Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Diplomarbeit

Distributed Accounting for P2P Content Services

Silvan Hofstetter, ETH Zürich, D-INFK

TIK No. DA-2004-15
Sommersemester 2004

Supervisor: Prof. Dr. B. Stiller, ETH Zürich, TIK
Tutor: D. Hausheer, ETH Zürich, TIK

5. August 2004

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Abstract

In the last few years, peer-to-peer systems became a hot topic in computer science. P2P file sharing applications like Napster or Gnutella, were even discussed in the general public. In a peer-to-peer system a group of computer users form a dynamic network, in which they can offer each other resources or can consume resources offered by other users. The main difference between a peer-to-peer system and a classical client/server system is that in a client/server system the role of each user in the system is fixed, while in a peer-to-peer system, each user is free to choose and change his role. In peer-to-peer systems peers can act as servers and/or as clients. As server they offer resources, as client they consume resources. Client/server applications are initialized by the clients and can only use resources a server offers. If the server is down or does not allow the use of his resources, the correct execution of a client/server application is not possible. In contrast to client/server applications, peer-to-peer applications can use resources on several peers and its correct execution is not depending on only one peer. Therfore, peer-to-peer applications can reach a better robustness and performance then classical client/server applications.

Peer-to-peer systems are typically designed around the assumption that all peers are disposed to offer approximately as much resources as they consume. In fact all peers act rational and they do not offer any of their resources, if they have no benefit of that. The described attitude of the peers tends to result in an overall performance of peer-to-peer applications that is not as good as expected. To handle this so called freerider problem, special mechanisms need to be integrated into peer-to-peer systems. One of these mechanisms is accounting. Accounting enables to monitor how many resources each peer consumed and how many resources the peer offered. The challenge of designing of an accounting mechanism is that it is reliable, scalable and available.

In this paper, the design, implementation and evaluation of an accounting mechanism is shown. A distributed accounting mechanism has been developed at TIK and is described in [1]. The mechanism realizes the concept of remote accounting, where an account for each peer in a peer-to-peer system is managed by one or more other peers in the system, called remote peers.

The distributed accounting mechanism is an accounting pattern, which implements an interface, called accounting API. Therefore, it is possible for peer-to-peer applications to use the distributed accounting mechanism for accounting without paying attention on the realization of the mechanism. Using the accounting API peer-to-peer applications can configure the distributed accounting mechanism, can create new accounting sessions and are able to update and query the account balances of peers. In this report, the realizations of all described func-

tions of the distributed accounting mechanism are illustrated. For the implementation of the mechanism the peer-to-peer infrastructure Pastry [8] is used. It is shown, how the functionalities of Pastry are used, to simplify the implementation of the distributed accounting mechanism.

A simulation component is described, which allows the evaluation of the implemented mechanism. The component creates a peer-to-peer network and generates randomly a number of interactions between the peers to simulate the execution of a peer-to-peer application that uses the distributed accounting mechanism. Furthermore, the distributed accounting mechanism the simulation component computes statistical values of the generated simulation process and compares them with the expected results.

Danksagung

Diese Arbeit hätte ohne die grosszügige Hilfe von verschiedenen Personen nicht realisiert werden können. Ich möchte mich insbesondere bei meinem Tutor David Hausheer und meinem Supervisor Prof. Dr. Burkhard Stiller dafür bedanken, dass sie mir jederzeit mit Rat und Tat zur Seite gestanden haben. Ein spezieller Dank gebührt auch Beat Herensperger, der mir bei der Korrektur dieser Dokumentation behilflich war, sowie meiner Mutter, die mich während der schönen und lehrreichen, aber teilweise auch anstrengenden Arbeit immer wieder moralisch unterstützt hat.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Peer-to-Peer Netzwerke avancierten in jüngerer Vergangenheit zu einem, auch in der breiten Öffentlichkeit, viel diskutierten Thema innerhalb der Informatik. Hauptsächlich dafür verantwortlich waren kontroverse und millionenfach besuchte Musiktauschbörsen wie Napster oder Gnutella. Diese Filesharing Systeme stellen auch zurzeit noch die mit Abstand am weitesten verbreitete P2P Applikation dar. Daneben gibt es aber eine ganze Reihe weiterer Anwendungen, bei denen Peer-to-Peer Systeme zum Einsatz gelangen können. Distributed Computing ist eine solche Anwendung, bei der die an der Applikation beteiligten Rechner ihre CPU-Zeit den anderen Peers zur Verfügung stellen. Dadurch können komplexe Berechnungen durchgeführt werden, die sonst entweder einen sehr viel grösseren Zeitaufwand, oder einen Rechner mit enorm viel Rechenleistung erfordern würden. P2P Content Storage ist eine weitere P2P Applikation, bei der ein oder mehrere Rechner anderen Rechnern anbieten, Daten für sie zu speichern. Solche Daten können beispielsweise ganze oder Teile von einzelnen Dateien sein. PAST [13] ist ein Beispiel für eine Umsetzung des P2P Content Storage. P2P Content Storage kann unter anderem zur Content Distribution genutzt werden, bei dem es darum geht Daten auf verschiedene Rechner zu verteilen. Als letzte P2P Anwendung sei hier das Instant Messaging genannt, bei dem die Kommunikation der Netzteilnehmer in Echtzeit im Mittelpunkt steht.

Anhand von Abbildung 1.1, in welcher sowohl ein Client/Server System und ein Peer-to-Peer System zu sehen ist, ist der Unterschied zwischen einer Client/Server Applikation und einer Peer-to-Peer Applikation gut ersichtlich. Ein grosser Unterschied zwischen den beiden Systemen ist die Verteilung der Ressourcen. Mit den Ressourcen sind die Leistungen gemeint, welche sich die einzelnen Rechner im System gegenseitig anbieten können. Wie bereits erwähnt,

können solche Ressourcen Dateien, aber auch zur Verfügung gestellte Rechenleistung oder Speicherkapazität sein. Mit Service wird im Übrigen das zur Verfügung stellen von Ressourcen bezeichnet. Im Client/Server Modell können nur nur die Server Ressourcen anbieten, während im Peer-to-Peer System alle Peers Ressourcen zur Verfügung stellen können. Eine Client/Server Applikationen läuft nun so ab, dass die Clients Anfragen an einen Server stellen, um seine angebotenen Ressourcen konsumieren zu können. Der Server generiert unter zu Hilfenahme seiner Ressourcen eine Antwort auf die Anfragen und schickt diese an die Clients zurück. Im Gegensatz zum Client/Server System sind die Rollen im Peer-to-Peer System nicht so eindeutig verteilt. Ein Peer kann hier sowohl als Client, als auch als Server in Erscheinung treten. Peer-to-Peer Applikationen machen sich genau diesen Aspekt zu Nutzen. Sie können von sämtlichen Peers gestartet werden und greifen während ihrer Ausführung auf die Ressourcen zu, welche andere Peers anbieten.

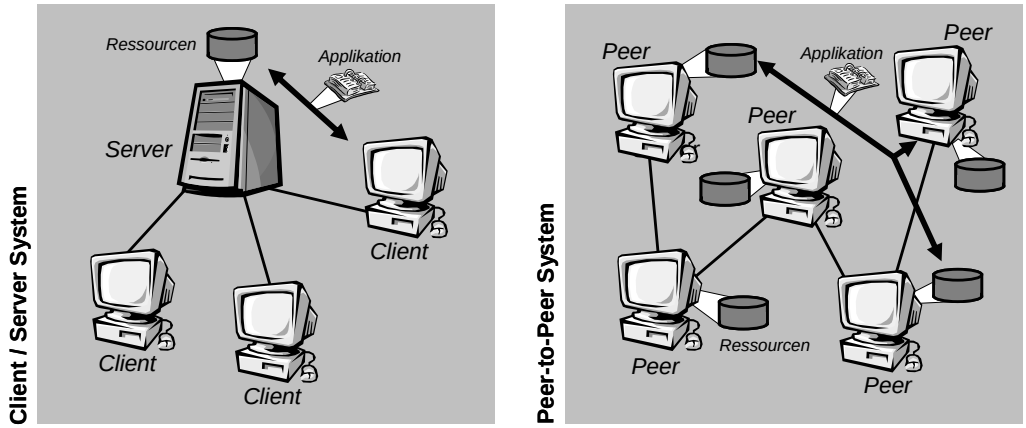

Abbildung 1.1: Vergleich von Client/Server- und Peer-to-Peer System

Im Vergleich zu Client/Server Applikationen, erreichen Peer-to-Peer Applikationen eine viel höhere Robustheit und Performance. Die bessere Robustheit, lässt sich durch die Struktur der beiden Systeme erklären. Fällt während der Ausführung einer Client/Server Applikation der Server aus, so kann sie nicht zu Ende gebracht werden, da nur der Server die benötigten Ressourcen anbieten kann. Eine Peer-to-Peer Applikation ist nicht auf einen zentralen Rechner angewiesen, weshalb sie mit dem Ausfall eines einzelnen Peers umgehen und trotzdem noch erfolgreich abgeschlossen werden kann. Die bessere Performance der Peer-to-Peer Applikationen ist ebenfalls auf die Verteiltheit der Ressourcen im Peer-to-Peer System zurückzuführen. Im Client/Server System kann der Server mit sehr vielen Anfragen konfrontiert werden, da alle Clients auf seine Ressourcen angewiesen sind. In Peer-to-Peer Systemen ist es unwahrscheinlicher, dass einzelne Peers mit diesem Problem zu kämpfen haben, da Ressourcen ja von vielen Peers zur Verfügung gestellt werden.

Die Gesamtperformance eines P2P System hängt sehr davon ab, wie viel Ressourcen die einzelnen Peers konsumieren und im Gegenzug den anderen Peers zur Verfügung stellen. Beziehen nämlich viele Peers mehr Ressourcen, als sie selbst anbieten, so kommt es zu einer ungleichen Verteilung der Ressourcen im System und die Performance sinkt relativ rasch. Mit diesem so genannten Freerider Problem sind die meisten gebräuchlichen Peer-to-Peer Anwendungen konfrontiert, da sich einfach sehr viele Peers rational verhalten und den anderen Peers keine eigenen Ressourcen zur Verfügung stellen, wenn sie daraus selbst keinen direkten Nutzen ziehen können. Daher liegt es nahe, Mechanismen in P2P Systeme zu integrieren, welche den Aufwand und den Nutzen gleichmässig über alle Peers verteilen. Einer dieser Mechanismen ist das Accounting, bei dem laufend registriert wird, wie viele Ressourcen die einzelnen Peers angeboten und konsumiert haben. Anhand dieser aufgezeichneten Accounts können dann die Peers entscheiden, ob sie einem bestimmten Peer ihre Ressourcen zur Verfügung stellen möchten oder nicht.

Damit ein Accounting Mechanismus effizient in einem Peer-to-Peer System eingesetzt werden kann, muss er gewisse Kriterien erfüllen. Zum einen muss er verlässlich sein. Er muss dafür garantieren können, dass die Accounts der einzelnen Peers korrekt aktualisiert und abgefragt werden können. Wäre dies nicht gegeben und könnten Accounts unberechtigterweise verändert werden, so liesse sich natürlich nicht mehr überprüfen, welche Peers wie viele Ressourcen bezogen, respektive angeboten haben. Um die Account Balance eines Peers in Erfahrung bringen zu können, muss dieser natürlich auch verfügbar sein. Die Verfügbarkeit der Accounts ist somit ein weiteres zentrales Kriterium an einen Accounting Mechanismus. Schliesslich soll ein Accounting Verfahren auch skalierbar sein, damit es unabhängig von der Anzahl Peers im P2P Systemen effizient eingesetzt werden kann.

Es gibt viele verschiedene Accounting Mechanismen, welche in [3] beschrieben sind. Der einfachste ist sicher das lokale Accounting, bei dem jeder Peer selbst für die Verwaltung und Aktualisierung seines Accounts verantwortlich ist. Das Local Accounting ist sehr leicht zu implementieren, hat jedoch den grossen Nachteil, dass die Vertrauenswürdigkeit des Accounts nicht garantiert werden kann. Das Public Accounting versucht dieses Problem zu lösen, indem sich die Peers von ihren Transaktionspartnern oder von verschiedenen, vertrauenswürdigen Peers das Beziehen und Anbieten von Ressourcen mit Zertifikaten bestätigen lassen. Durch das Verifizieren und Ausstellen von Zertifikaten entsteht allerdings ein sehr grosser, zusätzlicher Netzwerkverkehr, der überproportional mit der Anzahl, im System vorhandener Peers wächst. Das Verfahren ist deshalb nicht skalierbar. Ein weiterer Accounting Mechanismus ist das Central Accounting, bei dem ein bestimmter Peer als zentrale Bank die Accounts der einzelnen Peers verwaltet und so für die faire Verteilung der angebotenen Ressourcen sorgt. Durch

die Bank als zentrale Komponente krankt dieser Ansatz hingegen an der Verfügbarkeit. Das Token-based Accounting lehnt sich am Zahlungsverkehr in der realen Welt an. Die Peers bezahlen konsumierte Ressourcen bei den jeweiligen Anbietern mit Tokens. Ein Token ist dabei letztendlich ein Bitmuster, welches sich die einzelnen Peers zuschicken können. Die P2P Systeme, welche das Token-based Accounting verwenden, zeichnen sich durch eine gute Skalierbarkeit und eine hohe Verlässlichkeit aus. Sie müssen aber zusätzliche Verfahren enthalten, die das Fälschen und den mehrfachen Gebrauch von Token verhindern, was die Effizienz des Mechanismus schmälert. Beim Remote Accounting sind für die Verwaltung eines Peer Accounts schliesslich mehrere Remote Peers zuständig. Somit kann Verfügbarkeit und Vertraulichkeit der einzelnen Peers grundsätzlich ohne weitere, zusätzliche Sicherheitsmassnahmen garantiert werden. Allerdings hängt die Güte dieses Verfahren im Gegenzug immer von der Böswilligkeit der am Netzwerk beteiligten Peers ab.

1.1 Ziel der Arbeit

Im Rahmen dieser Arbeit soll der am TIK entworfene Distributed Accounting Mechanismus [1] welcher den Remote Accounting Ansatz verwendet, weiterentwickelt und anhand eines Prototyps evaluiert werden. Das daraus entstandene Accounting Pattern wird dann zu einem späteren Zeitpunkt in das Accounting und Charging Framework integriert, welches im Laufe des MMAPPS Projektes [2] entwickelt wurde.

Wichtig dabei ist, dass das Pattern als Komponente auf jedem Peer eingesetzt wird und gegen aussen gewisse Funktionalitäten anbieten soll. Diese Funktionalitäten umfassen das Konfigurieren des Accounting Mechanismus und der einzelnen Peers, das Erstellen von neuen Accounting Sessions und die Möglichkeit des Aktualisierens und Abfragens von Peer Accounts.

Bei der Evaluation sollen ausgehend von der Peer Uptime und der Böswilligkeit der Peers, sowie des Redundanz- und Verteiltheitsgrades des Accounting Verfahrens die Verlässlichkeit, Skalierbarkeit und Verfügbarkeit eines zugrunde liegenden P2P Systems untersucht werden.

1.2 Gliederung der Arbeit

Die Arbeit gliedert sich in sechs Kapitel. Im gleich folgenden Kapitel drei Accounting Mechanismen vorgestellt, welche mit dem, in dieser Arbeit beschriebenen Distributed Accounting Mechanismus verwandt sind. Die detaillierte Beschreibung des zu implementierenden Verfahrens folgt im nächsten Kapitel. Vorhandene P2P Infrastrukturen bieten eine Anzahl von nützlichen Funktionen, welche die Kommunikation und Interaktion der Peers untereinander vereinfachen

kann. Im Gegensatz zum stand-alone Ansatz bietet sich somit bei der Implementierung des Accounting Verfahrens die Verwendung einer solchen Infrastruktur an. Daher wird im dritten Kapitel beschrieben, welche P2P Infrastruktur sich für die Implementation des Accounting Mechanismus am besten eignet. Die eigentliche Implementation des Distributed Accounting Mechanismus ist im anschliessenden Kapitel beschrieben. In Kapitel 5 ist die Evaluation des Accounting Verfahrens erklärt, bevor dann im letzten Kapitel die Arbeit zusammengefasst und ein Ausblick auf zukünftige Erweiterungen des entwickelten Prototyps geworfen wird.

Kapitel 2

Verwandte Arbeiten

In diesem Kapitel werden drei verschiedene Accounting Mechanismen vorgestellt. Sie verdeutlichen die Probleme und Schwierigkeiten beim Accounting und geben für diese Lösungsansätze.

2.1 Token-based Accounting Pattern

Dieses Verfahren, welches in [3], [4] beschrieben wird, wurde im Rahmen des MMAPPS Projektes entwickelt und soll dabei helfen, ein Peer-to-Peer File Sharing Szenario zu realisieren, welches sich nach den in der Marktwirtschaft üblichen Mechanismen richtet. Im Wesentlichen heisst das, dass ein Konsument einer Leistung einen entsprechenden Betrag auf eine sichere Art an den Anbieter überweisen muss und diese Überweisung auch nachweisen kann. Das Pattern gewährleistet die Skalierbarkeit eines P2P Systems und sorgt für vertrauenswürdige Transaktionen zwischen den einzelnen Peers. Dafür werden drei Protokolle verwendet, welche für die Aggregation von Tokens, das Verhindern der Mehrfachverwendung von Tokens und den Zahlungsverkehr sorgen.

Beim Token-based Accounting Pattern müssen Peers einerseits die Konsumation von Ressourcen mit Token bezahlen, während sie andererseits für das Bereitstellen von Ressoucen Token erhalten. Damit die digitalen Geldstücke nicht gefälscht oder gestohlen werden können, müssen die Peers diese vor dem Ausgeben von Super Peers digital signieren lassen. Super Peers sind bestimmte Peers im Peer-to-Peer, die im Normalfall eine hohe Vertrauenswürdigkeit bei den anderen Peers besitzen. Der Ablauf dieser Aggregation der Token ist in Abbildung 2.1 dargestellt. Ein Peer schickt dabei zuerst diejenigen Token an einen Super-Peer, die er zuvor für das zur Verfügung stellen von Ressourcen verdient hat ①. Der Super

Peer überprüft die Gültigkeit der erhaltenen Token und generiert aus ihnen eine Menge neuer, unsignierter Token ②. Die Anzahl dieser wird durch eine spezielle Aggregationsfunktion bestimmt. Im Weiteren benutzt das Pattern die Threshold RSA Kryptographie, um die erzeugten Token mit einem Shared Key zu signieren. Dieses Verschlüsslungsverfahren erlaubt nicht nur die dezentrale Generierung eines geheimen Schlüssels und die Aktualisierung desjenigen, sondern garantiert auch, dass die einzelnen Super Peers die Token zwar signieren können, den vollständigen Private Key dabei aber nicht kennen. Die unsignierten Token gelangen also zu einer bestimmten Anzahl an weiteren Super Peers ④, werden dort partiell signiert ⑤ und an den ursprünglichen Peer zurückgesendet ⑥. Der Peer generiert schliesslich aus den signierten Token die neuen Token ⑦, mit Hilfe derer er weitere Ressourcen konsumieren kann.

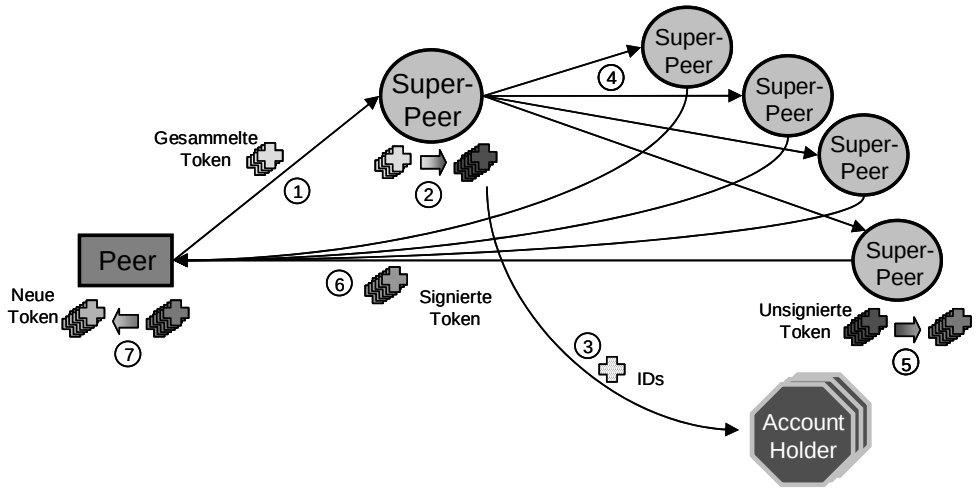

Abbildung 2.1: Token Aggregation des Token-based Accounting Pattern [3]

Damit eine Transaktion zwischen zwei Peers für beide Partner zufrieden stellend verläuft, stellt das Token-based Accounting Pattern ein Protokoll für den Zahlungsverkehr zur Verfügung. Dieses stellt sicher, dass sich weder der Service Anbieter, noch der Service Konsument einen Vorteil verschaffen können, indem sie ihrem jeweiligen Partner den vorher vereinbarten Service, respektive die vorher abgemachte Entlöhnung während der eigentlichen Transaktion verwehren. Zusätzlich wird während des Zahlungsverkehrs das Double Spending von Token verhindert.

Die Funktionsweise des Protokolls für den Zahlungsverkehr ist in Abbildung 2.2 dargestellt. Nachdem sich die Transaktionspartner auf den Service und dessen Kosten einigen konnten ①, schickt der Konsument dem Anbieter die IDs der von ihm für die Zahlung verwendeten Token, damit ein wiederholtes Verwenden dieser ausgeschlossen werden kann ②. Das Double Spending wird dabei von Account Holdern sichergestellt. Während der Aggregation der Token haben diese

nämlich die IDs der neu generierten Token bereits erhalten, wie in Abbildung 2.1 ersichtlich ist ③. So ist es ihnen nun möglich die entsprechenden Token während dem Zahlungsverkehr als ausgegeben zu markieren ④ und dem Service Anbieter eine Liste mit den Token zurückzuschicken, welche bisher noch nicht verwendet wurden ③. Nach dieser Überprüfung erfolgt die eigentliche Transaktion. Der Konsument und der Anbieter tauschen die vereinbarten Leistungen untereinander aus ⑤. Dabei schickt der Konsument dem Anbieter vorerst nur unsignierte Token, welcher dieser nicht für weitere Transaktionen verwenden kann und daher auch keinen Anreiz hat, dem Konsument seinen Service nicht zur Verfügung zu stellen. Erst nachdem der Konsument den vereinbarten Service erhalten hat, übermittelt er dem Anbieter die, mit seinem Private Key signierten Token ⑥. Nur diese werden von einem Super-Peer bei einer nächsten Token Aggregation als gültig erachtet. Bei eben diesem Prozess können die Account Holder dann die als ausgegeben markierten IDs wieder löschen ⑦.

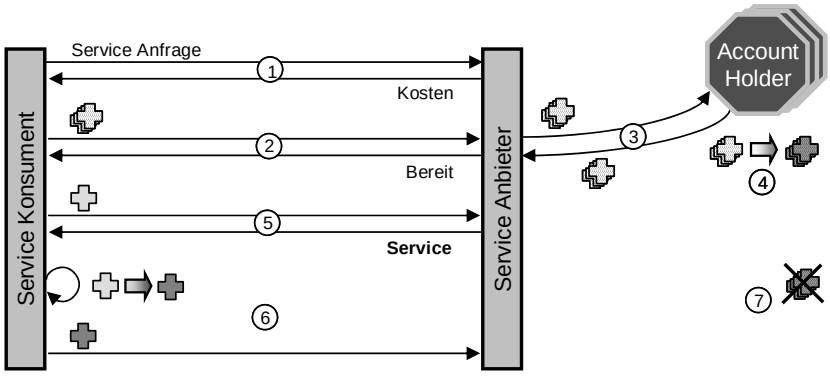

Abbildung 2.2: Zahlungsverkehr des Token-based Accounting Pattern [3]

2.2 PPay

Ein Micropayment Verfahren ist PPay [5]. Dabei können die Peers bei einem zentralen Broker gegen ein Entgelt so genannte Raw Coins beziehen. Ebenfalls ist es ihnen möglich, Coins, welche sie für das Anbieten von Ressourcen von anderen Peers erhalten haben, beim Broker gegen Cash umzutauschen. Damit der Broker nicht bei allen Transaktionen zwischen zwei Peers involviert sein muss und das System dadurch effizienter wird, ermöglicht das PPay Protokoll den Konsumenten von Ressourcen, selbstständig eigene Coins auf andere Peers zu überschreiben. In gewissen Spezialfällen funktioniert dies hingegen nicht und der Broker muss für den Austausch der Coins zusätzlich hinzugezogen werden. Für genau diese Fälle stellt das PPay Protokoll neben der Basis Implementation auch ein Downtime Protokoll zur Verfügung. Darüber hinaus nutzt das Verfahren für die Authentifizierung und Verschlüsselung von Nachrichten die Public Key Kryptographie.

Der Wechsel des Eigentümers eines Coins ist in der Abbildung 2.3 zu erkennen. Am Anfang des Protokolls steht dabei immer ein Raw Coin, der ein Peer vom Broker erhält ①. Dieser Raw Coin enthält die Seriennummer des Coins und die ID des Coin Eigentümers und wird mit dem Private Key des Brokers verschlüsselt. Konsumiert der Peer Ressourcen eines Peers V, so muss er diesem für seinen Service mit Coins belohnen. Er überschreibt einen seiner Raw Coins an den Peer V, indem er diesem die ID von V, sowie eine Sequenznummer hinzufügt und ihn mit seinem eigenen Private Key verschlüsselt ②. Den neu generierten Coin schickt U dann an V ③. Möchte der Peer V dann zu einem späteren Zeitpunkt einen Peer X für dessen Dienste entlöhnen, so kann er dafür den von U erhaltenen Coin verwenden. Er schickt deshalb eine Reassignment Anfrage an U ④. Diese Nachricht enthält neben dem Coin auch die ID von X und ist mit dem Privat Key von V verschlüsselt. Der Peer U speichert die Reassignment Anfrage und generiert einen neuen Coin, der neben dem ursprünglichen Raw Coin die ID von X und eine Sequenznummer enthält. Einmal mehr wird die Nachricht mit dem Private Key des entsprechenden Peers verschlüsselt ⑤. Den neuen Coin schickt U sowohl an V, als auch an X ⑥. V erhält den Coin deshalb, damit er beweisen kann, dass er X für seine Leistungen bezahlt hat, falls dieser später einmal das Gegenteil behaupten sollte. Weiter sollte V den Coin, welcher er im ersten Schritt von U erhalten hat, im weiteren Verlauf der Anwendung nicht mehr einsetzen können. Indem U bei der Generierung neuer Coins stetig eine höhere Sequenznummer erhält, kann dies sichergestellt werden. Dank der gespeicherten Reassignment Anfragen kann U darüber hinaus als Schlichtungsstelle bei einem allfälligen Streit zwischen V und X fungieren.

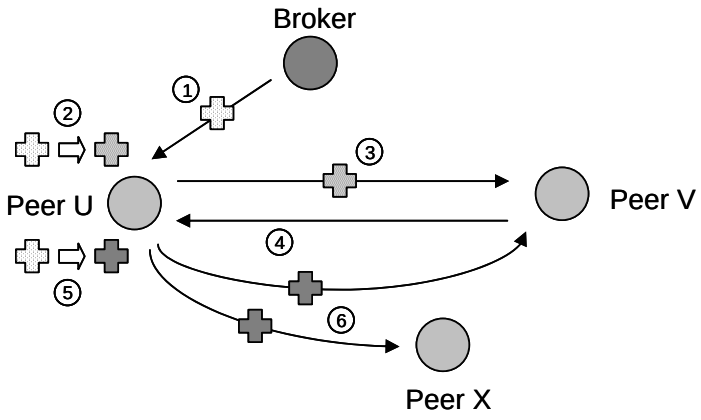

Abbildung 2.3: Basis Implementation von PPay

Das Überschreiben eines Coins von Peer V zu Peer X, kann nur dann funktionieren, wenn Peer U online und damit für V zu erreichen ist. Falls dies nicht der Fall ist, stellt PPay ein weiteres Protokoll zur Verfügung. Wie dieses funktioniert, ist in Abbildung 2.4 zu sehen. Peer V möchte einen Coin an Peer X überschreiben, den er ursprünglich von Peer U erhalten hat. Da er aber festgestellt hat, dass X

nicht online ist, schickt er eine Anfrage an den Broker, in welcher der Coin von U enthalten ist ①. Mit Hilfe dieses Coins generiert der Broker daraufhin einen neuen Coin ② und schickt ihn an V und X ③. Die Generierung verläuft dabei äquivalent zu jener des Peers U, aus der Basis Implementation, mit dem Unterschied, dass der Broker natürlich nicht den Private Key von U benutzen kann, sondern seinen eigenen verwenden muss. Sobald der Peer U wieder online ist informiert ihn der Broker über die eingegangene Anfrage ④. Anhand dieser Nachricht kann U überprüfen, ob ein Fehlverhalten der beiden involvierten Peers vorliegt, oder ob die Zahlung rechtens war.

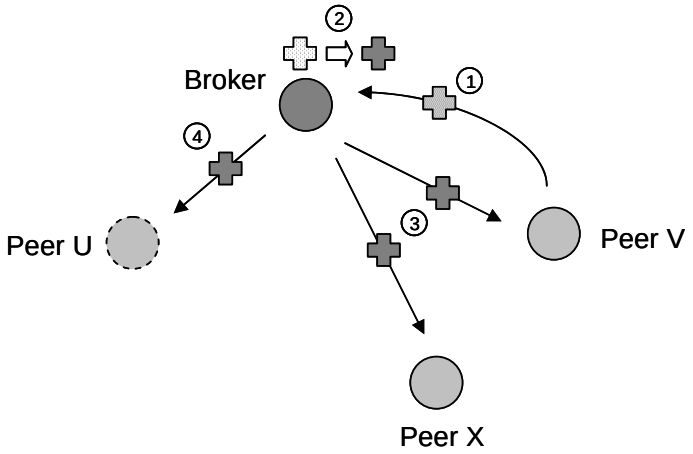

Abbildung 2.4: Downtime Protokoll von PPay

Im Gegensatz zu dem im Abschnitt 2.1 beschriebenen Token-based Accounting Pattern, kann bei PPay das Fälschen oder das wiederholte Ausgeben von Coins, respektive von Token nicht von Vornherein verhindert werden. Das Verfahren macht einen solchen Betrug aber unrentabel, indem jeder Missbrauch entdeckt, zum fehlbaren Benutzer zurückverfolgt und bestraft werden kann.

2.3 KARMA

KARMA [6] ist ein Peer-to-Peer Mechanismus, das einen Remote Accounting Ansatz verwendet. Jeder P2P Knoten hat einen skalaren Wert, welcher Karma genannt wird und den Stellenwert oder Rang des Knotens im System angibt. Für die Aktualisierung und Verwaltung der einzelnen Werte sind im Framework mehrere Knoten verantwortlich, die eine Bank für den entsprechenden Peer bilden.

Stösst ein neuer Knoten zum P2P-Netz hinzu, so durchläuft er zuerst eine Initialisierungsphase, in welcher er sich einen Public und Private Key generiert. Als nächstes errechnet er sich aufgrund eines kryptographischen Puzzles und seines Public Key eine zufällige ID. Durch die so erhaltene ID sind die Knoten, welche

für den Account des Peers zuständig sind, eindeutig bestimmt. Andererseits wird ein neuer Knoten selbst zu einem Remote Peer und als solcher für das Verwalten des Karmas von verschiedenen anderen Knoten zuständig. Da immer gleich viele Knoten die Bank eines Peers bilden, verändern sich mit dem neuen Knoten natürlich auch die Banken. Eine Synchronisation der betroffenen Peers ist deshalb nötig.

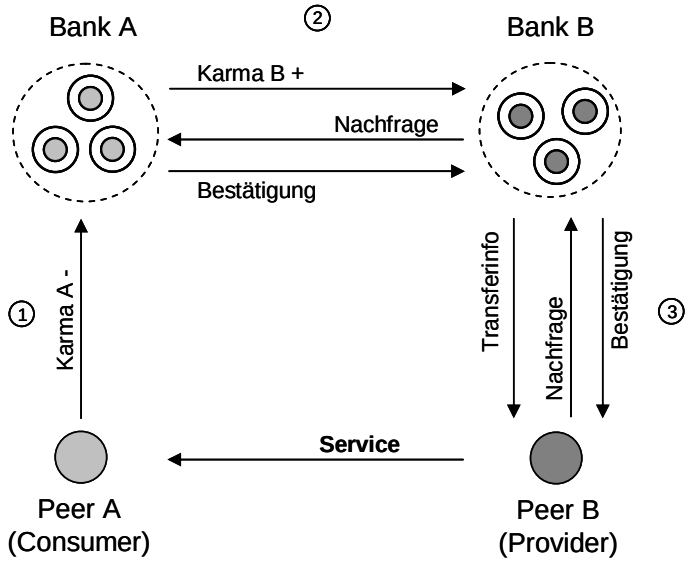

Abbildung 2.5: Accounting Mechanismus von KARMA

Der eigentliche Austausch von Ressourcen und der, damit einhergehende Accounting Mechanismus ist in Abbildung 2.5 dargestellt. Peer A tritt dabei als Consumer auf, welcher einen Service von Peer B in Anspruch nimmt. Folglich müssen die Karmas von Peer A und B aktualisiert werden. Das geschieht, indem Peer A eine entsprechende Nachricht an die P2P-Knoten in seiner Bank schickt ①, welche das Karma von A gemäss der Abmachung von A und B reduzieren. In einer zweiten Phase ② informieren die Peers in der Bank A sämtliche Peers in der Bank B über das Update. Hierfür garantiert ein spezielles Verfahren, das in [6] beschrieben ist, eine authentische Kommunikation. Haben die Peers in der Bank B das Karma von B erhöht, so wird der Provider in der dritten Phase ③ über diese Aktualisierung benachrichtigt. Zur Authentifizierung wird das gleiche Verfahren wie in der vorherigen Phase verwendet.

Kapitel 3

Konzept und Design

Wie bereits in Kapitel 1 beschrieben, ist das Ziel dieser Arbeit, den in [1] beschriebenen Distributed Accounting Mechanismus so zu implementieren, damit dieser P2P Applikationen die Möglichkeit bietet, von den Vorteilen des Accountings zu profitieren, ohne sich um dessen konkrete Ausführung kümmern zu müssen. Um dies zu ermöglichen, muss der Distributed Accounting Mechanismus P2P Applikationen eine Schnittstelle zur Verfügung stellen, über welche sie das Verfahren steuern können und über die sie sich die, für ihre Ausführung nötigen Informationen beschaffen können. Die Schnittstelle, als Accounting API bezeichnet, ist so allgemein definiert, dass sie für die Implementation von beliebigen weiteren Accounting Mechanismen ebenfalls verwendet werden kann. Konkret bietet sie den P2P Applikationen vier Methoden an, die es ihnen ermöglichen den Accounting Mechanismus zu konfigurieren, neue Sessions zu kreieren, sowie die Account Balance eines bestimmten Peers zu aktualisieren oder abzufragen. Unter einer Session versteht man einen Vorgang zwischen zwei Peers, während dem ein Peer die Rolle des Providers übernimmt und dem anderen Peer, welcher als Consumer bezeichnet wird, eine Reihe von Services anbieten kann. Jedes Accounting Verfahren, welche das beschriebene Accounting API verwendet, bezeichnet man als Accounting Pattern.

Um die Erstellung von Applikationen zu erleichtern, welche auf einem P2P Netzwerk basieren, existieren diverse, umfangreiche P2P Infrastrukturen. Sie übernehmen für den Benutzer nicht nur den Aufbau und die Verwaltung eines Peer-to-Peer Netzwerkes, sondern stellen dem Anwender darüber hinaus auch eine Reihe weiterer nützlicher Funktionalitäten zur Verfügung. Die Kommunikation zwischen den einzelnen Peers vereinfacht sich damit wesentlich. Aus diesem Grund bietet sich bei der Umsetzung des Distributed Accounting Mechanismus die Verwendung einer solchen Infrastruktur an.

Die so entstandene Architektur der einzelnen Peers ist in Abbildung 3.1 zu sehen. P2P Applikationen können über das Accounting API auf die Accounting Funktionalitäten des Distributed Accounting Mechanismus oder einen beliebigen weiteren Accounting Mechanismus zugreifen. Der Mechanismus selbst basiert seinerseits auf einer P2P Infrastruktur und nutzt deren Funktionen über ein Infrastruktur API. Ebenso können auch die P2P Applikationen auf die Schnittstelle der Infrastruktur zugreifen und deren Funktionen verwenden. Ebenso ist es möglich, dass die Applikationen direkt über die Netzwerkschicht kommunizieren.

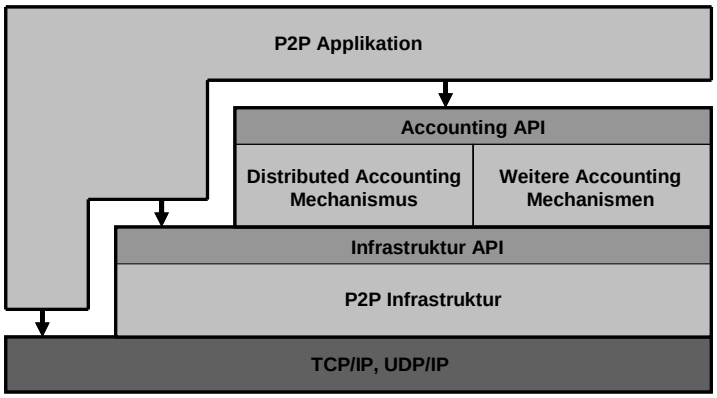

Abbildung 3.1: Architektur des Peers

In den nächsten Abschnitten sollen nun die einzelnen Bestandteile des Distributed Accounting Mechanismus erläutert werden. Gleich anschliessend wird die grundsätzliche Idee des Distributed Accounting Mechanismus vorgestellt. Im darauf folgenden Abschnitt wird mit Pastry die P2P Infrastruktur betrachtet, welche für die Implementation des Distributed Accounting Mechanismus verwendet wurde. Auf den Konzepten von Pastry basiert dann der anschliessende Abschnitt 3.3, in welchem das Accounting API und deren einzelne Methoden beschrieben sind. In den nächsten beiden Abschnitten folgen Ausführungen zur Verlässlichkeit des Accounting Mechanismus und zu den Synchronisationsverfahren, welche in diesem nötig sind. Schliesslich wird im Abschnitt 3.6 gezeigt, welche Designalternativen es in Bezug auf die P2P Infrastruktur gegeben hätte und weshalb Pastry für die Implementierung des Distributed Accounting Mechanismus gewählt wurde.

3.1. Distributed Accounting Mechanismus

Der zu implementierende Distributed Acccounting Mechanismus ist ein typisches Remote Accounting Verfahren [3]. Unter Remote Accounting versteht man einen Accounting Mechanismus, bei dem der Account eines Peers nicht von ihm selbst, sondern von einem oder mehreren fremden Peers, verwaltet werden. Die für die Verwaltung zuständigen Peers werden als Remote Peers bezeichnet. Im Übrigen wird die Menge aller Remote Peers eines bestimmten Peers als Remote Set des

Peers bezeichnet. In einem ersten Schritt bestimmen die einzelnen Peers ihre Remote Peers und sorgen dafür, dass diese einen entsprechenden Account für sie anlegen. Damit die einzelnen Peers ihr Remote Set nicht selbst bestimmen können, muss eine Mapping Rule definiert sein, welche allen Peers auf eine nicht zu beeinflussende Art und Weise ein Remote Set zuweist. Abbildung 3.2 zeigt das an die Konfigurationsphase folgende Accounting Szenario des Mechanismus. Dabei hat das Verfahren für die korrekte Aktualisierung und Verwaltung der Accounts der einzelnen Peers durch die Remote Peers zu sorgen. Jeder Account besitzt eine Account Balance, welche angibt, wie viele Ressourcen der entsprechende Peer konsumiert, respektive angeboten hat. Die Account Balance wird erhöht, sobald ein Peer als Provider Ressourcen angeboten hat und wird vermindert, wenn er als Consumer einen Service eines anderen Peers in Anspruch genommen hat. Im Beispiel aus Abbildung 3.2 möchte Peer B einen Service von Peer A beziehen. Dazu müssen die beiden Peers zuerst eine Session generieren. Sie vereinbaren ein Service Level Agreement (SLA), in welchem die Details der Session, wie der Tarif oder die Art der konsumierten, respektive angebotenen Ressource beschrieben. Nachdem sich die Peers auf eine Session geeinigt haben, stellt der Provider dem Consumer die soeben spezifizierten Service zur Verfügung. Gemäss dem SLA werden zu Beginn, während oder nach der Session die Accounts der Peers A und B aktualisiert. Dies geschieht, indem Update Nachrichten an die eigenen Remote Peers, sowie an diejenigen des Sessionspartners geschickt werden. Erhält ein Remote Peer sowohl vom Consumer, als auch vom Provider eine solche Meldung, so kann er sich sicher sein, dass beide Partner mit der Erhöhung, respektive der Verringerung der Account Balance des entsprechenden Peers einverstanden sind und er führt das Update durch. Remote Peers können jederzeit offline gehen. Damit diese ihre Account Balances aktuell halten können, müssen sie sich mit den entsprechenden Remote Peers synchronisieren, sobald sie wieder online kommen.

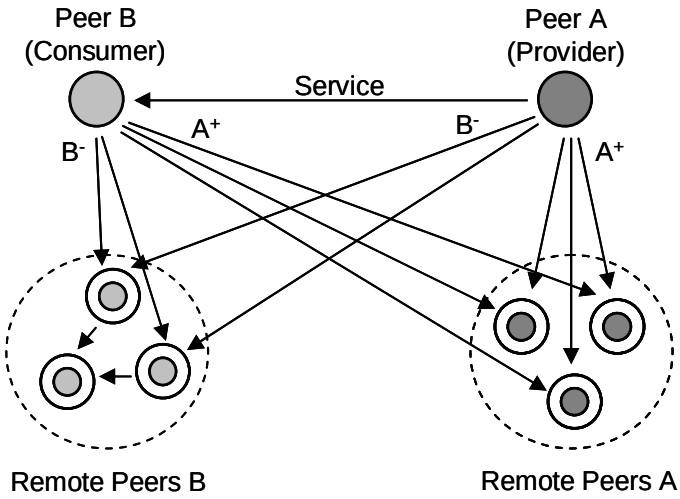

Abbildung 3.2: Remote Accounting Szenario [1]

3.2 Pastry

Pastry [8] ist die P2P Infrastruktur, auf welcher der Distributed Accounting Mechanismus aufbaut. Sie stellt eine Reihe von Funktionen zur Verfügung, welche für die Umsetzung des Mechanismus zentral sind. Darum soll diese Infrastruktur im Folgenden beschrieben werden.

Pastry ist eine skalierbare und dezentrale P2P Infrastruktur, welche für die Objekt Lokalisierung und das Routing in weiträumigen P2P Systemen entwickelt wurde. Dazu verwendet es ein Overlay Netzwerk. Die am P2P System beteiligten Knoten, welche durch das Internet miteinander verbunden sind, müssen darum auf das Pastry Netzwerk abgebildet werden. Das geschieht, indem jeder Knoten eine eindeutige ID erhält, die in einem zirkulären Namensraum liegt und 128 Bit lang ist.

Pastry ist in der Lage anhand eines 128 Bit langen Keys eine Nachricht an denjenigen Pastry Knoten zu routen, dessen ID am nächsten beim Key liegt und der im Allgemeinen als Root Node bezeichnet wird. In [8] wurde gezeigt, dass für das Routing höchstens eine, zur Menge am Netzwerk beteiligter Knoten logarithmische Anzahl an Hops im Pastry Netzwerk notwendig sind. Um dieses effiziente Routing zu gewährleisten, verwenden die Pastry Knoten ein Leaf Set und eine Routing Tabelle. Das Leaf Set enthält die Nachbarknoten, des entsprechenden Pastry Knotens. Diese Nachbarknoten setzen sich allerdings, im Anbetracht der ID des Pastry Knotens, immer zur einen Hälfte aus Knoten mit einer kleineren ID und zur anderen Hälfte aus solchen mit einer grösseren ID zusammen. Erhält ein Knoten nun eine Nachricht mit einem Key, der im Bereich des Leaf Sets liegt, so ist es ihm möglich den Root Node herauszufinden und ihm die Meldung zukommen zu lassen. Liegt der Key der erhaltenen Nachricht nicht innerhalb des Leaf Set Bereichs, so kommt die Routing Tabelle zum Einsatz. In dieser Tabelle sind eine bestimmte Anzahl an weiteren Pastry Knoten gespeichert, die das Weiterrouten der Nachricht an einen Knoten sicherstellen, der mit Sicherheit näher beim Root Node liegt, als der eben erreichte. Pastry sorgt selbst dafür, dass die Leaf Sets und die Routing Tabellen der Knoten in dem sich dynamisch ändernden Netzwerk ständig aktualisiert und so konsistent gehalten werden.

Ein effizientes Routing im virtuellen Pastry Netzwerk entfaltet nur dann seine volle Wirkung, wenn die Nachrichten auch im zugrunde liegenden, realen Netz auf einem möglichst direkten Weg vom Absender zum Empfänger gelangen. Durch die Wahl von geeigneten Knoten in der Routing Tabelle, kann Pastry zwar nicht garantieren, dass eine Nachricht auf dem kürzest möglichen Weg zum Empfänger gelangt, kann aber immerhin sicherstellen, dass diese sich bei jedem Routing Schritt weiter vom Absender entfernt.

3.3 Accounting API

Der Distributed Accounting Mechanismus bietet P2P Applikationen wie bereits erwähnt eine Schnittstelle mit vier Funktionen an. Das sind die Methoden *configure*, *createSession*, *updateAccountBalance* und *queryAccountBalance*. In den folgenden Abschnitten wird deren Sinn und Zweck, sowie ihre Funktionsweise genau erklärt. Der Begriff Knoten, wird dabei als Synonym für Peer verwendet und soll die Tatsache verdeutlichen, dass die Peers in einem Peer-to-Peer System Knoten eines Peer-to-Peer Netzwerks sind. Zur Umsetzung der Methoden wird auf die Funktionalitäten von Pastry zurückgegriffen.

3.3.1 Methode *configure*

Über diese Methode übergibt der Benutzer des Distributed Accounting Mechanismus dem Pattern die benötigten Parameter für den Accounting Mechanismus. Für den Distributed Accounting Mechanismus sind dies zum Beispiel der Redundanz- und Verteiltsheitsgrad des Verfahrens.

Die Auswahl der Remote Peers ist eine zentrale Komponente des Remote Accounting. Allfällige, böswillige Peers dürfen auf der einen Seite keinen Einfluss auf die Mapping Rule haben. Ansonsten könnten sie das Verfahren so beeinflussen, dass ihre Accounts von Peers verwaltet werden, die vertrauenswürdig sind. Auf der anderen Seite hat ein ehrlicher Peer natürlich auch gewisse Anforderungen an seine Remote Peers. Sie sollen seinen Account wahrheitsgetreu verwalten und aktualisieren. Dank der Verwendung von Pastry ist es relativ einfach, ein solches Verfahren zu kreieren. Den Peers ist es nämlich nicht möglich, sich im Overlay Netzwerk an einer selbst gewählten Stelle zu platzieren, da IDs der Knoten in Pastry absolut gleichmässig verteilt werden. Wird nun die ID des Knotens dazu verwendet die Remote Peers zu bestimmen, so ist die erste Anforderung an die Mapping Rule bereits erfüllt da kein Peer Einfluss auf die Wahl seiner Remote Peers nehmen kann. Weiter sind dank der zufälligen IDs auch die böswilligen Peers im ganzen Netzwerk gleichmässig verteilt. Werden Knoten als Remote Peers gewählt, die benachbart sind, muss kein Peer befürchten, die Wahrscheinlichkeit, dass diese Peers böswillig sind, sei viel höher als die Wahrscheinlichkeit von böswilligen Peers im ganzen System.

Abbildung 3.3 zeigt, wie ein Peer bei der Konfiguration seine Remote Peers in Erfahrung bringt und wie diese davon erfahren, dass sie den Account des Peers verwalten müssen. Der zu konfigurierende Knoten errechnet einen Hashwert seiner ID und verwendet diesen als Key für eine Nachricht, welche im Folgenden an den entsprechenden Root Node geroutet wird ①. Der Root Node erfährt aus der Nachricht den Redundanzgrad *r* des Accounting Mechanismus, welcher die Anzahl der verwendeten Remote Peers angibt. Der Root Node ermittelt anhand sei-

nes Leaf Sets seine *r* nächsten Nachbarn. Die so erhaltenen Remote Peers teilt er dem Peer mit ②. Anschliessend benachrichtigt er die Remote Peers darüber, dass sie für die Verwaltung seiner Account Balance zuständig sind ③. Alle Remote Peers legen daraufhin einen Account für den Peer an.

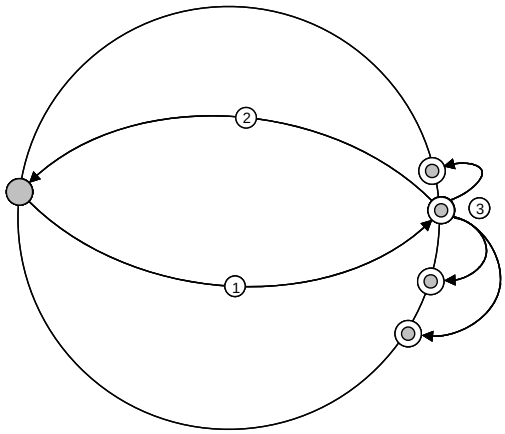

Abbildung 3.3: Konfiguration des Accounting Mechanismus

Damit das beschriebene Verfahren zur Ermittlung der Remote Peers funktioniert, muss der Root Node notwendigerweise auf die Anfrage des Peers antworten. Verweigert ein Root Node aus Böswilligkeit die gewünschte Auskunft, oder ist er offline, muss sich der Peer anderweitig nach seinen Remote Peers erkundigen. Dies könnte er bewerkstelligen, indem er noch einmal eine Nachricht an den Root Node schickt, wobei er diesmal sämtliche Knoten, welche im Laufe des Routings die Meldung erhalten, darum bittet, ihm ihr Leaf Set zu melden. Falls eines der Leaf Sets den Root Node enthält, so kennt der Peer nun mindestens einen der beiden direkten Nachbarn des Root Node. Diesen direkten Nachbarn kann er nun nach seinen Remote Peers befragen. Ist der Root Node nicht in dem erhaltenen Leaf Sets enthalten, so kann sich ein Peer über den Knoten, welcher am nächsten am Key liegt, nähere Informationen zu sein Remote Set beschaffen, indem sich der Peer das Leaf Set dieses Knotens zuschicken lässt. Zwangsläufig findet der Peer so ebenfalls einen direkten Nachbarn vom Root Node. Sollte die neue Anfrage an den Root Node Nachbarn wieder fehlschlagen, kann das Verfahren rekursiv wiederholt werden.

Im Weiteren soll ein Peer überprüfen können, ob ihm der Root Node die richtigen Remote Peers zurückliefert. Falls dies nicht gegeben wäre, hätte ein böswilliger Root Node die Chance, einem Peer eine Reihe von verbündeten Knoten als Remote Peers zurückzuliefern. Um solches zu unterbinden, kann ein Peer erst einmal die Distanzen zwischen den einzelnen Remote Peers betrachten. Unterscheiden sich diese nicht wesentlich von den Abständen der Knoten in seinem Leaf

Set, so sind die Angaben des Root Node mit grosser Wahrscheinlichkeit korrekt. Zeigen sich hingegen grosse Abweichungen, müssen bei einem Knoten, welcher mit dem Root Node benachbart ist, die Remote Peers zum Vergleich erneut angefordert werden. Dies kann mit der im vorherigen Abschnitt beschriebenen Methode realisiert werden.

Ein letztes Problem entsteht dann, wenn ein Peer dem Root Node nicht den richtigen Redundanzgrad mitteilt und dementsprechend eine andere Anzahl an Peers als Remote Peers zugeteilt bekommt. Hat ein Peer auf der einen Seite sehr wenige Remote Peers, so ist die Verlässlichkeit seines Account Balance sehr niedrig. Wenn ein Peer ein sehr grosses Remote Set hat, muss ein Peer für die Kommunikation mit den Remote Peers und die Remote Peers selbst für die Synchronisation untereinander sehr viele Nachrichten aufwenden. Aufgrund dieser beiden Tatsachen wird ersichtlich, dass die Peers daran gehindert werden müssen, den Redundanzgrad frei zu wählen. Dies kann nur dadurch garantiert werden, indem jeder Peer im System den gleichen Redundanzgrad erhält. In diesem Fall ist ein Root Node nicht auf den Erhalt der Redundanzgrades angewiesen, wenn sich ein entsprechender Peer konfiguriert und ihn um seine Remote Peers bittet.

3.3.2 Methode *createSession*

Die Methode *createSession* dient dazu, eine neue Accounting Session zu erstellen. Solche Sessionen werden in P2P Systemen natürlich immer auf die Initiative des Consumers erstellt, weshalb die Funktion auch nur auf diesem Peer aufgerufen wird. Dieser informiert dann den Provider selbstständig über die nötigen Schritte. Als Parameter werden dem Accounting Pattern die Art und der Tarif des während der Session benutzten Service, sowie die ID des Sessionspartners übergeben.

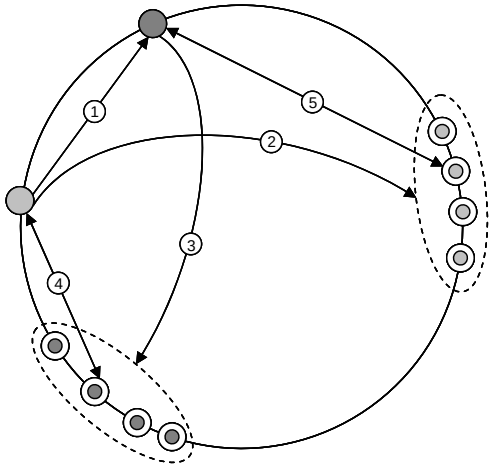

Abbildung 3.4: Kreieren einer Session

In Abbildung 3.4 ist zu erkennen, welche Schritte für das Erzeugen einer Session nötig sind. Bevor der Consumer den Provider über die neue Session informiert, erstellt er zuerst eine eindeutige Session ID. Diese kann erstellt werden, indem die ID des Consumers und des Providers, sowie eine Systemzeit des Rechners, auf dem der Consumer läuft, zusammengehängt werden. Neben den weiteren notwendigen Angaben zur Session teilt er diese dem Provider mittels einer Nachricht mit ①. Des Weiteren müssen nun die Remote Peers über die Session ins Bild gesetzt werden. Beide Sessionspartner schicken eine entsprechende Meldung an ihre Remote Peers ②, ③. Schliesslich müssen der Consumer und der Provider die Remote Peers des jeweiligen Partners, gemäss der im vorherigen Abschnitt beschriebenen Methode in Erfahrung bringen. Vorausgesetzt natürlich, diese sind ihnen noch nicht bekannt. Die Remote Peers werden zu einem späteren Zeitpunkt zur Aktualisierung des fremden Accounts gebraucht.

3.3.3 Methode *updateAccountBalance*

Nachdem der Provider dem Consumer seine Ressourcen erfolgreich zur Verfügung gestellt hat, ist es nötig, die Account Balance der Peers zu verringern, respektive zu erhöhen. Dies geschieht mittels der *updateAccountBalance* Methode, die nun bei beiden Sessionpartnern aufgerufen werden kann. Dabei wird dem Pattern mitgeteilt, im Rahmen welcher Session das Update durchzuführen ist und in welchem Umfang der Consumer Ressourcen konsumiert hat.

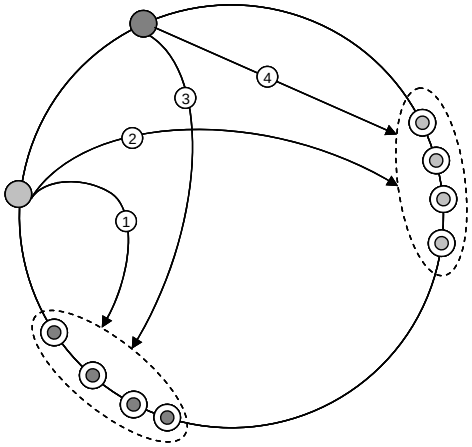

Abbildung 3.5: Aktualisieren der Account Balance

Wie die Account Balance des Consumers und des Providers im System aktualisiert wird, ist in Abbildung 3.5 zu sehen. Dabei schicken die beiden Peers eine Update Nachricht sowohl an alle ihre eigenen Remote Peeers ②, ③, als auch an die Remote Peers ihres Sessionpartners ①, ④. Anhand der Session ID, welche die Meldung enthält, finden die Remote Peers die zuvor gespeicherten Informationen

zur Session. Mit Hilfe dieser Daten und dem in der Nachricht mitgelieferten Hinweis darüber, in welchem Ausmass der Consumer Services vom Provider bezogen hat, können die Remote Peers schliesslich die Account Balance der jeweiligen Peers aktualisieren. Indem die Remote Peers die Update Nachricht vom Consumer und vom Provider erhalten, ist es böswilligen Peers nicht möglich, eine ungerechtfertigte Erhöhung oder Verminderung der eigenen oder einer fremden Account Balance zu erwirken.

Beim Update einer Account Balance kann dann ein Problem auftauchen, wenn zwei Peers gemeinsam eine Session generieren und während deren Verlauf nur die Remote Peers des Providers um ein Update der Account Balance bitten, nicht aber die des Consumers. Wiederholen die Peers diesen Vorgang, wobei sie abwechslungsweise Consumer oder Provider sind, können sie ihre Account Balance beliebig erhöhen. Um das zu verhindern, muss ein zusätzlicher Synchronisationsschritt zwischen den Remote Peers des Consumers und des Providers stattfinden. Indem sich die Peers der beiden Remote Sets gegenseitig die Korrektheit des Updates versichern, führen sie das Update auch aus und können damit das oben beschriebene Problem vermeiden.

3.3.4 Methode *queryAccountBalance*

Die *queryAccountBalance* Methode ist die letzte, vom Accounting Pattern angebotene Methode. Über sie kann man die Account Balance eines beliebigen Knotens im P2P Netzwerk ermitteln. Die Funktion erwartet als Parameter lediglich die ID des Knotens, dessen Account Balance abgefragt werden soll.

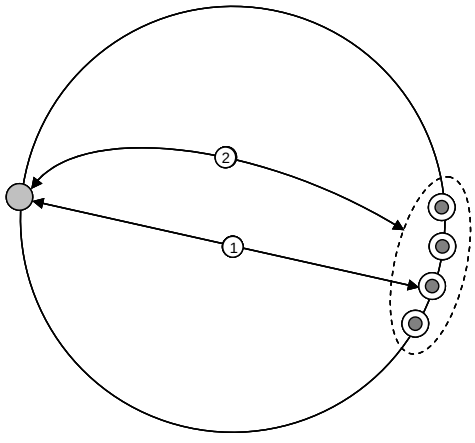

Abbildung 3.6: Abfrage der Account Balance

In Abbildung 3.6 ist der Vorgang bei der Abfrage einer Account Balance zu sehen. Zuerst vergewissert sich der Peer, ob ihm die jeweiligen Remote Peers, wel-

che er für die Abfrage benötigt bekannt und aktuell sind. Ist dies nicht der Fall, so kann er sie gemäss der im Abschnitt 3.1.1 beschriebenen Methode in Erfahrung bringen ①. Die Aktualität der Remote Peers wird im Wesentlichen davon bestimmt, wann er diese in Erfahrung gebracht hat. Denn je mehr Zeit seit der Anfrage nach dem Remote Set verstrichen ist, je wahrscheinlicher ist es, dass sich die Remote Peers inzwischen verändert haben. Mit der Definition eines Thresholds, welcher angibt, wie lange die angefragten Remote Peers gültig bleiben, kann die Aktualität der Remote Peers überprüft werden. Als nächstes schickt der Peer eine Nachricht an sämtliche ermittelten Remote Peers und bittet sie darum, ihm die Account Balance des entsprechenden Peers mitzuteilen ②.

$$\text{Threshold} > \frac{\text{Anzahl erhaltene Nachrichten}}{\text{Anzahl erwartete Nachrichten}} \cdot \frac{\text{Anzahl der Werte, die dem Mehrheitsentscheid entsprechen}}{\text{Anzahl erhaltene Werte}} \rightarrow \text{Wert glaubwürdig}$$

Abbildung 3.7: Überprüfung der Glaubwürdigkeit einer Account Balance

Anhand der erhaltenen Werte rekonstruiert sich der Peer die gesuchte Account Balance und ermittelt dessen Glaubwürdigkeit. Ersteres geschieht mit einem einfachen Mehrheitsentscheid. Wie der Peer überprüft, ob er den berechneten Wert für die Account Balance akzeptiert ist in Abbildung 3.7 dargestellt. Für die Berechnung der Glaubwürdigkeit stellt er zum einen fest, wie gross das Verhältnis zwischen den, von den Remote Peers tatsächlich zurück erhaltenen Nachrichten und den eigentlich erwarteten Rückmeldungen ist. Zum anderen berechnet er, wie viel Prozent der erhaltenen Werte identisch zu der von ihm berechneten Account Balance sind. Indem der Peer die beiden Prozentsätze miteinander multipliziert, erhält er ein Mass für die Glaubwürdigkeit der errechneten Account Balances, welche er mit dem Threshold vergleichen kann.

3.4 Verlässlichkeit des Mechanismus

Neben der Anforderung der Skalierbarkeit und Verfügbarkeit, muss der entwickelte Accounting Mechanismus vor allem verlässlich sein. Das Verfahren muss mit einer möglichst hohen Wahrscheinlichkeit garantieren können, dass die korrekte Account Balance der einzelnen Peers jederzeit in Erfahrung gebracht werden kann. Dafür sorgt in erster Linie die redundante Verwaltung der Accounts auf verschiedenen Remote Peers. Damit dieses Konzept allerdings überhaupt greifen kann, dürfen die einzelnen Komponenten des Accounting Mechanismen weder manipuliert, noch missbraucht werden können. Es braucht daher geeignete Verfahren, welche das ausschliessen können. Solche Verfahren, welche die spezifischen Probleme bei der Umsetzung der einzelnen Methoden der Pattern Schnittstelle behandeln, wurden in den vorherigen Abschnitten bereits vorgestellt. Es folgen daher hier nur noch übergreifende Betrachtungen zur Kommunikation zwischen einzelnen Peers und zur Sicherheit von Pastry.

Damit der Accounting Mechanismus wie gewünscht funktioniert, ist ein authentischer Austausch von Nachrichten nötig. Ist dies nicht der Fall, wäre es für einen Angreifer leicht, sich als anderen Peer auszugeben und seine Remote Peers beispielsweise zur Erhöhung seiner Account Balance zu bewegen. Die gewünschte authentische Kommunikation kann mit Hilfe von Public Key Kryptographie erreicht werden. Jeder Knoten im Netzwerk braucht dazu einen geheimen und einen, dazu passenden, öffentlichen Schlüssel. Verschickt ein Peer eine Nachricht, kann er diese mit seinem Private Key verschlüsseln. Die Meldung kann dann in der Folge nur mit dem, zum Peer passenden Public Key schnell und effizient wieder entschlüsselt werden. Um garantieren zu können, dass alle Peers sicher zu den richtigen Public Keys der anderen Peers im System gelangen, können beispielsweise eine Reihe von Certification Authorities verwendet werden. Diese haben typischerweise im System eine hohe Vertrauenswürdigkeit und stellen die Public Keys von entsprechenden Peers zur Verfügung.

Eine weitere Gefahr für das Accounting Verfahren stellen Replay Attacken dar. Fängt ein böswilliger Peer zum Beispiel Update Nachrichten von anderen Peers ab, so kann er diese zu einem späteren Zeitpunkt noch einmal an die Remote Peers schicken. Falls die Remote Peers die Nachrichten nicht wieder erkennen, führen sie das entsprechende Update ein weiteres Mal durch. Replay Attacken können relativ einfach verhindert werden, indem jede verschlüsselte Nachricht zusätzlich eine Sequenznummer enthält. Anhand der Sequenznummer können dann allfällige duplizierte Meldungen aussortiert werden.

Die heute gebräuchlichen Overlay Netzwerke wie Pastry sind grundsätzlich nicht sicher. Bereits eine relativ kleine Anzahl an böswilligen Peers kann die korrekte Übertragung von Nachrichten unterbinden. In [12] werden die vorhandenen Sicherheitslücken in Overlay Netzwerken diskutiert und Lösungen präsentiert, welche diese schliessen. Im Detail wird anhand von Pastry gezeigt, wie IDs von Knoten sicher vergeben werden können, wie eine sichere Wartung der Routing Tabellen gewährleistet und eine sichere Weiterleitung der Nachrichten während des Routing Vorgangs garantiert werden kann.

3.5 Synchronisation

Aufgrund der Dynamik eines Peer-to-Peer Systems, sind die Peers im Accounting Pattern dazu gezwungen, sich untereinander zu synchronisieren. Dies muss dann geschehen, wenn ein Peer, welcher zuvor offline war, wieder online kommt, oder wenn neue Knoten zum P2P Netzwerk hinzukommen oder dieses verlassen. Die Peers müssen darüber informiert sein, welches ihre Remote Peers sind. Sie müssen weiter wissen, welchen Knoten sie als Remote Peers dienen. Schliesslich müssen sie sich, wenn nötig, nach den neuen Account Balances der von ihnen verwalteten Accounts und nach allfälligen, von ihnen betreuten, neuen Sessions

erkundigen. Wie dies alles realisiert werden kann, soll im Weiteren erklärt werden.

Erhält das Peer-to-Peer Netzwerk einen neuen Knoten oder verlässt ein Peer das System, so werden in Pastry sämtliche Peers, deren Leaf Set sich aufgrund der Modifikation verändert hat, über die Neuigkeit informiert. Dies ist für die Synchronisation der Peers sehr nützlich.

Hat sich ein neuer Knoten ans P2P Netz angeschlossen, können die benachrichtigten Peers aufgrund der Mitteilung von Pastry überprüfen, ob es Änderungen in den, von ihnen gespeicherten und verwendeten Remote Sets gibt. Die Voraussetzung dafür ist allerdings, dass ein Remote Set weniger Peers beinhaltet, als ein Leaf Set. Ist dies gegeben, lässt sich die erwähnte Überprüfung mit den Peer IDs und den gespeicherten Remote Sets der verwalteten Accounts, sowie der ID des neu zum System hinzu gestossenen Knotens einfach bewerkstelligen. Dank dieser Werte kann der Peer nämlich errechnen, ob er im Gegensatz zum neuen Peer und den bisherigen Remote Peers, weiter vom Hash des Knotens, dessen Account er verwaltet, entfernt liegt. Trifft dies zu, ist er nicht mehr Remote Peer des Knotens und kann dessen Account löschen. Bleibt er hingegen Remote Peer und ist der neue Knoten nun Mitglied des jeweiligen Remote Sets, so teilt er dem Knoten die neuen Remote Peers mit. Befindet sich der benachrichtigte Peer zurzeit in einer Session mit einem anderen Peer, muss er diesem die Neuerung natürlich mitteilen.

Neben der Synchronisation der bisherigen Knoten des Netzwerkes, muss auch der neu zum System hinzugekommene Peer herausfinden, in welchen Remote Sets er sich befindet und wessen Accounts er damit zu verwalten hat. Er bittet deshalb seine beiden unmittelbaren Nachbarknoten, ihm diese Information mitzuteilen.

Verlässt ein Knoten das P2P Netzwerk, verändern sich die Remote Sets erneut. Im Gegensatz zum vorherigen Szenario muss jedoch nicht herausgefunden werden, welche Knoten nicht mehr Remote Peers bleiben, sondern es muss ermittelt werden, welche Peers nun weitere Accounts zu verwalten haben. Die Lücke, welcher der weggehende Knoten in verschiedenen Remote Sets hinterlässt, muss wieder gefüllt werden. Die Aufgabe, das zu bewerkstelligen, wird allen Peers übertragen, deren Remote Sets nun nicht mehr komplett sind. Von Pastry erfahren nämlich alle vom Weggang des Knotens betroffenen Peers über die allfällige Änderung ihres Leaf Sets. Anhand der erhaltenen Information können die Knoten überprüfen, ob der verschwundene Knoten in einem ihrer Remote Sets gewesen ist. Wenn dies zutrifft, dann muss das jeweilige Remote Set aktualisiert werden, indem ihm ein neuer Knoten hinzugefügt wird. Für das Finden eines solchen Knotens ist nun der entsprechende Root Node zuständig, der den im Remote Set verbliebenen Knoten den neuen Remote Peer liefert. Wie beim Hinzufügen eines

Knotens zum Netzwerk, informieren die Knoten des neu konstruierten Remote Sets den jeweiligen Knoten über seine neuen Remote Peers. Der wiederum leitet sie an seine Sessionpartner weiter. Auch diesem Szenario liegt eine bestimmte Annahme zu Grunde. Ist nämlich die Anzahl der Peers in einem Remote Set grösser als die halbe Anzahl der Knoten im Leaf Set, bemerken anhand ihres Leaf Sets nicht alle Peers, welche zu aktualisierenden Remote Sets angehören, dass eines ihrer Remote Sets aktualisiert werden muss. Folgendes Beispiel soll dies deutlich machen. Gegeben seien zwei Peers A und B, welche in einem Remote Set mit vier weiteren Peers liegen und in diesem die Knoten sind, welche am weitesten voneinander entfernt sind. Weiter nimmt man an, die Leaf Sets würden acht Peers enthalten. Verlässt der Peer B nun das Netz, so bemerkt dies der Peer A nicht, obwohl er in einem Remote Set liegt, welches aktualisiert werden muss. Pastry benachrichtigt A nicht über den Weggang von B, da der Knoten nicht im Leaf Set von A liegt.

Kommt ein Peer wieder online, so versucht er herauszufinden, ob sich die Account Balance der von ihm betreuten Accounts verändert haben und ob die entsprechenden Peers an neuen Sessions teilnehmen. Er schickt an alle Peers, welche ebenfalls Remote Peer eines, von ihm gehaltenen Accounts sind, eine Nachricht. Diese Meldung enthält einerseits die Peer ID und das ihm bekannte Remote Set des zu überprüfenden Accounts. Andererseits sind in der Message die, zum Account gehörenden Sessions aufgelistet. Hat sich während der Abwesenheit des Peers das Remote Set nicht geändert, so erhält er von den eben befragten Knoten die gewünschte, aktuelle Account Balance und die angeforderten Sessions. Hat jedoch ein Knoten während der Abwesenheit des Peers das Netzwerk verlassen oder ist ein neuer zum System hinzu gestossen, so muss der Peer erst einmal das entsprechende Remote Set bereinigen. Zu diesem Zweck senden ihm die gerade kontaktierten Knoten, sofern sie noch Mitglied in diesem sind, das aktuelle Remote Set. Der Peer kontrolliert, ob er in dem erhaltenen Remote Sets noch enthalten ist. Wenn dies nicht der Fall ist, kann er den jeweiligen Account löschen. Ist er jedoch nach wie vor für den Account zuständig und stimmen die erhaltenen Remote Sets überein, versucht er sich bei den Knoten des neuen Remote Sets auf die gleiche Art und Weise noch einmal zu synchronisieren.

3.6 Designalternativen bezüglich der Infrastruktur

In Abschnitt 3.2 wurde Pastry [8] als die P2P Infrastruktur vorgestellt, welche dem zu implementierenden Distributed Accounting Mechanismus zu Grunde liegt. Neben Pastry gibt es eine Reihe weiterer Overlay Netzwerke, wie Tapestry [9], CAN [10] oder Chord [11], die ebenfalls als Infrastrukturen hätten verwendet werden können. Ebenso wäre JXTA [7] für die Implementierung eine Alternative gewesen. Im Weiteren soll nun JXTA etwas detaillierter vorgestellt werden.

Schliesslich sollen die Vor- und Nachteile von Pastry und JXTA aufgezeigt werden und die Wahl von Pastry als P2P Infrastruktur für die Implementierung des Distributed Accounting Mechanismus begründet werden.

3.6.1 JXTA

JXTA ist ein P2P Framework und wurde ursprünglich von Sun Microsystems Inc. entwickelt. Ziel war es, eine Infrastruktur zu schaffen, welche die Interoperabilität der Peers unabhängig von ihren verwendeten P2P Applikationen gewährleisten kann, die Plattform unabhängig ist und die auf sämtlichen digitalen Geräten implementiert werden kann, seien sie auch noch so klein.

Neben grundsätzlichen Funktionen für die Kommunikation zwischen den Peers, bietet das Framework auch ganze Peer-to-Peer Applikationen an. Das P2P Netzwerk von JXTA basiert auf Peer Groups, die sich aus einer bestimmten Anzahl an Peers zusammensetzen. Die beteiligten Peers haben sich dabei auf eine Menge von P2P Services geeinigt, die sie sich gegenseitig anbieten. Jede Peer Group hat einen oder mehrere Rendezvous Peers. Die Rendezvous Peers ermöglichen es einem Peer sich an eine Peer Group anzugliedern. Ebenso nehmen sie bei der Lokalisierung und Suche nach bestimmten Ressourcen, sowie bei der Kommunikation zwischen Peers aus verschiedenen Peer Groups eine zentrale Rolle ein.

3.6.2 Auswahl der Infrastruktur

JXTA ist ein sehr breit abgestütztes Framework, das in unzähligen Peer-to-Peer Anwendungen verwendet wird. Es ist sehr umfangreich und bietet eine Unmenge von Funktionen und Möglichkeiten. Nur ein Bruchteil davon kann für die Implementierung des Accounting Patterns verwendet werden. Pastry ist momentan noch nicht so verbreitet wie JXTA und hat keine so grosse Lobby wie das Projekt von Sun. Gleichwohl existieren schon diverse Peer-to-Peer Applikationen, die auf Pastry basieren. Ebenso glänzt die Infrastruktur mit einer einfachen Schnittstelle. Der entscheidende Vorteil von Pastry liegt allerdings in der Struktur seines Overlay Netzwerkes. Diese vereinfacht beim Remote Accounting die Definition einer Mapping Rule, durch welche die Remote Peers eines Knotens bestimmt werden, entscheidend. Der Ansatz von JXTA mit den verwendeten Peer Groups und Rendezvous Peers macht das Finden einer guten Mapping Rule schwierig. Aus diesem Grund wurde Pastry als zugrunde liegende Infrastruktur für die Implementation des Remote Accounting Pattern ausgewählt.

Kapitel 4

Implementation

Dieses Kapitel zeigt die Implementation des, im vorherigen Kapitel beschriebenen Prototyps. Um die bereits erwähnten Vorzüge von Pastry als P2P Infrastruktur nutzen zu können, wurde dazu die Open Source Software FreePastry verwendet. Über eine einfache Schnittstelle können sowohl das Remote Accounting Pattern und der darin umgesetzte Distributed Accounting Mechanismus, als auch die Simulationskomponente auf die zur Verfügung gestellten Funktionalitäten zugreifen.

Im weiteren Verlauf des Kapitels wird zuerst einmal das API von FreePastry erläutert und erklärt, wie der Prototyp sich dieses zu Nutzen macht. Der Abschnitt 4.2 zeigt die Realisierung des Remote Accounting Pattern. Gezeigt ist dabei, welche vereinfachenden Annahmen, betreffend dem zu simulierenden P2P System gemacht wurden.

4.1 FreePastry API

Wie der Zugriff der beiden Komponenten des Prototyps auf das API von FreePastry gewährleistet wird, ist in Abbildung 4.1 zu sehen. Die Schnittstelle von Pastry ist dabei definiert durch diverse Interfaces und eine Testklasse *DistCommonAPITest*. Mit der Methode *start* der Testklasse kann ein Pastry Netzwerk mit einer bestimmten Anzahl an Peers erzeugt werden. Wird die Funktion *start* aufgerufen, so erstellt FreePastry intern als erstes einen Pastry Knoten vom Typ *Node* und fügt ihm dem Overlay Netzwerk hinzu. Bevor ein Testvorgang mit dem Aufruf der Methode *runTest* gestartet wird, wird für jeden neu kreierten Knoten die Funktion *processNode* ausgeführt, welcher der Pastry Knoten als Parameter übergeben wird. Die Methoden *runTest* und *processNode* sind abstrakte Methoden,

welche folglich von einer Subklasse der Klasse *DistCommonAPITest* implementiert werden müssen. Die Klasse *RemoteAccountingTest* ist eine solche Subklasse, von der genau eine Instanz existiert, welche als Simulationskomponente des Prototyps eingesetzt wird. Sobald auf der beschriebenen Instanz die Methode *processNode* aufgerufen wird, sorgt die Simulationskomponente dafür, dass der entsprechende Pastry Knoten einem Accounting Pattern zugewiesen wird und damit als Remote Accounting Knoten im Peer-to-Peer System agieren kann. Die Funktionalitäten des Remote Accounting Pattern sind in der Klasse *RemoteAccountingPeer* enthalten, weshalb das Simulationsobjekt nun eine Instanz dieser Klasse kreiert. Das Objekt speichert während ihrer Entstehung den Pastry Knoten in ihrem Feld *myPastryNode* ab. Um als Applikation auf Pastry agieren zu können und als solche von den Änderungen und Ereignissen im Overlay Netzwerk zu erfahren, muss die Klasse des Objekts das Interface *Application* implementieren. Ebenso muss der jeweilige Pastry Knoten natürlich über die auf ihm aufsetzende Anwendung informiert werden, was diese über die Methode *registerApplication* bewerkstelligen kann. Als Resultat erhält die Instanz ein Objekt vom Typ *Endpoint*, über das sie in der Folge Pastry spezifische Funktionen ausführen kann und das sie in ihrem Feld *endpoint* speichert.

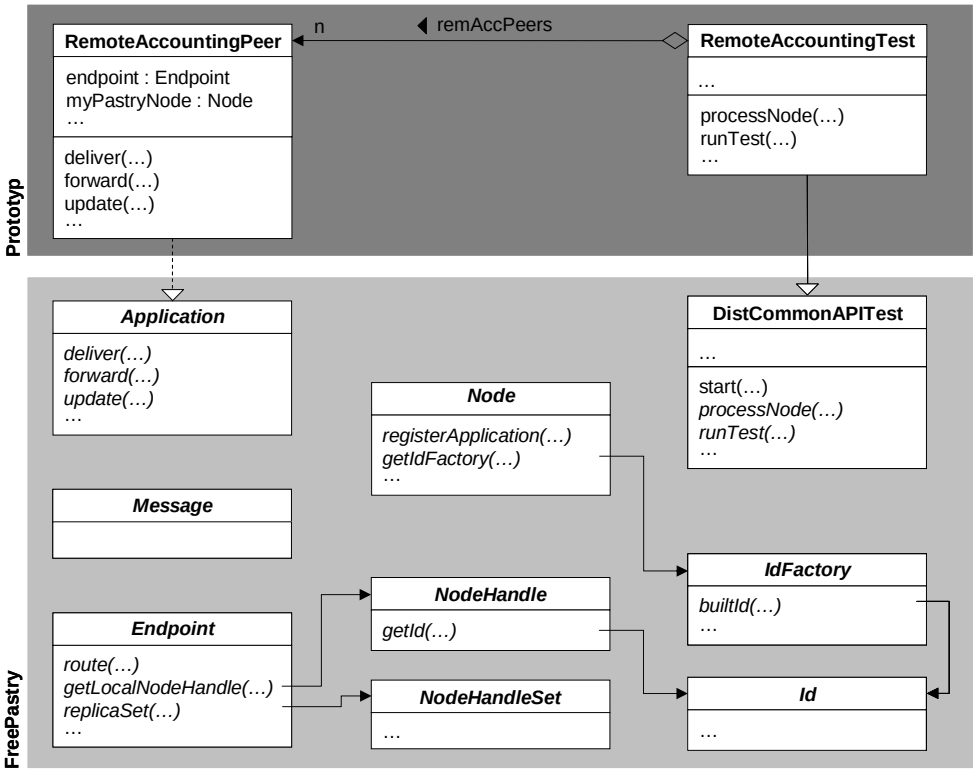

Abbildung 4.1: Schnittstelle von FreePastry

Während des eigentlichen Simulationsvorgangs, bei dem der Distributed Accounting Mechanismus getestet wird, interagieren der Prototyp und FreePastry weiter

miteinander. Dabei bedient sich FreePastry den Methoden *deliver*, *forward* und *update*, welche die Klasse *RemoteAccountingPeer* zu implementieren hat, da sie das Interface *Application* verwendet. Die Funktion *deliver* ruft FreePastry auf dem Objekt des Prototyps auf, wenn eine Nachricht über das Overlay Netzwerk zum Objekt des Prototyps, respektive dessen Pastry Knoten gelangt ist. Die Methode *forward* führt FreePastry aus, wenn eine Nachricht beim Accounting Peer eingetroffen ist, welche weitergeroutet werden muss. Der Peer kann entscheiden, ob, aber nicht wie die Nachricht weitergeleitet wird. Die Funktion *update* wird schliesslich abgearbeitet, wenn sich das Leaf Set des eigenen Pastry Knotens verändert hat.

Im Gegenzug stellt FreePastry dem Prototyp diverse Methoden zur Verfügung, die für die Durchführung des Distributed Accounting Mechanismus nötig sind und welche über die beiden Felder *endpoint* und *myPastryNode* von den Accounting Peers benutzt werden können. Über das, im Feld *endpoint* gespeicherte Objekt kann die Methode *route* aufgerufen werden, welche das routen von Nachrichten über das Pastry Netzwerk ermöglicht. Dazu muss der Methode als Parameter ein Key vom Typ *Id*, welcher den Zielort der Nachricht angibt und eine Nachricht vom Typ *Message*, welche nur serialisierbare Objekte beinhalten darf, übergeben werden. Optional kann der Funktion der Knoten mitgeteilt werden, repräsentiert durch ein Objekt vom Typ *NodeHandle*, zu welchem der erste Hop im Routing Prozess führen soll. Kennt man den Node Handle eines Peers kann man dank diesem optionalen Parameter eine Message direkt an den gewünschten Knoten versenden. Den eigenen Node Handle kann ein Accounting Peer über die Methode *getLocalNodeHandle* herausfinden, die ihm abermals das Interface *Endpoint* anbietet. Im Gegensatz zu seinem Pastry Endpoint kann ein Peer seinen Node Handle an andere Peers versenden, da Objekte vom Typ *NodeHandle* serialisierbar sind, solche vom Typ *Endpoint* jedoch nicht. Eine weitere, für das Accounting Verfahren wichtige Funktion ist *replicaSet*, welche das Finden der Remote Peers während der Konfiguration des Accounting Pattern sehr einfach macht. Ein nach den Remote Peers befragter Knoten kann diese Methode über seinen Pastry Endpoint aufrufen und diese liefert ihm die gewünschte Anzahl an nächsten Nachbarn. An ein letztes nützliches Feature von FreePastry gelangt ein Remote Accounting Peer über den, im Feld *myPastryNode* abgespeicherten Pastry Knoten. Er bietet eine Methode *getIdFactory* an, über welche man an ein Objekt vom Typ *IdFactory* kommt. Das erhaltene Objekt stellt weiter eine Methode *buildId* zur Verfügung, welche einen Hashwert zur ID eines beliebigen Knotens berechnet und diesen als Objekt vom Typ *Id* zurückliefert.

4.2 Remote Accounting Peers

Durch das zuordnen einer Instanz der Klasse *RemoteAccountingPeer* zu einem Pastry Knoten, entstehen Peers, welche es P2P Anwendungen ermöglichen, vom

Distributed Accounting Mechanismus Gebrauch zu machen. Wie bereits im Kapitel 3 beschrieben, ist die Verwendung und Konfiguration des Mechanismus durch eine einfache Schnittstelle, welche die einzelnen Peers anbieten, durch die erwähnten Applikationen möglich. Alle notwendigen, weiteren Schritte leiten die Knoten selbstständig in die Wege.

4.2.1 Steuerung des Simulationsablaufs

Da zwischen den einzelnen Funktionen des Accounting API Abhängigkeiten bestehen, können die Remote Accounting Peers die Methoden nicht einzeln für sich betrachten, sondern müssen deren Ausführung aufeinander abstimmen. Die Aufgabe der einzelnen Peers ist es, den von der Simulationskomponente generierten Simulationsablauf umzusetzen. Die Aufgaben der Simulationskomponente werden in Kapitel 5 noch einmal detaillierter beschrieben. Der Aufbau der Komponente wird hier erwähnt, um erklären zu können, welchen Input die einzelnen Peers, welche den Distributed Accounting Mechanismus umsetzen erhalten. Jeder Peer hat eine Action Queue, in welche die Simulationskomponente die jeweils auszuführenden Aktionen einfügt. Diese Aktionen sorgen schliesslich für den Aufruf von einzelnen Methoden der, zur Verfügung gestellten Schnittstelle des Accounting Patterns. Die Ausführung der vier API Methoden *configure*, *createSession*, *updateAccountBalance* und *queryAccountBalance* können dabei nicht isoliert voneinander betrachtet werden. Damit die an einer Session beteiligten Peers ihre Account Balance mit der Methode *updateAccountBalance* aktualisieren können, müssen die beiden Peers die Remote Peers des Sessionpartners kennen. Diese Information besorgen sie sich bei der Erstellung der Session, welche durch die Funktion *createSession* in die Wege geleitet wird. Das Kreieren von Sessions kann wiederum erst dann erfolgen, wenn ein Peer seine Remote Peers kennt und sie über die neue Session informieren kann. Das in Erfahrung bringen der Remote Peers ist Bestandteil der Konfiguration der Accounting Peers, welche mit der Methode *configure* in die Wege geleitet wird. Die beschriebenen Abhängigkeiten führen dazu, dass ein Peer die Abarbeitung der erhaltenen Aktionen koordinieren muss. Im Übrigen kann es bei der Beschaffung der nötigen Informationen von anderen Peers passieren, dass ein Peer die gewünschten Daten nicht erhält, weil der angefragte Peer zum Zeitpunkt des Eintreffens der Nachricht offline war oder er absichtlich keine Antwort auf die Anfrage gibt. Damit der Ablauf der Simulation in einem solchen Fall gewährleistet bleibt, muss ein Peer mit geeigneten Massnahmen auf das Problem darauf reagieren können.

In Abbildung 4.2 ist zu sehen wie die nötigen Steuerungsmechanismen im Prototyp umgesetzt sind. Erhält ein Accounting Peer eine Aktion von der Simulationskomponente, so überprüft er, ob andere Aktionen Informationen wie beispielsweise das Wissen darüber, welche Remote Peers ein Peer hat benötigen, welche von der neuen Aktion in Erfahrung gebracht werden. Dies trifft zu, wenn die Ak-

tion für die Konfiguration des Accounting Peers oder das Erstellen einer neuen Session zuständig ist. Wenn eine solche Aktion ermittelt werden kann, wird ihr ein Controller zugeordnet, der Auskunft darüber gibt, ob dem Peer die zu beschaffenden Daten bereits vorliegen. Weiter wird allen Aktionen, welche auf die Daten von anderen Aktionen angewiesen sind, mitgeteilt, welche Controller sie zu überwachen haben. Sind alle nötigen Informationen für eine Aktion vorhanden, so starten die Aktionen ihre Ausführung selbst, indem sie die entsprechende Methode des Pattern API aufrufen. Erhält ein Peer angeforderte Informationen von anderen Peers, so benachrichtigt das Message Memory die nötigen Controller darüber. Damit dies möglich ist, speichert der Peer jede verschickte Nachricht im Message Memory und weist ihr den passenden Controller zu. Trifft die erhoffte Antwort zu einem späteren Zeitpunkt beim Peer ein, so initiiert das Message Memory die nötige Aktualisierung des entsprechenden Controllers. Falls die gewünschte Nachricht nicht beim Peer eintrifft, so löst ein Timer Task nach einer bestimmten Zeit ein Timeout aus. Das Message Memory entscheidet in einem solchen Fall, was zu tun ist, wobei es im Wesentlichen zwei Möglichkeiten hat. Entweder die jeweiligen Informationen werden noch einmal angefordert, oder aber die Aktion und alle, von ihrer Ausführung abhängenden Aktionen werden abgebrochen und die Simulationskomponente wird über diesen Schritt informiert.

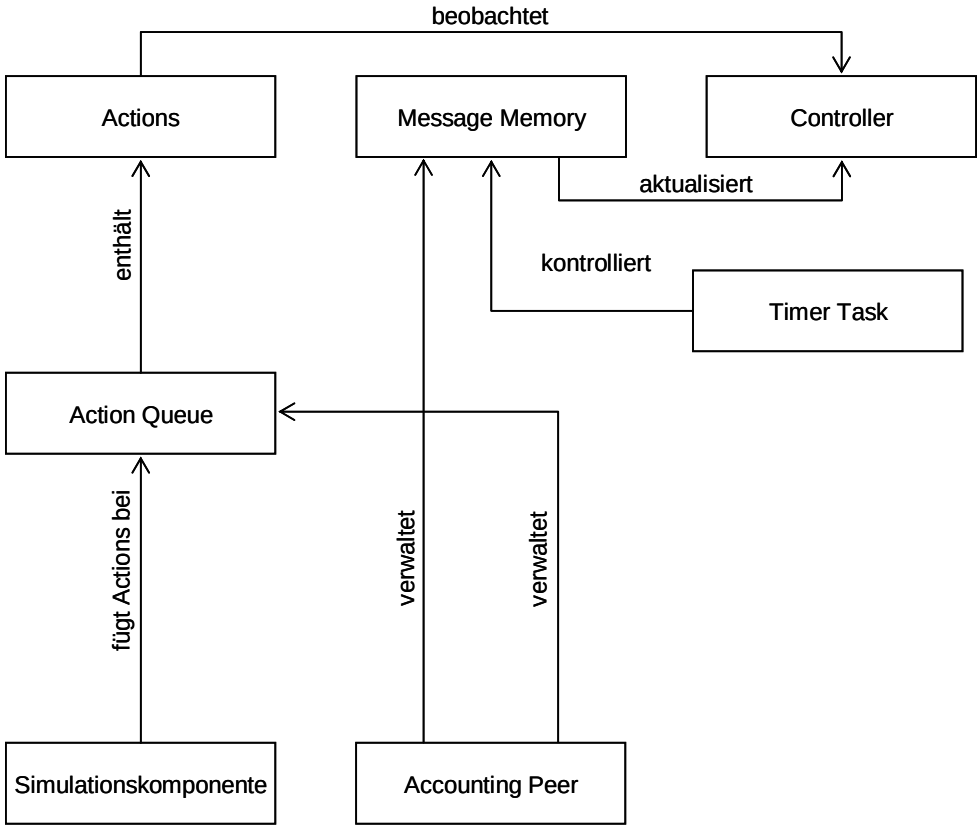

Abbildung 4.2: Steuerung des Simulationsablaufs

4.2.2 Vereinfachende Annahmen zum P2P System

Bei der Umsetzung des Prototyps agieren die Knoten im Peer-to-Peer Netzwerk nicht exakt so, wie dies in einem echten P2P System der Fall wäre. Diese Annahmen wurden getroffen, um in der zur Verfügung stehenden Zeit einen funktionierenden Accounting Mechanismus implementieren zu können, da sich dadurch die Synchronisation zwischen den einzelnen Peers vereinfacht, sondern auch der Simulationsablaufs. Folglich wurden nicht alle der Synchronisationsmassnahmen, welche in Abschnitt 3.5 beschriebenen wurden, bei der Implementation umgesetzt.

Die Implementierung der Simulationskomponente basiert, wie bereits erklärt, auf der Testklasse *DistCommonAPI* von FreePastry. Da diese Klasse keine Funktionen zur Verfügung stellt, mit denen neue Knoten zum Pastry Netzwerk hinzugefügt werden können oder mit denen einzelne Peers das System während der Simulation verlassen können, hat das P2P Netzwerk im Prototypen eine statische Struktur. Deshalb bleiben während dem gesamten Evaluationsprozess die Remote Sets der einzelnen Peers unverändert. Ebenso werden gewisse Einschränkungen an die Böswilligkeit und die Peer Uptime eines Peers gemacht. Während der Konfiguration und dem Erstellen von neuen Sessions agieren die Peers weder böswillig, noch sind sie während der Ausführung dieser Aktionen jemals offline. Dadurch wird den Remote Peers mit Sicherheit mitgeteilt, welche Account Balance sie zu verwalten haben und sie erhalten auf jeden Fall die Information zu neuen Sessions, die sie betreffen. Der Peer, der die erwähnten beiden Aktionen ausführt, kann zudem davon ausgehen, dass er mit Sicherheit erfährt, wer seine eigenen Remote Peers sind, respektive welches das Remote Set eines allfälligen Sesssionpartners ist. Die Böswilligkeit der Peers beschränkt sich nur auf das Ignorieren einer Update Nachricht für eine verwaltete Account Balance. Auch ein allfälliger Offline Zyklus des Peers wirkt sich nur auf das Aktualisieren und das Abfragen eines Accounts aus. Ein Peer muss sich darum mit den entsprechenden Remote Peers synchronisieren, um die neuen Werte für seine gehaltenen Accounts zu erfahren. Gleichwohl erhält ein Peer nicht immer eine Antwort, wenn er einen Remote Peer um eine Account Balance bittet. In einem realen Peer-to-Peer System würden die Peers, welche offline sind, natürlich auch alle Nachrichten nicht erhalten, welche aufgrund der Konfiguration des Accounting Mechanismus oder beim Erstellen von neuen Sessions verschickt werden.

4.2.3 Konfiguration der Peers

Im Folgenden wird die Umsetzung der Konfiguration des Remote Accounting Peers beschrieben, die in Abschnitt 3.3.1 bereits schematisch erklärt wurde. Das Verfahren wird durch einen Aufruf der API Methode *configure* gestartet, welcher als Parameter den Redundanzgrad des Distributed Accounting Mechanismus, der

Quotient für die Peer Uptime und einen boolschen Wert übergeben, welcher angibt ob der Peer böswillig ist oder nicht.

Abbildung 4.3 zeigt den Ablauf der Funktion. Mittels Zugriff auf die Methode *buildId* des Pastry Knotens errechnet sich der Peer zuerst einen Hash für seine eigene ID. Dieser bestimmt die Position seiner Remote Peers im Overlay Netzwerk. Bevor der Peer eine Nachricht an den Root Node seines Remote Sets schickt, speichert er die Message im Message Memory und weist ihr den Controller der Konfigurationsaktion zu. Das geschieht im Hinblick auf eine erwartete Antwort des Root Node, in welcher dem Peer die eigenen Remote Peers mitgeteilt werden. Jede Message, welche ein Peer verschickt, hat einen Typ, anhand dem der jeweilige Empfänger weiss, was er zu tun, respektive welche Methode er aufzurufen hat. Der Einfachheit halber sind der Name der auszuführenden Funktion und der Typ der ankommenden Nachricht identisch. Für die Konfiguartion verschickt der Peer eine Message vom Typ *initRemoteAccount* an seinen Root Node, in dem er ihm den Redundanzgrad des Accounting Mechanismus weiterleitet.

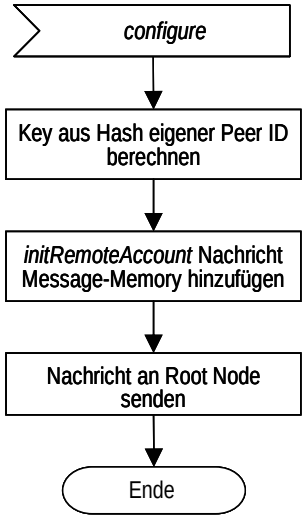

Abbildung 4.3: Methode *configure*

Abbildung 4.4 zeigt die Schritte, welche ein Root Node einleitet, wenn er eine *initRemoteAccount* Nachricht erhält. Über die Schnittstelle von FreePastry greift er auf die Funktion *replicaSet* zu, die ihm, in Abhängigkeit des erhaltenen Redundanzgrades, die gewünschten Remote Peers liefert und die er sogleich über ihre neue Aufgabe orientiert. Dazu schickt er allen eine Message vom Typ setRemotePeers, die alle Remote Peers in Form eines Objekts vom Typ *NodeHandleSet* und den Node Handle des Peers enthält, dessen Account es im Folgenden zu verwalten gilt. Am Schluss teilt der Root Node dem Absender der *initRemoteAccount* Nachricht das eigene Remote Set mit.

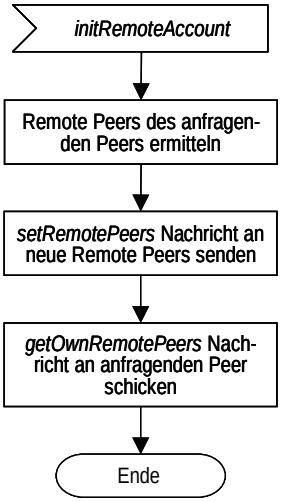

Abbildung 4.4: Methode *initRemoteAccount*

Ein Peer erhält seine Remote Peers in einer *getOwnRemotePeers* Message, worauf er eine die zur Nachricht gehörende Methode aufruft. Wie in Abbildung 4.5 zu sehen ist, speichert er sich während deren Ausführung als erstes die erhaltenen Informationen. Dann informiert er die Message Memory über die Ankunft der angeforderten Daten. Das Message Memory wiederum aktualisiert den Controller für die Konfigurationsaktion, womit es in der Folge möglich ist, Sessions zu kreieren. Die eigentlichen Aktionen werden mit einem Update der Action Queue gestartet, welche die *getOwnRemotePeers* Funktion zum Schluss durchführt.

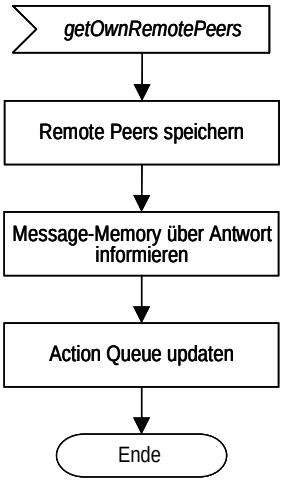

Abbildung 4.5: Methode *getOwnRemotePeers*

Die Methode *setRemotePeers* ruft ein Peer auf, wenn er darüber in Kenntnis gesetzt wird, das er in Zukunft die Account Balance eines bestimmten Peers verwalten soll. Die Abbildung 4.6 zeigt, dass im Verlauf der Funktion ein entsprechender Account bereitgestellt wird.

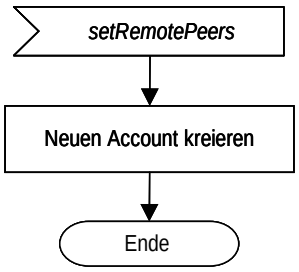

Abbildung 4.6: Methode *setRemotePeers*

4.2.4 Erstellen von Sessions

Nachdem alle Peers konfiguriert wurden, kann ein Knoten über die Methode *createSession* der Accounting Schnittstelle aufgefordert werden, eine Session mit einem anderen Peer aufzubauen. Der benachrichtigte Knoten agiert dabei als Consumer und soll den Provider über die von ihm erwünschte Session informieren.

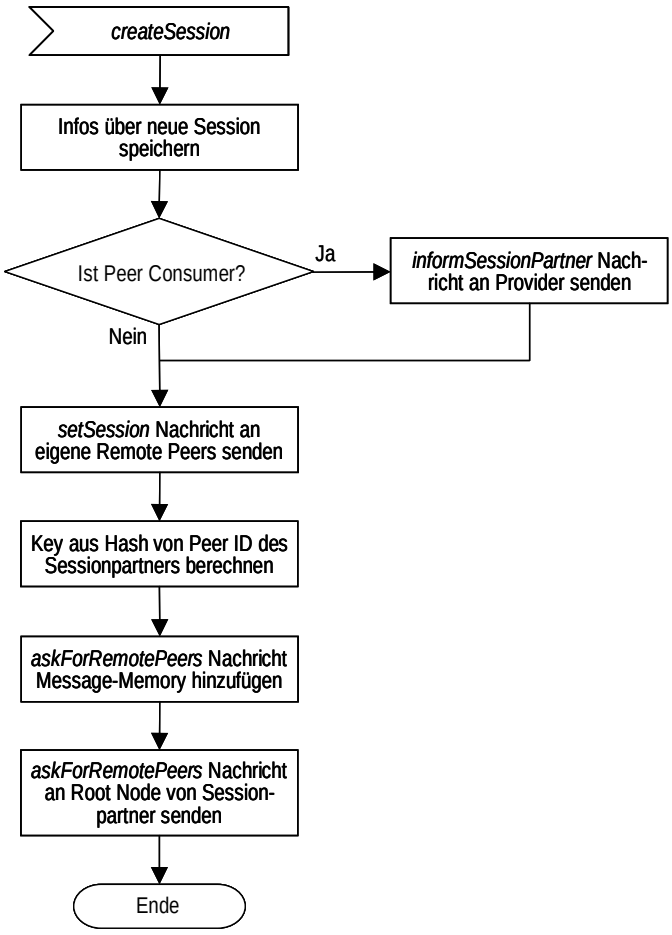

Abbildung 4.7: Methode *createSession*

In Abbildung 4.7 ist der Ablauf der Methode *createSession* zu sehen. Als Parameter übergibt ihr eine P2P Applikation im Wesentlichen den Node Handle des Sessionpartners, sowie die Art und den Tarif des Service, über den sich die Peers geeinigt haben. Wie bereits erwähnt, bittet der Consumer den Provider darum, mit ihm eine neue Session erstellen zu können. Dazu verschickt er ihm eine *informSessionPartner* Nachricht, worauf der Provider, wie in Abbildung 4.8 zu sehen ist, eine entsprechende Aktion in seine Action Queue einführt. Das führt schliesslich auf dem Peer ebenfalls zum Aufruf der API Methode *createSession*. Nachdem der Peer die Angaben zur neuen Session gespeichert und eine anfällige Nachricht an den Provider geschickt hat, muss er sich während der Ausführung der Funktion als Nächstes um die *setSession* Message an seine Remote Peers kümmern. Ihnen teilt er mit einer solchen Nachricht mit, mit welchem Peer er sich auf eine Session geeinigt hat und welchen Service der Provider dem Consumer im Verlaufe dieser zu welchem Tarif anzubietet. Für ein späteres Update der Account Balances müssen die beiden involvierten Peers natürlich die Remote Peers des jeweiligen Sessionpartners kennen. Dank dem bekannten Node Handle des Partners und der Methode *buildId* kann der Key berechnet werden, welcher der *askForRemotePeers* Nachricht mitgegeben wird und die dadurch an den Root Node des gewünschten Remote Sets geroutet wird. Bevor einem Peer die Remote Peers des Sessionpartners nicht bekannt sind, kann er die Aktionen nicht durchführen, welche sich allenfalls in der Action Queue befinden und welche ein Update der Account Balances bewirken. Es ist deshalb nötig die *askForRemotePeers* Nachricht dem Message Memory hinzuzufügen, das beim Erhalt der gewünschten Informationen den richtigen Controller aktualisiert und damit die Update Aktionen startet.

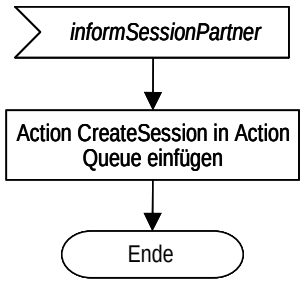

Abbildung 4.8: Methode *informSessionPartner*

Werden die Remote Peers über die neue Session benachrichtigt, so rufen sie ihre *setSession* Methode auf. Die Abbildung 4.9 zeigt, dass sie während deren Ausführung lediglich die erhaltenen Daten über die Session zu speichern haben, welche aus der ID der Session, den Node Handle der beiden Sessionpartner, sowie der Art und dem Tarif des vereinbarten Service bestehen.

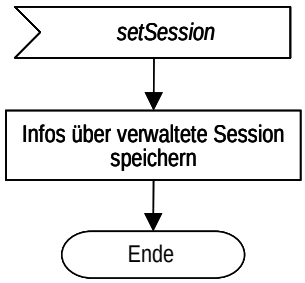

Abbildung 4.9: Methode *setSession*

Wie bereits erklärt, soll der Empfänger der *askForRemotePeers* Nachricht dem Absender die Remote Peers seines neuen Sessionpartners liefern. In der Abbildung 4.10 ist der Ablauf der aufgrund der Message aufgerufenen Funktion zu sehen. Die von FreePastry zur Verfügung gestellte Methode *replicaSet,* auf welche der Peer über seinen Pastry Endpoint zugreifen kann, liefert das gewünschte Remote Set. Dabei muss ihr als Parameter der Redundanzgrad des Accounting Mechanismus übergeben werden, der allen Peers in der Konfigurationsphase mitgeteilt wird. Die so berechneten Remote Peers werden dem anfragenden Peer mit der Nachricht *getExternalRemotePeers* mitgeteilt.

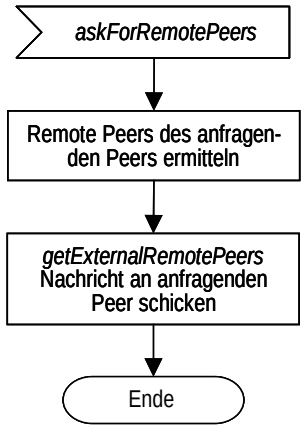

Abbildung 4.10: Methode *askForRemotePeers*

Erhält ein Peer das angeforderte Remote Set des Sessionpartners, arbeitet er die Methode *getExternalRemotePeers* ab. Während deren Verlauf, der in Abbildung 4.11 dargestellt ist, speichert sich der Peer vorerst die fremden Remote Peers. Dann benachrichtigt er das Message Memory über die erhaltenen Informationen, welches den Controller der entsprechenden Session aktualisiert. Mit dem Update der Action Queue werden von der Session abhängige Aktionen abgearbeitet, die das Update der Account Balances in die Wege leiten. Solche Aktionen sind das Produkt der erfolgreichen Interaktion des Consumers und Providers und können von einer P2P Applikation in die Action Queue eingefügt werden.

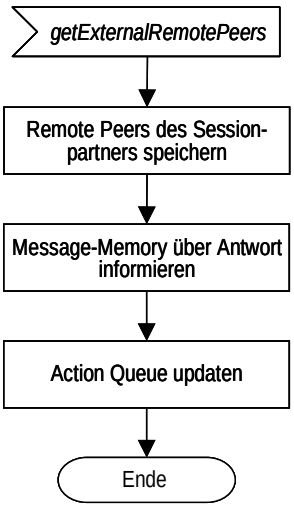

Abbildung 4.11: Methode *getExternalRemotePeers*

4.2.5 Account Balance aktualisieren

Als Nächstes soll die Implementierung des in Abschnitt 3.3.3 bereits erklärten Vorgangs zur Aktualisierung der Account Balances zweier Sessionpartner beschrieben werden. Das Update kann durch den Aufruf der Methode *updateAccountBalance* aus der Schnittstelle des Accounting Patterns gestartet werden. Der Consumer übergibt der Funktion die Menge an Ressourcen, welche er vom Provider bezogen hat, während der Provider der Funktion als Parameter angibt, wie viele Leistungen er dem Consumer angeboten hat.

In Abbildung 4.12 ist zu sehen, was der Peer während dem Ablauf der *updateAccountBalance* Methode alles unternimmt. Er schickt zum einen eine *updateAccount* Nachricht an die eigenen Remote Peers und übermittelt eine Message vom gleichen Typ an sämtliche Peers im Remote Set seines Sessionpartners.

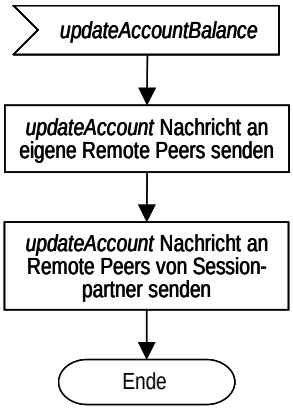

Abbildung 4.12: Methode *updateAccountBalance*

Erfährt ein Peer von einer angeblichen Aktualisierung eines Peer Accounts, welchen er verwaltet, so ruft er die Funktion *updateAccount* auf. Wie in Abbildung 4.13 gezeigt ist, überprüft ein Peer zuerst, ob er die Session, aufgrund derer das Update durchgeführt werden soll, überhaupt erfasst hat. Wenn dies nicht der Fall ist, könnte die Nachricht einerseits böswillig an den Peer gesandt worden sein, um ihn zu einem falschen Update einer Accounting Balance zu bewegen. Andererseits ist auch denkbar, dass der Peer vom jeweiligen Sessionpartner des Absenders der *updateAccount* Nachricht noch nicht über die Session informiert wurde. Dies tritt immer dann ein, wenn einer der beiden Sessionspartner die Aktionen in seiner Action Queue schneller abarbeiten kann als der andere. Deshalb muss der Peer die Informationen über seine angeblich verwaltete Session speichern und kann die bearbeitete Methode nicht vorzeitig beenden. Die nötigen Daten zur Session müssen somit in jeder *updateAccount* Message enthalten sein. Im weiteren Verlauf der betrachteten Funktion muss der Peer als Nächstes herausfinden, ob er bereits eine, zur jetzigen Nachricht äquivalente *updateAccount* Message vom entsprechenden Sessionpartner erhalten hat. Ist dies nicht geschehen, speichert er den Update Antrag, um beim Erhalt des dazu passenden Antrags des Sessionpartners die verwaltete Account Balance aktualisieren zu können. Ist ein Update tatsächlich notwendig, erhöht, repektive belastet der Peer die richtige Account Balance, je nachdem, ob es sich beim überwachten Account um den des Providers oder um den des Consumers handelt. Der Wert, den es der Account Balance abzuziehen oder zuzurechen gilt, kann dabei aus dem abgemachten Tarif der Sessionpartner und der Menge der zur Verfügung gestellten Ressourcen berechnet werden.

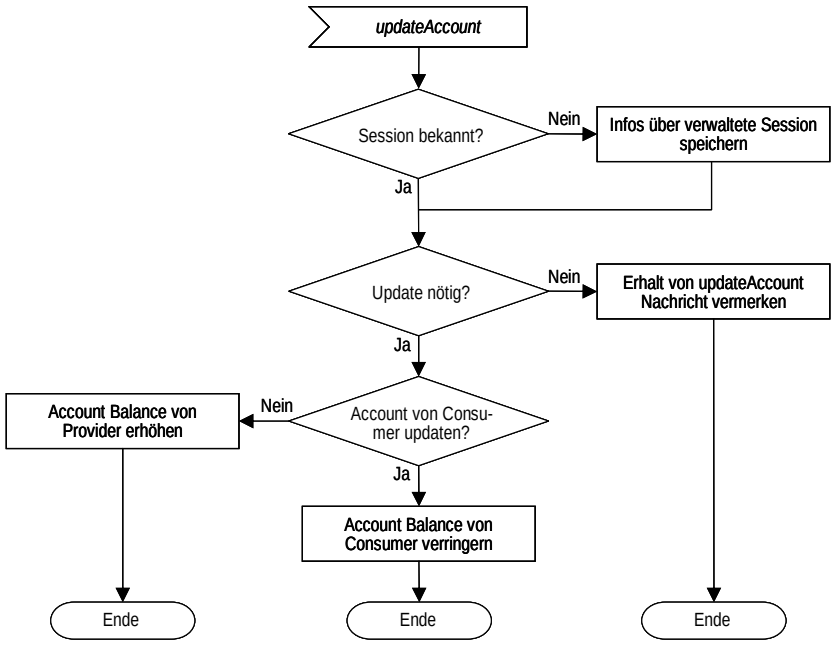

Abbildung 4.13: Methode *updateAccount*

4.2.6 Account Balance abfragen

Schliesslich wird die Umsetzung der API Methode *queryAccountBalance* beschrieben. Sie wurde vorangehend im Abschnitt 3.3.4 erläutert und kann von einer, auf den Remote Accounting Peers aufbauenden Applikation dazu verwendet werden, die Account Balance eines bestimmten Peers in Erfahrung zu bringen. Da diese Funktion im Prototyp von der Simulationskomponente nur zur Ermittlung der Account Balance des Peers benutzt wird, auf welchem die Methode auch aufgerufen wird, muss ihr als Parameter lediglich ein Listener Objekt mitgegeben werden, das dem Peer dazu dient, der Simulationskomponente den errechneten Wert für die Account Balance zurückliefern zu können.

In Abbildung 4.14 sind die Schritte beschrieben, welche ein Peer bei der Ausführung der *queryAccountBalance* Methode auszuführen hat. Bevor er eine *askForAccountBalance* Nachricht an alle seine Remote Peers sendet, muss er die Message dem Message Memory hinzufügen. Dies ist nötig, damit der Peer die erhaltenen Antworten an das Message Memory weiterleiten kann, welche das Message Memory wiederum an die vorher abgespeicherte Anfrage bindet. Liefern alle Remote Peers dem anfragenden Peer die eigene Account Balance zurück, so wird das in der Message Memory bemerkt. Der Peer wird dazu aufgefordert aus den erhaltenen Werten gemäss der, in Abschnitt 3.3.4 beschriebenen Methode zu berechen, wie hoch seine Account Balance ist und wie gross die Chance für die Richtigkeit des ermittelten Wertes ist. Falls einzelne Remote Peers böswillig sind oder beim Eintreffen der Nachricht gerade offline sind, geben sie dem jeweiligen Peer auf seine Anfrage keine Antwort. Folglich wird die Berechnung der Account Balance nicht durch das Message Memory veranlasst. Dafür, dass die Simulationskomponente in diesem Fall trotzdem einen Wert für die Account Balance des Peers erhält, sorgt der Timer Task der abgeschickten Nachricht. Dieser veranlasst das Message Memory nach einem Timeout, das Ermitteln der Account Balance des Peers lediglich anhand der bisher vorliegenden Werte vorzunehmen.

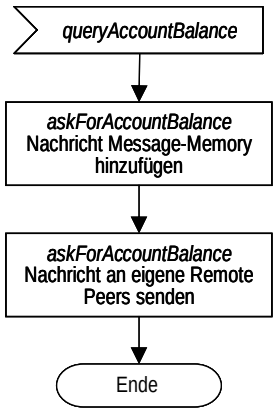

Abbildung 4.14: Methode *queryAccountBalance*

Trifft eine *askForAccountBalance* Nachricht bei einem Remot Peer ein, so wird die der Message entsprechende Methode aufgerufen und gemäss der Abbildung 4.15 abgehandelt. Der Empfänger meldet dem Absender mit einer *getAccountBalance* Nachricht die gewünschte Account Balance.

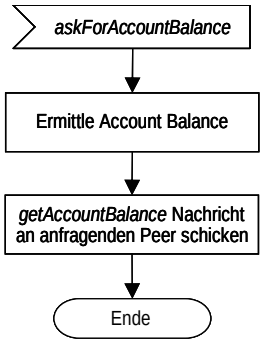

Abbildung 4.15: Methode *askForAccountBalance*

Erhält ein Peer von einem seiner Remote Peers seine Account Balance, so führt er die Funktion *getAccountBalance* aus. Während deren Ausführung, die in Abbildung 4.16 zu sehen ist, teilt der Peer lediglich dem Message Memory den erhaltenen Wert mit. Wie bereits zu Beginn dieses Abschnitts beschrieben, werden dann alle weiteren Aktionen, die für das Berechnen der Account Balance und das Überweisen dieser an die Simulationskomponente nötig sind, vom Message Memory selbstständig veranlasst.

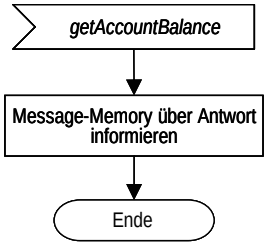

Abbildung 4.16: Methode *askForAccountBalance*

Kapitel 5

Evaluation

Der in dieser Arbeit implementierte Prototyp besteht im Wesentlichen aus zwei Komponenten. Dies ist zum einen der Distributed Accounting Mechanismus, welcher in den beiden vorhergehenden Kapiteln eingehend beschrieben wurde. Zum andern ist das die Simulationskomponente, welche bei der Durchführung der Evaluation des Accounting Mechanismus eine zentrale Rolle übernimmt und in diesem Kapitel beschrieben werden soll. Im nächsten Abschnitt werden zuerst die wesentlichen Aufgaben der Simulationskomponente besprochen, bevor im Abschnitt 5.2 die Implementation der Komponente beschrieben wird. Da für die konkrete Durchführung der Evaluation am Ende der Arbeit die Zeit fehlte, ist schliesslich in Abschnitt 5.3 lediglich beschrieben, wie bei der eigentlichen Evaluationsphase vorgegangen werden muss, um eine Aussage über die zu bestimmenden Grössen machen zu können und welche Daten dafür notwendig sind.

5.1 Aufgaben der Simulationskomponente

Die Simulationskomponente erfüllt zwei Aufgaben. Als Simulator ahmt sie ein P2P System nach und als Evaluator ermittelt sie die für die Evaluation des Distributed Accounting Mechanismus notwendigen Simulationsergebnisse. Ihre Funktionsweise kann anhand der Abbildung 5.1 erklärt werden. Mittels eines benutzerdefinierten Inputs fliessen Daten in die Komponente und ein Simulator startet den Simulationsprozess. Der Input beinhaltet dabei sowohl Angaben zur Konfiguration der Peers, als auch Angaben, welche für die Konfiguration des Distributed Accounting Mechanismus gebraucht werden. Für die Konfiguration der Peers ist es nötig, eine Annahme über den Anteil der Peers im System zumachen, welche böswillig sind und die mittlere Peer Uptime der Peers im System zu definieren.

Ebenso wird bei der Konfiguration der Redundanz- und Verteiltheitsgrad des Distributed Accounting Mechanismus festgelegt. Der Redundanzgrad gibt dabei die Anzahl der Peers an, welche in einem Remote Set sind. Der Verteiltheitsgrad definiert, ob ein Account zentral, hybrid oder verteilt verwaltet wird. Während des eigentlichen Simulationsprozesses, welcher durch den Simulator gestartet wird tauscht die Simulationskomponente über das Accounting API Informationen mit dem Peer aus. Am Schluss des Vorgangs steht die Auslieferung der vom Benutzer gewünschten Resultate durch den Evaluator. Um das garantieren zu können, muss der Evaluator sowohl über die erwarteten, als auch über die effektiven Resultate der Simulation verfügen.

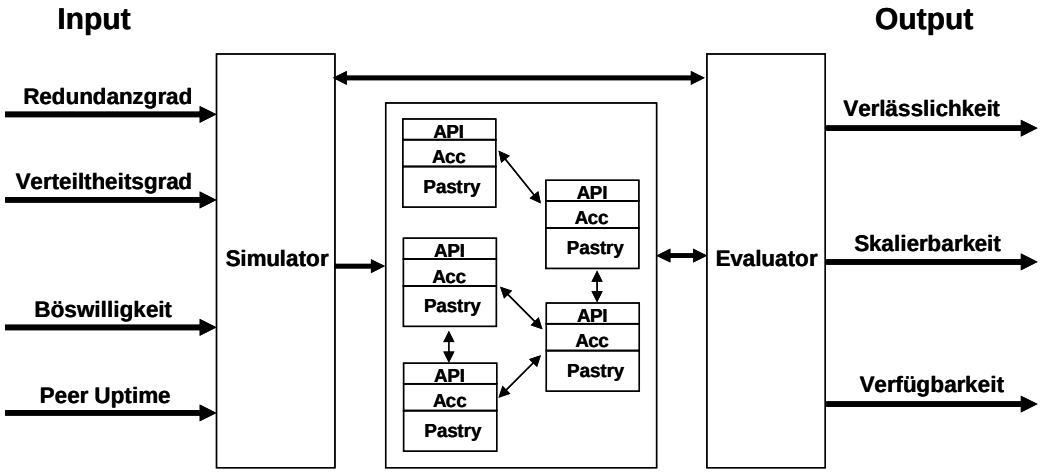

Abbildung 5.1: Evaluation des Accounting Mechanismus

5.1.2 Simulator

Für die Anlayse von dynamischen Systemen bietet sich eine Simulation förmlich an. Sie ermöglicht es, realitätsnahe Experiment-Setups zu wählen, welche bei der Evaluation des Distributed Accounting Mechanismus zu aussagekräftigen Resultaten führen. Die Peers agieren auf der einen Seite selbst dynamisch. Sie können böswillig sein, gehen immer wieder offline und kommen zu einem späteren Zeitpunkt wieder online. Sie verlassen oder treten einem Peer-to-Peer Netzwerk bei. Auf der anderen Seite interagieren die Peers auch miteinander. Sie können sich gegenseitig Ressourcen zur Verfügung stellen oder Leistungen von anderen Peers in Anspruch nehmen. Die hier verwendete Simulation muss diesen beiden Aspekten, der Dynamik des eigentlichen Peer-to-Peer Netzwerkes einerseits, den dynamischen Abläufen von Peer-to-Peer Applikationen andererseits, gerecht werden.

Die Umsetzung des beschriebenen Modells geschieht in der Simulationskomponente mit einem Simulator. Dieser sorgt zuerst einmal für die Erstellung des Peer-

to-Peer Netzwerks, wozu ihm Pastry sehr gute Dienste leisten kann. Über das Accounting API kann dann festgelegt werden, ob die beteiligten Peers böswillig sind und wie oft sie im System on- beziehungsweise offline sein sollen. Der eigentliche On-/Offline-Zyklus, wird von den Peers jedoch intern erzeugt. Während der Durchführung der Simulation generiert die Komponente laufend Sessions zwischen zwei zufälligen Peers. Dabei bestimmt sie auch, wie oft im Verlauf dieser, als Zeichen einer erfolgreichen Transaktion, ein Update der entsprechenden Account Balances durchzuführen ist. Die entsprechenden Informationen lassen sich abermals über das Accounting API an die Peers weiterleiten. Schliesslich soll die Simulationskomponente dafür sorgen, dass immer wieder neue Peers zum Netzwerk hinzukommen und andere dieses verlassen.

5.1.2 Evaluator

Die Simulationskomponente ist für die Beschaffung der für die Evaluation des Distributed Accounting Mechanismus nötigen Informationen zuständig. In erster Linie sollen Aussagen über die Verlässlichkeit und die Skalierbarkeit des Verfahrens gemacht werden können, weshalb die beiden zentralen Grössen die Anzahl der verschickten Nachrichten im System und die Account Balance der einzelnen Peers sind. Um die entsprechenden Daten zu erhalten, soll ein Evaluator zu gegebener Zeit mit den Peers in Verbindung treten. Die Account Balances und die zu ihnen gehörenden Thresholds, können über das Accounting API ermittelt werden, während zwecks Auskunft über den Nachrichtenversand separate Methoden zur Verfügung stehen müssen. Als Ergebnis der Simulation soll der Evaluator die ermittelten Werte in einer geeigneten Form wiedergeben. Dazu können die Nachrichten in verschiedene Kategorien, wie beispielsweise Synchronisationsnachrichten oder Updatenachrichten unterteilt werden. Die erhaltenen Account Balances können aufgrund eines Thresholds und des Simulationsablaufs auf ihre Korrektheit und Aussagekraft untersucht werden.

5.2 Implementation der Simulationskomponente

Wie bereits im Abschnitt 4.1 beschrieben, bietet FreePastry eine Testklasse *DistCommonAPITest* welche dazu verwendet werden kann, eine Pastry Netzwerk zu generieren. Die Simulationskomponente ist eine Subklasse *RemoteAccountingTest* der Klasse *DistCommonAPITest*. Entsprechend muss sie die beiden abstrakten Methoden *processNode* und *runTest* der Testklasse implementieren. Während die Funktion *processNode* immer dann aufgerufen wird, sobald im Laufe der Initialisierung des Pastry Netzwerkes ein neuer Pastry Knoten erstellt wurde, wird die Methode *runTest* aufgerufen, wenn die Konfiguration abgeschlossen ist. Mit dem Pastry Knoten, welche die Simulationskomponente in der *processNode* Methode erhält, sorgt sie dafür, dass diesem eine Accounting Komponente aufgesetzt

wird. Wenn das ganze Peer-to-Peer System konfiguriert ist, setzt die Simulationskomponente einen Simulationsprozess über die Funktion *runTest* in Gang, in dessen Verlauf über das Accounting API Gebrauch vom Distributed Accounting Mechanismus Gebrauch gemacht werden kann.

5.2.1 Manuelle Simulation

Zu zum Abschluss der Arbeit vorliegende Simulationskomponente kann noch keine zufälligen Simulationsprozesse generieren, bietet aber die Möglichkeit einer manuellen Simulation.

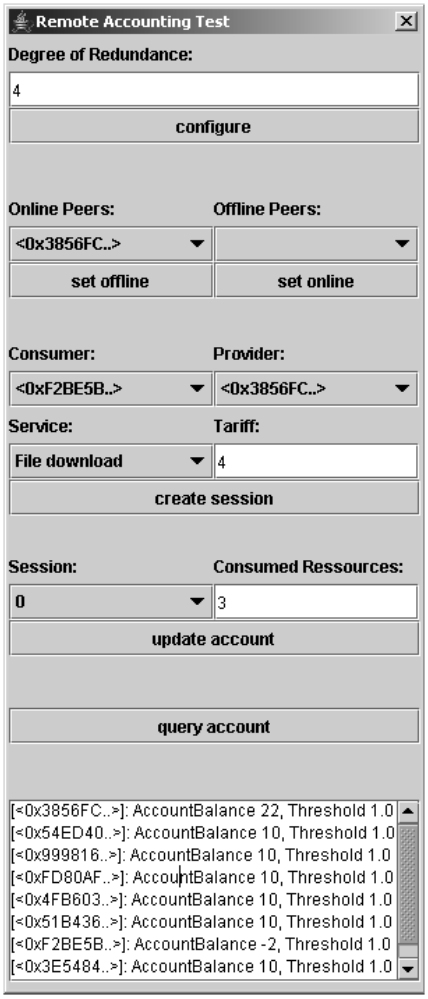

Abbildung 5.2: Manuelle Simulationssteuerung

Die Simulation startet beim Aufruf der Methode *runTest*, indem sie dem Benutzer ein GUI zur Verfügung stellt, welches in Abbildung 5.2 zu sehen ist. Über dieses GUI kann der Benutzer auf die verschiedenen API Methoden der Remote Accounting Peers zugreifen. Dem Beispiel aus Abbildung 5.2 liegt ein Peer-to-Peer

System zu Grunde, welches aus acht Peers besteht, die alle nicht böswillig sind und manuell on- und offline gesetzt werden können. Wie zu sehen ist, wurde der Distributed Accounting Mechanismus mit einem Redundanzgrad von vier konfiguriert, indem der Wert in das entsprechende Textfeld eingegeben und dann der Button *configure* betätigt wurde. Durch das Drücken auf einen Button wird auf auf der Instanz der Klasse *RemoteAccountingTest* eine der Aktion entsprechende Methode aufgerufen. Bei der Konfiguration des Accounting Mechanismus ist dies die Methode *runConfigure*, welche für alle Peers im System eine Configure Aktion generiert und diese in die Action Queue des jeweiligen Peers einfügt. Neben der Konfiguration, kann der Benutzer über das GUI Peers jederzeit manuell on- oder offline setzen, indem er über die entsprechende Drop-Down Liste den gewünschten Peer auswählt und den Button *set offline* oder *set online* betätigt. Als nächstes lassen sich Sessions erstellen. Dafür kann über eine Drop-Down Liste der Consumer und der Provider, sowie die Art des Services ausgewählt werden. In einem Textfeld muss der Benutzer überdies den Tarif der Session angeben. Im Beispiel möchte der Peer mit der ID *<0xF2BE5B>* Files vom Peer mit der ID *<0x3856FC>* downloaden, welche pro Megabyte 4 Währungseinheiten kosten. Nach dem Betätigen des *create session* Buttons wird auf der Simulationskomponente die Methode *runSession* ausgeführt, welche die entsprechenden Aktionen in die Action Queues der beiden an der Session beteiligten Peers einfügt. Im Weiteren kann man sich vorstellen, dass der Provider dem Consumer die gewünschten Files zur Verfügung stellt. Ist das geschehen, müssen die Account Balances der Peers aktualisiert werden. In unserem Beispiel hat der Consumer 3 MB Daten vom Provider erhalten. Durch das Drücken des Buttons *update account* wird auf der Instanz der Klasse *RemoteAccountingTest* die Methode *runUpdate* aufgerufen, welche eine Update Aktion in die Action Queue der Sessionpartner einfügt. Zum Ende des Beispiels werden die Account Balances der Peers abgefragt, wobei gesagt werden muss, dass diese bei der Konfiguration jeweils einen Wert von 10 Währungseinheiten erhalten haben, welche ihnen die jeweiligen Remote Peers zugewiesen haben. Mit dem Betätigen des Buttons *query account* kann eine Anfrage ausgelöst werden. Eine Methode *runQuery* wird ausgeführt, welche eine Query Aktion in die Action Queue sämtlicher Peers einfügt. Alle Peers ermitteln nun ihre eigene Account Balance, indem sie Nachrichten an ihre Remote Peers schicken und von diesen einen Wert für ihre Account Balance zurückerhalten. Da der Queryaktion eine Referenz auf die Simulationskomponente mitgegeben wird, können die Peers ihre berechneten Werte an die Instanz des *RemoteAccountingTest* zurückliefern, welche sie dann wiederum ins GUI schreiben. In Abbildung 5.2 ist die Account Balance für alle acht Peers zu sehen. Der Wert des Consumers mit der Peer ID *<0xF2BE5B>* hat sich dabei um 12 Währungseinheiten verringert, während die Account Balance des Providers mit der Peer ID *<0x3856FC>* nun auf 22 Währungseinheiten steht. Da kein Peer im System böswillig ist oder während der Ausführung des Beispiels offline war, beträgt die Wahrscheinlichkeit für die Korrektheit des ermittelten Account Balances aller Peers 100%.

5.2.2 Automatisierung der Simulation

Um noch aussagekräftigere Resultate zu erhalten, sollte die Simulation automatisiert werden, indem die Simulationsabläufe zufällig generiert werden. Im Folgenden soll deshalb beschrieben werden, wie die Simulation, welche auf dem implementierten Distributed Accounting Mechanismus aufbaut automatisiert werden kann. Eine Implementation des beschriebenen Modells liegt zum Zeitpunkt des Abschlusses der Arbeit noch nicht vor.

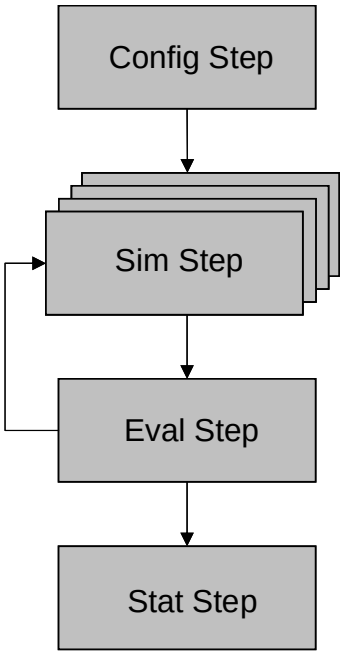

Abbildung 5.3: Steuerung der Simulation

In Abbildung 5.3 ist die Steuerung eines automatisierten Simulationsablaufs zu sehen. Die Simulationskomponente startet den Simulationsprozess mit einem Config Step. In diesem werden alle Peers im Peer-to-Peer System gemäss dem jeweiligen Setup der Simulation konfiguriert. In einem nächsten Schritt werden mehrere Sim Steps generiert, welche den eigentlichen Simulationsablauf beschreiben. Der Simulationsablauf besteht im Wesentlichen aus einer Menge von zufällig generierten Account Balance Updates welche die Peers innerhalb einer Session durchführen. Sind alle Sim Steps generiert so werden sie der Reihe nach abgearbeitet. Dazu werden die angesprochenen und zufällig generierten Aktionen in die Action Queues der entsprechenden Peers eingefügt. Haben alle Peers die Aktionen im SimStep beendet, so melden sie dies der Simulationskomponente. Anschliessend wird ein Eval Step durchgeführt, in welchem alle Peers dazu aufgefordert werden, ihre Account Balances und die Anzahl Nachrichten, welche sie während des Simulationsprozesses verschickt haben, der Simulationskomponente zu melden. Anhand des im Sim Step gespeicherten Simulationsablaufs können

die erwarteten Werte für die Account Balances berechnet werden und mit den effektiven Werten im Peer-to-Peer System verglichen werden. Das Erstellen solcher Statistiken für jeden Sim Step ist die Aufgabe des Stat Steps. Hat die Simulationskomponente schliesslich von allen Peers die gewünschten Daten für einen Sim Step erhalten, so kann der nächste Sim Step ausgeführt werden.

Die zentrale Komponente beim beschriebenen Konzept stellt der Sim Step dar, da er für die die Generierung des eigentlichen Simulationsablaufs zuständig ist. In Abbildung 5.4 ist zu sehen wie ein Sim Step diesen Ablauf erstellen kann. Gemäss dem Simulationssetup muss er sich dafür eine bestimmte Anzahl von Sim Sessions erstellen. Dazu werden zufällig zwei Peers aus dem Peer-to-Peer System ausgewählt, welche während der Ausführung des Simulationsschrittes eine gemeinsame Session kreieren sollen. Bei der Erstellung der Sim Sessions generieren diese wiederum eine Reihe von Update Aktionen, welche sie der zur SimSession gehörenden Session Aktion hinzufügen. Sind alle Sim Sessions erstellt, so bittet der Sim Step zufällig immer wieder eine andere Sim Session, ihre nächste kreierte Aktion in eine Action List einzufügen. Dieser Vorgang wird solange wiederholt, bis die Sim Sessions alle generierten Aktionen in der Action List abgelegt haben. Aus der Aneinanderreihung der verschiedenen Aktionen in der Action List ist ein zufälliger Simulationsablauf entstanden welche der Sim Step nun abarbeiten kann.

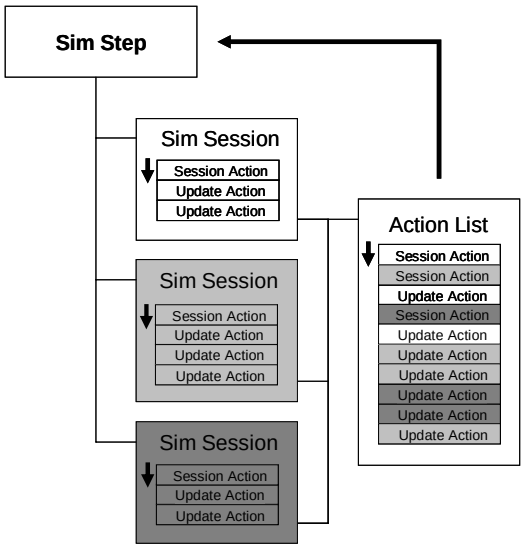

Abbildung 5.4: Generierung des Simulationsablaufs

5.3 Evaluationsphase

Anhand der aus der Simulation gewonnen Daten kann die Evaluation des Distributed Accounting Mechanismus vorgenommen werden. Dabei soll der Mecha-

nismus in Bezug auf seine Verlässlichkeit, seine Skalierbarkeit und seine Verfügbarkeit untersucht werden. Da in der zur Verfügung stehenden Zeit kein automatisierte Simulationskomponente implementiert werden konnte, war es während der Arbeit nicht möglich, genügend Daten zu sammeln, welche eine verlässliche Aussage über das Verhalten des Distributed Accounting Mechanismus zugelassen hätten. Im Folgenden soll jedoch nur beschrieben werden, welche Werte für die einzelnen, zu untersuchenden Grössen massgebend sind und wie diese mit den einzelnen Input Daten zusammenhängen.

5.3.1 Verlässlichkeit des Mechanismus

Mit der Verlässlichkeit des Distributed Accounting Mechanismus ist die Zuverlässigkeit gemeint, mit der festgestellt werden kann, wie die korrekten Account Balances der einzelnen Peers sind. Um dies mit Hilfe einer Simulation überprüfen zu können, müssen zwei Grössen ermittelt werden. Zum einen muss bekannt sein, wie gross die erwarteten Veränderungen in den Account Balances für einen durchgeführten Simulationsvorgang sind. Zum anderen müssen die Account Balances nach der Durchführung der Simulation über das Accounting API beschafft werden. Der Vergleich zwischen den beiden genannten Grössen liefert dann direkt eine Aussage über die Verlässlichkeit des Distributed Accounting Mechanismus. Dabei dürften die beiden wesentlichen Faktoren, welche auf die Verlässlichkeit einen Einfluss haben, die Böswilligkeit der Peers und der Redundanzgrad des Verfahrens sein. Je mehr böswillige Peers in einem System vorhanden sind, desto grösser ist die Wahrscheinlichkeit, dass die Account Balances nicht korrekt verwaltet werden. Dies ist deshalb so, da aufgrund der gleichmässigen Verteilung der Peers im Overlay Netzwerk von Pastry der Anteil an böswilligen Peers in einem Remote Set proportional zum Anteil der im gesamten System vorhandenen böswilligen Peers ist. Umgekehrt steigt die Wahrscheinlichkeit, dass in einem Remote Set eine Mehrheit an gutwilligen Peers vorhanden ist, wenn der Redundanzgrad des Accounting Mechanismus und damit auch die Remote Peers der einzelnen Peers vergrössert wird. Einen Einfluss auf die Verlässlichkeit des Distributed Accounting Mechanismus hat auch die durchschnittliche Peer Uptime. Je länger die einzelnen Peers im System nämlich offline sind, je weniger haben sie die Möglichkeit, von einem Update der von ihnen verwalteten Account Balances zu erfahren. Dies ist im Gegensatz zur Böswilligkeit der Peers deshalb nicht so gravierend, da sich die Remote Peers laufend synchronisieren.

5.3.2 Skalierbarkeit des Mechanismus

Unter der Skalierbarkeit des Distributed Accounting Mechanismus versteht man den Aufwand, welcher für die Ausführung des Verfahrens betrieben werden muss in Abhängigkeit der Anzahl im System beteiligter Peers. Wenn dieses Verhältnis

ungefähr konstant bleibt oder nicht mehr als logarithmisch zunimmt, so ist der Mechanismus skalierbar. Der Aufwand des Distributed Accounting Mechanismus ist charakterisiert durch die Anzahl Nachrichten, welche sich die Peers im System zuschicken müssen. Somit sollten während des Simulationsprozesses Informationen über die Anzahl verschickter Messageen gesammelt werden können. Eine zentrale Rolle für die Skalierbarkeit des Mechanismus spielt die Synchronisation zwischen den einzelnen Peers und damit die Peer Uptime. Je mehr Peers nämlich offline sind und dabei keine Nachrichten von anderen Peers mehr empfangen können, desto mehr Messageen müssen sie für ihre Synchronisation mit den anderen Remote Peers, welche die gleichen Account Balances verwalten verwenden, wenn sie wieder online kommen. Auch die Böswilligkeit der Peers hat einen negativen Einfluss auf die Skalierbarkeit des Mechanismus, weil böswillige Peers andere Peers dazu zwingen können, weitere Messageen verschicken zu müssen. Dies ist insbesondere dann der Fall, wenn ein Peer gewisse Remote Peers in Erfahrung bringen möchte und der jeweilige Root Node, an den er seine Anfrage richtet, böswillig ist. Auch ein steigender Redundanzgrad führt schliesslich zu einem Anstieg an versendeten Nachrichten. Denn je grösser der Redundanzgrad ist, desto grösser sind auch die Remote Sets, womit auch das Versenden von Informationen oder Anfragen an Remote Peers nachrichtenintensiver wird.

5.3.3 Verfügbarkeit des Mechanismus

Mit Verfügbarkeit des Mechanismus ist die Wahrscheinlichkeit gemeint, mit der ein Peer Zugang zur Account Balance eines beliebigen Peers im System hat. Der Wert, welcher über diese Wahrscheinlichkeit Auskunft gibt, ist die Anzahl der Peers, für welche während eines Simulationsprozess die Account Balance nicht bestimmt werden konnte. Dies passiert dann, wenn die Werte, welche einem Peer von Remote Peers gelieferten werden, keinen Mehrheitsentscheid erlauben oder wenn der Threshold für die errechnete Account Balance nicht genügend gross ist. Die Chance für eine hohe Verfügbarkeit des Distributed Accounting Mechanismus ist deshalb gross, da die Account Balances verteilt und redundant verwaltet werden. Dadurch ist es möglich, dass mehrere Peers Auskunft über die Account Balance eines Peers geben können. Wäre beispielsweise nur ein Peer für die Account Balance eines Peers zuständig und wäre dieser ständig offline oder böswillig, so würde die Account Balance dieses Peers wohl gar nie zur Verfügung stehen. Insofern sorgt ein grosser Redundanzgrad auch für eine hohe Verfügbarkeit des Mechanismus. Auf der anderen Seite schmälern ein hoher Anteil an böswilligen Peers und Peers, welche eine niedrige Peer Uptime haben die Verfügbarkeit des Verfahrens.

Kapitel 6

Zusammenfassung

Da sich viele Peers in realen Peer-to-Peer Systemen oft so verhalten, dass sie sehr viele Ressourcen konsumieren, selbst aber nichts zum System beitragen, ist die Gesamtperformance in solchen System oft viel kleiner als man erwarten könnte. Ein Verfahren, um dieses Freerider Problem zu lösen ist das Accounting. In dieser Arbeit wurde das Design, die Implementation und Evaluation eines solchen Mechanismus, des Distributed Accounting Mechanismus beschrieben.

Der Mechanismus wurde als Accounting Pattern umgesetzt. Dies bedeutet, dass das Verfahren eine Schnittstelle implementiert, welche es einer Peer-to-Peer Applikation erlaubt, während ihrer Ausführung Gebrauch vom Distributed Accounting Mechanismus zu machen, ohne sich dabei um die konkrete Umsetzung desjenigen kümmern zu müssen. Die Schnittstelle bietet vier Methoden an, über die der Distributed Accounting Mechanismus konfiguriert werden kann, über die neue Accounting Sessions kreiert werden können und über die man die Account Balance eines beliebigen Peers aktualisieren oder abfragen kann.

Als zu Grunde liegende Infrastruktur wurde bei der Implementation des Distributed Accounting Mechanismus Pastry verwendet. Pastry bietet nicht nur Funktionen an, welche den Aufbau und die Verwaltung eines Peer-to-Peer Netzwerks vereinfachen, und die Kommunikation zwischen den einzelnen Peers erleichtern, sondern enthält auch Methoden, welche die Konzeption und das Design des Distributed Accounting Mechanismus einfacher machen.

Zur Evaluation wurde eine Simulationskomponente implementiert, welche das manuelle Simulieren von Peer-to-Peer Applikationen möglich macht, die auf den Distributed Accounting Mechanismus zugreifen. Ebenso wurde ein Modell beschrieben, welches die Automatisierung des Erstellens von Simulationsprozessen

beschreibt. Schliesslich wurde gezeigt, welche Grössen bei der Evaluation des Distributed Accounting Mechanismus untersucht werden sollen und wie mit Hilfe von Daten, welche während einer Simulation ermittelt wurden, über diese eine Aussage gemacht werden kann.

6.1 Zukünftige Arbeiten

Die Implementierung und Evaluation des Distributed Accounting Mechanismus konnte in dieser Arbeit nicht immer im Sinne der Aufgabenstellung umgesetzt werden. Während bei der Implementation gewisse vereinfachende Annahmen an das, dem Peer-to-Peer System zugrunde liegende System gemacht wurden, konnten bei der Evaluation des Verfahrens noch keine konkreten Aussagen über dessen Verfügbarkeit, Verlässlichkeit und Skalierbarkeit gemacht werden.

Da eine konkrete Evaluation des implementierten Distributed Accounting Mechanismus im Rahmen dieser Arbeit wie bereits erwähnt noch nicht vorgenommen werden konnte, bietet sich die Durchführung einer solchen in einem nächsten Schritt an. Dazu muss die implementierte Simulationskomponente dahingehend abgeändert werden, dass sie die automatische Generation von verschiedenen Simulationsabläufen unterstützt und die Resultate der Simulation dem Benutzer in einer geeigneten Form zur Verfügung stellt.

Weiter basiert die Implementierung wie erklärt auf gewissen vereinfachenden Annahmen zum Peer-to-Peer System. So kann ein Peer nur böswillig oder offline sein, wenn er als Remote Peer ein Update einer Account Balance durchführen oder wenn er einem Peer eine von ihm verwaltete Account Balance mitteilen soll. Sämtliche Aufgaben im Zusammenhang mit der Konfiguration eines Peers oder dem Erstellen von neuen Sessions erfüllen die Peers jedoch korrekt und auf jeden Fall. Tritt ein Peer als Root Node auf, welcher eine Nachricht an die Remote Peers weiterleiten, oder einem Peer das entsprechende Remote Set liefern soll, so agiert er ebenfalls nie bösartig. Daher wäre es interessant, das Modell des Peer-to-Peer System zu aktualisieren und dieses dynamischer zu gestalten, was gewisse Anpassungen in der Implementation des Distributed Accounting Mechanismus nötig macht. Die Peers müssten ein höheres Mass an Synchronisation anbieten und bei der Beschaffung von Informationen über die Remote Peers von anderen Peers nicht mehr auf den jeweiligen Roote Node angewiesen sein.

Das bisher implementierte Remote Accounting Verfahren bietet darüber hinaus keine Sicherheitsmechanismen an. Die Kommunikation zwischen den einzelnen Peers ist komplett unverschlüsselt. Der Mechanismus müsste darum dahingehend angepasst werden, dass er eine authentische Kommunikation unterstützt, welche das Fälschen von Nachrichten nicht mehr möglich macht. Dies könnte mit Hilfe der Public Key Kryptographie realisiert werden.

Ebenso könnte die von Freepastry verwendete Testklasse *DistCommon-APITest* so verändert werden, damit sie das Hinzufügen von Knoten in das Pastry Netzwerk oder das Wegnehmen von Knoten aus dem Pastry Netzwerk jederzeit unterstützt. Dadurch wäre das Peer-to-Peer System noch mehr an real vorhandene Systeme angepasst und der Distributed Accounting Mechanismus könnte in einer noch realitätsnaheren Umgebung getestet werden.

Literaturverzeichnis

[1] David Hausheer. *Accounting Patterns*. Deliverable 10.1 des MMAPPS Projekts, März 2004.

[2] MMAPPS. *Market Management of P2P Services*. EU Project, http://www.mmapps.org/.

[3] David Hausheer, Nicolas C. Liebau, Andreas Mauthe, Ralf Steinmetz, Burkhard Stiller. *Towards A Market Managed Peer-to-Peer File Sharing System Using Token-based Accounting and Distributed Pricing*. TIK Report Nr. 179, ETH Zürich, TIK, August 2003.

[4] David Hausheer, Nicolas Liebau, Andreas Mauthe, Ralf Steinmetz, Burkhard Stiller. *Token-based Accounting and Distributed Pricing to Introduce Market Mechanisms in a Peer-to-Peer File Sharing Scenario*. 3rd IEEE International Conference on Peer-to-Peer Computing, Linköping, Schweden, Seiten 200-201, September 2003.

[5] Beverly Yang, Hector Garcia-Molina. *PPay: Micropayments for peer-to-peer systems*. ACM conference on Computer and Communication Security (CCS ’03), Washington, DC, USA, Oktober 2003.

[6] Vivek Vishnumurthy, Sangeeth Chandrakumar, Emin Gün Sirer. *KARMA: A Secure Economic Framework for Peer-to-Peer Resource Sharing*. Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA, 5.-6. Juni 2003.

[7] Li Gong. Project *JXTA: A Technology Overview*. Sun Microsystems Inc., Palo Alto, CA, USA, Oktober 2002, http://www.jxta.org/.

[8] Antony Rowstron, Peter Druschel. *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*. 18th IFIP/ACM International Confer ence on Distributed Systems Platforms (Middleware 2001). Heidelberg, Deutschland, November 2001, http://research.microsoft.com/~antr/Pastry/.

[9] Ben Y. Zhao, John Kubiatowicz, Anthony D. Joseph. *Tapestry: An infrastructure for fault-resilient wide-area location and routing*. Tech. Rep. UCB//CSD-01-1141, UC Berkeley, CA, USA, 2001.

[10] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. *A Scalable Content-Addressable Network*. ACM conference on Special Interest Group on Data Communication (SIGCOMM), San Diego, CA, USA, August 2001.

[11] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, Hari Balakrishnan. *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications.* ACM conference on Special Interest Group on Data Communication (SIGCOMM 2001), San Diego, CA, USA, August 2001.

[12] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Anthony Rowstron, Dan S. Wallach. *Secure routing for structured peer-to-peer overlay networks*. 5th Usenix Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, Dezember 2002.

[13] Antony Rowstron, Peter Druschel. *Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility*. 18th ACM symposium on Operating systems principles (SOSP'01), Lake Louise, Alberta, Canada, Oktober 2001.

Abbildungsverzeichnis