet Abbildung 4.5. Die Abbildung zeigt grafisch, wie sich das Feedback mit der Zeit über die Peers ausbreitet. Hier wurde in regelmäßigen Zeitintervallen gemessen, wie hoch der Anteil allen Feedbacks ist, das genau einem, zwei, ... acht Peers bekannt ist. Eine Spitze bei einem Peer und 0 Runden zeigt, das am Beginn des Experiments alles Feedback nur einem Peer bekannt war. Bereits nach $0{,}1$ Runden, wenn nach Diagramm 4.4 der Durchschnitt des Feedbacks 3 Peers bekannt ist, hat 2% des Feedbacks alle 8 Nachbarn erreicht, und nach $0{,}2$ Runden ist die Hälfte allen Feedbacks im System 5 oder mehr Peers bekannt. Die Abbildung zeigt, dass einige Feedback-Objekte sehr schnell ihr Ziel erreichen, und dass es nur wenige 'Nachzügler' gibt, die mehr als $0{,}5$ Runden

bis zu allen Nachbarn des Feedback-Objekts benötigen.

Ausbreitungsgeschwindigkeit unter Überlast Nun soll untersucht werden, wie sich das Gossip-Protokoll beim Überschreiten der Kapazitätsrestriktion verhält. Insbesondere ist dabei von Interesse, wie sich die vorgestellten Auswahlstrategien verhalten, mit denen das anzuhängende Feedback bestimmt wird. Um eine Überlast-Situation herbeizuführen, wurde ein CAN mit 10.000 Peers und einem nun vierdimensionalen Schlüsselraum aufgesetzt. Das bedeutet, jeder Peer hat $\kappa = 80$ Nachbarn, die an Feedback über ihn interessiert sind, muss aber nur $\delta \approx 4$ Weiterleitungen pro Runde an andere Peers leisten. Im Repository jedes Peers befinden sich nun 5 Feedback-Objekte, die jeweils an alle 80 Nachbarn des Subjekts zu übertragen sind. Um das Kapazitätsproblem zu verschärfen, durfte jeder Peer nur maximal $\beta = 10$ Objekte an eine Nachricht anhängen. Gemessen wurde wieder, nach wie vielen Runden das Feedback wie viele Peers erreicht hat. Ohne Berücksichtigung der Kapazitätsrestriktion beträgt der Erwartungswert für die Zeit, in der ein Feedback-Objekt alle $\kappa = 80$ Nachbarn erreicht, $\lceil \log_{\delta/2} \kappa \rceil = 7$ Runden.

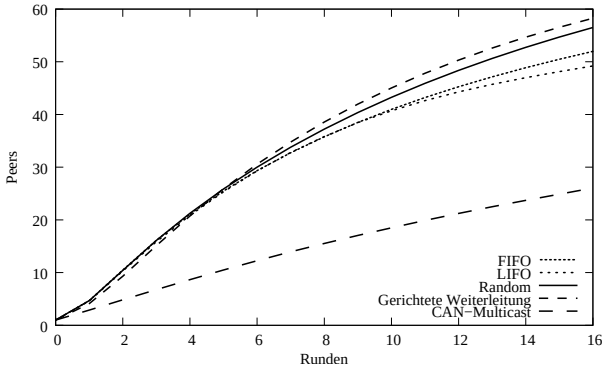

Abbildung 4.6: Einfluss der Auswahlstrategie auf die Ausbreitungsgeschwindigkeit.

Abbildung 4.6 zeigt das Ergebnis des Experiments. Zunächst ist auffällig, dass die Transportkapazität des Protokolls so weit überlastet wurde, dass auch 16 Runden nicht ausreichen, um jedem Peer alles Feedback über seine Nachbarn zuzustellen. Je nach Ansatz hat ein Feedback-Objekt nur 25 bis 60 von den 80 Nachbarn des Subjekts erreicht.

Weiterhin fällt auf, dass der CAN-Multicast die Feedback-Objekte sehr viel langsamer zustellt als die vier anderen Strategien. Dieser Befund liegt im Rahmen der Erwartungen. Werden je Runde nur Nachrichten an 4 von 80 Nachbarn geschickt, so ist eine Strategie benachteiligt, die starke Vorgaben für die Nachbarn macht, an die Feedback weitergeleitet werden darf. Dies haben begleitende Messungen bestätigt: die Kapazitätsrestriktion von $\beta = 10$ Attachments je Nachricht wurde vom Multicast nicht ausgeschöpft. Daneben zeigt Abbildung 4.6 einen leichten Vorsprung für die Strategien Random und gerichtete Weiterleitung. Während die Strategien FIFO und LIFO einfach jedes Objekt in bzw. entgegen der Einfügereihenfolge weitersenden, nutzt die Strategie der gerichteten Weiterleitung die Topologie des Schlüsselraums aus. Dabei wird über die Entfernungen zwischen Feedback-Erzeuger und aktuellem Peer bzw. Erzeuger

und möglichem Empfänger ermittelt, ob ein Peer ein Feedback-Objekt nicht über einen kürzeren Pfad von einem anderen Peer erhalten kann (vgl. Abb. 4.3). Die Random-Strategie vermeidet hingegen durch die zufällige Auswahl der zu übertragenden Objekte das Problem, dass mit einer starren Auswahlreihenfolge nur wenige Objekte viele Peers erreichen, der Großteil der Objekte aber nicht übertragen wird.

Ressourceneffizienz Ein Knoten kann mit seinem Übertragungs-Log zwar feststellen, ob er selbst ein Objekt an einen bestimmten Peer übermittelt hat. Er kann jedoch nicht ermitteln, ob nicht bereits ein anderer Knoten dasselbe Objekt an den Peer geschickt hat. Insbesondere in höherdimensionalen Schlüsselräumen können daher mehrfach an einen Peer übertragene Feedback-Objekte einen erheblichen Teil der verfügbaren Übertragungskapazität beanspruchen. Um die Auswirkungen der Auswahlstrategie auf die Ressourceneffizienz zu untersuchen, wurde die Zahl der Mehrfachübertragungen im letzten Experiment ermittelt. Um die unterschiedlichen Ausbreitungsgeschwindigkeiten zu kompensieren (vgl. Abb. 4.6), wurde jeweils gemessen, wie viele Feedback-Objekte im Durchschnitt mehrfach an einen Peer übertragen wurden, bis sie 30 Peers erreicht haben. Die Zahl von 30 Peers wurde willkürlich gewählt, um die Experimentdauer zu begrenzen: Strategien wie Random können nicht garantieren, dass die Feedback-Objekte überhaupt alle Nachbarn erreicht.

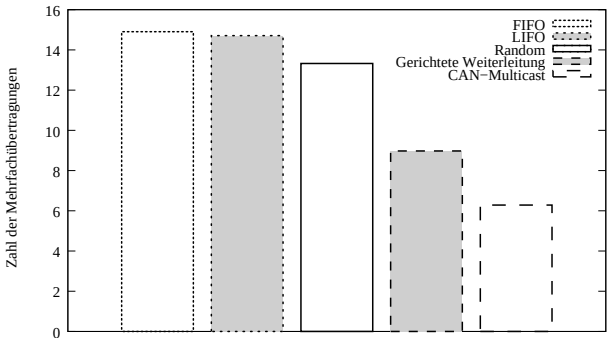

Abbildung 4.7: Einfluss der Auswahlstrategie auf die Zahl der Mehrfachübertragungen.

Das Ergebnis der Messung wird in Diagramm 4.7 präsentiert. Es zeigt, dass übereinstimmend mit den Erwartungen FIFO und LIFO viele Mehrfachübertragungen verursachen. Ein Feedback-Objekt muss dort im Durchschnitt 45 mal[4] von Peer zu Peer übermittelt werden, bis es die geforderten unterschiedlichen 30 Knoten erreicht hat. Der CAN-Multicast braucht für die gleiche Anzahl von Zielknoten nur 36 Übertragungen. Im Vergleich mit Abbildung 4.6 fällt jedoch auf, dass der Multicast dafür > 16 Runden benötigt, wohingegen alle anderen Auswahlstrategien mit ≈ 6 Runden für 30 Peers auskommen!

Vor diesem Hintergrund bietet die Strategie der gerichteten Weiterleitung einen sehr guten Kompromiss aus Effizienz und Ausbreitungsgeschwindigkeit: sie verursacht nur etwas mehr als halb so viele Duplikate wie FIFO oder LIFO, d.h. Übertragungen von Objekten an Peers,

[4] Übertragen an 30 Peers + 15 Duplikate = 45 Übertragungen insgesamt.

denen diese schon bekannt waren. Weiterhin weist die gerichtete Weiterleitung die höchste der hier gemessenen Ausbreitungsgeschwindigkeiten auf. Zudem ist die Strategie leicht zu realisieren: sie benötigt keine zusätzlichen Informationen darüber, in welcher Reihenfolge oder aus welcher Richtung ein Feedback-Objekt beim Peer eingetroffen ist.

4.5 Feedback-Verteilung in anderen P2P-Datenstrukturen

Der in diesem Kapitel vorgestellte Algorithmus zur Feedback-Verbreitung in FairNet ist ganz auf die Topologie und die Kontaktwahl in Content-Addressable Networks zugeschnitten: alle Peers sind im Schlüsselraum benachbart, die Feedback über einen bestimmten Peer erzeugen, und an diesem Feedback interessiert sind. Soll (wie in Abschnitt 3.7 beschrieben) FairNet zur Disziplinierung von unkooperativen Teilnehmern in anderen P2P-Datenstrukturen eingesetzt werden, so ist daher auch ein anderer Mechanismus zum Feedback-Transport zu entwickeln.

Die am Anfang dieses Kapitels beschriebenen Anforderungen nach einer schnellen, ressourcenschonenden Übertragung des Feedbacks an alle Kontakte des Feedback-Subjekts gelten auch in anderen Datenstrukturen, sind dort jedoch ggf. unterschiedlich zu realisieren.

Der Feedback-Transport in Chord In Anlehnung an den Abschnitt 3.8, in dem die Einsetzbarkeit von FairNet in Chord untersucht wird, soll hier diskutiert werden, wie der Feedback-Transport in dieser P2P-Datenstruktur realisiert werden kann. Im Gegensatz zum CAN werden Anfragen in Chord nur in einer Richtung durch den Schlüsselraum weitergeleitet. In Chord pflegt jeder Peer in seiner Finger-Table Kontakte zu Peers, die im Abstand $2^0, 2^1, \cdots, 2^x$ relativ zu seiner eigenen Position im Schlüsselraum liegen. Das bedeutet im Umkehrschluss, dass die an Feedback über einen bestimmten Knoten interessierten Peers im Abstand $-2^0, -2^1, \cdots, -2^x$ zu der Position des Feedback-Subjekts im Schlüsselraum liegen. Jeder Peer kann also bestimmen, welche Teilnehmer das von ihm generierte Feedback benötigen. Ein naiver Ansatz zum Feedback-Transport besteht darin, dass ein Knoten das von ihm generierte Feedback mittels separater Nachrichten an die so bestimmten Peers sendet.

Als eine ressourcenschonendere Alternative bieten sich inhaltsbasierte Multicast-Ansätze an. Ein auf Chord zugeschnittener Ansatz [Hue03] besteht darin, alle Empfänger einer Nachricht durch das umschließende Intervall im Schlüsselraum zu beschreiben. Nun wird die Nachricht zunächst an den Empfänger mit der kleinsten Position im Intervall gesendet. Ausgehend von diesem Knoten übermittelt dann jeder Peer die Nachricht an alle Kontakte in seiner Finger-Table, deren Position im Schlüsselraum innerhalb des Intervalls der Empfänger liegt. Um die Zahl der Duplikate zu reduzieren, wird dabei das Intervall der Empfänger jeweils um diejenigen Peers verringert, die die Nachricht von anderen Kontakten erhalten.

Ein grundlegender Nachteil der Verwendung solcher Multicast-Ansätze zum Feedback-Transport besteht darin, dass die Feedback-Empfänger in Chord nicht an aufeinander folgenden Positionen im Schlüsselraum liegen. Ausgefeiltere Ansätze wie ONP- (Overlay Network Provider) Multicast [Hue03] oder CAM-Chord [ZCLC05] verwenden jedoch zum Nachrich-

tentransport ein von der darunterliegenden Datenstruktur unabhängig gebildetes Overlay-Netzwerk (einen *Spanning Tree*), das sich in dieser Hinsicht anpassen ließe. Neben Multicast-Algorithmen sind auch weiterhin die in Abschnitt 4.1 beschriebenen und von der Datenstruktur unabhängigen Push-Pull-Dienste [KRD03] oder Gossip-Protokolle [CAPMN03] zum Feedback-Transport verwendbar.

4.6 Zusammenfassung

Das FairNet-Protokoll ist nur dann in der Lage, zeitnah auf dynamische Veränderungen im Verhalten der Peers zu reagieren, wenn das generierte Feedback rasch vom Erzeuger zu allen Peers gelangt, die es benötigen. Dieses Kapitel identifizierte zunächst die Anforderungen, die an den Feedback-Austausch zu stellen sind. Im Anschluss wurde ein Gossip-Protokoll beschrieben, das diese Anforderungen dadurch erfüllt, dass Feedback an Nachrichten angehängt wird, welche die Peers im regulären Betrieb untereinander austauschen.

Die Zahl der Feedback-Objekte, die auf diesem Wege übertragen werden können, ist jedoch begrenzt. Nach der überschlägigen Ermittlung der Kapazitätsrestriktionen des Gossip-Protokolls wurde daher diskutiert, mit welchen dezentralen Algorithmen die für FairNet geeignetste Auswahl von zu übertragenden Feedback-Objekten bestimmt werden kann. Eine Serie von Experimenten zeigte, dass eine Greedy-Strategie, bei dem das Feedback mit jeder Weiterleitung den Abstand zum Erzeuger im Schlüsselraum vergrößert, einen geeigneten Kompromiss aus Übertragungsgeschwindigkeit, Zahl der Empfänger und Zahl der Mehrfachübertragungen darstellt. Dabei wurde gezeigt, dass bei unterschreiten der Kapazitätsrestriktion wenige Runden ausreichen, um alles Feedback vom Erzeuger zu allen Kontakten des Feedback-Subjekts zu übermitteln. Abschließend wurde am Beispiel von Chord diskutiert, wie sich ein effizienter Feedback-Transport in anderen P2P-Datenstrukturen realisieren lässt.

Kapitel 5

FairNet: Umgang mit gefälschtem Feedback

In den vorangegangenen Kapiteln wurde stets davon ausgegangen, dass übermitteltes Feedback grundsätzlich wahrheitsgemäß generiert wird, also den tatsächlichen Ausgang einer Transaktion beschreibt. Feedback ist jedoch leicht zu fälschen: jeder Peer kann Feedback in beliebiger Menge, zu beliebigen Zeiten und mit beliebigem Inhalt generieren und an seine Nachbarn übermitteln. Dieses Kapitel untersucht zunächst, welchen Einfluss gefälschtes Feedback auf die Kosten kooperativer und unkooperativer Peers nehmen kann. Weiter wird beschrieben, wie sich gefälschtes Feedback durch den Vergleich zwischen dem vom Feedback prognostizierten und dem tatsächlichen Transaktionsausgang erkennen und von der Ermittlung vertrauenswürdiger Weiterleiter ausschließen lässt; und welche Änderungen an den Datenstrukturen und Methoden von FairNet dazu erforderlich sind. Des weiteren erläutert das Kapitel, wie das Verbreiten von falschem Feedback mittels Arbeitsbeweisen abgestraft werden kann. Eine analytische und experimentelle Evaluierung rundet das Kapitel ab.

Im folgenden wird die Unterscheidung *ehrlich* und *unehrlich* für Knoten gebraucht, die *korrektes* oder *verfälschtes* Feedback übermitteln. Zu verfälschtem Feedback zählt alles Feedback, das nicht mit Bezug auf eine abgeschlossene Transaktion erzeugt wurde oder den Ausgang der Transaktion nicht korrekt wiedergibt. Hier wird ein unehrlicher Peer so modelliert, dass er einen Anteil b gefälschtes und einen Anteil $(1-b)$ ehrliches Feedback generiert. Die Differenzierung zwischen ehrlich und unehrlich ist orthogonal zur Unterscheidung in kooperative und unkooperative Knoten; ein kooperativer Knoten kann durchaus unehrlich sein und gefälschtes Feedback in Umlauf bringen. Weiterhin sollen die Begriffe *positiver* und *negativer Transaktionsausgang* Operationen bezeichnen, die von kooperativen Peers erfolgreich abgeschlossen bzw. von unkooperativen Peers gestört wurden.

In Szenarien mit rationalen, unabhängigen Peers lässt sich gefälschtes Feedback leicht vermeiden. Für einen rational agierenden Peer, der sich ein kooperatives Rating erarbeitet hat, besteht kein Anreiz, gefälschtes Feedback über andere Peers zu verbreiten, da er daraus keinen Vorteil ziehen kann. Zur Absicherung würden also folgende Maßnahmen genügen:

- Peers dürfen kein Feedback über sich selbst verbreiten.
- Feedback wird nur von Peers akzeptiert, die kooperativ bewertet werden.
- Offensichtlich fehlerhaftes Feedback wird verworfen. Dazu zählt beispielsweise Feedback, das Peers bewertet, die der Feedback-Ersteller nicht beobachten konnte, oder Feedback mit unsinnigem Zeitstempel.

Solange davon ausgegangen werden kann, dass kooperative Peers keinen Anreiz zum Verbreiten von Falschaussagen haben, und solange unkooperative Peers nicht zusammenarbeiten, ist dieser Ansatz hinreichend. Dies ist in realistischen Szenarien aber nicht gegeben. Peers können zusammenarbeiten, und einzelne Peers können unter mehreren Identitäten bekannt sein oder andere Knoten bzw. das System als Ganzes zu beeinträchtigen versuchen.

Ein Koalitionsangriff auf den bisher beschriebenen Feedback-Mechanismus könnte beispielsweise wie folgt ablaufen: Gegeben sei eine Gruppe von unkooperativen, unehrlichen Peers, die den Feedback-Mechanismus unterlaufen wollen. Diese Peers erarbeiten sich bei ihren Nachbarn zunächst genügend positives Feedback, um als kooperativ zu gelten. Von nun an brauchen die Gruppenmitglieder nicht mehr kooperativ mitzuarbeiten. Es genügt, wenn die unehrlichen Peers insgesamt mehr gefälschtes positives Feedback über die Mitglieder ihrer Gruppe in Umlauf bringen, als echtes negatives Feedback wegen verweigerter Kooperation von anderen Peers über sie generiert wird. Da die Peers als kooperativ bewertet werden, wird das Feedback auch akzeptiert. Die einzige noch erforderliche Arbeitsleistung besteht im Weiterleiten der Anfragen und Ergebnissen der anderen Gruppenmitglieder. Angriffe, die andere Peers gezielt diskreditieren oder einzelne Knoten bevorzugen, sind auf diese Weise ebenfalls realisierbar.

5.1 Der Einfluss von gefälschtem Feedback in FairNet

Wenn Peer A mit einer Wahrscheinlichkeit b gefälschtes Feedback über Peer B verbreitet, und Peer B mit einer Wahrscheinlichkeit q Nachrichten nicht kooperativ beantwortet, aus welchen Feedback-Objekten setzt sich dann das Repository eines Peers C zusammen, der ein Nachbar von A und B ist? In diesem Abschnitt wird davon ausgegangen, dass das von A gefälschte Feedback stets positiv ist, um die Reputation von Peer B zu steigern. Da negatives Feedback analog gehandhabt wird, kann auf eine ausführliche Beschreibung dieses Falles verzichtet werden. Als weitere Vereinfachung wird angenommen, dass das gefälschte Feedback gleichmäßig generiert wird. Es wird also nicht eine Runde ausschließlich falsches und eine Runde nur korrektes Feedback an andere Peers übermittelt, und das Repository eines Peers enthält Feedback, das ihm andere Peers übermittelt haben, zu gleichen Teilen[1]. Die Wahrscheinlichkeit p_{pos}, mit der korrektes positives Feedback über Peer B generiert wird, wenn $b = 0$, wurde bereits in Abschnitt 3.4 ermittelt. Jeder Peer hat mindestens $3^d - 1$ Kontakte. Wie viele dieser Peers kommen nun als Quelle für Feedback über einen der Kontakte in Frage?

[1] Abschnitt 5.3 zeigt, wie dies technisch zu realisieren ist.

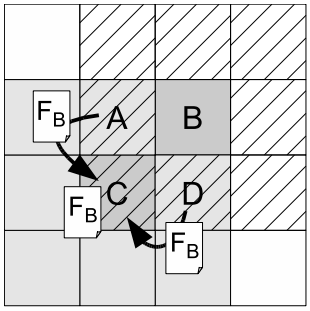

Abbildung 5.1: Feedback-Quellen in einem zweidimensionalen CAN.

Bild 5.1 skizziert die Problematik in einem zweidimensionalen CAN. Die Nachbarn von Peer C sind grau hinterlegt, und die Nachbarn von Peer B sind schraffiert dargestellt. In diesem Beispiel soll dargestellt werden, woher das Feedback stammt, das Peer C über Peer B besitzt. Zunächst generieren sämtliche Nachbarn von Peer B Feedback über ihn. Weil in dem hier vorgestellten Protokoll Feedback nur an Peers weitergegeben wird, die ebenfalls Nachbarn des Feedback-Subjekts sind, enthält das Repository von Peer C jedoch nur Feedback aus drei Quellen: die Beobachtungen von Peer C selbst, sowie den Peers A und D. Das bedeutet, dass die Peers A und D zum Einen bestimmen, *welches* Feedback sie weiterleiten. Zum Anderen können sie Feedback *manipulieren*, das andere Peers generiert haben, ohne dass Peer C diese Manipulation erkennen könnte. Es ist also beim Umgang mit gefälschtem Feedback sinnvoll, nur die direkten Quellen des Feedbacks zu berücksichtigen, und nicht von den Peers auszugehen, die es generiert haben. Gleichung 5.1 beschreibt die minimale Zahl[2] der Feedback-Quellen für das Repository eines Peers algebraisch:

$$c_f = \left\lfloor \frac{3^d - 1}{4} \right\rfloor + 1 \tag{5.1}$$

Somit erhöht sich die Wahrscheinlichkeit für positives Feedback bei einem Peer, der neben ehrlichem Feedback noch einen Anteil b gefälschtes positives Feedback verbreitet, auf den folgenden Wert:

$$\hat{p}_{pos} = \frac{c_f - 1}{c_f} \cdot p_{pos} + \frac{1}{c_f} \cdot \left(b \cdot 1 + (1 - b) \cdot p_{pos}\right) \tag{5.2}$$

Bei Verwendung dieser Annahmen erhält man eine Abschätzung für den schlimmsten Fall, d.h., wie sehr ein einzelner Peer die Gesamtaussage der Repositories seiner Nachbarn höchstens beeinflussen kann. Die Zahl der Feedback-Quellen erhöht sich sowohl bei unregelmäßiger Zonenaufteilung, als auch wenn der zu bewertende Nachbar mit mehr als nur einem Punkt an den Repository-Inhaber angrenzt. Existieren mehr verschiedene Feedback-Quellen, so reduziert sich der Einfluss eines einzelnen unehrlichen Peers.

[2] Bei einem CAN mit stochastischer Zonenaufteilung existieren möglicherweise mehr Feedback-Quellen, vgl. Abschnitt 2.3.

Für sich allein genommen stellt ein einzelner unehrlicher Peer in einem CAN mit mindestens drei Dimensionen nur ein geringes Bedrohungspotential dar. Beispielsweise beträgt für einen unkooperativen Peer mit einer lokalen Ausfallwahrscheinlichkeit von 50% in einem dreidimensionalen CAN[3] $p_{pos} = 0{,}05$. Gleichung 5.1 besagt, dass in diesem Falle Feedback aus mindestens sieben Quellen in das Repository eines Peers gelangt. Aus Gleichung 5.2 folgt, dass ein einzelner Lügner mit $b = 0{,}5$ die Wahrscheinlichkeit für positives Feedback nur auf $\hat{p}_{pos} = 0{,}11$ anheben kann. Mit Gleichung 5.3 lässt sich ermitteln, dass die Wahrscheinlichkeit dafür, dass der so begünstigte Peer fälschlich für kooperativ gehalten wird, nur von $1 \cdot 10^{-9}$ auf $1 \cdot 10^{-6}$ steigt.

$$P(|F_{pos}| \geq t) = \sum_{i=t}^{s} \binom{s}{i} \cdot p_{pos}^{i} \cdot (1 - p_{pos})^{s-i} \tag{5.3}$$

Für CAN mit höherdimensionalem Schlüsselraum, in denen also jeder Peer über noch mehr Nachbarn verfügt, die ihn mit Feedback versorgen, verringert sich diese Wahrscheinlichkeit sogar noch weiter.

Weitaus problematischer werden unehrliche Peers jedoch in Systemen mit einer hohen globalen Ausfallwahrscheinlichkeit, insbesondere wenn mehrere unehrliche Peers zusammenarbeiten, bzw. wenn ein einzelner Teilnehmer im Rahmen einer Sybil-Attacke [Dou02] mehrere Identitäten im CAN kontrolliert. Arbeiten z unehrliche Nachbarn eines Peers zusammen ($0 \leq z < c_f$), verändert sich Gleichung 5.2 wie folgt:

$$\hat{p}_{pos} = \frac{c_f - z}{c_f} \cdot p_{pos} + \frac{z}{c_f} \cdot \big(b \cdot 1 + (1 - b) \cdot p_{pos}\big) \tag{5.4}$$

Um den Schwellenwert t zu erreichen, bei dem ein Peer als kooperativ angesehen wird, muss nur ein ausreichender Anteil z/c_f aller möglichen Feedback-Quellen eines Peers zusammenarbeiten und gefälschtes positives Feedback über ihn generieren. Dieser Anteil lässt sich nach Umstellung von Gleichung 5.4 und Einsetzen von $t = s \cdot \hat{p}_{pos}$ folgendermaßen berechnen:

$$\frac{z}{c_f} = \frac{\frac{t}{s} - p_{pos}}{b - b \cdot p_{pos}} \tag{5.5}$$

Für das oben genannte Beispiel bedeutet dies, dass ein unkooperativer Peer, der nur die Hälfte seiner Anfragen korrekt verarbeitet oder weiterleitet, für kooperativ gehalten wird, sobald mindestens 58% der Peers, die Feedback über ihn abgeben können, ausschließlich (d.h. $b = 1$) gefälschtes positives Feedback verbreiten. Gleichung 5.5 demonstriert also, dass benachbarte Peers, die sich gegenseitig gefälschtes positives Feedback ausstellen, ihren Kooperationsgrad reduzieren können, ohne dafür einen Arbeitsbeweis befürchten zu müssen. Je mehr Nachbarn eines Peers zusammenarbeiten, desto weniger kooperativ kann sich dieser Peer verhalten.

[3] Parameter des Beispiels: 1.000.000 Peers, $t = 8$, $s = 10$, $p = 0{,}001$, $q = 0{,}5$.

5.2 Erkennung und Vermeidung von gefälschtem Feedback

Was die Erkennung von gefälschtem Feedback zu einer Herausforderung werden lässt, ist die Tatsache, dass Peers nicht stets kooperativ oder stets unkooperativ handeln, und ebenso wenig ausschließlich verfälschtes oder korrektes Feedback in Umlauf bringen müssen. Insgesamt ergeben sich daher folgende Anforderungen an eine Protokollerweiterung, die das Problem des unehrlichen Feedbacks lösen soll:

- Die Differenzierung von ehrlichem und unehrlichen Feedback soll mit einer hohen Genauigkeit erfolgen.
- Es muss für einen rationalen Peer ökonomisch sinnvoll sein, sich ehrlich zu verhalten. Ehrliches Feedback ist also zu belohnen, bzw. unehrliches zu bestrafen.
- Falsches Feedback darf nicht an andere Peers weitergegeben werden.
- Die Protokollerweiterung darf keine Schwachstellen aufweisen, die sich zum Angreifen des Protokolls ausnutzen ließen.
- Da sich ein Peer jederzeit entscheiden kann, ehrliches oder gefälschtes Feedback abzugeben, muss das Protokoll rasch auf Veränderungen reagieren.
- Bestehende Vertrauensbeziehungen zwischen Dritten (bzw. den Peers, die das Feedback-Subjekt von gefälschtem Feedback sind) sollen unbeeinträchtigt bestehen bleiben.

Neben diesen fachlichen Anforderungen sind noch eine Reihe von technischen Parametern zu berücksichtigen. So darf die Protokollerweiterung die P2P-Eigenschaften nicht verletzten (siehe Tabelle 2.1), muss also mit dezentralen Algorithmen arbeiten und ohne globales Wissen oder zentrale Koordination auskommen. Weiterhin muss das Protokoll effizient genug sein, um sinnvoll einsetzbar zu sein. Dies ist beispielsweise nicht gegeben, wenn die Protokollerweiterung mehr Ressourcen beansprucht, als das simple Wiederholen einer Anfrage verursacht.

Es existieren bereits Ansätze zum Umgang mit subjektiven, unehrlichen Informationen. Diese basieren zumeist darauf, ein zweites, 'übergeordnetes' Reputationssystem zu schaffen, das die Reputationsinformationen des 'untergeordneten' Reputationssystems bewertet. Dem entsprechend basieren die meisten Lösungen gegen gefälschtes Feedback auf den selben Grundlagen, die bereits in Abschnitt 2.6 vorgestellt wurden. Auf dem Bayes-Theorem bauen *Bayesian Truth Serum* [Pre04] und *Peer-Prediction* [MRZ05] auf. Der Nachteil dieser Systeme besteht darin, dass der Bayes-Ansatz nicht gut mit dynamischen Verhaltensänderungen umgehen kann. Ein anderes Beispiel ist CONFESS [JF04], das auf einem Spieltheorie-Ansatz basiert. Das Hauptproblem dieser Lösungen besteht darin, dass ein übergeordnetes Reputationssystem das Problem des gefälschten Feedbacks nicht löst, sondern nur auf eine neue semantische Ebene befördert: als Konsequenz würde ein drittes Reputationssystem benötigt, das die Vertrauenswürdigkeit eines Peers ermittelt, Angaben über die Vertrauenswürdigkeit der Reputationsinformationen eines anderen Peers zu machen.

Eine effektive Lösung für das Problem des gefälschten Feedbacks muss sich die Eigenschaften von FairNet zunutze machen. Hier gibt es zwei Möglichkeiten, gefälschtes Feedback zu erkennen: durch Vergleich mit den Aussagen anderer Peers, und durch den Abgleich von Feedback mit dem Ausgang einer Transaktion.

Vergleich mit den Erfahrungen anderen Peers. Diese Art der Erkennung wird beispielsweise hier [BB03a] angewandt: Ein Peer, dessen Aussagen um einen (vom Misstrauen des Teilnehmers) bestimmten Betrag von der durchschnittlichen Bewertung des fraglichen Knotens abweicht, wird von der Ermittlung des Vertrauenskoeffizientens ausgenommen. Auf das hier zugrunde gelegte Protokoll bezogen hieße das beispielsweise, einen Peer von der Vertrauensbestimmung auszuschließen, wenn er positives Feedback über einen anderen Peer abgibt, während alle anderen Kontakte mit negativen Feedback-Objekten gewertet haben.

Diese Herangehensweise hat den Vorteil, dass die Erkennung von unehrlichem Feedback bereits vor dessen Verwendung stattfindet. Das heißt, der Peer braucht weder selbst die Erfahrung zu machen, dass einer seiner Kontakte unkooperativ ist, noch dass das Feedback von einem Kontakt nicht korrekt ist. Dem stehen aber erhebliche Nachteile gegenüber: Hat ein Peer sein Verhalten verändert, so wird der erste ehrliche Peer, der diese Verhaltensänderung durch nun vom Durchschnitt abweichendes Feedback dokumentiert, als unehrlich eingestuft. Zu vergleichbaren Problemen kommt es, wenn sich ein Peer nur gegenüber einzelnen Kontakten unkooperativ verhält, beispielsweise indem er nur die Nachrichten eines Peers nicht bearbeitet, und sich ansonsten kooperativ verhält. Weil diese Manipulationsmöglichkeiten den Einsatz von Anreiz- oder Strafmechanismen verbieten, ohne die ehrliches Feedback jedoch nicht gefördert werden kann, soll dieser Weg nicht weiter verfolgt werden.

Erkennung nach Transaktionsende. Diese Art der Bewertung von Feedback wird beispielsweise in [LI04] verwendet. Der Ansatz basiert darauf, den *beobachteten Ausgang* einer Transaktion mit dem vorher von den Teilnehmern *vorhergesagten Ausgang* zu vergleichen. Teilnehmer, die dabei richtig prognostiziert haben, werden bei der nächsten Reputationsberechnung höher gewichtet. Um kurzfristige Schwankungen herauszufiltern, wird die Wichtung mittels exponentiellem gleitendem Durchschnitt (EMA) gefiltert.

Im hier verwendeten Protokoll zum Umgang mit unkooperativen Teilnehmern wird jeder an einer Transaktion involvierte Peer über dessen Ausgang informiert. Entweder, weil er als Initiator ein bzw. kein Anfrageergebnis (bzw. Rückmeldung bei anderen Operationen) erhält, oder weil er eine Feedback-Benachrichtigung bekommt. Daher kann jeder Peer das Feedback im eigenen Repository kurz nach Abschluss einer Transaktion überprüfen. Tabelle 5.1 zeigt die vier zu unterscheidenden Fälle.

	positiver Transaktionsausgang	**negativer Transaktionsausgang**
positives Feedback	Fb. als zutreffend erkannt	Fb. als *nicht* zutreffend erkannt
negatives Feedback	Fb. als *nicht* zutreffend erkannt	Fb. als zutreffend erkannt

Tabelle 5.1: Erkennung von (nicht) zutreffendem Feedback.

Von einer einzelnen Transaktion lassen sich noch keine sinnvollen Aussagen ableiten [JF04]. Über mehrere Transaktionen betrachtet sollte die Zahl des als nicht zutreffend erkannten Feedbacks bei unehrlichen Teilnehmern jedoch deutlich von ehrlichen Feedback-Quellen abweichen. Die algebraische und experimentelle Evaluierung am Ende dieses Kapitels wird demonstrieren, dass die im folgenden vorgestellte Protokollerweiterung diesen Anforderungen gerecht wird.

5.3 Der Umgang mit gefälschtem Feedback in FairNet

Um falsches Feedback zuverlässig zu erkennen und zu vermeiden, sind die folgenden Erweiterungen am in Kapitel 3 vorgestellten FairNet-Protokoll sowie den zugehörigen Datenstrukturen erforderlich:

Zuordnung von Feedback und Feedback-Versender. Beim Umgang mit möglicherweise gefälschtem Feedback ist es wichtig, die Herkunft des Feedbacks klar einem Kontakt zuschreiben zu können. Weiterhin dürfen sich von unterschiedlichen Peers übermittelte Feedback-Objekte nicht aus dem Repository verdrängen. Es muss möglich sein, alles (falsche) Feedback von einer Feedback-Quelle zu löschen. Genügte im ursprünglichen Protokoll noch ein gemeinsames Repository für alles Feedback über einen bestimmten Peer, muss nun also jeder Peer ω ein *getrenntes Repository* $R_{\omega,\lambda,\phi}$ für jede mögliche Kombination aus Feedback-Subjekt λ und Feedback-Quelle ϕ verwalten.

Wichtung des Feedbacks nach Vertrauenswürdigkeit des Versenders. Für jedes dieser Repositories ist ein *Wichtungsfaktor* $w_{\omega,\lambda,\phi}$ mit $0 \leq w \leq 1$ zu speichern, der die Vertrauenswürdigkeit des darin enthaltenen Feedbacks angibt. Der Wichtungsfaktor wird zum Herausfiltern von Schwankungen, die sich aus der zufälligen Streuung der Feedback-Objekte ergeben, mittels exponentiellem gleitendem Durchschnitt geglättet.

Bestrafung von Peers, die falsches Feedback versenden. Ein wichtiger Bestandteil des erweiterten Protokolls ist ein *Bestrafungsmechanismus*, der die Aufgabe hat, unehrliches Feedback wirtschaftlich unattraktiv zu machen. Dazu wird der in Kapitel 3 eingeführte Arbeitsbeweis wiederverwendet, allerdings ohne dass dafür positives Feedback generiert würde. Ein Schwellenwert t_p bestimmt, wie viel falsches Feedback ein Peer übermitteln darf, bis von ihm ein Arbeitsbeweis verlangt wird. Führt der Peer den Arbeitsbeweis nicht durch, so soll alles Feedback über ihn gelöscht werden, damit er sich die erforderliche Reputation zum Absenden von eigenen Anfragen erneut erarbeiten muss. Der Bestrafungsmechanismus baut somit auf existierende Strukturen auf; Änderungen am Protokoll sind nicht erforderlich.

Motivation zum Feedback-Verbreiten Der Bestrafungsmechanismus führt dazu, dass auch ehrliche Peers mit einer geringen Wahrscheinlichkeit für übermitteltes Feedback mit Arbeitsbeweisen bestraft werden könnten. Daher ist ein Anreiz erforderlich, der die Peers

motiviert, dennoch Feedback abzugeben. Ein Peer soll danach nur dann Feedback von anderen erhalten, wenn er selbst auch eigenes Feedback an andere Knoten weitergibt.

Im folgenden soll diskutiert werden, wie die Repository-Gewichte bestimmt werden, wie die gewichtete Anzahl des positiven Feedbacks im Repository bestimmt wird, wann ein Arbeitsbeweis als Strafe für falsches Feedback verlangt wird, und welche Änderungen am Gossip-Protokoll zur Feedback-Weiterleitung vorzunehmen sind, damit Peers kein gefälschtes Feedback 'aus zweiter Hand' weiterleiten. Des weiteren soll untersucht werden, wie der Anreizmechanismus ausgestaltet werden muss.

Bestimmung der Repository-Gewichtung. Mit Hilfe der Wichtungsfaktoren $w_{\omega,\lambda,\phi}$ kann ein Peer ω bestimmen, wie nützlich das Feedback in seinem Repository ist, das er von einem Kontakt ϕ über ein Feedback-Subjekt λ erhalten hat. Die 'Nützlichlichkeit' von Feedback lässt sich leicht bestimmen: ein Feedback-Objekt ist nützlich, wenn ein Peer auf Basis dieses Feedbacks eine richtige Entscheidung getroffen hat. Eine richtige Entscheidung wäre zum Beispiel, einen bestimmten Knoten zu vermeiden bzw. an einen anderen weiterzuleiten. Die Information, ob eine Entscheidung richtig oder falsch war, wird in Form der Feedback-Benachrichtigung nach Abschluss jeder Transaktion geliefert. Erhält ein Peer eine positive Benachrichtigung, so war seine Entscheidung für einen bestimmten Weiterleiter richtig, bei einer negativen Benachrichtigung falsch.

Eine P2P-Datenstruktur ist ein dynamisches System: Peers können das System jederzeit verlassen oder neu beitreten, und können ihr Verhalten zu jeden beliebigen Zeitpunkt ändern. Daher müssen die Wichtungsfaktoren geglättet werden, um stochastische Einflüsse herauszufiltern. Dabei sind Verfahren zu bevorzugen, die keine weiteren Datenbestände erfordern (wie das beispielsweise beim arithmetisches Mittel auf der Basis von Zeitreihen der Fall ist). Um das Protokoll möglichst ressourcensparend und kompakt zu gestalten, soll hier ein Verfahren nach [LI04] eingesetzt werden, bei dem der Wert für das neue Gewicht w' je nach Transaktionsausgang T und Anzahl positiven Feedbacks $|R^{pos}|$ auf 0 oder 1 gesetzt und mittels exponentiellem gleitendem Durchschnitt mit dem alten Gewicht w aggregiert wird. Der Parameter z_t bestimmt dabei, wie schnell neue Beobachtungen die alten verdrängen. Gleichung 5.6 beschreibt diesen Vorgang genauer:

$$w'_{\omega,\lambda,\phi} = (1-z_t)\cdot w_{\omega,\lambda,\phi} + z_t \cdot \begin{cases} 0 \text{ wenn } \{T \in T^{pos} \wedge |R^{pos}_{\omega,\lambda,\phi}| < t\} \vee \{T \in T^{neg} \wedge |R^{pos}_{\omega,\lambda,\phi}| \geq t\} \\ 1 \text{ wenn } \{T \in T^{pos} \wedge |R^{pos}_{\omega,\lambda,\phi}| \geq t\} \vee \{T \in T^{neg} \wedge |R^{pos}_{\omega,\lambda,\phi}| < t\} \end{cases} \tag{5.6}$$

Der Pseudocode in Abbildung 5.2 zeigt den vollständigen Aktualisierungsvorgang der Repositories eines Peers. Die Methode *updateRepositoryWeights* wird aufgerufen, wenn nach Beendigung einer Transaktion eine Benachrichtigung über den Transaktionsausgang vorliegt. Der Methode werden als Parameter der Peer, an den die ursprüngliche Transaktion weitergeleitet wurde, sowie der Transaktionsausgang (positiv oder negativ) übergeben. Beginnend mit Zeile 6 wird zunächst über alle Peers iteriert, die Feedback über den Weiterleiter übermittelt

haben. Bei richtigem Feedback wird in Zeile 13 das Gewicht des Repositories erhöht. Stimmt das Feedback nicht mit dem Transaktionsausgang überein, wird ab Zeile 16 die Wichtung verringert. Ab Zeile 19 wird nun geprüft, ob der Schwellenwert t_p zur Bestrafung falschen Feedbacks überschritten wird. Ist das der Fall, wird das falsche Feedback aus dem eigenen Repository gelöscht und ein Arbeitsbeweis vom unehrlichen Peer eingefordert. Falls dieser Beweis nicht erbracht wird, wird das Gewicht des Repositories auf 0 verringert und alles Feedback über ihn gelöscht.

```
1  updateRepositoryWeights(NextForwarder f, NotificationType n) {
2      // hole die gewichtete Anzahl positiven Feedbacks über Peer f
3      double v := getNumOfPosFb(f);
4
5      // iteriere über alle Peers, die Feedback über x gesendet haben
6      forall (q ∈ getFbSources(this.contactList, x)) {
7          Repository r := getRepository(f, q);
8          int l := r.getNumOfPosFb();
9
10         // Gewichte aktualisieren
11         if ((l ≥ t ∧ n = pos) ∨ (l < t ∧ n = neg)) {
12             // Feedback war richtig, also Gewicht erhöhen
13             r.weight := (1 − z_t) ∗ r.weight + z_t ;
14         } else {
15             // Feedback war falsch, also Gewicht verringern
16             r.weight := (1 − z_t) ∗ r.weight;
17
18             // wird das Feedback als gefälscht eingestuft?
19             if (l < v − t_p ∨ l > v + t_p) {
20                 // das falsche Feedback über f von q löschen
21                 deleteFeedback(f, q);
22
23                 // Arbeitsbeweis abfordern
24                 requestProW(q);
25                 waitForProWAnswer(timeout);
26                 if (ProW wurde nicht rechtzeitig beantwortet) {
27                     // kein ProW, also setze Gewicht auf 0 und lösche sein Fb
28                     r.weight := 0;
29                     deleteFeedback(q, this);
30                 }
31             }
32         }
33     }
34 }
```

Abbildung 5.2: Aktualisierung der Feedback-Repositories.

Gewichtung des Feedbacks Um die in Kapitel 3 beschriebenen Analysen und Formeln zur Bestimmung optimaler Parameterwerte weiterhin anwenden zu können, ist es erforderlich, die nach ihrer Quelle getrennten Repositories für das gleiche Feedback-Subjekt unter Berücksichtigung der Wichtungsfaktoren zusammenzuführen. Sei $Q_{\omega,\lambda}$ die Menge aller Peers, die Feedback über Peer λ an einen Knoten ω übermittelt haben, und $|R^{pos}_{\omega,\lambda,\phi}|$ die Anzahl positiven Feedbacks in einem Repository, das Peer ϕ übermittelt hat. Dann ist die Anzahl positiven Feedbacks über Knoten λ die gewichtete und mit der Summe der Wichtungsfaktoren der Quell-Peers normierte Anzahl des Feedbacks, wie in Gleichung 5.7 beschrieben.

$$f_{pos}(\lambda, Q_{\omega,\lambda}) = \frac{\sum\limits_{i \in Q_{\omega,\lambda}} |R^{pos}_{\omega,\lambda,i}| \cdot w_{\omega,\lambda,i}}{\sum\limits_{i \in Q_{\omega,\lambda}} w_{\omega,\lambda,i}} \tag{5.7}$$

Abbildung 5.3 illustriert diese Vorgehensweise. In dem Beispiel hat ein Peer Feedback von den Peers P_0, P_1 und P_2 über einen seiner Kontakte erhalten. Das Feedback von P_2 wird am höchsten gewichtet, das von P_1 am geringsten. Nach der Zusammenführung liegt die Anzahl positiven Feedbacks unter dem Schwellenwert t für kooperatives Verhalten.

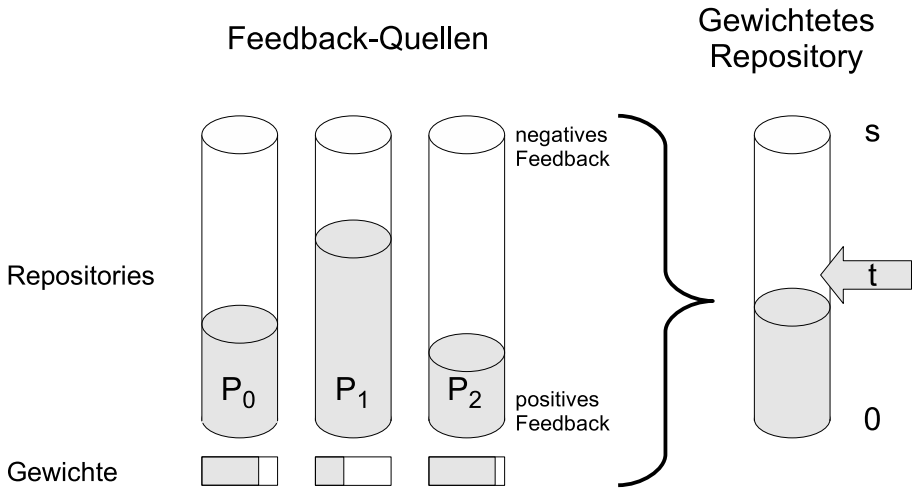

Abbildung 5.3: Zusammenführen von Feedback von unterschiedlichen Peers.

Die Abbildung 5.4 zeigt den Vorgang als Pseudocode. Die Methode *getNumOfPosFb()* ermittelt die gewichtete Anzahl positiven Feedbacks über Peer x. Damit überschreibt diese Methode *getNumOfPosFb()* die gleichnamige Methode im Pseudocode 3.5 der Methode *handleQuery()*. Die Methode erhält als Parameter den Peer übergeben, dessen Reputation bestimmt werden soll. Die Methode iteriert nun über alle Repositories, die Feedback über diesen Peer speichern, bestimmt die Zahl des positiven Feedbacks in den Repositories und gewichtet diese Zahl mit den Wichtungsfaktoren der Feedback-Quellen.

```
1  getNumOfPosFb(Peer x) {
2
3      // hole die Menge aller Peers, die Feedback über x gesendet haben
4      Peers Q := getFbSources(this.contactList, x);
5
6      double i, j := 0;
7      forall ( q ∈ Q) {
8
9          // hole das Repository für Peer x von Quelle q
10         Repository r := getRepository(x, q);
11
12         i := i + r.getNumOfPosFb() ∗ r.weight;
13         j := j + r.weight;
14     }
15     return i / j;
16 }
```

Abbildung 5.4: Methode *getNumOfPosFb*.

Wann wird ein Arbeitsbeweis als Strafe verlangt? Neben dem *Arbeitsbeweis für unkooperatives Verhalten* gibt es nun auch einen *Arbeitsbeweis für unehrliches Feedback*. Angenommen, ein Peer versucht die Zahl seiner Arbeitsbeweise für unkooperatives Verhalten durch das Verbreiten von gefälschtem Feedback zu reduzieren. Dann wird das Verbreiten von gefälschtem Feedback genau dann wirtschaftlich unattraktiv, wenn die Arbeitsbeweise für unehrliches Feedback teurer sind als kooperative Mitarbeit am Protokoll. Ebenso wie beim Arbeitsbeweis wegen unkooperativem Verhalten gilt jedoch: wird der Arbeitsbeweis von ehrlichen Peers verlangt, sinkt die Motivation der Teilnehmer rapide. Der Frage, wann ein Peer von einem anderen einen Arbeitsbeweis einfordert, kommt daher eine besondere Bedeutung zu.

Ein offensichtlicher Ansatz besteht darin, bei negativem Transaktionsausgang T^{neg} alle Peers zu strafen, die deutlich positiv gewertet, also mehr als die Schwelle zur Erkennung kooperativen Verhaltens t plus ein Schwellenwert t_p positives Feedback übermittelt haben. Bei positivem Transaktionsausgang T^{pos} würde analog vorgegangen. Ein Peer ϕ muss also einen Arbeitsbeweis für Peer ω nach Ausgang einer Transaktion T erbringen, wenn Peer ω an Peer λ weitergeleitet hat, und $\{T \in T^{pos} \wedge |R^{pos}_{\omega,\lambda,\phi}| < t - t_p\} \vee \{T \in T^{neg} \wedge |R^{pos}_{\omega,\lambda,\phi}| > t + t_p\}$. Dieser Ansatz hat jedoch einen schweren Nachteil: hat der Besitzer des Repositories ω seinen Schwellenwert t für die Erkennung kooperativen Verhaltens schlecht gewählt[4], und damit selbst eine falsche Einschätzung verursacht, würde der Arbeitsbeweis ehrliche Peers bestrafen.

Eine geeignetere Vorgehensweise besteht darin, nicht den Schwellenwert t als Basis für die Entscheidung heranzuziehen, sondern den bereits in Gleichung 5.7 berechneten gewichteten Durchschnittswert für die Anzahl positiven Feedbacks $f_{pos}(\lambda, Q_{\omega,\lambda})$. Der Vorteil liegt darin, dass dieser Wert die Meinung aller Feedback-Quellen $Q_{\omega,\lambda}$ sowie deren Glaubwürdigkeit wi-

[4] In dieser Arbeit verfügen alle Peers über den selben Parametersatz. Dies ist jedoch keine systemimmanente Einschränkung von FairNet; vgl. Ausblick auf Folgearbeiten, Abschnitt 6.1.

derspiegelt. Das führt dazu, dass nur noch Peers bestraft werden, die eine stark (siehe Schwelle t_p) vom Durchschnitt abweichende Meinung vertreten. Ein Peer ϕ muss also einen Arbeitsbeweis leisten, wenn $\{T \in T^{pos} \wedge |R^{pos}_{\omega,\lambda,\phi}| < f_{pos}(\lambda, Q_{\omega,\lambda}) - t_p\} \vee \{T \in T^{neg} \wedge |R^{pos}_{\omega,\lambda,\phi}| > f_{pos}(\lambda, Q_{\omega,\lambda}) + t_p\}$.

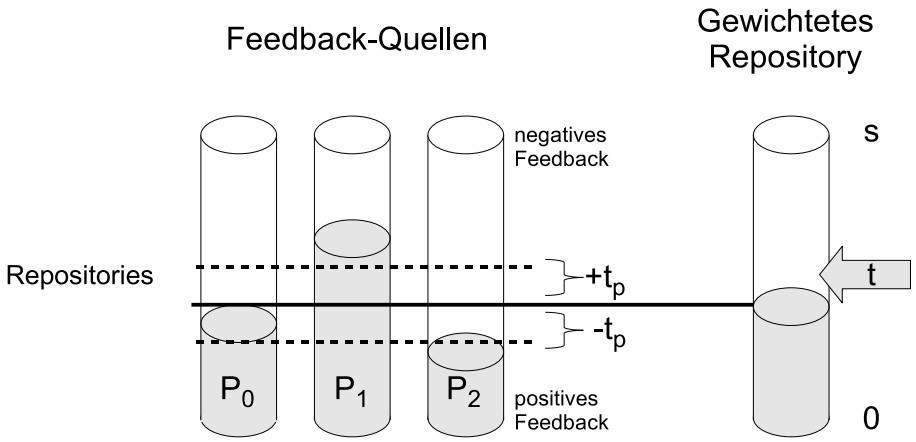

Abbildung 5.5: Bestimmung der zu bestrafenden Peers.

Abbildung 5.5 illustriert diesen Sachverhalt. Die gewichtete Anzahl an positivem Feedback, die in Abbildung 5.3 ermittelt wurde, ist als durchgezogene Linie dargestellt. Die gestrichelten Linien skizzieren die durch den Schwellenwert t_p festgelegten Toleranzen. Bei einem negativen Transaktionsausgang würde in diesem Beispiel also der Besitzer dieser Repositories von P_1 einen Arbeitsbeweis verlangen, weil er deutlich zu viel positives Feedback über den fraglichen Peer abgegeben hat, bei positivem Transaktionsausgang würde P_2 abgestraft.

Weiterleitung von Feedback. Wenn ein Peer das Feedback von einem anderen Knoten als falsch erkennt, löscht er dieses Feedback aus seinem Repository. Damit verhindert der Peer, dass er (1) das falsche Feedback an Dritte weitergibt, und (2) dass ein Knoten für das selbe falsche Feedback mehrfach bestraft wird. Darum kann der Mechanismus zur Weiterleitung von Feedback an andere Peers unverändert bleiben.

Anreize für das Weitergeben von Feedback Bisher wurde in FairNet davon ausgegangen, dass alle Peers freiwillig Feedback verbreiten. Nun besteht jedoch eine geringe Wahrscheinlichkeit, dass ein Peer für falsches, aber nicht unbedingt gefälschtes (vgl. Tabelle 5.1) Feedback mit Arbeitsbeweisen bestraft wird. Darum ist ein Anreiz für das Versenden von Feedback zwingend notwendig. In Abschnitt 3.5 wurde dazu eine kleine Erweiterung diskutiert, nach der jeder Peer Feedback nur an die Kontakte weiterleitet, die ihm im Gegenzug ebenfalls Feedback gesendet haben. Die Erweiterung ist eine Adaption der Strategie 'Tit-for-Tat' [Axe84] aus der Spieltheorie. Danach übermittelt jeder Peer zuerst Feedback, und folgt dann der Verhaltensweise seines Transaktionspartners. Diese Erweiterung wird nun obligatorisch für alle Peers.

Die bis zu dieser Stelle beschriebenen Algorithmen verfolgen zwei Ziele: (1) jeder Peer soll mittels Wichtungsfaktoren das Feedback höher werten, welches öfter zu richtigen Entscheidungen führt, und (2) Peers, die falsches Feedback abgeben, sollen mit Arbeitsbeweisen bestraft werden. Dies ist jedoch schwierig zu realisieren. Ehrlich abgegebenes Feedback kann durchaus als falsch klassifiziert werden (vgl. Tabelle 5.1), wenn sich das Feedback-Subjekt nicht vollständig kooperativ oder unkooperativ verhält, sondern beispielsweise die Hälfte der Nachrichten korrekt verarbeitet. Im folgenden Abschnitt soll analysiert werden, ob die hier vorstellten Algorithmen dazu führen, dass die Wichtungsfaktoren gefälschtes Feedback wirksam ausschließen können. Repositories mit gefälschtem Feedback müssen eine geringere Gewichtung erhalten als solche mit ehrlichem Feedback. Des weiteren soll gezeigt werden, dass bei einer konservativen Einstellung der Parameter ehrliche Peers nur mit einer sehr geringen Wahrscheinlichkeit mit einem Arbeitsbeweis abgestraft werden, unehrliche Knoten jedoch durch viele Arbeitsbeweise diszipliniert werden.

5.4 Analyse der Maßnahmen gegen gefälschtes Feedback

Die Analyse der Protokollerweiterung geht von den selben Annahmen aus wie die Analyse des Protokolls in Kapitel 3. Tabelle 5.2 zeigt die externen Parameter des um Maßnahmen zum Umgang mit unehrlichen Peers erweiterten FairNet-Protokolls. Im Vergleich zu Kapitel 3 neu hinzugekommen sind der Anteil des gefälschten Feedback b und die Wahrscheinlichkeit g für eine fehlgeschlagene Nachrichtenübermittlung. Die in der Tabelle angegebenen Vorgabewerte werden später bei der Evaluierung verwendet.

Parameter	**Symbol**	**Vorgabewert**
Dimensionalität	d	4
Anzahl Peers	N	10.000
Anteil unkooperativer Peers	u	5%
Globale Ausfallwahrscheinlichkeit über alle Peers	p	2,5%
Ausfallwahrscheinlichkeit unkooperativer Peers	q	50%
Anteil des gefälschten Feedbacks	b	$0 \cdots 1$
Wahrscheinlichkeit für eine fehlgeschlagene Nachrichtenübermittlung	g	$\approx 9{,}3\%$

Tabelle 5.2: Die exogenen Parameter des erweiterten FairNet-Protokolls.

Darüber hinaus setzt die Analyse auf die in Tabelle 5.3 aufgelisteten und bereits in Abschnitt 3.4 berechnete Parameter auf.

Parameter	**Symbol**	**Vorgabewert**
Schwellenwert zur Erkennung von kooperativen Peers	t	6
Größe der Feedback-Repositories	s	10
Mittlere Pfadlänge	δ	$\approx 3{,}9$
Wahrscheinlichkeit für positives Feedback im Repository	p_{pos}	$0 \cdots 1$

Tabelle 5.3: Bereits in Abschnitt 3.4 bestimmte Parameter.

Zuletzt führt die Protokollerweiterung einige neue Parameter ein, die die Reaktion der Peers auf möglicherweise gefälschtes Feedback steuern. Tabelle 5.4 zeigt eine Aufstellung der hinzugekommenen Parameter.

Parameter	**Symbol**	**Vorgabewert**
Schwellenwert für einen Arbeitsbeweis wegen falschem Feedback.	t_p	3,0
Faktor für die Glättung der Wichtungsfaktoren.	z_t	0,1

Tabelle 5.4: Die neuen Optimierungsparameter des erweiterten FairNet-Protokolls.

Der Schwellenwert t_p bestimmt, wann eine Feedback-Quelle einen Arbeitsbeweis erbringen muss. Der Schwellenwert geht dabei vom in Abschnitt 5.3 ermittelten gewichteten Durchschnitt aller Feedback-Objekte über ein Feedback-Subjekt aus. Das bedeutet, dass bei einem $t_p = 3{,}0$ ein Peer erst dann bestraft wird, wenn er bei einem negativen Transaktionsausgang mindestens drei positive Feedback-Objekte mehr übermittelt hat als der Durchschnitt. Der Parameter z_t bestimmt, wie stark die Wichtungsfaktoren für das übermittelte Feedback geglättet werden. Die Glättung erfolgt dabei durch den exponentiellen gleitenden Durchschnitt. Je größer z_t, desto weniger Gewicht wird auf Werte aus der Vergangenheit gelegt.

Erwartungswert für die Repository-Gewichte Zunächst soll ermittelt werden, welches Gewicht ein Repository mit Feedback von einem ehrlichen Peer im Durchschnitt erhält. Gleichung 5.6 beschreibt, wie der Wichtungsfaktor berechnet wird. Dafür müssen zuerst die Wahrscheinlichkeit p_t für ein als unkooperativ gewertetes Repository (Gleichung 5.9) und die Wahrscheinlichkeit g für eine fehlgeschlagene Nachrichtenübermittlung über eine Anzahl von δ Weiterleitern (Gleichung 5.10) berechnet werden:

$$
\begin{aligned}
p_t &= P(|R^{pos}_{\omega,\lambda,\phi}| < t) && (5.8)\\
&= \sum_{i=0}^{t-1} \binom{s}{i} \cdot (p_{pos})^i \cdot (1-p_{pos})^{s-i} && (5.9)\\
g &= 1-(1-p)^{\delta} && (5.10)
\end{aligned}
$$

Der Wichtungsfaktor w ist nun die Wahrscheinlichkeit dafür, dass das von einem Peer übermittelte Feedback von einem anderen Peer als korrekt erkannt wird, weil es mit dem beobachteten Transaktionsausgang übereinstimmt. Das bedeutet, w ist die Summe der Wahrscheinlichkeit für ein positives Repository und einen positiven Transaktionsausgang ($p_t \cdot g$) und der Wahrscheinlichkeit ($(1-p_t)\cdot(1-g)$) für den entgegengesetzten Fall:

$$
\begin{aligned}
w &= P(\text{FB. als korrekt erkannt})\\
&= p_t \cdot g + (1-p_t)\cdot(1-g)
\end{aligned}
\qquad (5.11)
$$

Nun wird untersucht, wie sich das Gewicht für Feedback von einen unehrlichen Peer vom oben ermittelten Gewicht unterscheidet. Ein unehrlicher Peer überträgt einen Anteil von b falschem positivem oder negativem Feedback, sowie einen Anteil von $(1-b)$ ehrlichem Feedback, welches mit Wahrscheinlichkeit p_{pos} positiv ist. Die Wahrscheinlichkeit p^{dis}_{pos}, mit der ein unehrlicher Peer positives Feedback generiert, beträgt also:

$$
p^{dis}_{pos} = \begin{cases} b\cdot 1 + (1-b)\cdot p_{pos} & \text{für falsches positives FB} \\ b\cdot 0 + (1-b)\cdot p_{pos} & \text{für falsches negatives FB} \end{cases} \qquad (5.12)
$$

Um nun den Wichtungsfaktor w_{dis} für ein Repository mit Feedback zu erhalten, das von einem unehrlichen Peer übermittelt wurde, ist p^{dis}_{pos} in Gleichung 5.9 einzusetzen und mit dem Ergebnis die Gleichung 5.11 neu zu berechnen:

$$
\begin{aligned}
p^{dis}_t &= \sum_{i=0}^{t-1} \binom{s}{i} \cdot \left(p^{dis}_{pos}\right)^i \cdot \left(1-p^{dis}_{pos}\right)^{s-i} && (5.13)\\
w_{dis} &= p^{dis}_t \cdot g + (1-p^{dis}_t)\cdot(1-g) && (5.14)
\end{aligned}
$$

Kosten durch Arbeitsbeweise für falsches Feedback Nun wird bestimmt, mit welchen Zusatzkosten durch Arbeitsbeweise ehrliche Peers beaufschlagt werden, wenn der in Abschnitt 5.3 beschriebene Algorithmus eingesetzt wird. Dabei werden alle Peers abgestraft, die um den Schwellenwert t_p mehr oder weniger positives Feedback übermittelt haben als der Durchschnitt. Gleichung 5.7 zeigt, wie Anzahl positiven Feedbacks von einem einzelnen Peer berechnet wird. In diese Gleichung geht das von allen Nachbarn gesendete Feedback sowie alle Wichtungsfaktoren dieser Nachbarn ein. Eine einfachere Abschätzung für den Durchschnitt

$\overline{h_{pos}}$ ist jedoch möglich: hat ein Peer viele ehrliche Nachbarn, oder ist der Anteil b des von ihnen übermittelten gefälschten Feedbacks gering, so ist der Anteil des gefälschten Feedbacks insgesamt vernachlässigbar. Der Durchschnitt für die Anzahl positiven Feedbacks $\overline{h_{pos}}$ für ein Repository der Größe s kann nun wie folgt abgeschätzt werden:

$$\overline{h_{pos}} = s \cdot p_{pos} \tag{5.15}$$

Gleichung 5.17 beschreibt nun die Anzahl der Arbeitsbeweise h_{honest}, die ein *ehrlicher* Peer pro Runde erbringen muss, weil das von ihm übermittelte Repository als *gefälscht* eingestuft wurde. Dies passiert, sobald der Peer (1) mehr als $\overline{h_{pos}} + t_p$ positives Feedback übermittelt hat, die Transaktion jedoch negativ ausging, bzw. (2) wenn er weniger als weniger als $\overline{h_{pos}} - t_p$ positives Feedback übertragen hat und die Transaktion positiv ausging. Zwei benachbarte Peers verfügen über $c_f = \lfloor (3^d - 1)/4 \rfloor + 1$ gemeinsame Kontakte (vgl. Gleichung 5.1), über die sie Feedback austauschen. Das bedeutet, jeder Peer kann von c_f benachbarten Knoten für falsches Feedback über ein bestimmtes Feedback-Subjekt abgestraft werden. Die Zahl der Arbeitsbeweise wegen falschem Feedback pro Runde entspricht nun der Summe der Wahrscheinlichkeiten für die beiden oben genannten Fälle multipliziert mit c_f.

$$f_{\mathfrak{j}}(j,k) = \begin{cases} 0 & \text{wenn } j > s \vee k < 0 \\ \sum_{i=j}^{k} \binom{s}{i} \cdot (p_{pos})^i \cdot (1 - p_{pos})^{s-i} & \text{sonst} \end{cases} \tag{5.16}$$

$$h_{honest} = c_f \cdot \left(g \cdot f_{\mathfrak{j}}(\lceil \overline{h_{pos}} + t_p \rceil, s) \; + \; (1-g) \cdot f_{\mathfrak{j}}(0, \lfloor \overline{h_{pos}} - t_p \rfloor) \right) \tag{5.17}$$

Aus Gründen der besseren Lesbarkeit wurde die Hilfsfunktion $f_{\mathfrak{j}}(j,k)$ aus der Gleichung 5.17 herausgelöst. Sie beschreibt die Wahrscheinlichkeit für ein Repository mit j bis k positiven Feedback-Elementen. Daneben berücksichtigt die Hilfsfunktion, dass in einem Repository nicht weniger als 0 oder mehr als s Feedback-Objekte vorhanden sein können. Aus der Zahl der Arbeitsbeweise pro Runde lässt sich der Erwartungswert für die Kosten des Protokolls für *ehrliche* Knoten nun wie folgt bestimmen:

$$E(\text{Kosten ehrlicher Knoten}) = h_{honest} \cdot c_{ProW} \tag{5.18}$$

Um mit diesem Gleichungssystem die Zahl der Arbeitsbeweise zu ermitteln, die ein *unehrlicher* Peer pro Runde erbringen muss, genügt es, p_{pos}^{dis} (s. Gleichung 5.12) in die Hilfsfunktion 5.16 einzusetzen. Man erhält somit eine neue Hilfsfunktion $f_{\mathfrak{k}}(j,k)$ und das folgende Gleichungssystem:

$$f_{\mathfrak{k}}(j,k) = \begin{cases} 0 & \text{wenn } j > s \vee k < 0 \\ \sum_{i=j}^{k} \binom{s}{i} \cdot \left(p_{pos}^{dis}\right)^i \cdot \left(1 - p_{pos}^{dis}\right)^{s-i} & \text{sonst} \end{cases} \tag{5.19}$$

$$h_{dishonest} = c_f \cdot \left(g \cdot f_{\mathfrak{k}}(\lceil \overline{h_{pos}} + t_p \rceil, s) \; + \; (1-g) \cdot f_{\mathfrak{k}}(0, \lfloor \overline{h_{pos}} - t_p \rfloor) \right) \tag{5.20}$$

Der Erwartungswert für die Kosten durch Arbeitsbeweise, die wegen gefälschtem Feedback pro Runde von unehrliche Knoten verlangt werden, beträgt somit:

$$E(\text{Kosten unehrlicher Knoten}) = h_{dishonest} \cdot c_{ProW} \tag{5.21}$$

Dynamik der Erkennung von unehrlichen Peers Um die Fähigkeit der FairNet-Erweiterung zum Umgang mit dynamischen Verhaltensänderungen zu ermitteln, sind zwei Fälle zu untersuchen: (1) die Zeit, die vergeht, bis ein vormals ehrlicher und nun unehrlicher Peer Arbeitsbeweise wegen gefälschtem Feedback leisten muss, und (2) die Zeit, bis sich die Wichtungsfaktoren der anderen Peers auf diese Verhaltensänderung eingestellt und das Feedback des Peers herabgestuft haben.

Zunächst soll bestimmt werden, wie viel Zeit vergeht, bis ein unehrlich gewordener Peer Arbeitsbeweise erbringen muss. Der Erwartungswert für ehrliches Feedback in einem Repository der Größe s beträgt $p_{pos} \cdot s$. Ein unehrlicher Peer, der einen Anteil b gefälschtes Feedback abgibt, generiert positives Feedback mit der in Gleichung 5.12 angegebenen Wahrscheinlichkeit. Er muss Arbeitsbeweise wegen gefälschtem Feedback erbringen, wenn er bei positivem Transaktionsausgang weniger als $p_{pos} \cdot s - t_p$ und bei negativem Transaktionsausgang mehr als $p_{pos} \cdot s + t_p$ positives Feedback übermittelt hat. Die Gleichungen 5.22 und 5.23 bestimmen nun den Erwartungswert dafür, wie viele Runden vergehen (vgl. Gleichung 3.29), bis ein unehrlich gewordener Peer für falsches Feedback $\mathfrak{t}_\mathfrak{d}$ mit Arbeitsbeweisen bestraft wird. Dabei werden zwei Fälle unterschieden: (1) der Peer generiert einen Anteil b gefälschtes *negatives* Feedback (Gleichung 5.22), und (2) der Peer generiert einen Anteil b gefälschtes *positives* Feedback (Gleichung 5.23).

$$E(\text{Zeit bis } \mathfrak{t}_\mathfrak{d} < p_{pos} \cdot s - t_p) = \frac{-t_p}{(\sigma_{pos} + \sigma_{neg}) \cdot (b \cdot 0 + (1-b) \cdot p_{pos} - p_{pos})} \tag{5.22}$$

$$E(\text{Zeit bis } \mathfrak{t}_\mathfrak{d} > p_{pos} \cdot s + t_p) = \frac{+t_p}{(\sigma_{pos} + \sigma_{neg}) \cdot (b \cdot 1 + (1-b) \cdot p_{pos} - p_{pos})} \tag{5.23}$$

Wird das Feedback nicht über einen kooperativen, sondern über einen unkooperativen Peer generiert, so ändert sich die Wahrscheinlichkeit p_{pos} auf p'_{pos} wie in Abschnitt 3.4 beschrieben.

Jetzt soll die Zahl der Runden ermittelt werden, die vergehen, bis sich die Wichtungsfaktoren der anderen Peers auf eine Verhaltensänderung eingestellt und das Feedback des unehrlichen Peers herabgestuft haben. Bei jeder Interaktion zweier Peers, bei auch der Feedback generiert wird, wird die Feedback-Gewichtung neu berechnet. Das neue Gewicht w' wird aus dem alten Gewicht w ermittelt, indem zunächst das alte Gewicht um den Faktor $(1 - z_t)$ reduziert wird. Wenn das Feedback als zutreffend erkannt wurde (vgl. Tabelle 5.1), wird anschließend der Wert z_t addiert. Der Wert z_t sorgt also für eine Glättung des Wichtungsfaktors über den zeitlichen Verlauf hinweg. Die Gleichung 5.6 bestimmt formal, wann Feedback zutreffend ist

und wann nicht. Die Wahrscheinlichkeit, das Feedback als korrekt erkannt wird, setzt sich zusammen aus der Wahrscheinlichkeit g für eine unbeantwortete Anfrage (Gleichung 5.10) und der Wahrscheinlichkeit p_t für ein Repository mit weniger als t positiven Feedback-Elementen (Gleichung 5.9) sowie aus den Wahrscheinlichkeiten für den entgegengesetzten Fall. Gleichung 5.24 berechnet nun w' wie folgt:

$$\begin{aligned} w' = & \quad (1 - z_t) \cdot w + z_t \cdot \begin{cases} 0 \text{ für falsches Feedback} \\ 1 \text{ für korrektes Feedback} \end{cases} \\ = & \quad (1 - z_t) \cdot w + z_t \cdot \big(p_t \cdot g + (1 - p_t) \cdot (1 - g)\big) \end{aligned} \tag{5.24}$$

Eine algebraische Bestimmung von w' ist schwierig: durch die Verwendung des exponentiellen gleitenden Durchschnitts fließen in die Berechnung alle Permutationen aus Transaktionsausgang, Wichtungsfaktor und Feedback-Inhalt für jeden einzelnen Weiterleitungsvorgang seit dem Beitritt des Peers zur Datenstruktur ein. Daher soll auf eine algebraische Herleitung von w' verzichtet werden; für eine praxisgerechte Bestimmung eines geeigneten Wertes für den Glättungsfaktor z_t ist die numerische Interpretation von Gleichung 5.24 ausreichend.

5.5 Evaluierung der Gegenmaßnahmen

Mit Hilfe des eben aufgestellten algebraischen Modells soll nun ermittelt werden, wie sich das Protokoll zum Umgang mit unehrlichen Peers unter verschiedenen Umgebungsbedingungen verhält. Dabei tritt gefälschtes Feedback in vier unterscheidbaren Fällen auf:

1. gefälschtes *positives* Feedback über einen *unkooperativen* Peer
2. gefälschtes *negatives* Feedback über einen *kooperativen* Peer
3. gefälschtes *positives* Feedback über einen *kooperativen* Peer
4. gefälschtes *negatives* Feedback über einen *unkooperativen* Peer

Da in FairNet über kooperative Peers ohnehin ausreichend positives und über unkooperative Peers negatives Feedback in Umlauf ist, wird auf eine Untersuchung der letzten beiden Fälle verzichtet.

Durch den Einsatz von Wichtungsfaktoren wird die Pfadauswahl und damit die durchschnittliche Pfadlänge beeinflusst, wovon wiederum sämtliche anderen Umgebungsparameter des Protokolls dynamisch abhängen (vgl. Kapitel 3). Die nun folgenden algebraischen Untersuchungen gehen zur Vereinfachung von unveränderlichen Pfadlängen aus. Eine umfassende Untersuchung des Gesamtsystems wird im Anschluss experimentell erfolgen. Wenn nicht anders angegeben, werden die in den Tabellen 5.2 bis 5.4 aufgelisteten Parameter für das CAN verwendet. Diese Parameter führen zu einer Wahrscheinlichkeit $p_{pos} = 0{,}85$ für positives Feedback über kooperative Knoten und $p'_{pos} = 0{,}13$ für positives Feedback über unkooperative Knoten. Die globale Ausfallwahrscheinlichkeit, also die Wahrscheinlichkeit, dass der nächste Empfänger einer Nachricht diese nicht korrekt bearbeitet, beträgt hier $p = 2{,}5\%$. Das führt dazu, dass

gemäß Gleichung 5.10 bei einer mittleren Pfadlänge $\delta = 3{,}9$ insgesamt $9{,}3\%$ aller Anfragen nicht beantwortet werden.

Der Zusammenhang zwischen den Parametern der verwendeten CAN-Umgebung und den ermittelten Wichtungsfaktoren ist dabei nicht intuitiv erkennbar. Bei Verwendung der in Tabelle 5.4 beschrieben Werte erhält das Feedback, das ehrlichen Peers abgegeben haben, gemäß Gleichung 5.11 im Mittel eine Wichtung von $w = 0{,}90$ für Feedback über kooperative Knoten, und $w = 0{,}54$ für Feedback über unehrliche Peers. Das heißt, dass auch das Feedback eines vollkommen ehrlicher Peers kein Gewicht von $1{,}0$ erhält. Das ist jedoch nicht maßgeblich für das Protokoll; vielmehr kommt es darauf an, dass das Feedback von einem unehrlichen Peer eine *geringere* Wichtung erhält. Im Folgenden soll daher untersucht werden, ob die Anwendung der Protokollerweiterung zum Umgang mit unehrlichen Knoten unter allen Umgebungszuständen gefälschtes Feedback niedriger gewichtet als ehrliches Feedback.

Wichtung des Feedbacks von ehrlichen Peers. Zuerst soll ermittelt werden, welche Wichtungsfaktoren w ehrliche Teilnehmer erreichen können, wenn der Anteil der unkooperativen Teilnehmer u im CAN variiert wird. Abbildung 5.6 zeigt den Verlauf des Wichtungsfaktors für *ehrlich* abgegebenes Feedback über kooperative und unkooperative Peers bei einem Anteil von $u = 0$ bis 100% unkooperativen Teilnehmern, die je 50% Anfragen korrekt bearbeiten. Es ist zu sehen, dass der Wichtungsfaktor w über ein kooperatives Feedback-Subjekt zunächst sinkt, bei etwa 27% einen Minimumpunkt erreicht und dann wieder steigt, wohingegen der Wichtungsfaktor für Feedback über ein unkooperatives Feedback-Subjekt von $0{,}5$ an kontinuierlich steigt. Doch wie lassen sich diese Kurvenverläufe erklären?

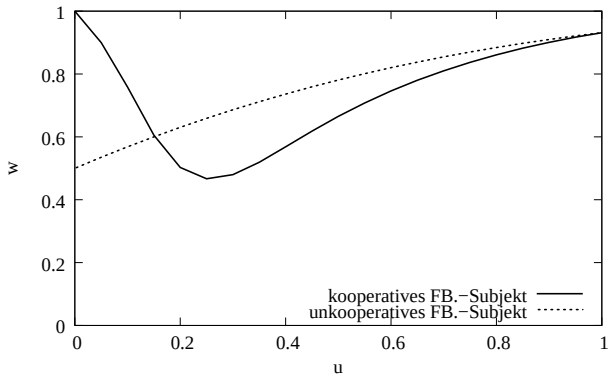

Abbildung 5.6: Der Wichtungsfaktor bei veränderlichem Anteil unkooperativer Teilnehmer.

Aus Gleichung 5.11 folgt, dass w von der Wahrscheinlichkeit abhängt, mit der eine Nachricht im CAN beantwortet wird. Daher ist zu erwarten, dass auch das Feedback von einem ehrlichen Peer eine niedrige Gewichtung erhalten kann, wenn sich das Gesamtsystem durch eine hohe Unsicherheit auszeichnet. Wenn eine weitergeleitete Nachricht mit der gleichen Wahrscheinlichkeit korrekt bearbeitet oder verworfen wird, ist *jedes* Feedback mit einer Wahrscheinlichkeit von 50% falsch, und $w = 0{,}5$. Dieser Fall tritt für unkooperative Feedback-Subjekte

ein, wenn es sehr wenige unkooperative Peers im System gibt, die mit 50% Nachrichten verwerfen. Daher beginnt die Kurve, die den Wichtungsfaktor für Feedback mit unkooperativen Feedback-Subjekten zeigt, bei $0{,}5$. Nimmt die Zahl der unkooperativen Teilnehmer zu, so sinkt zugleich die Unsicherheit: an unkooperative Peers weitergeleitete Anfragen bleiben häufiger als zu 50% unbeantwortet, also stimmt das das negative Feedback über unkooperative Teilnehmer besser mit der beobachteten Realität überein, und der Wichtungsfaktor für dieses Feedback steigt. Auf der anderen Seite führt auch die globale Ausfallwahrscheinlichkeit zu einem hohen Maß an Ungewissheit über das Feedback mit kooperativem Subjekt. Der Erwartungswert für den Anteil an unkooperativen Peers mit lokalen Ausfallwahrscheinlichkeiten $q = 0{,}5$ in einem CAN mit einer Pfadlänge $\delta = 3{,}9$, bei dem Anfragen mit 50% Wahrscheinlichkeit unbeantwortet bleiben, beträgt $u = (1 - \sqrt[\delta]{0{,}5})/q \approx 0{,}32$. Auf diese Weise entsteht der Minimum-Punkt im Verlauf des Wichtungsfaktors über Feedback mit kooperativem Subjekt: egal, welches Feedback über ein kooperatives Feedback-Subjekt in Umlauf ist, bei einem Anteil von einem Drittel unkooperativer Teilnehmer im System schlägt eine Transaktion mit einer Wahrscheinlichkeit von 50% fehl, und das gegebene Feedback stimmt in der Hälfte aller Fälle nicht mit den Beobachtungen überein.

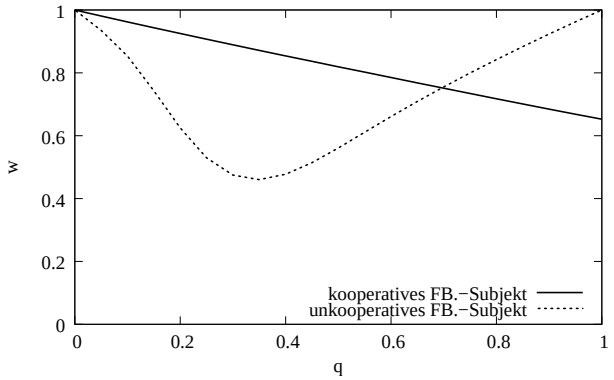

Abbildung 5.7: Der Wichtungsfaktor bei veränderlicher Ausfallwahrscheinlichkeit.

Die Abbildung 5.7 zeigt, dass diese Fälle nicht nur auftreten, wenn die Zahl der unkooperativen Teilnehmer variiert wird, sondern auch, wenn bei konstanter Zahl der unkooperativen Peers deren lokale Ausfallwahrscheinlichkeit verändert wird. Hier wurde ein Anteil von 10% unkooperativen Peers festgelegt. Je höher nun die lokale Ausfallwahrscheinlichkeit dieser Peers, desto geringer wird die Wahrscheinlichkeit, dass eine Nachricht beantwortet wird, und desto weniger stimmt das über kooperative Peers generierte positive Feedback mit den Beobachtungen der Teilnehmer überein.

Wichtung des Feedbacks unehrlicher Peers. Der vorangegangene Abschnitt hat gezeigt, dass Feedback von ehrlichen Knoten unterschiedliche Wichtungen erhalten kann. Die Wichtung hängt von dem Maß an Unsicherheit im System ab, wobei die Unsicherheit am größten ist, wenn eine Nachricht mit 50% Wahrscheinlichkeit beantwortet wird oder nicht. Nun kommt es darauf an, dass gefälschtes Feedback von unehrliche Peers auch unter unsicheren Umstän-

den nicht höher gewichtet wird als das Feedback von ehrlichen Knoten, d.h., dass gefälschtes Feedback keinen Einfluss auf Routing-Entscheidungen und die Zahl der verlangten Arbeitsbeweise wegen unkooperativem Verhalten nehmen kann.

Feedback wird in FairNet jedoch primär dazu generiert, unkooperative Teilnehmer zu disziplinieren. Es spiegelt nicht zwingend die beobachtete Realität wieder, sondern soll nur dafür sorgen, dass unkooperative Peers viele und kooperative Knoten wenige Arbeitsbeweise erbringen müssen. Das bedeutet, dass über einen Knoten mit einer lokalen Ausfallwahrscheinlichkeit von 20% nicht auch 20% negatives und 80% positives Feedback generiert wird, sondern erheblich mehr negatives (vgl. Kapitel 3). Deswegen besteht die Gefahr, dass unter Unsicherheit gefälschtes Feedback eventuell höher gewichtet wird als ehrliches Feedback. Die Abbildungen 5.8 und 5.9 untersuchen diese Problemstellung. Dabei entsprechen die Kurvenverläufe für ehrliche Peers ($b = 0$) dem Abbildung 5.6 zugrundeliegenden Experiment. Davon ausgehend wird nun untersucht, welche Gewichte das Feedback unehrlicher Peers erhält, wenn diese mit einem Anteil von b gemäß Gleichung 5.12 gefälschtes negatives Feedback über einen kooperativen Peer (Abb. 5.8) bzw. gefälschtes positives Feedback über einen unkooperativen Peer (Abb. 5.9) erzeugen.

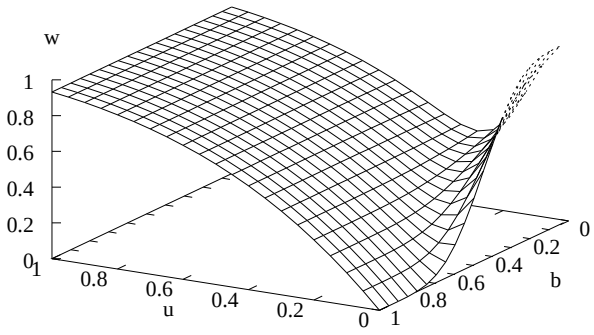

Abbildung 5.8: Der Wichtungsfaktor bei gefälschtem negativem FB über kooperative Peers.

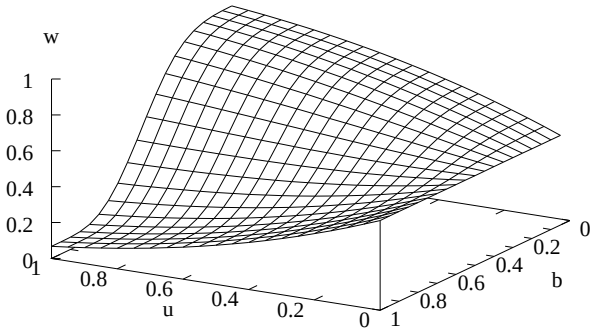

Abbildung 5.9: Der Wichtungsfaktor bei gefälschtem positivem FB über unkooperative Peers.

Die Abbildungen 5.8 und 5.9 zeigen ein sehr positives Resultat: die Gewichte w für Peers, die gefälschtes Feedback abgeben, sind höchstens ebenso groß wie die von ehrlichem Feedback. Das bedeutet, dass bei der Anwendung des hier analysierten Algorithmus unehrliche Teilnehmer die Effektivität von FairNet bei der Disziplinierung oder Umgehung unkooperativer Knoten nur unter statistisch unsicheren Umständen mit $g \approx 50\%$ beeinträchtigen können. Dies sind jedoch extreme Zustände: sie bedingen entweder eine außerordentlich hohe globale Ausfallwahrscheinlichkeit, Weiterleitungen über sehr viele Peers oder sehr viele unkooperative Peers mit hoher lokaler Ausfallwahrscheinlichkeit. Unter realistischen Bedingungen (kurze Pfade zwischen Anfragesteller und Beantworter, geringe Ausfallwahrscheinlichkeiten) wird gefälschtes Feedback nur mit einer sehr geringen Gewichtung in die Berechnung der Vertrauenswürdigkeit einbezogen.

Wie viele Arbeitsbeweise werden für unehrliches Feedback verlangt? Damit es unattraktiv wird, gefälschtes Feedback in Umlauf zu bringen, sollen Arbeitsbeweise von Peers verlangt werden, die Feedback verbreitet haben, das nicht den direkt beobachteten Gegebenheiten entspricht. Gemäß Abschnitt 5.3 wird dieser Arbeitsbeweis immer dann von einem Peer A verlangt, wenn A um den Wert t_p mehr positives Feedback über einen Peer B generiert hat als der Durchschnitt der anderen Peers, und eine an B weitergeleitete Transaktion fehlgeschlagen ist. Ebenso soll der Arbeitsbeweis verlangt werden, wenn B eine Transaktion korrekt ausgeführt hat, A jedoch um t_p weniger positives Feedback generiert hat als der Durchschnitt.

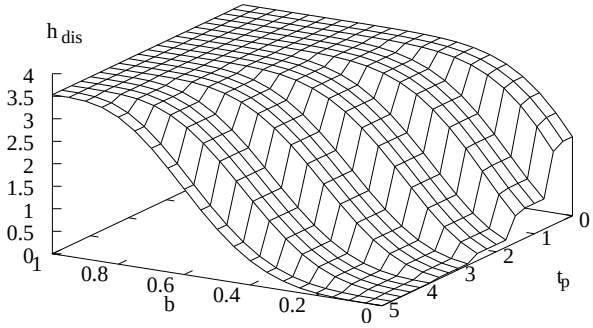

Abbildung 5.10: Zahl der Arbeitsbeweise bei unehrlichem Feedback über kooperative Peers.

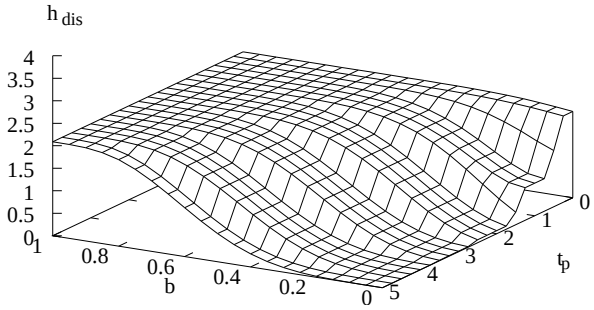

Abbildung 5.11: Zahl der Arbeitsbeweise bei unehrlichem Feedback über unkooperative Peers.

Die Abbildungen 5.10 und 5.11 beschreiben die Zahl der verlangten Arbeitsbeweise pro Runde in Abhängigkeit von dem Schwellenwert t_p und dem Anteil des gefälschten Feedbacks b. Ehrliche Peers werden dabei durch $b = 0$ repräsentiert. Da in der hier verwendeten CAN-Umgebung jeder Peer pro Runde durchschnittlich $\delta = 3{,}9$ Nachrichten weiterleitet, können von jedem Peer auch bis zu $3{,}9$ Arbeitsbeweise verlangt werden. Das Diagramm 5.10 zeigt die Zahl der Arbeitsbeweise, wenn mit Anteil b negatives Feedback über einen kooperativen Peer gefälscht wird. Das Diagramm zeigt, dass eine scharfe Differenzierung zwischen ehrlichen ($b = 0$) und unehrlichen ($b > 0$) Peers möglich ist: ab $t_p > 3$ werden von ehrlichen Peers keine Arbeitsbeweise mehr verlangt. Unehrliche Peers werden jedoch schnell abgestraft, wenn sie gefälschtes Feedback in relevanten Größenordnungen verbreiten.

Diagramm in Abbildung 5.11 zeigt die Zahl der verlangten Arbeitsbeweise für gefälschtes positives Feedback über unkooperative Peers. Wie aus den vorangegangenen Untersuchungen hervorgeht, stellt dieser Fall eine Herausforderung für das Protokoll dar: da die unkooperative Peers hier mit Wahrscheinlichkeit $q = 50\%$ Nachrichten verwerfen, ist die Differenzierung zwischen gefälschtem und ehrlichem Feedback schwierig. Die Abbildung 5.11 zeigt nun ein sehr positives Resultat: unter dieser Unsicherheit halbiert sich die Zahl der verlangten Arbeitsbeweise von unehrlichen Peers zwar nahezu, ehrliche Peers werden jedoch nicht mit zusätzlichen Kosten beaufschlagt! Ein Schwellenwert von $t_p > 3$, der in Abbildung 5.10 als geeignet identifiziert wurde, ist also auch hier uneingeschränkt anwendbar.

Die algebraische Evaluierung hat gezeigt, dass die vorgestellte Protokollerweiterung in der Lage ist, auch unter stark wechselnden Umgebungsbedingungen zwischen ehrlichen und un-

ehrlichen Peers zu differenzieren. Allerdings mussten dabei einige praxisferne Annahmen getroffen werden, zum Beispiel Peers mit gleich großen Zonen, unveränderliche Pfadlängen und eine rundenbasierte Abarbeitung von Anfragen. Der folgende Abschnitt soll nun mit Hilfe von Experimenten untersuchen, wie sich eine Implementierung dieses Protokolls in der Praxis verhält.

Experimentelle Evaluierung Die vordringlichste Frage im Zusammenhang mit der Erweiterung des FairNet-Protokolls lautet: zahlt es sich in einer realen FairNet-Implementierung für einen rationalen Peer aus, nur noch ehrliches Feedback abzugeben? Ein Experiment soll diese Frage beantworten. Dazu wird ein CAN mit 1.000 Peers, einem dreidimensionalen Schlüsselraum und einer Repository-Größe von $s = 10$ herangezogen.

Es war technisch nicht möglich, dieses Experiment mit dem selben Setup wie die Experimente in Kapitel 3 durchzuführen. Konnte bisher die Feedback-Ausbreitung simuliert werden, benötigt hier nun jeder Peer eigene Arbeitskopien der Feedback-Objekte. Bei 10.000 Peers wird dabei die Hauptspeichergröße von 2 GB je Rechner im Cluster überschritten. Die durchschnittliche Pfadlänge als wichtigster Parameter ist jedoch in einem dreidimensionalen CAN und 1.000 Peers vergleichbar mit einem vierdimensionalen mit 10.000 Knoten (3,6 vs. 3,8), daher lassen sich die gewonnenen Erkenntnisse übertragen.

In dieser Umgebung wurde nun ein Anteil von 5% unehrlicher, aber kooperativer Peers eingebracht. Diese Knoten variierten die Rate des abgegebenen gefälschten Feedbacks von $b = 0$ bis $b = 1$. Um die Erkennung von gefälschtem Feedback zu erschweren, verhielt sich ein Anteil von 5% der Knoten im System unkooperativ ($q = 1{,}0$), wobei es jedoch keine unehrlichen und zugleich unkooperativen Knoten gab. Die Wahrscheinlichkeit für eine unbeantwortete Nachricht liegt hier bei 16%, d.h., sie ist weit entfernt von einer Wahrscheinlichkeit von 50%, bei der sich gefälschtes Feedback nicht mehr von ehrlich abgegebenem unterscheiden lässt.

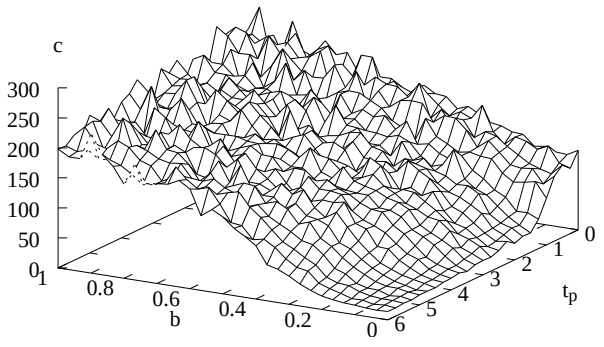

Abbildung 5.12: Die Gesamtkosten unehrlicher kooperativer Peers.

Nun wurde über 500.000 Anfragen gemessen, welche Gesamtkosten einem unehrlichen, aber kooperativen Knoten entstehen, wenn die anderen Peers unehrliches Feedback anhand eines Schwellenwertes von t_p =0 bis $t_p = 6$ erkennen. Abbildung 5.12 zeigt das Ergebnis des Experiments. Die Abbildung zeigt, dass ab einer Schwelle $t_p \approx 4$ ehrliche Peers ($b = 0$) nur die Kosten der kooperativen Mitarbeit tragen müssen, aber keine Arbeitsbeweise wegen falschem

Feedback befürchten müssen. Des weiteren ist zu sehen, dass selbst bei einem sehr konservativen Schwellenwert von $t_p \geq 6$ unehrliche Peers bereits erhebliche Mehrkosten in Kauf nehmen müssen, selbst wenn sie nur 30% falsches und 70% richtiges Feedback abgeben. Des weiteren zeigt Abbildung 5.12, dass die aus der numerischen Analyse der FairNet-Erweiterung abgeleiteten Erkenntnisse in die Praxis übertragbar sind: Abbildung 5.10 gibt die numerisch bestimmte Anzahl der Arbeitsbeweise (die den Großteil der Kosten unehrlicher Peers ausmachen) unter vergleichbaren Umgebungsbedingungen an, und zeigt einen angenähert identischen Verlauf wie Abbildung 5.12.

Nachdem experimentell bestätigt wurde, dass die Protokollerweiterung das Generieren von gefälschtem Feedback für *einzelne* rationale Peers unattraktiv macht, soll nun geprüft werden, ob es auch gegen *Kollaborationsangriffe* wirksam ist. Hierbei generieren Peers über einen benachbarten unkooperativen Knoten fortwährend gefälschtes positives Feedback. Der Angriff ist erfolgreich, wenn der Knoten trotz unkooperativem Verhalten keine Arbeitsbeweise erbringen muss. Das heißt, bei einem erfolgreichen Angriff kann der Knoten durch unkooperatives Verhalten Einsparungen realisieren, ohne durch Arbeitsbeweise Mehrkosten davonzutragen.

Ein Experiment soll die Durchführbarkeit eines derartigen Angriffs überprüfen. Hierzu werden wieder 1.000 Peers in einem CAN mit dreidimensionalem Schlüsselraum verwendet. 5% der Knoten verhalten sich unkooperativ, indem sie die Hälfte der Nachrichten nicht beantworten ($q = 0{,}5$). Im Experiment werden die unkooperativen Peers nun von $0, 1, \cdots, 3^d - 1$ unehrlichen Peers umgeben, die für diesen unkooperativen Peer korrektes Feedback mit einem Anteil $1 - b$ und gefälschtes positives Feedback mit einem Anteil b generieren. Gemessen werden die absoluten Kosten der unkooperativen Peers, d.h. die Kosten für das Beantworten und Weiterleiten von 50% der Anfragen sowie das Ausführen von Arbeitsbeweisen. Abbildung 5.13 zeigt das Ergebnis des Experiments. Aus der Abbildung wird ersichtlich, dass die Kosten des unkooperativen Teilnehmers ohne Unterstützung durch gefälschtes Feedback etwa $c = 400$ betragen. Die Abbildung zeigt auch, dass solche Kollaborationsangriffe prinzipiell möglich sind, aber einen großen Anteil von unehrlichen Teilnehmern mit hohen Anteilen gefälschten Feedbacks benötigen. Im verwendeten dreidimensionalen Schlüsselraum hat jeder Peer 26 Nachbarn – damit ein unkooperativer Peer von gefälschtem Feedback profitieren kann, müssen aber wenigstens 18 seiner Nachbarn fast ausschließlich ($b \geq 0{,}9$) gefälschtes positives Feedback abgeben. Betrachtet man die in Abbildung 5.12 abgetragenen Kosten für unehrliches Feedback, wird deutlich, dass dieser Angriff für die Angreifer grundsätzlich nicht ökonomisch rational ist: damit ein Peer Kosten von $c < 5$ realisiert[5], müssen 18 andere Arbeitsbeweise wegen falschem Feedback auf sich nehmen.

Abbildung 5.13 zeigt jedoch auch, dass die Kosten für unkooperatives Verhalten bereits bei einem geringeren Anteil unehrlicher Peers signifikant unter $c = 400$ sinkt. Durch den Einsatz der in diesem Kapitel beschriebenen Wichtungsfaktoren sollte die Zahl der Arbeitsbeweise, die ein unkooperativer Knoten erbringen muss, zumindest so lange von gefälschtem Feedback unabhängig bleiben, wie die Mehrheit seiner Nachbarn ehrlich ist.

[5] Zum Vergleich: die Kosten der kooperativen Mitarbeit betragen $c = 11{,}7$ pro Runde.

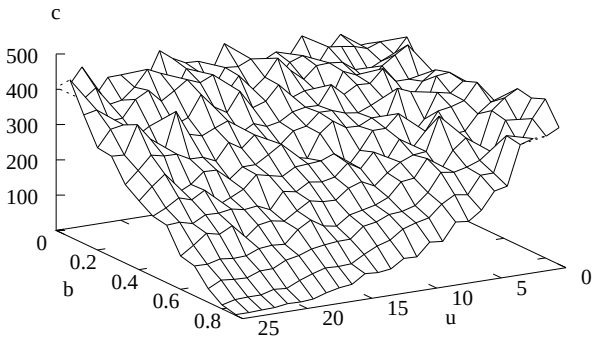

Abbildung 5.13: Kollaborationsangriff ohne Wichtungsfaktoren.

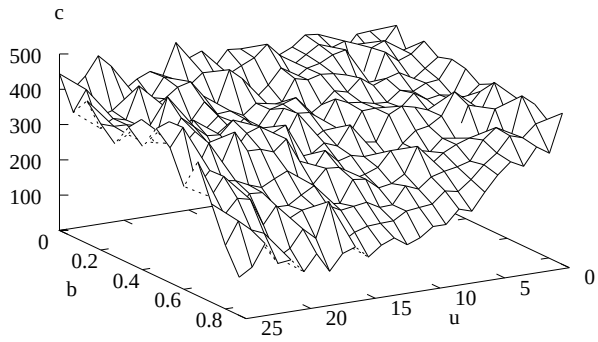

Abbildung 5.14: Kollaborationsangriff mit Wichtungsfaktoren.

Für das Diagramm in Abbildung 5.14 wurde das oben beschriebene Experiment nun unter Einsatz der Wichtungsfaktoren wiederholt. Die Abbildung zeigt ein sehr positives Resultat: selbst bei 25 von 26 unehrlichen Nachbarn sind die Kosten des unkooperativen Knotens noch immer erheblich höher als die Kosten der kooperativen Mitarbeit! Des weiteren zeigt der direkte Vergleich der Abbildungen 5.13 und 5.14, dass sich die Kosten des unkooperativen Teilnehmers bei Einsatz der Wichtungsfaktoren erst bei einem erheblich größeren Anteil von unehrlichen Knoten und bei einer hohen Rate von gefälschtem Feedback unter die Marke von $c = 400$ fallen.

Die Evaluierung hat gezeigt, dass die FairNet-Erweiterung effektiv wirksam ist im Umgang mit unehrlichem Feedback. Es wurde nachgewiesen, dass gefälschtes Feedback erst bei einem sehr großen Anteil vom Gesamtfeedback dazu führt, dass die Kosten von unkooperativen Teilnehmern reduziert werden. Weiter konnte gezeigt werden, dass Teilnehmer, die verfälschtes Feedback in praktisch wirksamen Mengen verbreiten, sehr hohe Kosten tragen müssen, während ehrlichen Knoten keine Mehrkosten entstehen.

5.6 Diskussion der Design-Entscheidungen

Wie bisher gezeigt wurde, ist die vorgestellte Protokollerweiterung zur Vermeidung von gefälschtem Feedback leicht implementierbar, kommt mit wenigen zusätzlichen Datenstrukturen aus und ist nur schwer zu überwinden. Nun stellt sich jedoch die Frage, ob es nicht einen anderen Algorithmus gibt, der zu einer besseren Differenzierung zwischen ehrlichem und gefälschtem Feedback führt. Aus diesem Grund sollen in diesem Abschnitt einige grundsätzliche Überlegungen diskutiert werden, die die Ausgestaltung des Algorithmus maßgeblich bestimmt haben.

Hat der Bestrafungsmechanismus Auswirkungen auf das abgegebene Feedback? Arbeitsbeweise für falsches Feedback sind problematisch, wenn die Gefahr besteht, dass damit ehrliche Peers bestraft werden. Wenn ein rationaler Peer mit einem Arbeitsbeweis rechnen

muss, obwohl er Feedback exakt protokollgerecht erzeugt hat, so wird er sein abgegebenes Feedback anpassen. Der Peer muss Feedback übermitteln, um selbst welches zu erhalten. Er wird jedoch nur noch Feedback an seine Nachbarn weiterleiten, das mit einer sehr hohen Wahrscheinlichkeit als korrekt erkannt wird. Dies führt dazu, dass nur noch Feedback über 'unkritische' Peers ausgetauscht wird, die vollständig kooperativ oder unkooperativ sind und ihr Verhalten über einen längeren Zeitraum nicht mehr verändert haben. Feedback über neue Peers oder solche, deren Verhalten häufigen Schwankungen unterliegt, wird nicht mehr übertragen – genau dieses Feedback ist jedoch in der Praxis am wichtigsten. Deswegen ist eine sehr sorgfältige Bestimmung des Schwellenwertes für den Arbeitsbeweis wegen gefälschtem Feedback von großer Bedeutung.

Warum kann auf einen Bestrafungsmechanismus nicht verzichtet werden? Der Bestrafungsmechanismus sorgt dafür, dass es für einen rationalen Peer *wirtschaftlich* wird, protokollgerecht erzeugtes Feedback zu verbreiten. Mit den Wichtungsfaktoren können die Peers zwar verhindern, dass gefälschtes Feedback von *wenigen* unehrlichen Peers ihre Entscheidungen beeinflusst. Dies funktioniert aber nur so lange, wie die Mehrheit der Peers korrektes Feedback abgibt. Es kann jedoch argumentiert werden, dass für einzelne kooperative Peers kein Anreiz besteht, gefälschtes Feedback zu generieren. Wenn also ausgeschlossen werden kann, dass mehrere Peers im Rahmen eines Kollaborationsangriffs mit gefälschtem Feedback gemeinsame Ziele verfolgen, kann auf einen Bestrafungsmechanismus verzichtet werden. In den meisten Anwendungsgebieten ist dies jedoch nicht gegeben.

Warum wird kein Feedback für ehrliches oder unehrliches Feedback generiert? Auf den ersten Blick erscheint es sinnvoll, ehrliches oder unehrliches Verhalten ebenso wie die Kooperationsbereitschaft eines Peers mit positivem oder negativem Feedback zu belohnen oder zu bestrafen. Allerdings kann diese Vorgehensweise zu schwerwiegenden Problemen führen. Zunächst könnten sich Peers entscheiden, weniger kooperativ mitzuarbeiten und stattdessen vermehrt ehrliches Feedback zu verbreiten. Weitaus problematischer ist, dass es eine ganze Reihe von Angriffen gibt, die dazu führen können, dass ein Peer falsche Informationen über seine Nachbarn erhält, beispielsweise Masquerading (ein Peer bearbeitet Operationen, die nicht für seinen Datenbereich bestimmt sind [RBBW05]), Man-in-the-Middle-Angriffe (ein Peer verändert den Inhalt einer Nachricht, die er nur weiterleiten sollte [CDV+02]) oder Sybil-Angriffe (ein Teilnehmer bucht sich mehrfach unter verschiedenen Identitäten im CAN ein bzw. kontrolliert mehrere abhängige Peers, die dann koordiniert Angriffe ausführen [Dou02]). Wird nun für falsches oder richtiges Feedback ebenfalls Feedback generiert, multiplizieren sich die Auswirkungen dieser Angriffe, und da Feedback zwischen den Peers ausgetauscht wird, können Angriffe eine große Anzahl Peers betreffen und etablierte Vertrauensbeziehungen zwischen Dritten beeinträchtigen. Mit dem in diesem Kapitel vorgeschlagenen Protokoll bewertet jeder Peer unabhängig von anderen Knoten die Nützlichkeit von erhaltenem Feedbacks. Die Auswirkungen der genannten Angriffe sind somit lokal auf einzelne Knoten begrenzt, und der Feedback-Mechanismus als Ganzes bleibt stabil und unbeeinträchtigt.

Warum gewichtet der Algorithmus nicht nach dem Feedback-Erzeuger? Jeder Peer bestimmt seine Wichtungsfaktoren nach dem Feedback-Übermittler, nicht nach dem Feedback-Erzeuger. Dies hat vier Gründe:

1. Jeder Peer entscheidet, welches Feedback er an seine Nachbarn weiterleitet und welches nicht. Auf diese Weise kann ein Peer also das Feedback von Dritten zensieren, und damit das übertragene Feedback bestimmen.
2. Peers könnten Feedback fälschen und die Urheberschaft abstreiten, d.h., als Feedback-Erzeuger einen anderen Nachbarn angeben. Wird jedoch der Feedback-Übermittler für falsches Feedback zur Verantwortung gezogen, ist dieser Angriff unmöglich.
3. Die Peers sollen motiviert werden, weitergegebenes Feedback von Dritten sorgfältig auszuwählen, und kein offensichtlich gefälschtes Feedback übermitteln.
4. Ein Peer kann seinen Nachbarn, von dem er das Feedback erhalten hat, direkt mit einem Arbeitsbeweis sanktionieren. Der Knoten, der das Feedback generiert hatte, ist nicht unbedingt auch ein Nachbar des Peers, der die Inkorrektheit des Feedbacks feststellt, und hat somit keine Möglichkeit, Arbeitsbeweise zu verlangen.

Können nicht auch die Feedback-Benachrichtigungen gefälscht sein? In diesem Kapitel wurde davon ausgegangen, dass nur das übermittelte Feedback inkorrekt sein kann, nicht aber die Feedback-Benachrichtigungen, mit denen die Anfragesteller den Weiterleitern den Transaktionsausgang mitteilen. Es ist jedoch möglich, dass ein Peer nach dem erfolgreichen Erhalt einer Antwort trotzdem eine negative Feedback-Benachrichtigung übermittelt, wodurch die Weiterleiter die Wichtungsfaktoren der falschen Peers steigern bzw. absenken.

Die Kernproblematik besteht darin, dass es 'teuer' ist, die Korrektheit eines Anfrageergebnisses zu überprüfen. Nur der Anfragesteller kann mit Sicherheit feststellen, ob das Anfrageergebnis gültig ist, und zur Anfrage passt (vgl. Man-in-the-Middle-Angriffe, Abschnitt 3.6). Wenn die Anwendung, in der FairNet eingesetzt wird, eine einfache Überprüfung der Anfrageergebnisse zulässt, kann eine kleine Änderung am Protokoll (siehe Abschnitt 3.7) Feedback-Benachrichtigungen obsolet machen.

In den meisten Anwendungsszenarien ist das jedoch nicht möglich. Hier kann jedoch argumentiert werden, dass es für einen kooperativen Peer, der auch weiterhin Anfrageergebnisse bekommen möchte, nicht rational ist, nach dem Erhalt einer korrekten Antwort eine falsche Benachrichtigungen zu versenden. Des weiteren lässt sich der hier beschriebene Mechanismus zum Umgang mit gefälschtem Feedback auch auf gefälschte Feedback-Benachrichtungen erweitern: wenn bei einer Weiterleitung über einen bestimmten Nachbarn die Mehrheit mit einer positiven Benachrichtigung reagiert, muss der Peer unehrlich sein, der überdurchschnittlich oft negative Benachrichtigungen abgibt.

5.7 Zusammenfassung

Eine wesentliche Herausforderung für jedes dezentrale Reputationssystem besteht im Umgang mit (absichtlich oder unabsichtlich) falschen Rückmeldungen von Einzelnen oder von ganzen

Gruppen zusammenarbeitender Teilnehmer. Dieses Kapitel stellte eine kleine Ergänzung des FairNet-Protokolls vor, die gefälschtes Feedback erkennt und die Versender des falschen Feedbacks abstraft. Zunächst wurde dazu ermittelt, welche Auswirkungen das gefälschte Feedback einer Anzahl von zusammenarbeitenden unehrlichen Peers in FairNet haben kann. Dabei konnte zunächst festgestellt werden, dass ein einzelner Peer nur geringen Einfluss auf die Repositories seiner Kontakte nimmt, eine Gruppe von unehrlichen Knoten jedoch die Reputation eines Peers beträchtlich beeinflussen kann.

Im Anschluss wurde ein dezentraler Algorithmus vorgestellt, bei dem jeder Peer den durch Feedback vorhergesagten mit dem tatsächlichen Transaktionsausgang vergleicht. Das Feedback der Peers, die häufig positives Feedback für kooperative und negatives Feedback für unkooperative Knoten abgegeben haben, erhält dabei eine höhere Gewichtung als das der Knoten, die falsches Feedback generiert haben. Des weiteren werden diejenigen Peers mit einem Arbeitsbeweis bestraft, dessen Feedback um mehr als einen Schwellwert vom Durchschnitt aller Peers abweicht, die einen bestimmten Knoten bewertet haben. Diese Maßnahmen gegen falsches Feedback wurden einer ausführlichen Evaluierung unterzogen. Insbesondere konnte festgestellt werden, dass der erwähnte Kollaborationsangriff durch eine Gruppe von unehrlichen Knoten nun (nahezu) ohne Auswirkungen bleibt und hohe Kosten für alle beteiligten unehrlichen Teilnehmer nach sich zieht. Selbst wenn 25 von 26 Nachbarn eines unkooperativen Peers fast ausschließlich positives Feedback abgeben, muß der unkooperative Peer höhere Kosten tragen, als wenn er sich vollständig kooperativ verhalten hätte. Das Kapitel wurde von einer Diskussion der Design-Entscheidungen abgeschlossen, die beim Entwurf von Maßnahmen gegen unehrliche Teilnehmer zu treffen sind.

Kapitel 6

Zusammenfassung und Ausblick

Unkooperative Peers, die Operationen nicht protokollgerecht verarbeiten um Ressourcen zu sparen, sind ein ernstes Problem für alle Arten von P2P-Datenstrukturen. Beispielsweise können bereits 5% unkooperative Teilnehmer in einem Content-Addressable Network dazu führen, dass eine Anfrage mit einer Wahrscheinlichkeit von 40% nicht beantwortet wird. Insbesondere das Fehlen von zentralisierten Strukturen, welche Aufgaben wie Monitoring, Authentifizierung oder Accounting übernehmen könnten, machen den Umgang mit unkooperativen Teilnehmern in P2P-Datenstrukturen zu einer Herausforderung. Hinzu kommt, dass die Peers jederzeit ihr Verhalten ändern oder das System verlassen und unter einer neuen Identität wieder eintreten können. Es existieren keine einfachen Maßnahmen gegen unkooperatives Verhalten. Schwarze Listen nicht anwendbar in Systemen, in denen die Teilnehmer ihre Identität wechseln können. Bezahlverfahren sind ressourcenintensiv in der Realisierung bzw. benötigen eine vertrauenswürdige zentrale Instanz. Verfahren aus verwandten Disziplinen – z.B. mobile Ad-Hoc Netze, elektronische Märkte oder Sensornetze – gehen von einer abweichenden Systemarchitektur aus und sind daher ebenfalls nicht unmittelbar auf P2P-Datenstrukturen übertragbar.

In dieser Arbeit wurde *FairNet* vorgestellt, ein Protokoll, das unkooperatives Verhalten in P2P-Datenstrukturen für rationale Peers unattraktiv macht. FairNet basiert darauf, dass Peers Beobachtungen über das Verhalten von anderen in Form von Feedback untereinander austauschen. Nur Peers, über die genug positives Feedback vorliegt, dürfen an der Anfrageverarbeitung teilnehmen. Alle anderen Peers müssen einen Arbeitsbeweis erbringen, bevor sie Anfragen absetzen oder Anfrageergebnisse erhalten können. Der Arbeitsbeweis macht unkooperatives Verhalten für rationale Peers unwirtschaftlich. Zu den Beiträgen dieser Arbeit zählt ein *algebraisches Modell*, mit dem FairNet so parametrisiert werden kann, dass kooperative Teilnehmer nur definierte Mehrkosten tragen, während unkooperativen Peers hohe Kosten entstehen. So ist es möglich, dass selbst Peers, die noch 90% der eingehenden Anfragen beantworten, doppelt so hohe Kosten aufbringen müssen wie kooperative Knoten, denen dabei nur 10% Mehrkosten durch das Protokoll entstehen. Daneben ermöglicht das algebraische Modell die Vorhersage der Kosten für Neueinsteiger und die Bestimmung der Zeitspanne, in der FairNet auf Änderungen im Verhalten der Peers reagiert. Eine Serie von *Experimenten mit bis zu 160.000 Peers* konnte nachweisen, dass FairNet auch unter realistischen Umgebungsbedingun-

gen wie ungleich verteilte Anfrageschlüssel oder unterschiedlich aktive Peers wie berechnet arbeitet. Es konnte weiterhin experimentell bestätigt werden, dass das Protokoll stabil ist gegenüber Änderungen in den Umgebungsbedingungen: ein Parametersatz, der für ein System mit 10.000 Peers optimiert wurde, kann unkooperatives Verhalten auch dann wirksam disziplinieren, wenn die Zahl der Teilnehmer in FairNet von 80 bis 160.000 variiert wird.

Zum *kostenneutralen Austausch von Feedback* in FairNet wurde ein Gossip-Protokoll entwickelt, bei dem die Peers Feedback an Nachrichten anhängen, die sie im Rahmen der Anfrageverarbeitung ohnehin versenden. Da Feedback-Objekte nur wenige Bytes groß sind, ist der Ressourcenverbrauch durch das Anfügen weniger Feedback-Objekte an eine Nachricht vernachlässigbar. Es wurde ermittelt, wie groß die Übertragungskapazität des Feedback-Transports ist. Des weiteren wurden verteilte Algorithmen evaluiert, mit denen ein Peer bestimmt, welche Feedback-Objekte beim Überschreiten der Übertragungskapazität zuerst an Nachrichten angehängt werden. Der *Umgang mit gefälschtem Feedback*, das von einem Einzelnen oder einer Gruppe von unehrlichen Teilnehmern verbreitet wird, ist ein großes Problem für jedes dezentrale Reputationssystem. In FairNet wird dazu der durch positives oder negatives Feedback prognostizierte Transaktionsausgang mit dem tatsächlichen verglichen. Feedback von unehrlichen Peers wird mit Hilfe von Wichtungsfaktoren herabgestuft, weiterhin werden unehrliche Peers mit Arbeitsbeweisen diszipliniert. Eine ausführliche Diskussion von Design-Entscheidungen, Angriffsmöglichkeiten, schlechtestmöglichen Umgebungsbedingungen sowie alternativen Implementierungsvarianten hat gezeigt, dass FairNet *flexibel an viele Einsatzszenarien angepasst* werden kann. So konnte unter anderem gezeigt werden, dass nur eine kleine Änderung in den Protokollparametern erforderlich ist, damit FairNet in einem Chord-Netzwerk eine vergleichbare Wirksamkeit gegen unkooperative Teilnehmer zeigt wie in einem Content-Addressable Network. Damit hat FairNet einen Entwicklungsstand erreicht, der eine Umsetzung in die Praxis ermöglicht.

6.1 Ausblick auf Folgearbeiten

Aufgrund der Breite des Themas musste eine Auswahl der im Rahmen dieser Arbeit zu behandelnden Fragestellungen getroffen werden. Demzufolge bietet diese Arbeit einen Ausgangspunkt für zahlreiche mögliche Folgearbeiten.

Umsetzung in die Praxis Nur die Umsetzung in die Praxis kann zeigen, ob FairNet in realen Anwendungen gegen unkooperatives Verhalten wirksam ist. Für P2P-Dateitauschbörsen existieren bereits zahlreiche Untersuchungen über das Nutzerverhalten [AH00, ZYF02]. Für Anwendungen von P2P-Datenstrukturen existieren zwar sehr viele interessante Vorschläge, es gibt aber erst wenige große Installationen. Daher stehen auch Untersuchungen über das Nutzerverhalten in P2P-Datenstrukturen noch aus; insbesondere Maßnahmen gegen unkooperatives Verhalten konnten noch nicht in der Praxis evaluiert werden.

Individuelle Kostenfunktionen Dieser Arbeit wurde eine sehr einfache Kostenstruktur für Arbeitsbeweise sowie das Weiterleiten und Beantworten von Anfragen verwendet. In der Realität sind jedoch für CPU-Zeit, Speicherbedarf und Netzwerkbandbreite mehrdimensionale und

für jeden Peer individuelle Kosten anzusetzen – beispielsweise kann ein Rechner über eine schnelle DSL-Verbindung in das Internet verfügen, während ein anderer nur an ein Modem angeschlossen ist, aber einen schnelleren Prozessor besitzt. Des weiteren ist die Modellierung der Ressourcen in Form von Kosten problematisch, wenn ungenutzte Ressourcen für die Teilnahme an P2P-Datenstrukturen aufgewendet werden. Zudem lassen sich diese Ressourcen nicht ansparen, und unterliegen keinem messbaren Verschleiß durch die Teilnahme an der P2P-Datenstruktur. Diesem Problem kann durch die Einführung von Kostenfunktionen begegnet werden, die von Peer zu Peer verschieden sind und individuell bestimmt werden.

Selbstoptimierung Im hier vorgestellten Protokoll verfügt jeder Teilnehmer über den gleichen unveränderlichen Satz von Konfigurationsparametern. Die Umgebungsbedingungen in einer P2P-Datenstruktur sind jedoch permanenten Veränderungen unterworfen: Die Zahl der Teilnehmer fluktuiert, die Ausfallwahrscheinlichkeiten ändern sich, benachbarte Peers modifizieren ihr Verhalten und die Zonenaufteilung wird durch ein- und austretende Peers geändert. Zwar konnte im Rahmen dieser Arbeit gezeigt werden, dass FairNet sehr stabil ist gegenüber solchen Veränderungen. Dennoch können die Diskriminierung verbessert bzw. die Overheadkosten gesenkt werden, wenn jeder Teilnehmer seinen Parametersatz permanent an die jeweils beobachteten Umgebungsbedingungen anpasst. Jeder Peer kann die Zahl der Teilnehmer anhand der Pfadlängen bei der Weiterleitung abschätzen, und kann die globale Ausfallwahrscheinlichkeit am Anteil der abgesendeten und beantworteten Anfragen bestimmen. Auf Basis dieser Daten können nun Strategien zur Selbstoptimierung von FairNet entwickelt werden.

Fairer Lastausgleich Beim praktischen Einsatz einer P2P-Datenstruktur sind Lastausgleichsmechanismen erforderlich, um auf Überlast-Situationen wegen beliebterer Datentupel oder aktiverer Peers zu reagieren. Es bestehen zahlreiche Möglichkeiten, einen Lastausgleich zu erzielen: Anfragen können beispielsweise über weniger belastete Peers weitergeleitet werden, die Zahl der Replikate kann dynamisch an die Lastsituation angepasst werden, oder Peers können die Größen ihrer Zonen variieren. Lastausgleichsmechanismus und FairNet stehen im Konflikt: während es der Lastausgleichsmechanismus ermöglicht, dass Peers von der Anfrageverarbeitung zurücktreten um Ressourcen zu sparen, soll in FairNet jeder Peer jede Anfrage verarbeiten, um eben dieses Verhalten verhindern. Daher besteht ein wichtiges Thema für eine Folgearbeit darin, Lastausgleichsmechanismen und FairNet zusammenzuführen.

Übertragbarkeit auf andere Systeme In dieser Arbeit wurde FairNet auf der Basis von Content-Addressable Networks evaluiert und die Übertragbarkeit auf unterschiedliche P2P-Datenstrukturen am Beispiel von Chord exemplarisch diskutiert. Es gibt jedoch darüber hinausgehende Ansätze, um die Anwendbarkeit von FairNet in *allgemeinen* P2P-Datenstrukturen zu demonstrieren. In [AAG+05] wird anhand einer Referenzarchitektur gezeigt, wie ein P2P-Netzwerk losgelöst von einer konkreten Implementierung spezifiziert werden kann. Eine Verallgemeinerung von FairNet für diese Referenzarchitektur würde das Protokoll für alle P2P-Systeme öffnen, die auch von der Referenzarchitektur unterstützt werden. Zum Anderen könnte FairNet in ein bestehendes P2P-Framework wie JXTA[1] integriert werden. JXTA stellt eine Java-Schnittstelle für P2P-Applikationen zur Verfügung und abstrahiert von konkreten Implementierungen.

[1] http://www.jxta.org

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ACK ..... Acknowledge (Kontrollnachricht beim TCP/IP Protokoll)
CAN ..... Content-Addressable Network
CPU ..... Central Processing Unit (Zentrale Recheneinheit)
DHT ..... Distributed Hashtable (Verteilte Hashtabelle)
DSL ..... Digital Subscriber Line (eine Datenverbindung mit hoher Bandbreite)
FIFO ..... First In First Out
FIN ..... Finish (Kontrollnachricht beim TCP/IP Protokoll)
IP ..... Internet Protocol
LIFO ..... Last In First Out
P2P ..... Peer-to-Peer
PC ..... Personal Computer
RAM ..... Random Access Memory (der Hauptspeicher im PC)
SYN ..... Synchronize (Kontrollnachricht beim TCP/IP Protokoll)
TCP ..... Transmission Control Protocol
URL ..... Uniform Resource Locator (Adresse und Zugriffsprotokoll einer Webseite)
WWW .... World Wide Web

Literaturverzeichnis

[AAG+05] ABERER, Karl ; ALIMA, Luc O. ; GHODSI, Ali ; GIRDZIJAUSKAS, Sarunas ; HARIDI, Seif ; HAUSWIRTH, Manfred: The Essence of P2P: A Reference Architecture for Overlay Networks. In: *Proceedings of the 5th International Conference on Peer-to-Peer Computing (P2P'05)*, 2005, S. 11–20 13, 93, 139

[AB05] APEL, Sven ; BUCHMANN, Erik: Biology-Inspired Optimizations of Peer-to-Peer Overlay Networks. In: *Praxis der Informationsverarbeitung und Kommunikation (PIK)* Bd. 28(4/05), 2005, S. 199–205 29

[Abe01] ABERER, Karl: P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In: *Proceedings of the 9th International Conference on Cooperative Information Systems (CoopIS'01)*, 2001, S. 179–194 2, 11, 13, 20, 94

[ABHL03] AHN, Luis von ; BLUM, Manuel ; HOPPER, Nicholas J. ; LANGFORD, John: CAPTCHA: Using Hard AI Problems for Security. In: *Lecture Notes in Computer Science (LNCS)* 2656 (2003), Mai 3, 48, 75

[AD01] ABERER, Karl ; DESPOTOVIC, Zoran: Managing Trust in a Peer-2-Peer Information System. In: *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM'01)*, 2001 37, 89

[ADHS05] ABERER, Karl ; DATTA, Anwitaman ; HAUSWIRTH, Manfred ; SCHMIDT, Roman: Indexing Data-Oriented Overlay Networks. In: *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, 2005, S. 685–696 20

[AH00] ADAR, E. ; HUBERMAN, B.: Free Riding on Gnutella. In: *First Monday* 5 (2000), Oktober, Nr. 10 2, 5, 7, 27, 138

[AH02] ABERER, Karl ; HAUSWIRTH, Manfred: An Overview on Peer-to-Peer Information Systems. In: *Proceedings of the 4th International Workshop on Distributed Data and Structures (WDAS'02)*, 2002 4, 9

[ATS04] ANDROUTSELLIS-THEOTOKIS, Stephanos ; SPINELLIS, Diomidis: A Survey of Peer-to-Peer Content Distribution Technologies. In: *ACM Computing Surveys* 36 (2004), Dezember, Nr. 4, S. 335–371 9, 16

[Axe84] AXELROD, Robert: *The Evolution of Cooperation*. New York : Basic Books, 1984 29, 34, 120

[BA05] BUCHMANN, Erik ; APEL, Sven: Piggyback Meta-Data Propagation in Distributed Hash Tables. In: *Proceedings of the 1st International Conference on Web Information Systems and Technologies (WEBIST'05)*, 2005 97, 99

[Bac02] BACK, Adam: *Hashcash - A Denial of Service Counter-Measure*. http://hashcash.org, August 2002 48

[BAS03] BURAGOHAIN, Chiranjeeb ; AGRAWAL, Divy ; SURI, Subhash: A Game-Theoretic Framework for Incentives in P2P Systems. In: *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P'03)*, 2003 34

[BB02] BUCHEGGER, Sonja ; BOUDEC, Jean-Yves L.: Performance Analysis of the CONFIDANT Protocol. In: *Proceedings of the 3rd Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC'02)*, 2002, S. 226–236 89

[BB03a] BUCHEGGER, Sonja ; BOUDEC, Jean-Yves L.: The Effect of Rumor Spreading in Reputation Systems for Mobile Ad-Hoc Networks. In: *Proceedings of the 1st International Symposium on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks (WiOpt'03)* (2003) 114

[BB03b] BUCHMANN, Erik ; BÖHM, Klemens: Effizientes Routing in verteilten skalierbaren Datenstrukturen. In: *Tagungsband der 10. GI-Fachtagung für Datenbanksysteme für Business, Technologie und Web (BTW'03)*, 2003 18, 79, 102

[BB04a] BUCHEGGER, Sonja ; BOUDEC, Jean-Yves L.: A Robust Reputation System for P2P and Mobile Ad-hoc Networks. In: *Proceedings of the 2nd Workshop on the Economics of Peer-to-Peer Systems (P2PEcon'04)*, 2004 38, 90

[BB04b] BUCHMANN, Erik ; BÖHM, Klemens: FairNet - How to Counter Free Riding in Peer-to-Peer Data Structures. In: *Proceedings of the 12th International Conference on Cooperative Information Systems (CoopIS'04)*, 2004 43

[BB04c] BUCHMANN, Erik ; BÖHM, Klemens: How to Run Experiments with Large Peer-to-Peer Data Structures. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004 3, 76, 77

[BB06] BÖHM, Klemens ; BUCHMANN, Erik: Free Riding-Aware Forwarding in Content-Addressable Networks. In: *International Journal on Very Large Data Bases (VLDB)* (2006), Januar 43

[BEG03] BAEHNI, S. ; EUGSTER, P. T. ; GUERRAOUI, R.: Data-Aware Multicast / Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. 2003 (IC/2003/73). – Forschungsbericht. 98

[Blo70] BLOOM, Burton H.: Space/Time Trade-offs in Hash Coding with Allowable Errors. In: *Communications of the ACM (CACM)* 13 (1970), Nr. 7, S. 422–426 15

[BMT+05] BENDER, Matthias ; MICHEL, Sebastian ; TRIANTAFILLOU, Peter ; WEIKUM, Gerhard ; ZIMMER, Christian: MINERVA: Collaborative P2P Search. In: *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, 2005, S. 1263–1266 16

[BMT+06] BENDER, Matthias ; MICHEL, Sebastian ; TRIANTAFILLOU, Peter ; WEIKUM, Gerhard ; ZIMMER, Christian: P2P Content Search: Give the Web Back to the People. In: *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS '06)*, 2006 16

[BOGKW88] BEN-OR, Michael ; GOLDWASSER, Shafi ; KILIAN, Joe ; WIGDERSON, Avi: Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions. In: *Proceedings of the 20th Symposium on Theory of Computing (STOC'88)* (1988), S. 113–131 3, 87

[CAPMN03] CUENCA-ACUNA, Francisco M. ; PEERY, Christopher ; MARTIN, Richard P. ; NGUYEN, Thu D.: PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In: *Proceedings of the 12th International Symposium on High Performance Distributed Computing (HPDC'03)*, 2003 99, 108

[CBG02] CARTER, Jonathan ; BITTING, Elijah ; GHORBANI, Ali A.: Reputation Formalization within Information Sharing Multiagent Architectures. In: *Computational Intelligence* 18 (2002), November, Nr. 4, S. 45–64 33

[CBN05] CHUN, Brent N. ; BUONADONNA, Philip ; NG, Chaki: Computational Risk Management for Building Highly Reliable Network Services. In: *Proceedings of the 1st Workshop on Hot Topics in System Dependability (HotDep'05)*, 2005 29

[CCF04] CAI, M. ; CHERVENAK, A. ; FRANK, M.: A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table. In: *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC'04)* (2004), S. 56–56 94

[CDG+02] CASTRO, Miguel ; DRUSCHEL, Peter ; GANESH, Ayalvadi ; ROWSTRON, Antony ; WALLACH, Dan S.: Secure Routing for Structured Peer-to-Peer Overlay Networks. In: *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)* 36 (2002), Nr. SI, S. 299–314. – ISSN 0163–5980 87

[CDK+03] CASTRO, Miguel ; DRUSCHEL, Peter ; KERMARREC, Anne-Marie ; NANDI, Animesh ; ROWSTRON, Antony ; SINGH, Atul: SplitStream: High-Bandwidth Content Distribution in a Cooperative Environment. In: *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003 15

[CDKR03] CASTRO, Miguel ; DRUSCHEL, Peter ; KERMARREC, Anne-Marie ; ROWSTRON, Antony: Scalable Application-Level Anycast for Highly Dynamic Groups. In: *Proceedings of the 5th International Workshop on Networked Group Communications (NGC'03)*, 2003, S. 47–57 98

[CDV+02] CORNELLI, Fabrizio ; DAMIANI, Ernesto ; VIMERCATI, Sabrina De C. ; PARABOSCHI, Stefano ; SAMARATI, Pierangela: Choosing Reputable Servents in a P2P Network. In: *Proceedings of the 11th International World Wide Web Conference (WWW'02)*, 2002, S. 376–386 134

[CJT01] CABRERA, Luis F. ; JONES, Michael B. ; THEIMER, Marvin: Herald: Achieving a Global Event Notification Service. In: *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS'01)*, 2001, S. 87 15

[CL99] CASTRO, Miguel ; LISKOV, Barbara: Practical Byzantine Fault Tolerance. In: *Proceedings of the 3th Symposium on Operating Systems Design and Implementation (OSDI'99)*, 1999 87

[Cli00] CLIP2.COM, Inc.: *Gnutella: To the Bandwidth Barrier and Beyond*. http://dss.clip2.com, November 2000 12

[CN03] COX, Landon P. ; NOBLE, Brian D.: Samsara: Honor Among Thieves in Peer-to-Peer Storage. In: *Proceedings of the 19th Symposium on Operating Systems Principles (SOSP'03)*, 2003, S. 120–132 31, 86

[DGH+87] DEMERS, Alan ; GREENE, Dan ; HAUSER, Carl ; IRISH, Wes ; LARSON, John ; SHENKER, Scott ; STURGIS, Howard ; SWINEHART, Dan ; TERRY, Doug: Epidemic Algorithms for Replicated Database Maintenance. In: *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC'87)*, 1987, S. 1–12 99

[DHA03] DATTA, Anwitaman ; HAUSWIRTH, Manfred ; ABERER, Karl: Beyond Web of Trust: Enabling P2P E-Commerce. In: *Proceedings of the 5th IEEE Conference on E-Commerce (CEC'03)* (2003) 32

[DKK+01] DABEK, Frank ; KAASHOEK, M. F. ; KARGER, David R. ; MORRIS, Robert ; STOICA, Ion: Wide-Area Cooperative Storage with CFS. In: *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP'01)*, 2001 15, 94

[Dou02] DOUCEUR, John R.: The Sybil Attack. In: *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002, S. 251–260 47, 87, 112, 134

[DR01] DRUSCHEL, Peter ; ROWSTRON, Antony: PAST: A Persistent and Anonymous Store. In: *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS'01)*, 2001 15

[DT04] *Kapitel* 4. In: DALLY, William J. ; TOWLES, Brian P.: *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004, S. 75–86 22

[DY95] DARWEN, Paul J. ; YAO, Xin: On Evolving Robust Strategies for Iterated Prisoner's Dilemma. In: *Selected papers from the AI'93 and AI'94 Workshops on Evolutionary Computation*, 1995, S. 276–292 35

[Eur98] EUROPEAN CENTRAL BANK (HRSG.): Report on Electronic Money / European Central Bank, Postfach 16 03 19 , D-60066 Frankfurt am Main. 1998 (ISBN 92-9181-012-61). – Forschungsbericht. 6

[FLSC04] FELDMAN, Michal ; LAI, Kevin ; STOICA, Ion ; CHUANG, John: Robust Incentive Techniques for Peer-to-Peer Networks. In: *Proceedings of the 5th ACM Conference on Electronic Commerce (EC'04)* (2004) 38

[FNSY96] FERGUSON, Donald F. ; NIKOLAOU, Christos ; SAIRAMESH, Jakka ; YEMINI, Yechiam: Economic Models for Allocating Resources in Computer Systems. (1996), S. 156–183 29

[FR98] FRIEDMAN, Eric ; RESNICK, Paul: The Social Cost of Cheap Pseudonyms. In: *Journal of Economics and Management Strategy (JEMS)* 10 (1998), Nr. 2 30, 41, 49, 89

[FST04] FIGUEIREDO, Daniel R. ; SHAPIRO, Jonathan ; TOWSLEY, Don: Payment-based Incentives for Anonymous Peer-to-Peer Systems / Computer Science Department, University of Massachusetts. 2004 (UMass CMPSCI 04-62). – Forschungsbericht. 6, 30

[GGG+03] GUMMADI, K. ; GUMMADI, R. ; GRIBBLE, S. ; RATNASAMY, Sylvia ; SHENKER, Scott ; STOICA, Ion: The Impact of DHT Routing Geometry on Resilience and Proximity. In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'03)*, 2003 14

[GLBM01] GOLLE, Philippe ; LEYTON-BROWN, Kevin ; MIRONOV, Ilya: Incentives for Sharing in Peer-to-Peer Networks. In: *Lecture Notes in Computer Science (LNCS)* 2232 (2001) 6, 30

[GMSK03] GARCIA-MOLINA, Hector ; SCHLOSSER, Mario T. ; KAMVAR, Sepandar D.: The EigenTrust Algorithm for Reputation Management in P2P Networks. In: *Proceedings of the 12th International World Wide Web Conference (WWW'03)* (2003), April 56

[GWB+01] GRIBBLE, S. ; WELSH, Matt ; BEHREN, J. R. ; BREWER, Eric A. ; CULLER, David E. ; BORISOV, N. ; CZERWINSKI, Steven E. ; GUMMADI, R. ; HILL, Jon R. ; JOSEPH, Anthony D. ; KATZ, Randy H. ; MAO, Z. M. ; ROSS, S. ; ZHAO, Ben Y.: The Ninja Architecture for Robust Internet-Scale Systems and Services. In: *Journal of Computer Networks* 35 (2001), März, Nr. 4, S. 473–497 16, 87, 88

[Hue03] HUEBSCH, Ryan: Content-Based Multicast: Comparison of Implementation Options / EECS Department, University of California, Berkeley. 2003 (UCB/CSD-03-1229). – Forschungsbericht. 15, 107

[JF04] JURCA, R. ; FALTINGS, B.: CONFESS: Eliciting Honest Feedback without Independent Verification Authorities. In: *Proceedings of the 6th International Workshop on Agent Mediated Electronic Commerce (AMEC'04)* (2004) 113, 115

[JJ99] JAKOBSSON, M. ; JUELS, A.: Proofs of Work and Bread Pudding Protocols. In: *Proceedings of the 4th International Conference on Communications and Multimedia Security (CMS'99)*, 1999 3, 48

[Knö03] KNÖFEL, Stefan: *Lastverteilung in Peer-to-Peer Architekturen*. Diplomarbeit, Fakultät für Informatik, Technische Universität Dresden, November 2003 88

[KR97] KHARE, Rohit ; RIFKIN, Adam: Weaving a Web of Trust. In: *World Wide Web Journal* 2 (1997), Nr. 3, S. 77–112. – ISSN 1085–2301 31, 36

[KRD03] KHAMBATTI, Mujtaba ; RYU, Kyung D. ; DASGUPTA, Partha: Push-Pull Gossiping for Information Sharing in Peer-to-Peer Communities. In: *Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03)*, 2003, S. 1393–1399 98, 108

[KS05] KENT, Stephen ; SEO, Karen: RFC 4301: Security Architecture for the Internet Protocol. In: *The Internet Society, Standards Track* (2005), Dezember 86

[KZ04] KOESSLER, Frederic ; ZIEGELMEYER, Anthony: Parimutuel Betting under Asymmetric Information / Max Planck Institute of Economics, Strategic Interaction Group. 2004 (2003-34). – Forschungsbericht. 2

[LEB+03] LILLIBRIDGE, Mark ; ELNIKETY, Sameh ; BIRRELL, Andrew ; BURROWS, Michael ; ISARD, Michael: A Cooperative Internet Backup Scheme. In: *Proceedings of the 2003 USENIX Annual Technical Conference (USENIX'03)* (2003), S. 29–41 86

[LI04] LIU, Jinshan ; ISSARNY, Valérie: Enhanced Reputation Mechanism for Mobile Ad Hoc Networks. In: *Proceedings of the 2nd International Conference on Trust Management (iTrust'04)*, 2004, S. 48–62 114, 116

[Lit80] LITWIN, Witold: Linear Hashing: A New Tool For File And Table Addressing. In: *Proceedings of the 6th International Conference on Very Large Data Bases (VLDB'80)*, 1980, S. 212–223 18

[LK00] LEE, Seungjoon ; KIM, Chongkwon: Neighbor Supporting Ad Hoc Multicast Routing Protocol. In: *Proceedings of the 1st Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC'00)*, 2000, S. 37–44 4, 99

[LMR02] LYNCH, Nancy A. ; MALKHI, Dahlia ; RATAJCZAK, David: Atomic Data Access in Distributed Hash Tables. In: *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002, S. 295–305 25

[LNS93] LITWIN, Witold ; NEIMAT, Marie-Anne ; SCHNEIDER, Donovan A.: LH* - Linear Hashing for Distributed Files. In: *Proceedings of the 9th International Conference on Management of Data (SIGMOD'93)*, 1993 18

[Mar94] MARSH, S.: Formalising Trust as a Computational Concept. In: *Ph.D. Thesis., Department of Mathematics and Computer Science, University of Stirling* (1994) 32

[MGLB00] MARTI, Sergio ; GIULI, T. J. ; LAI, Kevin ; BAKER, Mary: Mitigating Routing Misbehavior in Mobile Ad-Hoc Networks. In: *Proceedings of the 6th International Conference on Mobile Computing and Networking (MobiCom'00)*, 2000, S. 255–265 5, 30

[MKL+02] MILOJICIC, Dejan S. ; KALOGERAKI, Vana ; LUKOSE, Rajan ; NAGARAJA, Kiran ; PRUYNE, Jim ; RICHARD, Bruno ; ROLLINS, Sami ; XU, Zhichen: Peer-to-Peer Computing / HP Labs. 2002 (HPL-2002-57). – Forschungsbericht. 4, 9, 11

[MNR02] MALKHI, Dahlia ; NAOR, Moni ; RATAJCZAK, David: Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In: *Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC'02)*, 2002, S. 183–192 22

[MPS+02] MAINWARING, Alan ; POLASTRE, Joseph ; SZEWCZYK, Robert ; CULLER, David E. ; ANDERSON, John: Wireless Sensor Networks for Habitat Monitoring. In: *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, 2002 30

[MRZ05] MILLER, Nolan ; RESNICK, Paul ; ZECKHAUSER, Richard: Eliciting Informative Feedback: The Peer Prediction Method. In: *Management Science* 51 (2005), September, Nr. 9, S. 1359–1373 113

[Nas51] NASH, John: Non-Cooperative Games. In: *Annals of Mathematics* 54 (1951), S. 286–295 34, 39

[Nec97] NECULA, George C.: Proof-Carrying Code. In: *Proceedings of the 24th Symposium on Principles of Programming Languages (POPL'97)*, 1997, S. 106–119 30

[NT04] NTARMOS, Nikos ; TRIANTAFILLOU, Peter: SeAl: Managing Accesses and Data in Peer-to-Peer Sharing Networks. In: *Peer-to-Peer Computing*, 2004, S. 116–123 30, 31

[Odl03] ODLYZKO, Andrew M.: The Case Against Micropayments. In: *Proceedings of the 7th International Conference on Financial Cryptography (FC'03)* Bd. 2742, 2003, S. 77–83 7, 31

[Per04] PERLEGOS, Pete C.: DoS Defense in Structured Peer-to-Peer Networks / EECS Department, University of California, Berkeley. 2004 (UCB/CSD-04-1309). – Forschungsbericht. 88

[Pre04] PRELEC, Drazen: A Bayesian Truth Serum for Subjective Data. In: *Science* 306 (2004), Oktober, Nr. 5695, S. 462–466 113

[PRR97] PLAXTON, C. G. ; RAJARAMAN, Rajmohan ; RICHA, Andrea W.: Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In: *Proceedings of the 9th Symposium on Parallel Algorithms and Architectures (SPAA'97)*, 1997, S. 311–320 21

[PS05] PAPAIOANNOU, Thanasis G. ; STAMOULIS, George D.: Optimizing an Incentives Mechanism for Truthful Feedback in Virtual Communities. In: *Proceedings of the 4th International Workshop on Agents and Peer-to-Peer Computing (AP2PC'05)*, 2005 34

[RBBW05] REIDEMEISTER, Thomas ; BÖHM, Klemens ; BUCHMANN, Erik ; WARD, Paul: Malicious Behaviour in Content-Addressable Peer-to-Peer Networks. In: *Proceedings of the 3rd Conference on Communication Networks and Services Research (CNSR'05)* (2005) 86, 87, 134

[RD01] ROWSTRON, Antony ; DRUSCHEL, Peter: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: *Proceedings of the 21st International Conference on Distributed Systems (ICDCS'01)*, 2001 2, 15, 21, 94

[RFH+01] RATNASAMY, Sylvia ; FRANCIS, Paul ; HANDLEY, Mark ; KARP, Richard ; SHENKER, Scott: A Scalable Content-Addressable Network. In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, 2001 2, 3, 11, 16, 17, 18, 25, 46, 60, 86, 101

[RHKS01] RATNASAMY, Sylvia ; HANDLEY, Mark ; KARP, Richard ; SHENKER, Scott: Application-Level Multicast Using Content-Addressable Networks. In: *Lecture Notes in Computer Science (LNCS)* 2233 (2001) 4, 15, 98, 102

[RKCD01] ROWSTRON, Antony ; KERMARREC, Anne-Marie ; CASTRO, Miguel ; DRUSCHEL, Peter: Scribe: The Design of a Large-Scale Event Notification Infrastructure. In: *Proceedings of the 3rd International Workshop on Networked Group Communication (NGC'01)* Bd. 2233, 2001, S. 30–43 15

[RKZF00] RESNICK, Paul ; KUWABARA, Ko ; ZECKHAUSER, Richard ; FRIEDMAN, Eric: Reputation Systems. In: *Communications of the ACM (CACM)* 43 (2000), Nr. 12 33

[RL03] RAMASWAMY, L. ; LIU, Ling: Free Riding: A New Challenge to Peer-to-Peer File Sharing Systems. In: *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)* (2003) 24

[RV03] REYNOLDS, Patrick ; VAHDAT, Amin: Efficient Peer-to-Peer Keyword Searching. In: *Proceedings of the 4th International Middleware Conference (Middleware'03)*, 2003 15

[RWE+01] RHEA, Sean ; WELLS, Chris ; EATON, Patrick ; GEELS, Dennis ; ZHAO, Ben Y. ; WEATHERSPOON, Hakim ; KUBIATOWICZ, John D.: Maintenance-Free Global Data Storage. In: *IEEE Internet Computing (IC)* 5 (2001), Nr. 5, S. 40–49 15, 25

[RWSB05] RÖSCH, Philipp ; WETH, Christan von d. ; SATTLER, Kai-Uwe ; BUCHMANN, Erik: Verteilte Anfrageverarbeitung in DHT-basierten P2P-Systemen. In: *Tagungsband der 12. GI-Fachtagung für Datenbanksysteme für Business, Technologie und Web (BTW'05)*, 2005 16

[SBM02] SCHWEITZER, Frank ; BEHERA, Laxmidhar ; MÜHLENBEIN, Heinz: Evolution of Cooperation in a Spatial Prisoner's Dilemma. In: *Computing Research Repository (CoRR)* cond-mat/0211605 (2002) 34

[Sch03] SCHWEITZER, Frank: Meinungsbildung, Kommunikation und Kooperation aus physikalischer Perspektive. In: *Physik Journal* 5 (2003), S. 57–62 29

[SCL+05] STRIBLING, Jeremy ; COUNCILL, Isaac G. ; LI, Jinyang ; KAASHOEK, M. F. ; KARGER, David R. ; MORRIS, Robert ; SHENKER, Scott: OverCite: A Cooperative Digital Research Library. In: *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS '05)*, 2005 16

[SLB06] SHERWOOD, Rob ; LEE, Seungjoon ; BHATTACHARJEE, Bobby: Cooperative Peer Groups in NICE. In: *Journal of Computer Networks* 50 (2006), Nr. 4, S. 523–544 31

[SMLN+01] STOICA, Ion ; MORRIS, Robert ; LIBEN-NOWELL, David ; KARGER, David R. ; KAASHOEK, M. F. ; DABEK, Frank ; BALAKRISHNAN, Hari: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, 2001 2, 15, 19, 94

[SRBB04] SATTLER, Kai-Uwe ; RÖSCH, Philipp ; BUCHMANN, Erik ; BÖHM, Klemens: A Physical Query Algebra for DHT-based P2P Systems. In: *Proceedings of the 6th International Workshop on Distributed Data and Structures (WDAS'04)*, 2004 16, 79

[SS01] *Kapitel* 15. In: SAAKE, G. ; SATTLER, Kai-Uwe: *Algorithmen und Datenstrukturen – Eine Einführung mit Java*. 2. Auflage. dpunkt.verlag, 2001, S. 393–412 13

[TD04] TANG, Chunqiang ; DWARKADAS, Sandhya: Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. In: *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004, S. 211–224 15

[TG97] TAUSCHER, Linda ; GREENBERG, Saul: How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems. In: *International Journal of Human-Computer Studies* 47 (1997), Nr. 1, S. 97–137 79

[TP05] TANIS, Martin ; POSTMES, Tom: A Social Identity Approach to Trust: Interpersonal Perception, Group Membership and Trusting Behavior. In: *European Journal of Social Psychology* 35 (2005), S. 413–424 29

[Tsu88] TSUCHIYA, P. F.: The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks. In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'88)*, 1988, S. 35–42 21

[WV03] WANG, Yao ; VASSILEVA, Julita: Bayesian Network-Based Trust Model. In: *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'03)*, 2003, S. 372–378 35, 36

[XL02] XIONG, Li ; LIU, Ling: Building Trust in Decentralized Peer-to-Peer Electronic Communities. In: *Proceedings of the 5th International Conference on Electronic Commerce Research (ICECR'02)*, 2002 89

[XL04] XIONG, Li ; LIU, Ling: PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 16 (2004), Nr. 7 38

[YGM03] YANG, Beverly ; GARCIA-MOLINA, Hector: PPay: Micropayments for Peer-to-Peer Systems. In: ATLURI, Vijay (Hrsg.) ; LIU, Peng (Hrsg.): *Proceedings of the 10th Conference on Computer and Communication Security (CCS'03)*, 2003 6, 30

[ZCLC05] ZHANG, Zhan ; CHEN, Shigang ; LING, Yibei ; CHOW, Randy: Resilient Capacity-Aware Multicast Based on Overlay Networks. In: *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS'05)*, 2005, S. 565–574 107

[ZGG+04] ZHANG, Hui ; GOEL, Ashish ; GOVINDAN, Ramesh ; MASON, Kahn ; ROY, Benjamin V.: Improving Eigenvector-Based Reputation Systems Against Collusion. In: *Proceedings of the 3rd International Workshop on Algorithms and Models for the Web-Graph (WAW'04)* (2004), Oktober 48

[ZKJ01] ZHAO, Ben Y. ; KUBIATOWICZ, John D. ; JOSEPH, Anthony D.: Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing / University of California at Berkeley. 2001 (UCB//CSD-01-1141). – Forschungsbericht. 21, 94

[ZMM99] ZACHARIA, Giorgos ; MOUKAS, Alexandros ; MAES, Pattie: Collaborative Reputation Mechanisms in Electronic Marketplaces. In: *Proceedings of the 32th Hawaii International Conference on System Sciences (HICSS'99)*, 1999 30, 36

[ZYF02] ZEINALIPOUR-YAZTI, Demetris ; FOLIAS, Theodoros: A Quantitative Analysis of the Gnutella Network Traffic / University of California - Riverside, USA. 2002 (CS204). – Forschungsbericht. 12, 138

[ZZJ+01] ZHUANG, Shelley Q. ; ZHAO, Ben Y. ; JOSEPH, Anthony D. ; KATZ, Randy H. ; KUBIATOWICZ, John D.: Bayeux: An Architecture for Scalable and Fault-Tolerant Wide-Area Data Dissemination. In: *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, 2001, S. 11–20 15, 98