TECHNISCHE UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Fachgebiet für Datenbanken und verteilte Systeme
Prof. Alejandro P. Buchmann, Ph. D.

Betreuer Christof Leng

Diplomarbeit

Reputationssysteme in geclusterten Peer-to-Peer Netzwerken

Peter Dobschal

14. April 2008

Statutory Declaration

I declare that I have developed and written the enclosed work completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Diploma Thesis was not used in the same or in a similar version to achieve an academic degree nor has it been published elsewhere.

Eidesstattliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Peter Dobschal

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit Reputationssystemen in geclusterten Peer-to-Peer Netzwerken.
Reputationssysteme sammeln Informationen über das Verhalten eines Teilnehmers und versuchen daraus eine Prognose seines zukünftigen Verhaltens abzuleiten (Reputation). Diese Ansätze haben ihre Nützlichkeit bei Online-Communities und -Marktplätzen wie eBay bewiesen, stellen aber in einem Peer-to-Peer System aufgrund der fehlenden Kontrollinstanz eine besondere Herausforderung dar. Jeder einzelne Peer muss das Verhalten seiner Transaktionspartner bewerten und diese Information im Netzwerk verteilen. Bei der Auswertung dieser Informationen muss vor allem die Vertrauenswürdigkeit der einzelnen Peers und die Plausibilität, der von ihnen abgegebenen Bewertungen, hinterfragt werden. Im Mittelpunkt dieser Arbeit steht zum einen die effektive Informationsverteilung von Bewertungen, sowie die Untersuchung verschiedener Verfahren zur Berechnung der Reputation anhand der zuvor gesammelten Informationen, um einen möglichst fairen Datenaustausch zu gewährleisten.
Um die Effizienz der Informationsverteilung zu steigern und um die Plausibilität und Vertrauenswürdigkeit der Bewertung besser beurteilen zu können, sollen die Clusterstrukturen von Peer-to-Peer Netzwerken ausgenutzt werden. Diese entstehen unter anderem durch unterschiedliche Interessen der Benutzer. Es werden verschiedene Verfahren für die Informationsverteilung und die Reputationsberechnung vorgestellt, die sich an existierenden Reputationssystemen orientieren, sowie neue Verfahren, die aus Überlegungen während der Erstellung dieser Arbeit hervorgegangen sind.
Durch die Simulation verschiedener Szenarien, soll der Erfolg der einzelnen Verfahren analysiert werden, sowie die Auswirkungen der Clusterstrukturen untersucht werden. Des Weiteren soll, durch die Simulation verschiedener Angriffe auf das Reputationssystem, die Robustheit der Verfahren überprüft werden.

2. Motivation

2.1. Einleitung

Im heutigen Internetzeitalter haben Computernetzwerke in allen Lebensbereichen Einzug gehalten. Die klassische Architektur eines Computernetzwerks, mit einer zentralen Verwaltungseinheit dem Server als Leistungsanbieter und diversen Clients als Leistungsnutzer, war in der Vergangenheit das grundlegende Konzept für die Kommunikation innerhalb eines Netzwerks. Jedoch setzte sich in den letzten Jahren eine dezentrale Architektur in Form der sogenannten Peer-to-Peer (P2P) Netzwerke immer mehr durch. Bei dieser Netzwerkarchitektur übernimmt jeder Teilnehmer (Peer) zugleich einen Teil der Aufgaben des Leistungsanbieters (Provider) sowie des Leistungskonsumenten (Consumer). Somit ist jeder Teilnehmer in einem P2P Netzwerk gleichgestellt und in einem gewissen Maße für die Funktionsfähigkeit des Netzwerks mitverantwortlich. Gründe dafür, dass sich P2P Netzwerke in bestimmten Bereichen gegen Client-Server-Systeme durchsetzen, liegen vor allem in der erhöhten Ausfallsicherheit, den niedrigeren Kosten bei der Skalierung und in der Möglichkeit, dass jeder Teilnehmer die Serviceleistung des Netzwerks mitgestalten kann.

- **Ausfallsicherheit:** Durch die Dezentralisierung des P2P Netzwerks besteht keine Gefahr, dass durch den Ausfall einzelner weniger Server das komplette Netzwerk zusammenbricht.
- **Kosten bei der Skalierung:** Wenn sich in einem Client-Server Netzwerk die Anzahl der Clients erhöht, werden entsprechend mehr Server benötigt, um die Performanz konstant zu halten. In einem P2P Netzwerk hingegen dient ein neuer Client gleichzeitig als Server. Somit müssen keine zusätzlichen Serverkapazitäten bereitgestellt werden.
  Jedoch muss an dieser Stelle auch erwähnt werden, dass der Bedarf der Serverleistung, die die Peers erbringen, nicht proportional mit ihrer Anzahl steigt, da ein P2P Netzwerk eine Verwaltungsstruktur mit entsprechenden Verwaltungsaufwand benötigt, der überproportional mit der Netzgröße ansteigt.
- **Mitgestaltung:** Die Bedeutung der Möglichkeit zur Mitgestaltung einer Serviceleistung wird durch den Erfolg des WEB 2.0 deutlich. Bei dieser sogenannten 2.

Generation des Internets steht der User im Mittelpunkt. Er ist nicht nur Konsument, sondern er partizipiert an der Gestaltung und dem Inhalt des angebotenen Service. Zu den bekanntesten Webseiten des WEB 2.0 zählen Seiten wie Wikipedia[1], StudieVZ[2] oder YouTube[3].
Bei einer P2P Architektur ist die Möglichkeit zur Mitgestaltung, durch die Eigenschaft, dass jeder Peer Serviceleistungen anbietet, von Natur aus gegeben. Hingegen muss bei einem Client-Server-System diese Eigenschaft erst künstlich geschaffen werden.

Durch die Einführung neuer Systeme, wie dem P2P System, treten auch neue Probleme auf, mit denen sich in Theorie und Praxis auseinandergesetzt werden muss.
Durch das Fehlen einer zentralen Instanz in einem P2P Netzwerk, fehlt auch ein zentrales Kontroll- und Verwaltungsorgan. In einem Client-Server-System können die Server die Clients kontrollieren, für Sicherheit und für eine gerechte Leistungsverteilung sorgen. Dazu können die Server Informationen über das Konsumverhalten einzelner User sammeln und anhand dieser faire Entscheidungen treffen.
Hingegen muss in einem P2P Netzwerk jeder einzelne Peer diese Aufgabe übernehmen und darüber hinaus auch Informationen über das Kooperationsverhalten anderer User sammeln. Damit Peers ihre Entscheidung auf einer möglichst breiten Wissensbasis treffen können, ist es notwendig, dass sie die einzeln gewonnen Informationen untereinander austauschen. Dieser Austausch öffnet zu gleich einen Spielraum für Manipulationsversuche, die es gilt zu verhindern.
Ein System, welches diesen Anforderungen gerecht werden soll, ist ein verteiltes Reputationssystem. Es dient dazu, Informationen über das Verhalten von Usern zu sammeln, die Verteilung dieser Informationen im Netzwerk zu organisieren und sie zu analysieren.
Als Ergebnis dieser Analyse liefert dieses System Reputationswerte. Diese sind ein Maß für die Vertrauenswürdigkeit und Leistungsbereitschaft der User im Netzwerk. Anhand dieser Werte können einzelne Peers bevorzugt oder benachteiligt werden. Dies soll zu einem gerechteren und sichereren Leistungsaustausch innerhalb des P2P Netzwerks führen.

Arbeiten wie [3] haben aufgezeigt, dass P2P Netzwerke gewisse Strukturen aufweisen. Es bilden sich Gruppen von Peers heraus, in denen Peers häufiger miteinander Transaktionen[4] durchführen als mit Peers außerhalb dieser Gruppe.
Diese Strukturen kann sich ein verteiltes Reputationssystem zu nutze machen, um Informationen gezielt an User im Netzwerk zu versenden und Informationen bei der Reputationsberechnung unterschiedlich zu gewichten.

[1] www.wikipedia.de
[2] www.studivz.de
[3] www.youtube.com
[4] Als Transaktionen wird der Austausch von Leistungen zwischen zwei Peers bezeichnet.

In dieser Arbeit sollen verschieden Reputationsverfahren vorgestellt werden und unter dem Aspekt der Clustereigenschaften eines P2P Netzwerks untersucht werden. Zudem sollen neue Möglichkeiten aufgezeigt werden, die sich gezielt diese Clustereigenschaften zu nutze machen.

2.2. Hintergrund

Bevor einzelne Reputationsverfahren vorgestellt werden, sollen in diesem Abschnitt wichtige Grundlagen vermittelt werden, die dem besseren Verständnis der Problematik dienen. In 2.2.1 werden die eingangs erwähnten Strukturen eines P2P Netzwerks genauer betrachtet. In 2.2.2 wird mit der Spieltheorie ein soziales Phänomen beschrieben, welches den Erfolg eines Reputationssystem in einem P2P Netzwerk maßgeblich beeinflussen kann. Des Weiteren wird in 2.2.3 ein Einblick in die Public Key Kryptographie geliefert, die für ein sicheren Informationsaustausch innerhalb des P2P Netzwerks erforderlich ist. In 2.2.4 wird auf die Familie der kollaborativen Filter Algorithmen, zu denen auch die Reputationsverfahren gehören sowie die Recommander Systeme, die im weiteren Verlauf dieser Arbeit Verwendung finden, eingegangen.
Schließlich werden in 2.2.5 die allgemeinen Eigenschaften von Flussgraphen und die Berechnung des maximalen Flusses erläutert. Diese finden bei einigen Reputationsverfahren Verwendung.

2.2.1. Die Struktur von P2P Netzwerken

An dieser Stelle sollen die Strukturen von P2P Netzwerken genauer betrachtet werden. Im ersten Abschnitt 2.2.1.1 wird auf die Clustereigenschaften von P2P Netzwerken eingegangen. In 2.2.1.2 wird das Modell des Small World Networks erläutert. Dieses Modell beschreibt unter anderem elementare Struktureigenschaften von P2P Netzwerken. Im Abschnitt 2.2.1.3 wird auf die statistische Verteilung des Datenangebots der Peers eingegangen.

2.2.1.1. Clustereigenschaften von P2P Netzwerken

Die Netzwerkstruktur eines P2P Netzwerk kann als ein Graph, die Peers als Knoten und Transaktionen zwischen den Peers als Kanten abstrahiert werden.
In der Graphentheorie werden Cluster als Cliquen oder auch als Subgraphen bezeichnet. Sie besitzen eine hohe Dichte, d.h. es existieren viele Kanten zwischen den Knoten innerhalb eines Subgraphen. Die Subgraphen wiederum sind mit nur wenigen Kanten untereinander verbunden. Da im Graphenmodell Kanten Transaktionen zwischen den Peers beschreiben, werden Cluster, die in diesem Graphen auftreten, als Transaktionscluster

bezeichnet. Peers in dem selben Transaktionscluster führen weit aus häufiger miteinander Transaktionen durch, als Peers zwischen zwei verschiedenen Transaktionsclustern.
Die Ursache für die Bildung solcher Transaktionscluster ist vielschichtig. Eine Möglichkeit diese zu gliedern wird im Folgenden aufgeführt.

- **Infrastructure-Clustering:**
  Auf unterster Ebene kann die Infrastruktur des Netzwerks das Transaktionsclustering beeinflussen. Falls Peers schnelle Verbindung zueinander haben, sind sie in der Lage viele Transaktionen miteinander durchzuführen.
  Im Zusammenhang mit Reputationssystemen ist dabei zu beachten, dass ein Reputationssystem dafür sorgen muss, dass Peers, die über eine schnelle Verbindung verfügen, langsame Peers nicht ausnutzen. In [4] wird ein Szenario beschrieben in dem zwei Infrastrukturcluster existieren, ein schnelles Universitätsnetzwerk und ein außerhalb liegendes langsameres Cluster. Das Reputationssystem muss nun verhindern, dass Peers, die im Universitätsnetzwerk ohne große Mühen hohe Leistung erbringen und mit den dadurch einfach gewonnen guten Reputation, bei den Peers außerhalb des Universitätsnetzwerks bevorzugt werden. Hingegen müssen Peers, die sich außerhalb des schnellen Netzwerks befinden, mühselig eine gute Reputation erarbeiten und könnten dadurch benachteiligt werden.

- **Content-Clustering:**
  Peers die Interesse an ähnlichen Daten aufweisen und diese untereinander austauschen liegen im selben Contentcluster. Diese Daten entsprechen den Leistungen die ein Peer konsumiert bzw. anderen Peers zur Verfügung stellt.
  In [5] wurde festgestellt, dass Peers sich in der Regel für relativ wenige und meist sogar nur für ein Contentcluster interessieren. Diese Informationen beruhen auf der Untersuchung der Verteilung von Musikdateien in einer P2P Tauschbörse. Sie ordneten jede Musikdatei in eine von 26 verschiedenen Kategorien ein und haben festgestellt, dass fast 24% aller Benutzer Musikdateien aus nur einer Kategorie besitzen und 90% Dateien aus 8 oder weniger Kategorien. Diese Verteilung wird in Grafik 2.1 dargestellt. Auf der x-Achse wird die Anzahl der Cluster in denen ein Peer aktiv ist abgebildet und auf der y-Achse die Anzahl der Peers.

  Leider betrachtet diese Untersuchung nur einen Teil eines P2P File-Sharing-Netzwerks und lässt sich deshalb nur schlecht verallgemeinern, so wurden hier nur die Musikinteressen der User untersucht. Bei vielen P2P Tauschbörsen wird nicht nur Musik getauscht, sondern unter anderem auch Filme. Die Eigenschaft, dass User, die sich für die gleiche Musik interessieren auch für die gleichen Filme interessieren, ist nicht

[5] Leider fehlen in der Quelle [5] die Beschriftungen des Graphen. Die Interpretation der Werte ist dem Text zu entnehmen.

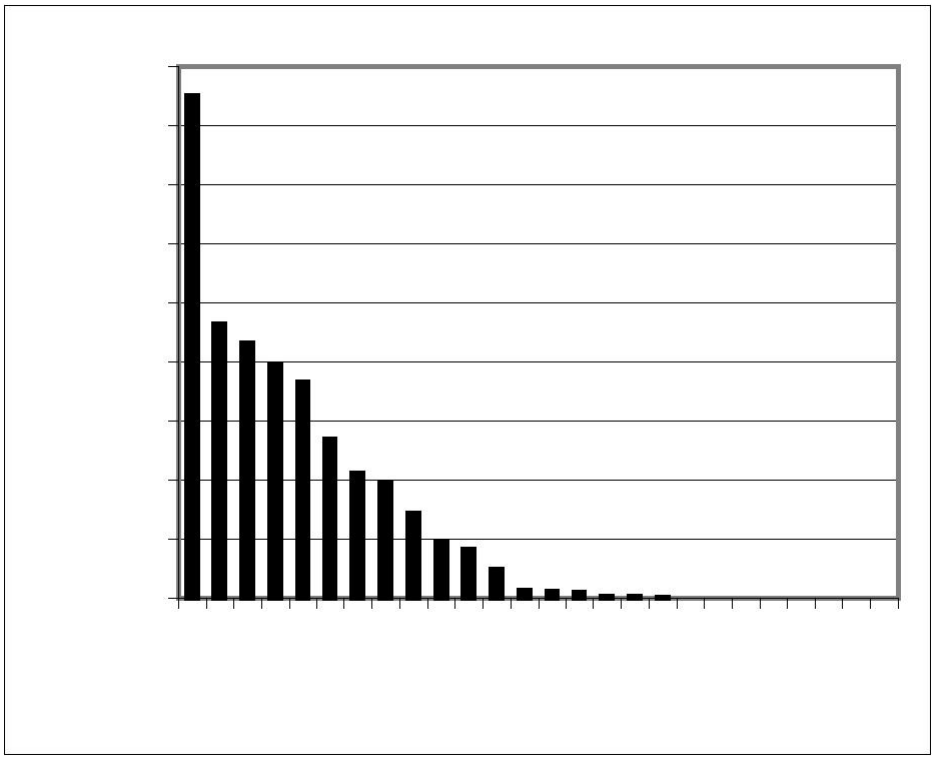

Abbildung 2.1.: Anzahl der Peers, die in x Clustern aktiv sind [5] [5]

zwangsläufig gegeben.

- **Reputation-Clustering:**
  Auf oberster Ebene zur Bildung von Transaktionsclustern liegt das Reputationclustering. Reputationscluster werden durch das Reputationssystem geprägt, so tauschen Peers die einander vertrauen häufiger Daten aus, als Peers die einander nicht vertrauen.
  Ein gutes Reputationssystem muss dafür sorgen, dass Peers, die neu in das Netzwerk kommen, schnell von Vertrauensgruppen aufgenommen werden, d.h. dass sie von vertrauenswürdigen Peers erfahren, welchen Peers sie trauen können. Die Peers, die sich schon länger im Netzwerk befinden, müssen schnell an Informationen gelangen, um die Vertrauenswürdigkeit des neuen Peers richtig einschätzen zu können.
  Des Weiteren muss ein gutes Reputationssystem in der Lage sein, einen bösartigen Peer schnell aufzuspüren und dafür sorgen, dass er aus dem Reputationscluster ausgestoßen wird, d.h. dass alle Peers in dem Reputationscluster möglichst schnell erfahren, dass sie ihm nicht vertrauen können und deshalb keine weiteren Transaktionen mit ihm durchführen.

2.2.1.2. P2P Network als Small World Network

Die Small World Theorie basiert auf ein Experiment das 1967 von Stan Milgram durchgeführt wurde und dessen Ergebnis unter dem Namen "Six Degrees of Seperation" bekannt wurde [6]. Dabei hat er 160 Personen aus dem Süden der US-Westküste ausgewählt und ihnen die Aufgabe zugeteilt, jeweils ein Päckchen an eine ihnen unbekannte Person, die an der nördlichen Ostküste lebt, zu schicken. Die Personen durften jedoch das Päckchen nur an ihnen bekannte Personen weiter geben. Milgram schätzte das über 100 Personen nötig wären, um das Päckchen quer durch die USA zu schicken. Jedoch wurde erstaunlicherweise festgestellt, dass weniger als 6 Personen nötig waren, dabei sind natürlich nur die Päckchen berücksichtigt worden, die angekommen sind.
Die Eigenschaft von kurzen Entfernungen innerhalb eines Graphen ist eine Eigenschaft des Small World Networks, jedoch reicht sie nicht als alleiniger Bedingung aus. In [7] wird ein hoher Clusterkoeffizient als zusätzliche Eigenschaft eines Small World Network genannt. Er gibt an, wie stark einzelne Knoten und deren Nachbarn untereinander verbunden sind. Es gibt verschiedene Formalisierungen zur Berechnung des Clusterkoeffizienten. Die folgende Formalisierung 2.1 stammt von Watts und Strogatz [8].

Definition 2.1 *Sei $G = (V, E)$ ein Graph mit der Menge V als Knoten und der Menge E als Kanten. Die Nachbarn[6] eines Knotens v_i seien definiert durch $V_i = v_j : v_j \in V, e_{ij} \in E$. Der Grad (degree) des Knotens v_i ist definiert als $d_i = |V_i|$. D_i entspricht der maximalen Anzahl Kanten zwischen den Nachbarn des Knotens i. In einem gerichteten Graphen entspricht $D_i = d_i(d_i - 1)$. $E_i = \{e_{jk} : v_j, v_k \in V_i, e_{jk} \in E\}$ ist die Menge aller Kanten zwischen den Nachbarn von v_i. Der Cluster-Koeffizient sei definiert als $C = \frac{\sum_i C_i}{|V|}$, mit $C_i = \frac{|E_i|}{D_i}$.*

Ein hoher Clusterkoeffizient wurde in vielen ganz unterschiedlichen Netzwerken nachgewiesen. Besonders im Bereich der sozialen Netzwerke wurde er immer wieder beobachtet. So gibt es einige Internetseiten, die dieses Phänomen anschaulich verdeutlichen, hier sei kurz die Bacon Numbers[7] und XING[8] verwiesen.

Aus Sicht der Graphentheorie entspricht ein Small World Network Graph, einer Kombination von einem regulären und einem Zufallsgraphen. In Abbildung 2.2 wird dies verdeutlicht.
Der reguläre Graph besitzt einen hohen Clustering-Koeffizienten, was die Cliqueneingenschaft des Small World Networks widerspiegelt. Jedoch besitzt ein regulärer Graph nicht die Eigenschaft, dass die Entfernung zwischen zwei Knoten sehr niedrig ist.

[6] Für den Clusterkoeffizienten sind im Kontext der P2P Netzwerke die Nachbarn Transaktionspartner, also Peers die untereinander Leistungen ausgetauscht haben
[7] www.oracleofbacon.org
[8] www.xing.com

Ein Zufallsgraph wiederum besitzt eine kurze Entfernung zwischen den Knoten, jedoch keinen hohen Clustering-Koeffizienten.
Um die beiden positiven Eigenschaften der beiden Graphen zu kombinieren, kann man einen regulären Graphen erstellen und zufällige Kanten, zu zufällig anderen Knoten umbiegen. Diese Kanten führen dazu, dass die Entfernungen zwischen den Knoten sinken und können deshalb auch als Abkürzungen betrachtet werden.

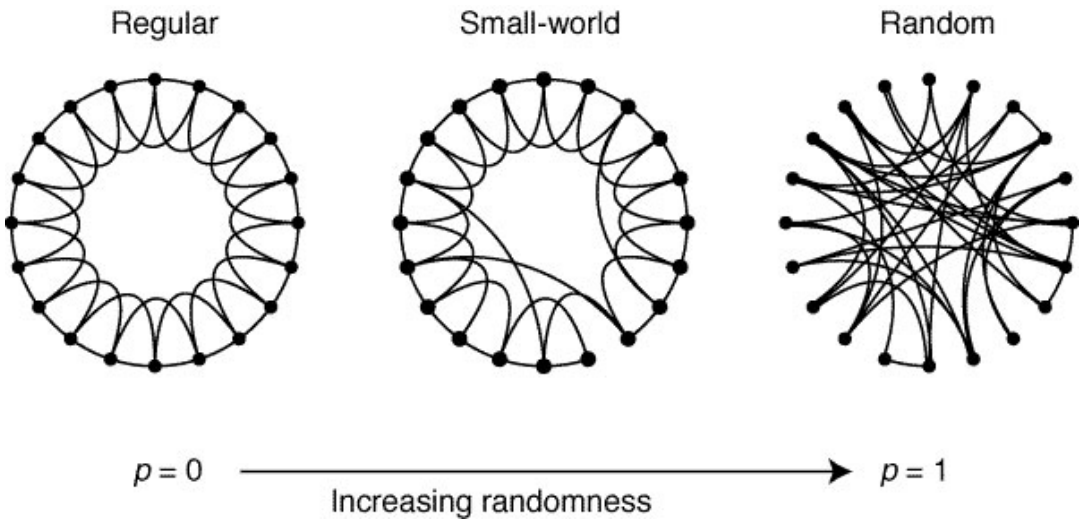

Abbildung 2.2.: Darstellung eines regulären, small-world und eines zufälligen Graphen [1]

Das Small World Modell, lässt sich auf viele P2P Anwendungsgebiete übertragen. Dies wurde unter anderem in einer Studie über das Freenet Netzwerk in [9] festgestellt.
Eine Tauschbörse weist eine große soziale Komponente auf, d.h. die Interessen eines Peers entsprechen den sozial geprägten Interessen eines Menschen. Deshalb eignet sich eine P2P Tauschbörse, um auf ein Small World Network Modell abgebildet zu werden.
Die beiden Eigenschaften des Small World Network Graphens lassen sich leicht auf P2P Tauschbörsen übertragen. Die Eigenschaft des hohen Clusterings wurde durch [5] erläutert und die Eigenschaft der kurzen Wege lässt sich dadurch erklären, dass Peers, die in verschiedenen Clustern agieren, als Abkürzungen dienen und somit die Entfernungen innerhalb des Netzwerks verkürzen. Dies wurde in [4] ansatzweise anhand einer P2P Tauschbörse untersucht, hier wurde festgestellt, dass die Entfernung der Transaktionspartnern bei 60% aller Transaktionen bei 2 oder weniger lag. Das bedeutet, dass die Transaktionspartner sich entweder direkt oder über einen Nachbarn kennen. Leider werden bei dieser Untersuchung keine weiteren Angaben über die Entfernung der restlichen 40% gemacht.
Diese Small World Eigenschaften sind nicht nur für P2P Tauschbörsen spezifisch, sondern können auch bei anderen P2P Netzwerken wiedergefunden werden, jedoch muss dies für jedes P2P Netzwerk individuell überprüft werden.

2.2.1.3. Statistische Verteilung der Daten in P2P Netzwerken

Verschiedene Studien haben ergeben, dass ein Großteil der Daten in einem P2P Netzwerk von wenigen Providern zur Verfügung gestellt werden [10]. Dieses typische Verteilung wird als Pareto-Verteilung bezeichnet. Ihren Ursprung hat sie in der Finanzmathematik und wurde von Vilfredo Pareto entdeckt. Er untersuchte die Verteilung des Kapitals in Italien und stellte fest, dass 20% der Einwohner im Besitz von 80% des Kapitals sind. Neben dem Gebiet der Finanzmathematik wurde diese Verteilung in vielen weiteren Anwendungsbereichen entdeckt.
Die Bestimmung einer Pareto-verteilten Zufallszahl erfolgt durch,

$$P(x) = c * (1 - x)^{\frac{-1}{\tau}} \tag{2.1}$$

wobei x eine gleichverteilte Zufallszahl im Intervall $[0 \dots 1]$ ist. τ und c sind Trimfaktoren, die dem entsprechenden Anwendungsszenario angepasst werden müssen.

2.2.2. Spieltheorie

Die Spieltheorie befasst sich mit der Frage, inwieweit autonome Einheiten unter gewissen Rahmenbedingungen miteinander kooperieren, wenn sie auf ihren eigenen Vorteil bedacht sind.
In einem P2P Netzwerk definiert das Reputationssystem diese Rahmenbedingungen. Ein gutes Reputationssystem muss dafür sorgen, dass sich ein positives Kooperationsverhalten vorteilhaft für einen Peer auswirkt und entsprechend, dass unkooperative Peers bestraft werden. Auf Grund dieses Zusammenhangs bildet die Spieltheorie den Grundbaustein für jedes Reputationssystems in einem P2P System und Bedarf deshalb einer genaueren Betrachtung.

2.2.2.1. Gefangenendilemma

Das Gefangenendilemma ist zentraler Bestandteil der Spieltheorie. Es ist ein Paradoxon, das zeigt, wie individuell optimale Entscheidungen zu einem kollektiv schlechten Ergebnis führen können.
In diesem anschaulichen Beispiel geht es um zwei Gefangene, die unter Verdacht stehen gemeinsam eine Straftat begangen zu haben. Beide werden vor die Entscheidung gestellt ein Geständnis abzugeben, um sich selber zu entlasten und damit den anderen zu belasten. Beide können sich nicht absprechen und müssen unabhängig von einander eine Entscheidung treffen. Wenn beide schweigen und kein Geständnis abgeben, können beide nur auf Grund von Indizien zu jeweils zwei Jahren Haft verurteilt werden. Wenn einer von beiden gesteht, wird seine Strafe auf ein Jahr reduziert und die Strafe für den anderen auf fünf Jahre festgelegt. Falls jedoch beide gestehen und sich damit gegenseitig belasten,

steigt die Strafe für beide auf jeweils vier Jahre an.
Können sich beide nicht blind vertrauen, führt dies dazu, das beide gestehen. Da keiner von beiden davon ausgehen kann, dass der andere schweigt. Paradox ist nun, dass die individuell rational sinnvollste Entscheidung nicht zu einem für das Kollektiv optimalen Ergebnis führt, welches erreicht werden würde, wenn beide schweigen würden.

2.2.2.2. Vertrauen

Eine wichtige Rolle spielt, bei der Entscheidung den kollektiven Nutzen über den eigenen Nutzen zu stellen, das Vertrauen. Wenn sich beide Gefangene vertrauen könnten, würden beide nicht gestehen, um so die kollektiv beste Entscheidung zu treffen. Wenn sie sich jedoch misstrauen und befürchten von ihrem Kumpanen betrogen zu werden, müssen sie die für sich beste Entscheidung treffen und gestehen, um der Gefahr der fünf Jahre Strafe aus dem Weg zu gehen.
Daraus kann man schließen, dass Vertrauen eine hohe Relevanz dafür hat, ob ein optimales Ergebnis erreicht wird. Dieses Vertrauen kann nur Aufgrund von gesammelten Erfahrungen bzw. Beurteilungen anderer entstehen und der Gewissheit, dass für jeden die Entscheidung mit dem höchsten kollektiven Nutzen auch den höchsten Eigennutzen besitzt.
Um Vertrauen in einem P2P Netzwerk aufbauen zu können, müssen Informationen über das Verhalten der Teilnehmer ausgetauscht werden. Dabei ist es wichtig, zwischen dem Austausch von Erfahrungen[9] und dem Austausch von Beurteilungen[10], wie sie hier benutzt werden, zu unterscheiden. Erfahrungen sind Informationen über stattgefunden Ereignisse. Hingegen ist die Beurteilung das Ergebnis der Auswertung dieser Informationen. Sie kann im Gegensatz zu den Erfahrungen durch die subjektive Wahrnehmung des Wertenden verfälscht werden und ist entsprechend weniger vertrauenswürdig.
Die Beurteilungen die von anderen Peers eingeholt werden, müssen in Frage gestellt werden und je nachdem inwieweit sie dem Betrachter als plausibel erscheinen, werden sie von ihm gewichtet.
Damit die ausgetauschten Erfahrungen authentisch sind und nicht manipuliert werden können, ist es notwendig, dass jede Transaktion zwischen zwei Peers protokolliert wird und dieses Protokoll von den beiden Handlungspartner signiert wird, um dessen Authentizität verifizieren zu können. Der Nachteil dieser Methode ist, dass ein zusätzlicher Overhead durch das Protokollieren entsteht.

[9] Erfahrungen entsprechen den Transaktionsinformationen in P2P Netzwerken
[10] Beurteilung und Bewertung werden in dieser Arbeit synonym verwendet

2.2.3. Public Key Kryptographie

Public Key Kryptographie ist eine weit verbreitete Technik zur Verschlüsselung von Daten und um digitale Signaturen zu erstellen. Sie ist auch als asymmetrische Kryptographie bekannt. Im Gegensatz zur symmetrischen Kryptographie besitzt der User bei der asymmetrische Kryptographie ein Paar von Schlüsseln, dem öffentlichen Schlüssel der im Netzwerk verbreitet wird und dem privaten Schlüssel, der geheim gehalten wird.
Bei Verwendung der symmetrischen Kryptographie muss der User für jeden Kommunikationspartner einen neuen Schlüssel generieren und diesen geheim mit dem Kommunikationspartner austauschen. Diese Eigenschaft macht die symmetrische Kryptographie unpraktikabel für große Netzwerke, wie einem P2P Netzwerk, weil jeder User einen Schlüssel für jeden Kommunikationspartner benötigt. Dies würde zu einem exponentiellem Wachstum der Anzahl der Schlüssel im Netzwerk führen.
In der asymmetrischen Kryptographie muss der User lediglich seinen öffentlichen Schlüssel im Netzwerk verbreiten. Diesen Schlüssel benutzen andere User, um dem Besitzer des öffentlichen Schlüssel verschlüsselte Daten zu schicken. Dieser entschlüsselt diese wiederum mit seinem passenden privaten Schlüssel.
In dem Anwendungsgebiet der digitalen Signatur verschlüsselt der User seine Nachricht mit seinem privaten Schlüssel. Dies ist der eigentliche Signaturvorgang. Andere User können die Authentizität dieser Signatur überprüfen, indem sie die Nachricht mit dem passenden öffentlich Schlüssel entschlüsseln.

2.2.4. Kollaboratives Filtern

Das Kollaborative Filtern ist ein Analyseverfahren, bei dem vom Verhaltensmuster bzw. der Meinung einer Gruppe auf das zukünftige Verhalten einzelner Personen geschlossen wird. Bei diesem Verfahren geht man von folgender Annahme aus.
Das Verhalten einer Person, welches in der Vergangenheit dem Verhalten einer Gruppe ähnelte, wird auch in Zukunft diesem Verhalten ähneln. Ein anschauliches Beispiel für solch ein Verfahren ist die Kaufempfehlung von Amazon[11], die unter dem Slogan steht: "Kunden, die diesen Artikel gekauft haben, haben auch jene Artikel gekauft!" Diese Empfehlung entspricht dem Grundkonzept des kollaborativen Filterns.
Beim kollaborativen Filtern wird zwischen aktivem und passiven Filtern unterschieden. Beim passivem Filtern werden die User von einer zentralen Einheit überwacht. Diese zeichnet das Verhalten der User auf und wertet sie aus. Solch eine zentrale Einheit ist in der Regel der Server der einen Service anbietet.
Hingegen sammelt beim aktiven Filtern jeder User lokal seine eigene Erfahrungen bzw.

[11] www.amazon.de

Bewertungen und tauscht sie mit anderen Usern aus. Die Auswertung erfolgt durch jeden einzelnen User und nicht mehr durch die zentrale Verwaltungseinheit. Diese ist dann nicht mehr von Nöten.
Der Nachteil beim aktiven Filtern im Vergleich zum passiven Filtern ist, dass die Berechnung von Reputationen deutlich aufwändiger ist, da die Informationen erst einmal gesammelt und ausgetauscht werden müssen, bevor sie ausgewertet werden können. Erschwerend kommt hinzu, dass den Informationen, die zwischen den Usern ausgetauscht werden, nicht zwangsläufig vertraut werden kann, da sie manipuliert sein könnten und müssen deshalb in Frage gestellt werden.
Beim passiven Filtern sind die Nachteile hingegen weitaus elementarer, falls in der Umgebung, in der das kollaboratives Filtern stattfinden soll, gar keine zentrale Überwachungseinheit existiert, ist ein passives Filtern technisch gar nicht realisierbar.
Unabhängig davon wird zwischen explizitem und impliziten Filtern unterschieden. Beim implizitem Filtern wird die Bewertung durch das Userverhalten ohne sein Eingreifen gemacht.
Beim explizitem Filtern gibt der User gezielt eine Bewertung ab. Jedoch kann dabei durch die subjektive Wahrnehmung, die Bewertung verfälscht werden. Dies gilt es bei der Auswertung entsprechend zu berücksichtigen.
Bei der Definition eines P2P Protokolls, kann kein konkreter Einfluss auf die implizite Bewertung des Clients genommen werden. So kann jeder User seinen persönlichen Client verwenden, dessen implizite Bewertungslogik sich von den anderen unterscheidet und darüber hinaus kann ein User seinen Client so manipulieren, dass dessen impliziter Bewertungsmechanismen durch die Subjekte Meinung des User verfälscht wird, d.h. man kann in einem P2P Netzwerk nicht davon ausgehen, dass Bewertungen rein objektiv gemacht wurden.
Tabelle 2.1 stellt eine Übersicht über die Kombinationen der verschieden kollaborativen Filter Ausprägungen da.

	Aktiv	Passiv
implizit	Bewertung und Auswertung übernimmt der Client (Bsp. BitTorrent Rating)	Bewertung und Auswertung übernimmt der Server (Bsp. Amazon Produktempfehlung)
explizit	Bewertung übernimmt der User und Auswertung übernimmt der Client (Bsp. Qualitätsbewertung in P2P Filesharing)	Bewertung übernimmt der User und Auswertung übernimmt der Server (Bsp. ebay Userbewertung)

Tabelle 2.1.: Übersicht über die verschiedenen Ausprägungen des kollaborativen Filtern

2.2.4.1. Reputationssysteme

Neben dem zuvor in der Einleitung vermittelten Überblick über Reputationssysteme, soll an dieser Stelle ein strukturierter Einblick in Reputationssysteme gegeben werden.
Zunächst soll jedoch der Begriff der Reputation genauer erläutert werden.
Die Reputation ist das Ansehen, das eine Person bei anderen Person genießt. Dieses Ansehen wird zum einen durch seine vergangenen Handlungen und zum anderen durch ihr Umfeld geprägt, indem sie interagiert.
Übertragen auf das reale Leben kann ein Menschen sich Ansehen erarbeiten, indem er so handelt, dass er seine Mitmenschen beeindruckt. Das Umfeld in dem er verkehrt spielt dabei ein entscheidende Rolle, so z.B. erhält ein Mensch mehr Ansehen, wenn er in Kreisen verkehrt, in denen seine Mitmenschen ebenfalls ein hohes Ansehen genießen.
Man könnte nun annehmen, dass Personen, die ein hohes Ansehen genießen auch sehr vertrauenswürdig sind. Prinzipiell stimmt diese Annahme, jedoch sollte dieses Vertrauen auch hinterfragt werden.
Vor allem der Einflussfaktor des Umfeldes erschwert die Einschätzung des Vertrauens und muss entsprechend in Relation mit dem Umfeld des Betrachters gesetzt werden. So wäre es z.B. nicht ratsam einem Mathematiker der in entsprechenden Kreisen über eine hohe Reputation verfügt in medizinischen Fragen zu vertrauen.

In einem offenen System, wie einem P2P Netzwerk, dessen Grundprinzipien auf einem Geben und Nehmen zwischen den Teilnehmern innerhalb des Netzwerks beruht, besteht die Gefahr, dass einige Individuen versuchen auf Kosten Anderer, ihren eigenen Profit zu steigern. Vor allem eine hohe Anonymität und das beschränkte Wissen des einzelnen über sein Umfeld begründen diese Gefahr.
Ein Reputationssystem dient dazu, eine gerechte Interaktionsbasis zu schaffen und die Gefahr des Betruges zu minimieren. Es muss dafür sorgen, dass jeder einzelne Teilnehmer fair behandelt wird. Dies bedeutet, dass jeder Teilnehmer eine gewisse Gegenleistung, für die Leistung, die er der Gemeinschaft zur Verfügung gestellt hat, erhält. Dies wiederum setzt ein gewisses Vertrauen zwischen den Teilnehmern voraus.

Die Hauptaufgabe eines Reputationssystem liegt darin, User anhand ihres Verhaltens zu bewerten und zu bestimmten wie viel Vertrauen in sie gesetzt werden kann. Diese Bewertung wird durch den Reputationswert repräsentiert und mittels einer Reputationsfunktion bzw. eines Reputationsberechnungsalgorithmuses anhand von gesammelten Informationen über andere User berechnet. Diese Informationen können Erfahrungen oder Beurteilungen Dritter sein.
In einem P2P Netzwerk sind diese Informationen nicht frei verfügbar. Deshalb reicht hier die Definition einer Reputationsfunktion nicht aus. Zusätzlich muss bestimmt werden, wie

diese Informationen zwischen den einzelnen Peers ausgetauscht werden soll. Des Weiteren muss definiert werden, wie ein Peer einen Provider bzw. Consumer anhand dessen Reputationswert auswählen soll. Wie später noch deutlich wird, ist die Auswahl des Peers mit der höchsten Reputation nicht immer die beste Wahl.
Somit besteht ein Reputationssystem innerhalb eines P2P Netzwerk aus einer Reputationsfunktion, einer Informationsverteilungsstrategie und einer Provider- bzw. Consumerauswahlstrategie.

Viele Algorithmen zur Berechnung der Reputation sind eine Spezialisierung des kollaborativen Filterns.
Die verschiedenen Berechnungsalgorithmen unterscheiden sich vor allem in ihrer Komplexität. So interagieren z.B. Peers miteinander und nach jeder Transaktion bewerten sie sich gegenseitig. Diese Bewertungen werden zwischen den Peers ausgetauscht und ausgewertet. Ein einfacher Algorithmus gewichtet alle positiven und negativen Bewertungen gleich und summiert sie auf, ungeachtet dem Umfeld indem sie erstellt wurden. Solch ein Algorithmus wird von dem ebay[12] Reputationssystem für die Käufer- und Verkäuferbewertung benutzt.
Ein weitaus anspruchsvollerer Ansatz ist der PageRank [11] Algorithmus. Dieser geht davon aus, dass Bewertungen, die von Individuen mit hoher Reputation abgegeben wurden, höher gewichtet werden sollten, als Bewertungen die von Individuen mit niedriger Reputation abgegeben wurden. Somit wird zwar das Umfeld indem die Bewertung abgeben wurde berücksichtigt, jedoch wurde diese Betrachtungsweise nicht in Relation zum Umfeld des Betrachters gesetzt. Eine detaillierte Vorstellung von verschiedenen Reputationsberechnungsalgorithmen wird im Abschnitt 2.3 gegeben.

2.2.4.2. Recommender System

Eine anderes Anwendungsgebiet für kollaborative Filter Algorithmen sind die Recommender Systeme. Dies sind Empfehlungssysteme, die einem User auf Grund seines vergangenen Verhaltens und dem Verhalten ähnlicher User Produkte empfehlen. Ein anschauliches Beispiel ist hier das eingangs erwähnte Empfehlungssystem von Amazon.
Zentraler Bestandteil des dort verwendeten Algorithmuses ist es, die Ähnlichkeit von Usern zu bestimmen. Um diese Ähnlichkeit zu bestimmen, muss zuerst das Verhalten eines User mathematisch erfasst werden. Dazu werden Produktvektoren erstellt, bei denen jeder Eintrag der Beziehungsbewertung zwischen User und einem Produkt entspricht. Diese Bewertung kann durch explizites oder implizites Filtern entstanden sein.
Schließlich müssen mittels einer Ähnlichkeitsmetrik die beiden Produktvektoren mitein-

[12] www.ebay.de

ander verrechnet werden. Als Ergebnis erhält man die Ähnlichkeit der beiden User. Die Wahl der Ähnlichkeitsmetrik ist stark abhängig vom Anwendungsfall und in welcher Form die Produktvektoren vorhanden sind.

2.2.5. Flussgraphen und maximaler Fluss

In dieser Arbeit wird ein Reputationssystem vorgestellt, das auf der Berechnung des maximalen Flusses innerhalb eines Flussgraphens beruht. Deshalb sollen an dieser Stelle auf die grundlegenden Eigenschaften des Flussgraphen und des maximalen Flusses eingegangen werden.
Ein Flussgraph ist ein gerichteter Graph ohne Mehrfachkanten mit zwei ausgezeichneten Knoten s (Quelle), einer t (Senke) und ein einer Kapazitätsfunktion, die jeder Kante eine Kapazität aus den nicht reellen Zahlen zuordnet. Ein Fluss ist eine Funktion, die von den Kanten im Netzwerk in die Menge der nicht reellen Zahlen abbildet. Sie gibt an wie viele Einheiten über eine Kante fließen. Für diesen Fluss gelten zwei Bedingungen:

- Der Fluss über eine Kante darf deren Kapazität nicht überschreiten.
- Es gilt das Krichhoffsche Flusserhaltungsgesetz für alle Knoten bis auf Quelle und Senken. Die Summe der einströmende Flüsse ist gleich der Summe der ausströmenden Flüsse.

Der maximale Fluss entspricht dem minimalen Schnitt des Graphen. Ein Schnitt ist eine Teilmenge von Knoten und deren Kanten die s enthält aber nicht t. Die Kapazität eines Schnittes ist die Summe der Kapazitäten der aus dem Schnitt herausführenden Kanten. Der minimale Schnitt eines Graphen ist ein Schnitt, dessen Kapazität minimal ist.[13]
Der maximale Fluss von einem Knoten A zu einem Knoten B in einem Graphen G wird notiert als $mf(A \rightarrow B, G)$.

2.3. Kollaboratives Filtern in verteilten Netzwerken

In diesem Kapitel werden die Funktionsweisen und Einsatzgebiete verschieden kollaborative Filter Verfahren in P2P Netzwerken vorgestellt. Mit 2.3.1 wird ein einfaches und sehr verbreitetes Reputationssystem vorgestellt, dass jedoch keine Informationen zwischen den Peers austauscht.
Erst mit den beiden Reputationssystemen 2.3.2 und 2.3.3 werden Bewertungen zwischen Peers ausgetauscht. Beide Verfahren sind eng miteinander verwandt und basieren auf einer transitiven Reputationsberechnung.
Im Gegensatz dazu werden in 2.3.4 keine Reputationen zwischen den einzelnen Peers ausgetauscht, sondern konkrete Informationen über stattgefundene Transaktionen.

[13] Max-Flow-Min-Cut Theorem

In 2.3.5 wird ein Reputationsberechnungsverfahren vorgestellt, dass anstatt einer transitiven Berechnungsmethodik die Reputation mithilfe eines maximalen Flusses berechnet. Schließlich wird in 2.3.6 ein Recommander System vorgestellt, das einem User für ihn interessante Dateien empfiehlt.

2.3.1. Tit-for-tat

Tit-for-tat Reputationssysteme gehören zu den simpelsten Reputationssysteme. Sie zählen zu der ersten Generation von Reputationssystemen in P2P Netzwerken und finden eine weite Verbreitung, wie bei eMule [12] und BitTorrent [13].
Jeder Peer verwaltet seine eigenen Reputationswerte, die er lediglich aus seinen eigenen Erfahrungen berechnet hat. Erfahrung oder Beurteilungen, die andere Peers gemacht haben, werden nicht berücksichtigt.
Dies macht die Implementierung sehr einfach, es wird kein zusätzlicher Mechanismus benötigt, der für die Verteilung der Informationen zwischen den einzelnen Peers zuständig ist.
Streng genommen handelt es sich bei diesem Verfahren nicht um kollaboratives Filtern und somit auch nicht um ein verteiltes Reputationssystem nach 2.2.4.1, da keine Informationen zwischen den Peers ausgetauscht werden.

2.3.2. Eigentrust

Orientiert am PageRank-Algorithmus [14] wurde der Eigentrust-Algorithmus [15] entwickelt. Das dahinter liegende Prinzip ist, dass jede Peer eine Reputation besitzt, die umso größer ist, je mehr Peers mit hoher eigener Reputation ihn gut bewerten. Grundbaustein zur Berechnung der Reputation ist das transitive Vertrauen. Vertraut ein Peer i einem Peer k, so vertraut er auch den Peers denen k vertraut.
Um diese transitive Reputation berechnen zu können, ist es zuvor notwendig, die lokalen Reputationen, die auf der Erfahrung von direkten Interaktionen basieren, zu bestimmen. Die lokale Reputation $s_{(ij)}$, die Peer i in Peer j setzt, wird berechnet, indem die Anzahl der nicht zufriedenstellenden Leistungen von j an i ($unsat_{(ij)}$) von den zufriedenstellenden Leistungen von j an i ($sat_{(ij)}$) subtrahiert wird.

$$s_{ij} = sat_{(ij)} - unsat_{ij} \tag{2.2}$$

Diese lokalen Reputationen entsprechen der Reputation im Tit-for-tat Verfahren. Beim Eigentrust-Verfahren werden nun die lokalen Reputationen im Netzwerk verteilt. So kann ein Peer die lokalen Reputationen vieler Peers sammeln und sie in einer Matrix C spei-

chern.[14] Bevor diese Matrix für weitere Berechnungen verwendet werden kann, ist es zuvor notwendig die Matrix zu normalisiern, damit die Ergebnisse in den weiteren Rechenschritten konvergieren. Dazu werden die Zeilensummen der Matrix auf 1 normiert. Dadurch erhält man den normierten lokalen Reputationswert c_{ij}.

$$c_{ij} = \frac{max(s_{ij}, 0)}{\Sigma_j max(s_{ij}, 0)} \tag{2.3}$$

Um nun die transitiven Reputationen zwischen den Peers i und j zu berechnen, müssen die Erfahrungen der Peers zu Rate gezogen werden, die mit Peer i und Peer j interagiert haben.

$$t_{ik} = \Sigma_k c_{ik} c_{kj} \tag{2.4}$$

t_{ik} repräsentiert das transitive Vertrauen, das Peer i in Peer k besitzt. Zur Vereinfachung können die gesamten Reputationen die Peer i in andere Peers besitzt in dem Vektor $\vec{t_i}$ zusammengefasst werden.

$$\vec{t_i} = C^T \vec{c_i} \tag{2.5}$$

Will man nun die transitiven Reputation nicht nur über eine Generation berechnen, d.h. will man nicht nur die Bewertungen der Peers einfließen lassen denen man vertraut, sondern auch die Bewertungen von den Peers, denen die vertrauten Peers vertrauen, so kann man die abgegebenen Bewertungen seiner Nachbarn in dem Maße gewichten, indem man ihnen vertraut und es fließen in die Bewertung die lokalen Reputationen von 2 Nachbargenerationen mit ein. Will man alle transitiven Bewertungen berechnen, so lässt sich dies auch mittels Matrixmultiplikation darstellen, indem die Matrix C^T mit sich selbst multipliziert wird. Betrachtet man 2 Nachbargenerationen, so kann man dies wie in 2.6 berechnen.

$$\vec{t_i} = (C^T)^2 \vec{c_i} \tag{2.6}$$

Die Anzahl der Nachbargenerationen kann verallgemeinert werden:

$$\vec{t_i} = (C^T)^n \vec{c_i} \tag{2.7}$$

Wobei n für die betrachteten Nachbargenerationen bzw. für die Traversierungstiefe steht. Falls nun n groß genug ist, konvergiert der Vektor $\vec{t_i}$ aufgrund der Normalisierung 2.3 zum linken Eigenvektor der Matrix C. Dieser Eigenvektor ist ein globaler Reputationsvektor, d.h alle Peers, die mithilfe der gleichen Tokenmenge bzw. dem gleichen lokalen Wissen die Reputationen berechnen, kommen zu einem identischen Reputationsvektor. Dies ist, wie wir noch sehen werden, nicht bei allen Reputationsverfahren der Fall.

[14] In [15] werden zwei Möglichkeiten zur Verteilung der lokalen Reputation vorgestellt, auf die an dieser Stelle jedoch nicht weiter eingegangen werden soll.

2.3.3. Multilevel Tit-for-tat

Der Multilevel Tit-for-tat ist ein Ansatz, der die Vorteile des Tit-for-tat und des Eigentrusts vereint.
Der einfache Tit-for-tat Ansatz hat ein großes Problem mit der Skalierbarkeit der Netzwerkgröße, denn ein Peer begründet seine Entscheidungen nur auf seinen eigenen Erfahrungen. Da dieser in einem großen Netzwerk mit einem sehr kleinen Teil der Peers bereits interagiert hat, besitzt er oftmals keine Informationsbasis um eine Entscheidungen zu treffen, dies trifft vor allem auf die Peers zu, mit denen er noch nicht interagiert hat. Die Konsequenz ist, dass der Peer seine Entscheidungen blind vornehmen muss.
Im Gegensatz dazu benutzt ein Peer bei einem Eigentrust Reputationssystem auch die Erfahrungen, die andere Peers gemacht haben, dies hat zwar den Vorteil, dass ein Peer sein Handeln auf einer breiteren Informationsbasis stützen kann, jedoch besteht die Gefahr, dass die Bewertungen, die er von anderen Peers erhält nicht zwangsläufig für ihn gelten, d.h. die Bewertungen können durch die jeweilige subjektive Wahrnehmung getrübt sein.
In [4] wird ein Beispiel erläutert, bei dem dieses Problem auftritt. In diesem Beispiel wird von einem Szenario mit zwei Clustern ausgegangen, in dem einen Cluster besteht eine sehr schnelle Verbindung zwischen den einzelnen Peers. In dem anderen Cluster sind die Verbindungen deutlich schlechter. Die Verbindung zwischen den beiden Clustern ist ebenfalls langsam. Die Peers in dem schnellen Cluster können ohne großen Aufwand eine riesige Menge an Daten austauschen und entsprechend einfach können sie eine sehr hohe Reputation erhalten. Jedoch können die Peers in dem schlechten Cluster nicht von der Leistung, die der guten Reputation zu Grunde liegt, profitieren, da sie zu den Peers mit der hohen Reputation nur eine schlechte Verbindung besitzen. Dies führt zwangsläufig zu einer unfairen Leistungsverteilung im Netzwerk.
Im Multilevel Tit-for-tat Ansatz [4] wird, ähnlich wie beim Eigentrust Verfahren, eine Matrix angelegt. Doch hier wird sie nicht solange potenziert, bis diese konvergiert, sondern lediglich n mal. Wobei n ein relativ kleiner Wert ist und bestimmt, wie weitreichend die Erfahrung anderer Peers genutzt werden sollen, bzw. von wie vielen Nachbargenerationen die lokalen Reputationen in die eigene Entscheidung mit einfließen sollen. So bedeutet ein Wert 2 für n, dass ein Peer lediglich die Reputationen seiner Nachbarn und deren Nachbarn berücksichtigt.
Da nun nicht mehr der Eigenvektor gebildet wird, sondern ein Reputationsvektor der für jeden Peer unterschiedlich ist, gibt es keinen allgemeinen Reputationsvektor. Zwei Peers, die das gleiche lokale Wissen besitzen, müssen nicht zwangsläufig den gleichen Reputationsvektor berechnen, d.h. der Reputationsvektor ist abhängig von der Perspektive bzw. dem Umfeld des Betrachters.

2.3.4. STEP

In [16] wird das STEP-Protokoll beschrieben. Basierend auf dem Gnutella-Protokoll sorgt es dafür, dass ein Peer möglichst vertrauenswürdige Nachbarn besitzt, um dadurch Nachbarschaften zu schaffen in denen großes Vertrauen herrscht und es so Angreifern möglichst schwer macht.
Bei diesem Protokoll werden nicht, wie bei den voran gegangen Reputationssystemen die subjektiv gebildeten Reputationen weitergeben, sondern es werden die Transaktionen in sogenannten Token protokolliert und diese im Netzwerk verteilt. Dieser Token wird von beiden Transaktionspartnern signiert, um seine Korrektheit und Authentizität zu bestätigen.
Durch diese objektive Weitergabe von Transaktionsinformationen ist es jedem Peer selbst überlassen, wie er die Erfahrungen der anderen Peers auswertet.
Bei dem Gnutella-Protokoll besitzt jeder Peer eine gewisse Anzahl von Nachbarn. Über diese Nachbarn kann ein Peer Suchanfragen absenden und Token austauschen. Unkooperative Nachbarn würden einen Peer schädigen, deshalb ist es für ein Peer wichtig, vertrauenswürdige Peers als Nachbarn zu besitzen. Bei STEP versucht nun jeder Peer anhand seiner gebildeten Reputationen seine Nachbarn auszuwählen. Ist in seiner Nachbarschaft ein Platz frei geworden, so sucht er sich einen neuen Nachbar mit einer möglichst hohen Reputation aus. Dieser Nachbar akzeptiert ihn, falls er wiederum ein hohes Vertrauen in ihn hat. Wenn seine Nachbarschaft voll belegt ist, kündigt er die Nachbarschaft mit einem anderen Nachbar auf, dessen Reputation niedriger ist. Dies sorgt dafür, dass das Vertrauen innerhalb einer Nachbarschaft anwächst.

2.3.5. Stimulating Participation

In [17] wird ein Verfahren für Wireless Community Networks (WCNs) vorgestellt, das darauf abzielt, eine möglichst gerechte Leistungsverteilung innerhalb des Netzwerks zu gewährleisten. WCNs sind großflächige WLAN Netzwerke, deren Hardware von meist privaten Providern unentgeltlich bereitgestellt und gewartet werden. Diese Provider hoffen, durch die zur Verfügungstellung ihres lokalen Internetzugangs, im Gegenzug Zugang zum Internet über andere Provider in der Community zu erhalten, falls sich ihr eigener WLAN Access Point außerhalb ihrer Reichweite befindet.
Ein Provider erhält für die Bereitstellung seines Internetzugangs vom Consumer signierte digitale Belege mit deren Hilfe er wiederum leichter Zugang zum Internet bei anderen Providern erhält.
Um eine gewisse Sicherheit gegenüber Angriffen zu gewährleisten, werden die einzelnen Belege gewichtet. Diese Gewichtung erfolgt mittels eines Flussgraphenalgorithmuses.
Ein Knoten des Graphens entspricht einem Team, welches aus mehreren Provideren besteht. Eine gerichtete Kante von Knoten C (Consumer) nach Knoten P (Provider) ent-

spricht der Bereitstellung des Zugangs eines Mitglieds von Team P an ein Mitglied des Teams C. Die Kantengewichtung entspricht der Summe der zur Verfügung gestellten Bandbreite von Team P an Team C.
Durch die Zusammenfassung der Provider in Teams wird bezweckt, dass ein möglichst homogenes Verhältnis zwischen bereitgestellter und erhaltener Leistung entsteht. Dies bedeutet für den Graphen, dass das Verhältnis der Summe der eingehenden Kantengewichte und der Summe der ausgehenden Kantengewichte einem Verhältnis von 1:1 entspricht.
Es wird vorausgesetzt, dass sich die einzelnen Teammitglieder vertrauen und es wird ihnen überlassen, wie sie innerhalb der Gruppe die erbrachten und erhaltenen Leistungen gerecht verteilen.
In der einfachen Version akzeptiert ein Provider die Anfrage eines Consumers mit der Wahrscheinlichkeit p, die anhand der Funktion 2.8 bestimmt wird.

$$p = min(\frac{mf(P \to C)}{mf(C \to P)}, 1) \tag{2.8}$$

Dieser Ansatz verhindert verschiedene Angriffsmöglichkeiten im Vergleich zu einer ungewichteten Auswertung. Unter diesen Angriffen zählt eine Version der Sybil-Attacke, die im folgenden als Sybil-Attacke Version A bezeichnet wird, bei denen der angreifende Consumer C Scheingeschäfte generiert, indem er imaginäre Knoten C_is, d.h. Teams erzeugt, die ihm Belege ausstellen, die ihm eine nicht bereitgestellte Leistung bescheinigen. Würde man nun alle erbrachten Leistungen eines Consumers ohne Gewichtung aufsummieren, so könnte sich ein Angreifer, durch die Generierung von Belegen aus Scheingeschäften, eine beliebig hohe positive Bewertung verschaffen.
Durch die Berechnung des maximalen Flusses von P nach C fließen jedoch diese Scheingeschäfte nicht in die Bewertung des Consumers mit ein, da die erzeugten C_is nur über ausgehende Kanten verfügen und sie somit nach Definition keine Auswirkung auf den maximalen Fluss von P nach C haben.
Eine weiter Sybil-Attacke, an der die einfache Version des Algorithmuses scheitert, die im folgenden als Sybil-Attacke Version B bezeichnet wird, sieht wie folgt aus. Anstatt dass der Angreifer C sich von seinen imaginäre Knoten Belege ausstellen lässt, stellt er seinen C_is Belege aus und wechselt, wenn er Anfragen an einen Provider stellt, seine Identität zu einem C_i'. Erhält nun dieser C_i' eine Leistung, wird dieser Knoten mit einer ausgehende Kante belastet und verringert seine Chancen eine Leistung von einem Provider zu erhalten. Deshalb wechselt C bei der nächsten Anfrage zu einem anderen C_i'', welcher noch über keine ausgehenden Kanten verfügt. Die Leistungen, die er mit der Identität C_i' erhalten hat, belasten lediglich C_i' und indirekt C, jedoch nicht C_i'' da keine zusätzlicher Fluss von C_i'' nach P über C_i' entstanden ist.
Die erweiterte Version des Algorithmuses soll nun dieser Sybil-Attacke der Version B entgegen wirken, indem versucht wird die Angreifer zu entlarven. Für die Anwendung

dieser erweiterten Version wird vorausgesetzt, dass alle Teams eine möglichst homogene Konsumrate besitzen und das die Wahrscheinlichkeit, dass zwei Teams miteinander interagieren für alle Teams identisch ist. Diese Rahmenbedingung führen zu einem relativ gleichmäßigen Graphen, der keine Clusterstrukturen aufweist. Für diesen Graphen besitzen die Angreifer folgende Eigenschaften, zum einen beträgt die Summe der ausgehenden Kanten eines Angreifer C ein vielfaches der Summe der eingehenden Kanten, des weiteren ist die Entfernung zu einem imaginären C_i höher als zu einem realen C.
Diese beiden Eigenschaften fließen in die Berechnung ein. Hierfür wird der der GMF[15] definiert. Der im GMF verwendete Graph entspricht dem im einfachen Verfahren verwendeten Graphen, jedoch wird jede Kantengewichtung w_e des Graphen durch das Verhältnis $q_{(P,S_e)}$ und durch die Entfernung $d_{(P,S_e)}$ verringert. Das Verhältnis $q_{(P,S_e)}$ berechnet sich aus der Summe aller ausgehenden Kanten des Ursprungsknoten der zu bewertenden Kante, dividiert durch die Summe der ausgehenden Kanten des Providers. Das Kantengewicht w_e wird jedoch nur durch $q_{(P,S_e)}$ dividiert, falls $q_{(P,S_e)} > 1$, ansonsten wirkt sich das Verhältnis nicht auf w_e aus.
Die Entfernung $d_{(P,S_e)}$ entspricht der Anzahl der Kanten die zwischen dem Provider und der Ursprungsknoten S_e der zu bewertenden Kante liegen. Diese Entfernung wird mit w_e verrechnet, indem w_e durch $g^{d(P,S_e)}$ dividiert wird, wobei g eine GMF Konstante ist, die auf 2 festgelegt wurde.
Ein GMF Ergebnis, was deutlich niedriger ausfällt als den GMF Wert den ein Provider gewöhnlicherweise im Netzwerk feststellt, sollte einen Provider darauf hinweisen, dass der Consumer mit einer gewissen Wahrscheinlichkeit ein Angreifer ist. Deshalb akzeptiert er die Anfrage eines Consumer nur mit einer Wahrscheinlichkeit p_{gmf}, die sich nach 2.9 berechnet.

$$p_{gmf} = min(\frac{g \times GMF(P \to C)}{gmf}, 1) \tag{2.9}$$

Sobald der Provider ein positiven GMF berechnet, verändert sich gmf nach 2.10. Für a wurde ein Wert von 0.5 festgelegt.

$$gmf = a \times gmf_{old} + (1 - a) \times GMF(P \to C) \tag{2.10}$$

Die Wahrscheinlichkeit die mit 2.9 berechnet wird, betrachtet nur die Möglichkeit, ob der Consumer ein Angreifer ist, jedoch nicht ob dem Consumer aufgrund seiner erhaltener und erbrachter Leistungen weiter Leistungen zustehen. Deshalb erhält ein potentieller Consumer vom Provider nur eine Leistung mit einer Wahrscheinlichkeit von 2.8, wenn er zuvor mit einer Wahrscheinlichkeit von 2.9 nicht als Angreifer angesehen wurde.

[15] generalized maxflow

2.3.6. DeHinter

DeHinter [18] ist ein Entwurf für ein dezentrales Empfehlungssystem für Filesharing P2P Netzwerke. Die Idee die hinter diesem Ansatz steht ist, dass die Dateien, die sehr populär sind, empfohlen werden. Dabei wird den Dateien ein höheres Gewicht gegeben, die von Usern gespeichert werden, die eine hohe Ähnlichkeit mit dem User haben, dem die Dateien empfohlen werden sollen. Dazu werden die zu empfehlenden Dateien nach einem Empfehlungswert in einer Empfehlungsliste sortiert. Dateien mit einem höheren Empfehlungswert stehen weiter oben in dieser Liste und sind somit interessanter für den User, aus dessen Sicht die Liste erzeugt wurde.
Die Berechnungsfunktion 2.11 des Empfehlungswert $w_{(s_k)}$ einer Datei besteht aus zwei Komponenten, zum einen wird die Popularität $pop(s_k)$ der zu bewertenden Datei in der Nachbarschaft des Users u_x, dem die Datei empfohlen werden soll, bestimmt und zum anderen wird der Grad der Ähnlichkeit $deg(s_k)$ zwischen u_x und seinen Nachbarn bestimmt. Die Popularität einer Datei wird gemessen, indem gezählt wird, wie oft sie von Nachbarn von u_x zur Verfügung gestellt wird. Die Ähnlichkeit zwischen u_x und einem seiner Nachbarn wird berechnet, indem die Anzahl der von beiden Usern zur Verfügung gestellten gleichen Dateien bestimmt wird.
Daraus ergibt sich folgende Funktion zum Berechnen des Empfehlungswertes.

$$w(s_k) = pop(s_k) * deg(s_k) \tag{2.11}$$

Das Experiment das in [18] vorgestellt wurde, bezog sich auf Daten, die innerhalb von 7 Tagen innerhalb des Gnutella file-sharing-Netzwerk gesammelt wurden. Es wurden mehr als 100.000 Datensätze Mittels einer Vergleichstest analysiert. Dabei wurden die Daten in 10 Teilmengen gestückelt. Während den 10 Berechnungsdurchläufen wurde jeweils eine Teilmenge als Trainingsmenge und die Restmengen als Vergleichsmenge herangezogen. Als Ergebnis konnte eine durchschnittliche Genauigkeit von 81% erreicht werden, d.h. 81% aller Empfehlungen haben sich als sinnvoll herausgestellt.

2.4. Ziel der Arbeit

Wie zuvor beschrieben wurde, nutzen Reputationssystemen Strukturen in P2P Netzwerken, um Informationen effektiv im Netzwerk zu verteilen.
Durch die Erkenntnis, dass Peers sich nicht zwangsläufig nur in einem Cluster aufhalten, sondern in vielen verschiedenen und diese fließend ineinander übergehen, entsteht beim Austausch von Reputationsinformationen mit allen Nachbarn ein gewisser Overhead. Nicht jeder Nachbar eines Peers interessiert sich für diese Informationen, denn er muss nicht zwangsläufig in dem selben Cluster liegen, in dem diese Informationen zustande gekommen sind. Demnach muss ein Peer versuchen Informationen nur an die Peers zu

schicken, die im selben Cluster liegen, indem diese Informationen entstanden sind.
Des weiteren kann eine Peer Bewertungen, die von Peers abgegeben wurden, die in dem gleichen Cluster interagieren, wie der zu bewertende Peer, stärker berücksichtigen, als die Beurteilung von Peers die außerhalb des Clusters liegen. Damit ein Peer die Strukturen des Netzwerks analysieren kann, benötigt er eine gewisse Kenntnis über den Aufbau des Netzwerks, um diese Kenntnis zu erlangen, bietet sich ein Informationsaustausch auf Tokenbasis an, dieser kann im Vergleich zu einer reinen Reputationsverteilung, dem Peer zusätzliche Informationen über die Strukturen des Netzwerks liefern.

In dieser Arbeit soll ein tokenbasiertes Reputationsverfahren vorgestellt werden, das die Clustereigenschaften eines P2P Netzwerks ausnutzen soll, um Token gezielt im Netzwerk zu verteilen und diese auszuwerten.
Um die Effektivität dieses Reputationssystem zu messen, soll es mit bereits existierenden Verfahren verglichen werden. Bei diesem Vergleich soll zum einen die geschaffene Fairness der Reputationssysteme untersucht werden und zum anderen ihre Robustheit gegenüber Angriffen getestet werden.
Außerdem soll dieser Vergleich, durch die Simulation verschiedener Ausprägungen des Netzwerkclusterings erfolgen, um eine Vorstellung zu vermitteln, wie stark sich die Netzwerkstruktur auf die Repuationsberechnung auswirkt.

3. Konzept

In 2.2.4.1 wurde ein Überblick über Reputationssystem gegeben und in 2.3 wurden verschiedene Reputationssysteme für P2P Netzwerk, deren Aufbau und Funktionsweise vorgestellt. In diesem Kapitel soll nun ein Reputationssystem vorgestellt werden, welches die eingangs beschriebenen Verfahren mit neuen Ansätzen verknüpft. Dieses neue Reputationssystem soll gezielt die in 2.2.1 vorgestellten Transaktionsclustereigenschaften eines P2P Netzwerks ausnutzen, um den eigenen Wirkungsgrad zu erhöhen.
Die Clustereigenschaften können bei der Entwicklung eines Reputationssystems genutzt werden, um den Informationshorizont eines Peers einzuschränken, d.h. ein Peer braucht nicht über das Verhalten aller Peers im Netzwerk informiert zu sein. Er benötigt lediglich die Informationen, die in den Clustern anfallen, in denen er selbst aktiv ist.
Falls ein Reputationssystem es schafft, diese Eigenschaft sinnvoll auszunutzen, hat dies zur Konsequenz, dass deutlich weniger Informationen zwischen den Peers ausgetauscht werden müssen und somit wichtige Bandbreite eingespart werden kann.
Neben einer optimierten Informationsverteilung kann die Clustereigenschaft auch dafür genutzt werden, um die eigentliche Reputationsberechnung zu optimieren. Ein Peer könnte z.B. den Bewertungen eines anderen Peers mehr vertrauen, wenn er mit diesem häufiger interagiert, bzw. er eine ähnliche Nachbarschaft besitzt, d.h. wenn dieser Peer im gleichen Cluster liegt.
Bevor jedoch das Reputationssystem die Informationen der Clustereigenschaften des Netzwerks nutzen kann, muss es ein entsprechendes Wissen über den Aufbau der Clusterstrukturen des Netzwerks erlernen.
Bei einem reinen Austausch von Reputationen, wie es bei den vorgestellten Konzepten 2.3.2 und 2.3.3 verwendet wurde, kann anhand der ausgetauschten Informationen, ein nur sehr geringer Schluss auf die Struktur des Netzwerks gezogen werden. Bei diesem Konzept kann davon ausgegangen werden, dass ein Peer mit einem anderen Peer noch keine Transaktion durchgeführt hat, falls er ihn noch nicht bewertet hat. Besitzt er ein Bewertung über einen anderen Peer, weiß man nicht, ob er diese Reputation aus eigener Erfahrung berechnet hat, oder ob er sie sich nur durch Meinung Dritter gebildet hat. Des Weiteren erhalten diese Reputationen keine Informationen über die Intensität der Interaktion[1] zweier Peers. Das Schaffen eines gerechten Ausgleichs zwischen erbrachter und erhaltener

[1] Die Intensität der Interaktion ist gleichbedeutend mit der Anzahl der Transaktionen, die zwischen den Peers durchgeführt wurde.

Leistung, ist somit relativ schwer zu verwirklichen.

Als Lösung bietet sich ein Reputationssystem an, dass auf der Verteilung von sogenannten Token beruht, dieses Konzept wurde in Abschnitt 2.3.4 vorgestellt. Mithilfe der Token können zusätzliche Informationen über die Strukturen des Netzwerks und der Intensität der Interaktion zweier Peers gewonnen werden.
Dieses Tokenmodell soll nun genauer erläutert werden. In 3.1 wird ein detaillierter Einblick in die verschiedenen Komponenten eines Tokens und wie dieser erzeugt wird vermittelt.
Der darauf folgende Abschnitt 3.2 beschäftigt sich mit der eigentlichen Reputationsberechnung. Es werden die im ersten Kapitel beschriebenen Reputationsalgorithmen auf ein tokenbasiertes Informationssystem adaptiert. Zudem wird ein neuer Reputationsalgorithmus in 3.2.2.4 vorgestellt, der sich an den Recommender Systemen orientiert.
Abschnitt 3.3 und 3.4 beschäftigen sich mit der Fragestellung, wie ein optimaler Provider bzw. Consumer anhand des Reputationswertes ausgewählt werden kann.
Abschnitt 3.5 diskutiert verschieden Aspekte, die bei der Tokenverteilung beachtet werden können.
Der letzte Abschnitt 3.6 stellt verschiedene Tokenempfängerauswahlstrategien vor und erläutert ihre Vor- und Nachteile. Auch hier wird mit 3.6.1.2 ein neues Verfahren vorgestellt, welches die Clusterstrukturen während der Tokenverteilung effektiv ausnutzen soll.

3.1. Das Tokenmodell

Das Tokenmodell stellt ein alternatives Informationsverteilungsverfahren zu den herkömmlichen Reputationsmodellen dar, bei denen lediglich Reputationswerte zwischen den einzelnen Teilnehmern ausgetauscht werden. Beim Tokenmodell hingegen werden detaillierte Informationen über Transaktionen ausgetauscht, die zwischen zwei Peers stattfanden. Diese Informationen werden in den sogenannten Token gebündelt und im Netzwerk verteilt.
In Abschnitt 3.1.1 wird der Aufbau dieser Token beschrieben. Mit ihrer Hilfe kann ein Peer eine Einschätzung über die Leistungen eines anderen Peers vornehmen und dessen Reputationen berechnen, ohne dass diese durch die Meinung dritter verfälscht wurde.
Die Erzeugung der Token erfolgt nach einem speziellen Sicherheitskonzept, welches in 3.1.2 schematisiert dargestellt wird.
Durch die Analyse der gesammelten Token, können nicht nur andere Peers bewertet werden, sondern auch Informationen über die Transaktionsclusterstruktur[2] des Netzwerks gewonnen werden, so enthält jeder Token darüber Informationen, welche Peers miteinander Transaktionen durchgeführt haben und welche Intensität bzw. welches Volumen diese Transaktionen hatten. Mittels dieser Analyse ist ein Reputationssystem in der Lage

[2] siehe 2.2.1.1

Reputionswerte sicherer zu berechnen und Token gezielter im Netzwerk zu verteilen, um somit den Informationsgrad der Empfänger zu erhöhen. Aus diesem Grund sollte ein gutes Verfahren zur Analyse der Clusterstruktur zum einen dazu führen, dass Angriffe schwerer möglich sind und zum anderen für einen gerechteren Ausgleich der Leistungen innerhalb des Netzwerks sorgen.

3.1.1. Der Aufbau eines Tokens

Ein Token ist ein Informationscontainer, der Informationen über das Ausmaß und die Qualität der Transaktionen zweier Peers beinhaltet. Mit seiner Hilfe können sich Peers ein Bild über die erbrachten Leistungen der Tokenerzeuger machen.
Dieser Container besteht aus den beiden Summen der jeweils zur Verfügung gestellten Datenmenge und deren Qualität.

Definition 3.1 *Ein Token $t_{(i,j)}$ sei definiert durch den Tupel $(P_i, P_j, v_i, v_j, q_i, q_j)$. Wobei P_i und P_j die beiden IDs der Erzeuger des Tokens sind. v_i und v_j sind das bereitgestellte Datenvolumen von P_i für P_j bzw. P_j für P_i. Die Qualität des bereitgestellten Datenvolumens v_i wird durch q_i bzw. q_j für v_j bestimmt. Folgender Wertebereich sind für die Variablen zulässig: $v_i, v_j \mapsto \mathbb{N}_0$ und $q_i, q_j \mapsto [0, 1]$.*

Damit die Authentizität und Korrektheit des Tokens von anderen Peers verifiziert werden kann, muss der Token von beiden Peers signiert werden. Dies erfolgt mithilfe der Public Key Kryptographie. Der Public Key kann zugleich auch als PeerID verwendet werden. Dadurch wird eine komplexere Public Key Infrastruktur und ein damit verbundener zusätzlicher Datentransfer eingespart, da der öffentliche Schlüssel bzw. die ID ohnehin zusammen mit den Token versendet wird.
Um den Overhead, der durch das Verschicken der Token entsteht, möglichst klein zu halten, muss die Anzahl der Tokenempfänger möglichst gering gehalten werden. Auf diesen Aspekt wird in 3.5 näher eingegangen.
Eine weiter Möglichkeit den Overhead zu reduzieren, ist die Tokengröße klein zu halten. Die Größe des Informationsanteils eines Tokens kann je nach Anwendungsgebiet variieren. Zudem kann die Verwendung unterschiedlicher Komprimierungsmethoden zu verschiedenen Tokengrößen führen.
Bei einer P2P Tauschbörse könnte man für das Übertragungsvolumen von je 4 Bytes und für die Qualitätsangabe von je 1 Byte ausgehen. Zusätzlich kann der Token mit einem Zeitstempel der letzten Änderung versehen werden.
Ein weiterer fester Bestandteil eines Tokens stellen die beiden Schlüssel bzw. IDs dar. Diese benötigen nach dem heutigen Sicherheitsempfinden bei einem Elliptische-Kurven-

Kryptoverfahren Verschlüsselungsverfahren eine Größe von jeweils 20 byte (160 bit) [19][3]. Schließlich kommen noch die beiden Signaturen hinzu, die aus den verschlüsselten elementaren Tokeninformationen[4] bestehen, die eine Größe von 14 Byte besitzen.
Zusammengefasst führt dies zu einer Gesamtgröße von 82 Byte pro Token. In Tabelle 3.1 ist dieser beispielhafte Tokenaufbau dargestellt.

Komponente	Grösse
Schlüssel bzw. ID	2 x 20 Byte
Volumen	2 x 4 Byte
Qualität	2 x 1 Byte
Zeitstempel	1 x 4 Byte
Signatur	2 x 14 Byte
Summe	82 Byte

Tabelle 3.1.: Aufbau eines Tokens

3.1.2. Die Tokengenerierung

Die Token Generierung erfolgt in drei Schritten. Zuerst bittet der Consumer einen Provider ihm eine Leistung zu erbringen. Dazu sendet er ihm einen Rohtoken, den er signiert hat. Dieser Rohtoken enthält als Transaktionsvolumen, das vom Consumer gewünschte Volumen und als Qualitätsbewertung einen als Unbekannt definierten Wert.
Im nächsten Schritt überprüft der Anbieter, ob er dem Nutzer die Leistung erbringen will, wenn ja unterzeichnet er den Rohtoken, schickt ihn an den Consumer und erbringt ihm den Service.
Im letzten Schritt bewertet der Consumer die Qualität des erbrachten Services, trägt die tatsächlich erbrachte Leistung ein, signiert er erneut den Token und schickt diesen wieder zurück an den Anbieter. Nun besitzen Beide den aktuellen Token und können diesen im Netzwerk verteilen. Abbildung 3.1 stellt diesen Vorgang anschaulich dar.
Jeder einzelne dieser Schritte ist notwendig, damit die Transaktionspartner über ein Beweismittel verfügt, falls er von seinem Gegenüber betrogen werden sollte. Den Rohtoken, den der Provider nach dem ersten Schritt erhalten hat, benötigt er, falls der Consumer sich weigert den Token nach dem Erbringen des Services zu signieren. Den Rohtoken, den der Consumer im zweiten Schritt erhält, kann er nutzen, um den Anbieter anzuschwärzen, falls dieser ihm einen Service mit schlechter Qualität liefert. Mithilfe des zuletzt generierten Token kann schließlich der Anbieter beweisen, dass er eine gewisse Leistung mit einer gewissen Qualität erbracht hat.

[3] Diese Schlüssellänge von 160 Bit entspricht einer Sicherheit von 1024 Bit bei herkömmlichen asymmetrischer Verschlüsselungsverfahren [19]
[4] Die elementaren Tokeninformationen bestehen aus dem Volumen, der Qualität und dem Zeitstempel

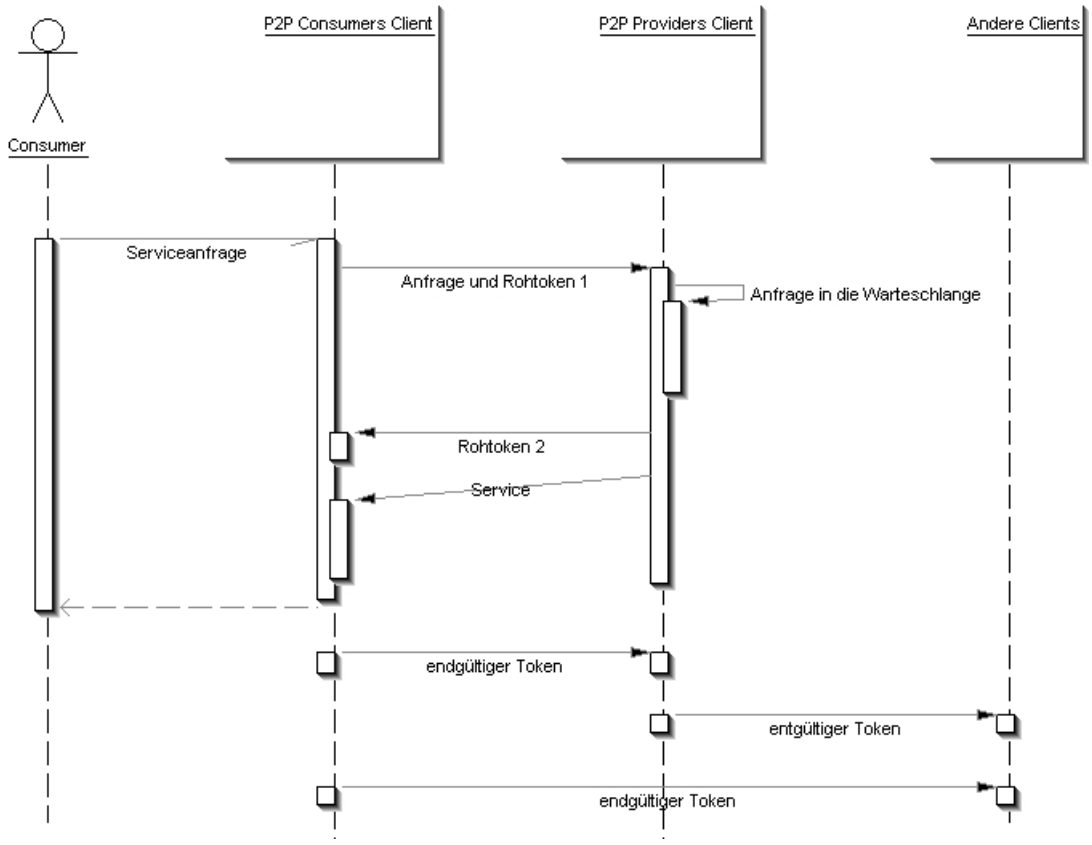

Abbildung 3.1.: Tokenaustausch bei impliziter Bewertung

Wenn trotz vollendeter Erstellung des Tokens einer der beiden Transaktionspartner, diesen nicht publiziert, sondern den Rohtoken, um dadurch seine Reputation zu verbessern, kann er entlarvt werden, indem bei der Auswertung der Token überprüft wird, wie oft und bei welchen Transaktionspartnern ein Peer den Rohtoken veröffentlicht. Falls ein Peer öfters Rohtoken verschickt von Transaktionspartner mit denen sonst kaum ein anderer Peer Probleme hat, ist dies ein Indiz dafür, dass der Peer, der die Rohtoken verschickt, ein Betrüger ist. Mögliche Angriffsszenarien auf dieses Tokengenerierungskonzept werden in dieser Arbeit nicht weiter betrachtet. Es wird davon ausgegangen, dass nur komplette Token und keine Rohtoken im Netzwerk publiziert werden.
Neben der Erzeugung neuer Token ist es auch möglich, Informationen eines Tokens zu aktualisieren, indem eine neue Tokenversion erzeugt wird. Eine Aktualisierung wird notwendig sobald zwei Transaktionspartner weitere Transaktionen nach der Tokengenerierung durchführen. Die neuste Version eines Tokens enthält alle Informationen über Transaktionen, die zwischen den beiden Tokenerzeugern bis zum Aktualisierungszeitpunkt stattfanden.
Für diese Aktualisierung ist ebenfalls ein Sicherheitskonzept notwendig. Dieses Konzept lässt sich von der hier schematisierten Darstellung der Generierung neuer Token ableiten und wird deshalb nicht näher erläutert.

3.2. Die Reputationsberechnung

Reputationen können für verschiedene Verwendungszwecke wie *Providerauswahl*[5], *Consumerauswahl*[6] oder *Tokenverteilung*[7] verwendet werden. Für jeden dieser Zwecke muss die Reputation eines Peers explizit berechnet werden. Bei der Providerauswahl kommt es darauf an, wie schnell und in welcher Qualität ein potentieller Provider Leistungen liefern kann. Dabei ist relativ unerheblich, wie viel der potentielle Provider bereits selbst konsumiert hat. Für die Consumerauswahl kann der gleiche Peer jedoch eine schlechte Reputation besitzen, da er z. B. viel mehr Leistungen erhalten hat, als er selbst geliefert hat.
Des Weiteren ist die Reputation abhängig von dem individuellen Wissen des Peers, der die Reputation berechnet. Dieser Peer wird im folgenden als Betrachter bezeichnet. Er kann je nach Situation die Rolle des Consumers, Providers oder Tokenversenders annehmen. Das Wissen, über welches der Betrachter verfügt, entspricht der Menge der Token, die dieser Peer gesammelt hat. Die Tokenmenge T_i ist die Menge aller Token die Peer i besitzt.

Aus dieser Abhängigkeit zwischen der Menge T_i, dem Verwendungszweck z und dem Repuatationswert ergibt sich Definition 3.2 für die Reputationfunktion.

Definition 3.2 *Die Reputationfunktion $rep_{z(i,j)}$ für die Berechnung des Reputationswertes des Peers j aus der Perspektive von Peer i mit seiner Tokenmenge T_i ist für den Verwendungszweck z definiert durch:*

$$rep_{z(i,j)} : T_i \mapsto [0, x] mit\ x \in \mathbb{R}, z \in \{pz, cz, tz\}$$

Durch diese Definition nehmen Reputationswerte keine negative Werte an. Peers, über die der Betrachter keine Informationen besitzt bzw. ihm unbekannt sind, erhalten mit einem Reputationswert von 0 den kleinst möglichen Reputationswert. Somit ist es nicht möglich einen schlechteren Reputationswert zu erhalten als Peers, die dem Betrachter unbekannt sind. Dies gilt im Besonderen für bösartige Peers und liegt darin begründet, dass Peers, die durch ein negatives Verhalten schlechter eingestuft werden müssten als unbekannte Peers jederzeit ihre Identität aufgeben könnten und mit neuer Identität und einem Reputationswert von 0 beginnen könnten. Dieser Angriff wird als *"White washing"* bezeichnet. Um diesen Angriff im Vorfeld zu vermeiden, werden bösartige Peers nie schlechter bewertet als neue Peers. Dies hat zur Folge, dass sich Peers zuerst ein gewisses Vertrauen durch die Bereitstellung von Leistungen erarbeiten müssen bevor sie selbst in den Genuss von Leistungen kommen.

[5] Abk. pz
[6] Abk. cz
[7] Abk. tz

Um weiter Formalisierungen zu vereinfachen, werden alle Reputationswerte, die ein Peer i in andere Peers besitzt, zu einem Reputationsvektor $\vec{rep}_{z(i)}$ zusammengefasst, 3.3, wobei der j-te Eintrag dem Reputationswert für den j-ten Peer entspricht.

Definition 3.3 *Für den Reputationsvektor $\vec{rep}_{z(i)}$ des Peers i gilt:*

$$\vec{rep}_{z(i)} = \left\{ \begin{array}{l} rep_{z(i,1)} \\ \vdots \\ rep_{z(i,n)} \end{array} \right\}$$

Die eigentliche Reputationsberechnung lässt sich in drei Teilbereiche gliedern.
In dem ersten Bereich 3.2.1 werden verschiedene Tokenbewertungsfunktionen vorgestellt. Diese interpretieren die Tokeninformationen in Abhängigkeit verschiedener Kontexte.
Im zweiten Bereich, der im Folgenden als Plausibilitätsprüfung bezeichnet wird, werden die abgegebenen Bewertungen eines Peers unabhängig vom Transfervolumen betrachtet und es wird überprüft, ob sie dem Betrachter als plausibel erscheinen. Diese Betrachtung entspricht einer rein qualitativen Analyse der Tokenbewertungen. Die genaue Bestimmung welche Bewertungen als plausibel angesehen werden, ist abhängig von dem jeweiligen verwendeten Plausibilitätsprüfungsverfahren. Diese Verfahren werden in 3.2.2 vorgestellt.
Der letzte Abschnitt, der im Folgenden als Vertrauenswürdigkeitsprüfung bezeichnet wird, betrachtet die quantitative Bewertung der Token. Er berechnet in welchem Maße dem angegeben Transfervolumen vertraut werden kann. Nicht jedem Peer, dessen Bewertungen dem Betrachter als plausibel erscheinen, sollte er auch vertrauen. Auch hier bieten sich verschiedene Verfahren an, die in 3.2.3 vorgestellt werden.

3.2.1. Die Tokenbewertung

Für die Tokenbewertung bietet sich eine Vielzahl verschiedener Entscheidungsmöglichkeiten an, die in dieser Arbeit nicht alle betrachtet werden können, so wird z. B. das Alter der Token außer Acht gelassen. Würde man diesen Aspekt berücksichtigen, könnte man ältere Token weniger stark in die Berechnung einfließen lassen als aktuellere.

Die elementaren Eigenschaften auf denen die Tokenbewertung in dieser Arbeit beruhen soll, ist zum einen, ob der Betrachter die abgegebenen Bewertungen des Tokenerzeugers als plausibel einschätzt. Dies wird in 3.2.2 genauer erläutert.
Zum anderen ist die Tokenbewertung abhängig von dem Transfervolumen, dessen Qualität und in welchem Kontext der Token betrachtet wird. Als Kontext steht zur Auswahl der Kontext der Leistungsansprüche, der Leistungsverpflichtungen, des Tokenvolumens, des Bewertungsverhältnisses und des Bewertungsverhältnisses mit undefinierten Werten.
In welchem Kontext ein Token betrachtet wird, ist abhängig vom Verwendungszweck und

den eingesetzten Algorithmen. Für die Reputationsberechnung der Consumerauswahl z.B. spielt das Verhältnis zwischen den Leistungsansprüchen und den Leistungsverpflichtungen des potentiellen Consumers eine entscheidende Rolle. Bei der Providerauswahl hingegen sind die Leistungsverpflichtungen eines potentiellen Providers unerheblich für die Reputationsberechnung.

Im folgenden Abschnitt werden die Tokenbewertungen in den jeweiligen Kontexten beschrieben und anschließend in der Liste 3.4 formal dargestellt.

Wird ein Token t_{ij} im Kontext der *Leistungsansprüche*[8] betrachtet, wird bestimmt, wie hoch die Leistungsansprüche des Peers i bzw. des Peers j sind, die aus diesem Token hervorgehen. Die Ansprüche für Peer i sind abhängig von dem von Peer i für Peer j zur Verfügung gestellten Datenvolumen, deren Qualität, und der Glaubwürdigkeit von Peer j, bzw. wie plausibel dem Betrachter die Bewertungen von Peer j erscheinen. Erachtet der Betrachter die abgegeben Bewertungen von Peer j als plausibel, werden die Leistungsansprüche von Peer i an Peer j im vollen Umfang angerechnet. Dabei werden die korrekt erbrachten Leistungen mit den fehlerhaften Leistungen im gleichen Maße verrechnet. Somit müssen die von Peer i und Peer j gelieferten Leistungen zu mehr als zur Hälfte gut bewertet worden sein, damit Peer i Ansprüche gelten machen kann. Erscheinen die Leistungen dem Betrachter als nicht plausibel, dann können für Peer i keine Ansprüche angerechnet werden.

Im Kontext der *Leistungsverpflichtungen*[9] wird bestimmt, wie hoch die Verpflichtungen von Peer i gegenüber Peer j bzw. von Peer j gegenüber Peer i sind, die aus dem zu bewertenden Token t_{ij} entstehen. Die Verpflichtungen von Peer i gegenüber Peer j sind abhängig von dem Datenvolumen, das Peer i von Peer j erhalten hat und deren Qualität. Erscheinen die Bewertungen von Peer i dem Betrachter als nicht plausibel, so wirkt sich dies Peer i negativ aus, d.h. die für Peer i berechneten Verpflichtungen erhöhen sich, da davon ausgegangen werden muss, dass er sich durch die nicht plausible Bewertung einen Vorteil verschaffen wollte und deßhalb seine Verpflichtungen gegenüber Peer j schmälern wollte.
An dieser Stelle soll darauf hingewiesen werden, dass die Leistungsverpflichtungen von Peer i gegenüber Peer j nicht zwangsläufig identisch mit den Leistungsansprüchen von Peer j an Peer i sein müssen, auch wenn Peer j korrekte Leistungen liefert. Sobald der Betrachter die Bewertungen von Peer i als nicht plausibel ansieht, reduzieren sich die Ansprüche von Peer j auf 0 und die Verpflichtungen von Peer i werden im vollen Umfang angerechnet. Somit wirken sich die als nicht plausibel eingeschätzten Bewertungen auf

[8] Abk.: ak
[9] Abk.: vk

beide Transaktionspartner negativ aus. Deshalb hat kein Peer Interesse weder selbst als nicht plausibel eingestuft zu werden, noch mit ein Peer zusammen zuarbeiten, der keine plausiblen Bewertungen abgibt.

Im Kontext des *Tokenvolumens*[10] wird die Qualität der Leistungen außer Acht gelassen. Lediglich die Intensität der Kommunikation zwischen den Tokenerzeugern wird in diesem Kontext gemessen.

Der Kontext des *Bewertungsverhältnisses*[11] betrachtet das Verhältnis zwischen den abgegeben positiven Bewertungen und den abgegebenen negativen Bewertungen. Dies entspricht q_i und ist gleichbedeutend mit der subjektiven Einschätzung der Qualität, des von Peer i geleisteten Services aus Sicht von Peer j. Falls in dem zu bewertenden Token q_i unbestimmt ist, d.h. wenn Peer i für Peer j noch keine Bewertung abgegeben hat, dann wird der Token mit 0 gewertet. Hingegen wird im Kontext des *Bewertungsverhältnisses mit undefinierten Werten*[12] bei einem unbestimmten q_i die Tokenbewertung als undefiniert bestimmt. Die letzten beiden aufgeführten Kontexte werden lediglich für die Plausibliätsprüfung verwendet und haben für andere Verwendungszwecke keine weitere Relevanz.

Definition 3.4 *Die Tokenbewertungsfunktion $tvf_{k(i,j)}$[13] für die Bewertung des Peers i anhand des Tokens $t_{(i,j)}$[14] aus der Tokenmenge T_m und der Plausibilität p_i bzw. p_j der abgegebenen Bewertungen von Peer i bzw. Peer j, ist wie folgt für den jeweiligen Kontext k definiert:*

- ***Kontext der Leistungsansprüche:*** *Die Leistungsansprüche von Peer i an Peer j werden berechnet, im Falle dass der Betrachter die Bewertungen von Peer j als plausibel einstuft, indem das bereitgestellte Volumen v_i des Peers i und der Qualität q_i multipliziert wird. Davon werden die schlechten Leistungen subtrahiert, die sich durch die Multiplikation des Volumens v_i mit $1 - q_i$ ergeben. Schließlich wird dieses Teilergebnis mit 0 maximiert, um negative Ergebnisse zu verhindern, die entstehen würden wenn $q_i < 0,5$.*
  Werden die Bewertungen von Peer j als nicht plausibel eingestuft, so kann Peer i keine Ansprüche aus dem Token gelten machen.

[10] Abk.: tk
[11] Abk.: bk
[12] Abk.: bk'
[13] Token Value Function
[14] siehe 3.1.1 für die genaue Beschreibung der Tokenparameter q_i, q_j, v_i und v_j

$$tvf_{ak(i,j)} =: t_{(i,j)} \mapsto [0, v_i]$$

$$tvf_{ak(i,j)} = \begin{cases} max(v_i * q_i - (1 - q_i) * v_i, 0) & falls\ p_j = 1 \\ 0 & sonst \end{cases} \tag{3.1}$$

- ***Kontext der Leistungsverpflichtungen:*** *Die Leistungsverpflichtungen die Peer i bei Peer j besitzt, werden wie folgt berechnet. Falls der Betrachter die Bewertung von Peer i als plausibel ansieht, entsprechen die Verpflichtung von Peer i gegenüber Peer j dem Volumen der korrekt erbrachten Leistungen von Peer j an Peer i.*
  Sobald Peer i für den Betrachter unglaubwürdige Bewertungen abgibt, entsprechen die Verpflichtungen von Peer i an Peer j dem gesamten Volumen der bereitgestellten Leistungen, unabhängig ihrer Qualität.

$$stvf_{vk(i,j)} =: t_{(i,j)} \mapsto [0, v_j]$$

$$tvf_{vk(i,j)} = \begin{cases} v_j * q_j & falls\ p_i = 1 \\ v_j & sonst \end{cases} \tag{3.2}$$

- ***Kontext des Tokenvolumens*** *Die Bewertung im Kontext des Tokenvolumens besteht lediglich aus der Summe v_i und v_j.*

$$tvf_{tk(i,j)} =: t_{(i,j)} \mapsto [0, v_j + v_i]$$

$$tvf_{tk(i,j)} = \begin{cases} v_i + v_j & falls\ t_{(i,j)} \in T_m \\ 0 & sonst \end{cases} \tag{3.3}$$

- ***Kontext des Bewertungsverhältnisses***

$$tvf_{bk(i,j)} =: t_{(i,j)} \mapsto [0, 1]$$

$$tvf_{bk(i,j)} = \begin{cases} q_i & falls\ q_i \neq null \\ 0 & sonst \end{cases} \tag{3.4}$$

- ***Kontext des Bewertungsverhältnisses mit undefinierten Werten***

$$tvf_{bk'(i,j)} =: t_{(i,j)} \mapsto [0, 1] \cup null$$

$$tvf_{bk'(i,j)} = \begin{cases} q_i & falls\ q_i \neq null \\ null & sonst \end{cases} \tag{3.5}$$

Die gesamten Tokenbewertungen, die für einen Peer i im Kontext k berechnet wurden, können in einem Tokenbewertungsvektor $\vec{tvv_{k(i)}}$[15] zusammengefasst werden. Dabei ist zu beachten, dass ein Betrachter Peer i anhand seines lokalen Wissens, welches seinen gesammelten Token T_i entspricht, auch den Tokenbewertungsvektor für jeden beliebigen Peer j bilden kann und nicht nur seinen eigenen $\vec{tvv_{k(i)}}$.

Definition 3.5 *Für den Tokenbewertungsvektor $\vec{tvv_{k(j)}}$ des Peers i im Kontext c gilt:*

$$\vec{tvv_{k(j)}} = \begin{pmatrix} tvf_{k(j,1)} \\ \vdots \\ tvf_{k(j,i)} \\ \vdots \\ tvf_{k(j,n)} \end{pmatrix}$$

Die Tokenbewertungsvektoren können wiederum zu einer Tokenbewertungsmatrix TVM[16] zusammengefasst werden. In dieser Matrix entspricht der Zeilenvektor i den Bewertungen, die Peer i erhalten hat und der Spaltenvektor i den Bewertungen die Peer i abgegeben hat. Somit entspricht der Eintrag $TVM_{k(i,j)}$ der Bewertung die Peer j Peer i im Kontext k gegeben hat.

Definition 3.6 *Für die Tokenbewertungsmatrix (Token Value Matrix) TVM_k gilt:*

$$TVM_k = \begin{pmatrix} \vec{tvv_{k(1)}} & \dots & \vec{tvv_{k(n)}} \end{pmatrix} \tag{3.6}$$

Diesen Tokenbewertungsmatrix kann Spaltenweise normiert werden, so dass die Summe jeder Spalte 1 ergibt. Die sich daraus ergebende Matrix wird im folgenden als TVM' bezeichnet und wird für verschiedene hier vorgestellte Verfahren benötigt. Diese Normierung ähnelt der in 2.3.2 vorgestellten Normierung.

$$TVM'_{k(i,j)} = \begin{cases} \frac{TVM_{k(i,j)}}{\sum_i TVM_{k(i,j)}} & \text{falls } \sum_i TVM_{k(i,j)} > 0 \\ 0 & \text{falls } \sum_i TVM_{k(i,j)} = 0 \end{cases} \tag{3.7}$$

[15] Token Value Vector
[16] Token Value Matrix

3.2.2. Verfahren zur Überprüfung der Plausibilität

Bei dieser Prüfung wird untersucht, wie verlässlich ein Betrachter die abgegebenen qualitativen Bewertungen anderer Peers ansieht.
Im Folgenden werden vier Alternativen für die Plausibilitätsprüfung vorgestellt. Die ersten drei orientieren sich an den Eingangs erwähnten Reputationssystemen. Die letzte erläutert einen neu entwickelten Ansatz, der sich an den Recommender Systemen orientiert und versucht die Plausibilität mittels einer Ähnlichkeitsbestimmung der abgegeben Bewertungen zu berechnen.
Als Ergebnis liefert dieser Schritt einen Vektor $p\vec{v}_i$[17], dessen $j-ter$ Eintrag die Plausibiliät der abgegebenen Bewertung des $j-ten$ Peers aus der Perspektive von Peer i beschreibt. Um die Komplexität weiterer Berechnungen zu vereinfachen, wird die Plausibilität als binär betrachtet und deshalb gilt $p\vec{v_{i(j)}} \in 0, 1$, d.h. entweder werden die abgegeben Bewertungen eines Peers als plausibel betrachtet oder nicht.

3.2.2.1. Das naive Verfahren

Bei diesem Verfahren sieht ein Peer alle Bewertungen anderer Peers als plausibel an.
Dies entspricht bei der Darstellung mittels der Vektorenschreibweise einem Einsvektor als Plausibilitätsvektor. Dieses simpelste Verfahren, dient hauptsächlich als Referenzalgorithmus, um den Erfolg der aufwändigeren Verfahren bzw. deren Performanz zu messen.
Angreifer haben bei diesem Verfahren leichtes Spiel, da ihre möglicherweise manipulierten Bewertungen nicht angezweifelt werden.

$$p\vec{v}_i = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \tag{3.8}$$

3.2.2.2. Transitive Plausibilitätsprüfung durch Eigenwertberechnung

Eine verbessertes Verfahren stellt die Plausibilitätsprüfung mithilfe des PageRank Algorithmuses da und beruht auf der Annahme, dass Bewertungen von den Peers hoch einzustufen sind, die von Peers gut bewertet wurden, die wiederum selbst hoch bewertet wurden.
Hierfür wird ähnlich wie beim Eigentrust Ansatz der Eigenvektor einer Matrix berechnet. Diese Matrix entspricht der TVM'. Dabei wird als Kontext für die Aufstellung der Tokenbewertungsmatrix der Kontext des Bewertungsverhältnisses gewählt, d.h. es fließen in die Berechnung nur qualitative Bewertungen ein und keine quantitativen.

$$p\vec{v}_i{}' = (TVM'_{bk})^n tv\vec{v}_{bk(i)} \tag{3.9}$$

[17] plausibility vector

$$p\vec{v}_{i(j)} = \begin{cases} 1 & p\vec{v}_{i(j)}{}' \geq \pi \\ 0 & \text{sonst} \end{cases} \tag{3.10}$$

Durch die normierte Matrix TVM' wird gewährleistet, dass die Matrix $(TVM'_{bk})^n$ bei entsprechend großem n zum linken Eigenvektor von TVM'_{bk} konvergiert. Bei der späteren Simulation wird aus Gründen der Laufzeit auf eine genaue Bestimmung des Eigenvektors verzichtet und lediglich n entsprechend groß gewählt, so dass sich die beiden Summen der Einträge der Ergebnisvektoren zweier Iterationsschritte nur geringfügig unterscheiden.
Damit der Plausibilitätsvektor nur die Werte 0 und 1 enthält, wird eine Schranke in der Höhe von π eingeführt, alle Werte des Eigenvektors die darunter oder gleich dieser Schranke liegen werden auf 0 abgebildet und alle darüber auf 1. Die Höhe der Schranke ist stark davon abhängig, wie misstrauig ein Peer ist, je misstrauiger er ist, desto höher muss er sie Ansätzen. Damit die verschiedenen Verfahren in den späteren Simulationen vergleichbar bleiben, wird der minimal Wert von 0 für π angesetzt.
Diese Höhe lässt sich damit rechtfertigen, dass in den folgenden Simulationen nur extrem Situationen simuliert werden. Situationen die nicht diesen Extremen entsprechen, benötigen eine genauere Betrachtung, die in dem nötig Umfang in dieser Arbeit nicht erfolgen kann. In der Praxis kann die Höhe von π durch Auswertung der Verteilung der Werte des Eigenvektor Erfolgen, so können z.B. die schlechtesten 10% als nicht plausibel betrachtet werden.
Ähnlich wie beim Eigentrustverfahren, wird bei diesem Verfahren eine globale Bewertung berechnet. Dieser Bewertung ist ausschließlich von der Tokenmenge des Betrachters abhängig und nicht vom Betrachter selbst, d.h. zwei Betrachter die über, die gleiche Tokenmenge verfügen, bewerten andere Peers identisch. Dies kann zu Problemen führen, vor allem bei Bewertungen, die stark von einer subjektiven Meinung abhängig sind und dementsprechend stark von den persönlichen Geschmack des einzelnen Peers beeinflusst werden.
Diese globale Betrachtungsweise birgt ein kleines Paradoxon. Peers die von anderen schlecht bewertet wurden und dadurch eine globale Plausibilität von 0 besitzen, glauben denen von ihnen abgegebenen Bewertungen nicht mehr. Dies führt bei der Vertrauenswürdigkeitsprüfung mittels maximalen Fluss, welche in 3.2.3 vorgestellt wird, zu Problemen.[18] Eine Lösung dieses Problem zu vermeiden ist, dass der Eintrag $p\vec{v}_{i(i)} = 1$ gesetzt wird und somit jeder Peer seine eigenen Bewertungen als plausibel ansieht. Dies bewirkt zwar, dass die globale Charakteristik des $\vec{pv}$ für dieses Verfahren verloren geht, jedoch unterscheiden sich die beiden Plausibilitätsvektoren zweier Betrachter, die über die gleiche Tokenmenge verfügen, in maximal zwei Einträgen. Deshalb kann man immer noch von einer nahezu

[18] Durch die als nicht plausibel betrachteten eigenen Bewertungen, besitzt dieser Knoten keine ausgehenden Kanten und kann somit auch keinen maximalen Fluss berechnen bzw. beträgt dieser für alle anderen Knoten einen Wert von 0.

globalen Charakteristik des $\vec{pv}$ sprechen.

Bei genauerer Betrachtung birgt dieses Verfahren ein weiteres Problem, welches nicht erst durch die Adaptierung auf das Tokensystem entstanden ist. Bereits das Eigentrust-Verfahren 2.3.2 beinhaltet dieses Problem. Es entsteht bei der Anpassung des PageRank Algorithmuses von der ursprünglich webseitenbasierten Anwendung auf ein P2P System und beruht auf der transitiven Berechnungsmethodik, die die einzelnen Bewertungen höher einstuft, wenn der jeweilige Peer, der die Bewertung abgegeben hat, ebenfalls gut bewertet wurde.
Bei Webseiten sind Empfehlungen - die Links Bestandteil der eigentlichen Serviceleistung der Webseite. Webseiten werden selbst als unseriös eingestuft, wenn sie auf unseriöse Webseiten verlinken.
Bei P2P Systemen hingegen sind die Empfehlungen kein elementarer Bestandteil der Serviceleistung und werden in der Regel nicht bewertet. Dies bedeutet, dass Bewertungen von Peers als plausibel angesehen werden, nur weil ihre bereitgestellten Daten gut bewertet wurden. Jedoch müssen deshalb ihre abgegebenen Bewertungen nicht zwangsläufig korrekt sein.

3.2.2.3. Transitive Plausibilitätsprüfung mithilfe des Multi-Level Ansatzes

Orientiert am Multilevel Tit-For-Tat Ansatz [4], wird bei diesem Verfahren die Reputation transitiv berechnet. Im Gegensatz zur einfachen transitiven Plausibilitätsprüfung wird nicht der Eigenvektor der Matrix bestimmt, sondern es fließt lediglich die Meinung von n Nachbargenerationen in die Berechnung mit ein. Dies hat zur Folge, dass der Plausibilitätsvektor nicht nur abhängig ist von der Tokenmenge T_i des Betrachters, sondern auch von ihm selbst, d.h. von seinem lokalen Umfeld im Netzwerk. Würde man den Eigenvektor bilden, wäre der Plausibilitätsvektor für alle Betrachter identisch, wenn sie über die gleiche Tokenmenge verfügen.
In der Untersuchung in [4] hat sich ein Wert von 1 für n als optimal herausgestellt. Dies entspricht einem Indirektionslevel von 1 und bedeutet dass die eigenen Bewertungen und die Bewertungen der direkten Nachbarn in die Reputationsberechnung einfließen. Die Bewertungen der Peers, die über zwei Indirektionen mit dem Betrachter in Verbindungen stehen, fließen nicht in die Bewertung mit ein. Diese Einlevelindirektion wird für diese Arbeit übernommen.
Zusätzlich wird, wie beim transitiven Verfahren mit Eigenwertberechnung, die π-Schranke eingeführt, für die die selben Eigenschaften gelten.
Durch diese eingeschränkte Betrachtungsweise, wird das in 3.2.2.2 beschriebene erste Problem behoben. Die Eigenwertbestimmung berechnet globale Reputationswerte, die für alle

Betrachter identisch sind, sofern sie über die gleiche Tokenmenge verfügen. Die Einschränkung der Indirektionslevel bewirkt, dass die Reputationswertberechnung einen lokalen Charakter bekommt, d.h. es wird berücksichtigt das Peers Leistungen aus subjektiver Sicht bewerten. Es fließen nur Bewertungen von den Peers in die Berechnung ein, von denen der Betrachter Leistungen erhalten hat.
Einen Einfluss, auf das zweite genannte Problem, hat dies nicht. Peers die eine gute Leistung erbringen, können trotzdem Bewertungen durch ihre subjektive Meinung beeinflussen bzw. mutwillig manipulieren und damit andere Peers betrügen, die in ihrer Nachbarschaft liegen.

$$\vec{pv_i}' = TVM'_{bk} * \vec{tvv_{bk(i)}} \tag{3.11}$$

$$\vec{pv_{i(j)}} = \begin{cases} 1 & \vec{pv_{i(j)}}' \geq \pi \\ 0 & \text{sonst} \end{cases} \tag{3.12}$$

3.2.2.4. Plausibilitätsprüfung mittels Ähnlichkeitsbewertung

Bei diesem neu entwickelten Ansatz wird davon ausgegangen, dass ein Peer i die Bewertungen eines anderen Peers j als Plausibel ansieht, wenn sie beide ähnliche Bewertungen abgegeben haben.
Diese Ähnlichkeit wird mithilfe der Ähnlichkeitsfunktion $sim_{(i,j)}$[19] berechnet. Für dieser Funktion werden die Bewertung im Kontext des Bewertungsverhältnisses mit undefinierten Werten betrachtet, d.h. es wird betrachtet, wie ähnlich die beiden Peers i und j die Qualität des Services anderer Peers bewertet haben.
Für eine übersichtliche Darstellung werden zwei Hilfsfunktionen benötigt, die im Folgenden erläutert werden. Die Ähnlichkeit der einzelnen von Peer i und j abgegebenen Bewertungen für Peer l wird mit der Funktion $rSim_{(i,j,l)}$[20] berechnet. Eine Ähnlichkeit von $rSim_{(i,j,l)} = 1$ bedeutet, dass Peer i und j den Peer l identisch bewerten haben.
Es werden nur Bewertungen betrachtet, die von beiden Peers für den selben Peer abgegeben wurden. Die Anzahl der Peers, für die beide Peers i und j eine Bewertung abgegeben haben, wird mit der Funktion $\sum_l def_{(i,j,l)}$[21] bestimmt. Falls $\sum_l def_{(i,j,l)} = 0$ existiert kein Peer für den beide Peers eine Bewertung abgegeben haben und es kann keine Aussage über die Ähnlichkeit der beiden Peers i und j getroffen werden. Um sich vor Angriffen zu schützen muss von einer Ähnlichkeit von 0 ausgegangen werden.

$$sim_{(i,j)} = \begin{cases} \frac{\sum_l rSim_{(i,j,l)}}{\sum_l def_{(i,j,l)}} & \text{falls } \sum_l def_{(i,j,l)} \neq 0 \\ 0 & \text{sonst} \end{cases} \tag{3.13}$$

[19] sim: similarity
[20] rSim: rating similarity
[21] def: defined

$$rSim_{(i,j,l)} = \begin{cases} 1 - |tvf_{bk'(i,l)} - tvf_{bk'(j,l)}| & \text{falls } tvf_{bk'(i,l)} \neq null \wedge tvf_{bk'(j,l)} \neq null \\ 0 & \text{sonst} \end{cases} \tag{3.14}$$

$$def_{(i,j,l)} = \begin{cases} 1 & \text{falls } tvf_{bk'(i,l)} \neq null \wedge tvf_{bk'(j,l)} \neq null \\ 0 & \text{sonst} \end{cases} \tag{3.15}$$

Damit die Ähnlichkeit dem Definitionsbereich des Plausibilitätsvektors $\vec{pv}_i$ entspricht, müssen die Ähnlichkeitswerte auf 0 und 1 abgebildet werden. Dies geschieht ähnlich, wie bei den transitiven Verfahren mit der π-Schranke, die auch hier auf 0 gesetzt wird. Im Gegensatz zu den anderen Verfahren, lässt sich hier der Wert für die π-Schranke leichter interpretieren, so bedeutet eine Schranke in Höhe von $0,9$, dass 90% der abgegebenen Bewertungen eines Peers identisch mit denen des Betrachters sein müssen, um als plausibel anerkannt zu werden.

$$\vec{pv_{i(j)}} = \begin{cases} 1 & \text{falls } sim_{(i,j)} \geq \pi \\ 0 & \text{sonst} \end{cases} \tag{3.16}$$

Durch dieses Verfahren erhalten die Plausibltätswerte einen lokalen Charakter, d.h. sie sind abhängig von den abgegebenen Bewertungen des Betrachters im Gegensatz zum transitiven Verfahren 3.2.2.2, bei dem die Bewertungen global berechnet werden. Ebenfalls entfällt die zweite Problematik, die durch die transitive Berechnungsmethodik entsteht. Es werden nun Bewertungen als gut bzw. plausibel befunden, falls sie von Peers abgegeben wurden, die ein ähnliches Bewertungsverhalten wie der Betrachter besitzen. Somit findet eine Trennung der Gewichtung der abgegeben Bewertungen eines Peers und der Bewertung seiner erbrachten Leistung statt.

3.2.3. Verfahren zur Überprüfung der Vertrauenswürdigkeit

Nachdem verschiedene Möglichkeiten zur qualitativen Bewertung der Token vorgestellt wurden, sollen in diesem Abschnitt zwei Verfahren zur quantitativen Auswertung der Token bzw. des Tokenvolumens betrachtet werden. Hier steht im Vordergrund die Frage, inwieweit den angegeben Tokenvolumen vertraut werden kann. Prinzipiell gilt, dass je größer das Transaktionsvolumen zwischen zwei Peers ist, desto eher können sie sich gegenseitig vertrauen.

Das erste Verfahren erzielt, ähnlich wie auch bei der Eigenwertberechnung für die Plausibiltätsprüfung globale Reputationswerte, die für alle Betrachter identisch sind, falls sie über die gleiche Tokenmenge verfügen. Diese Eigenschaft führt zu ähnlichen Problemen

wie bei der Eigenwertberechnung.
Das zweite Verfahren hingegen überprüft die Vertrauenswürdigkeit mittels eines lokalen Ansatzes und erzielt damit gewisse Vorteile.

3.2.3.1. Uneingeschränktes Vertrauen

Bei diesem einfachen Verfahren, werden alle Tokenbewertungen gleich gewichtet. Für die Providerauswahl wird der Kontext der Leistungsansprüche verwendet. Peers die über große Leistungsansprüche verfügen, müssen eine große Datenmenge in guter Qualität zur Verfügung gestellt haben, somit sind Peers mit hohen Leistungsansprüchen auch gute Provider. Alle Leistungsansprüche, die der Betrachter für einen zu bewertenden Peer i mittels eines Tokens belegen kann, werden aufsummiert. Die daraus resultierende Summe entspricht dem Reputationswert des Peers i.

$$\vec{rep}_{(pz)_i} = \sum_j tvf_{ak(i,j)} \tag{3.17}$$

Für die Consumerauswahl wird die Summe der Leistungsansprüche des Peers i von der Summe seiner Leistungsverpflichtungen subtrahiert. Damit Peers die mehr Leistungen erhalten haben, als sie erbracht haben, keine negativen Reputationswert erhalten, wird die Differenz mit 0 maximiert. Dies verhindert, dass New-Comer eine bessere Reputation erhalten können als Peers die bereits Transaktionen durchgeführt haben.

$$\vec{rep}_{(cz)_i} = max(\sum_j tvf_{ak(i,j)} - \sum_j tvf_{vk(i,j)}, 0) \tag{3.18}$$

Diese primitive Berechnungsmethodik birgt einige Risiken. So bietet sie keinen Schutz vor den in 2.3.5 erwähnten "Sybil-Attacken". Bei diesen Angriffen generiert der Angreifer mithilfe von imaginären Peers Token, die ihm eine hohe Leistungsbereitstellung bescheinigen. Wenn nun andere Peers die Reputation des Angreifers berechnen wollen, fließen auch diese vorgetäuschten Transfervolumen der imaginären Peers in die Berechnung mit ein.
Des Weiteren können durch diese Gewichtungsmethodik Probleme in geclusterten Netzwerken entstehen. Ein Peer i der in Cluster A und B interagiert, kann Token die für ihn positiv sind und die er in Cluster A gesammelt hat, auch in Cluster B verteilen. Jedoch werden Token die für ihn negative sind und die er in Cluster A erzeugt hat, nicht zwangsläufig auch in Cluster B verteilt, da die Peers aus Cluster A, die für die negativen Token verantwortlich sind nicht unbedingt auch in Cluster B aktiv sein müssen. Dies führt dazu, dass Peer i in Cluster B überbewertet wird, da die Peers dort die positiven Token aus Cluster A erhalten haben, aber nicht die negativen Token.

In wie weit dies die Gerechtigkeit des Netzwerks beeinflusst, soll durch Simulationen untersucht werden.

3.2.3.2. Vertrauenswürdigkeitsprüfung mittels maximalen Flusses

Das Ziel dieses Ansatzes liegt darin, einem Peer nur in dem Maße zu vertrauen, in dem er direkt oder indirekt dem Betrachter Leistungen zur Verfügung gestellt hat. Anders ausgedrückt, ein Peer muss sich das Vertrauen, welches der Betrachter in ihn setzten soll, durch das Erbringen von Leistungen erarbeiten. Dieser Ansatz orientiert sich an dem in 2.3.5 vorgestellten Verfahren.
Als Modell zur Berechnung der Reputation wird das Netzwerk als ein Flussgraph betrachtet in dem der Fluss Leistungen bzw. Vertrauen darstellt.
Bei der Providerauswahl basiert eine gute Reputation auf Vertrauen. Dieses Vertrauen entsteht indem korrekte Leistungen erbracht werden, je mehr korrekte Leistung ein Peer erbringt, desto höher ist seine Vertrauenswürdigkeit.
Übertragen auf das Modell des Flussgraphen entspricht die Quelle dem Consumer, Peer C, der zugleich auch der Betrachter ist. Die Senke entspricht dem potentiellen Provider Peer P, dessen Vertrauenswürdigkeit überprüft werden soll. Das Vertrauen, welches der Betrachter Peer C in Peer P setzt, kann direkt oder indirekt von Peer P erarbeitet worden sein. Das direkt erarbeite Vertrauen von Peer C in Peer P entsteht, indem Peer P Peer C korrekte Leistungen erbringt. Dies wird im Flussgraphen mittels einer gerichteten Kante von Knoten P nach Knoten C symbolisiert. Diese Kante besitzt eine Kapazität, die den Leistungsansprüchen von Peer P an C entspricht und mittels der Funktion $tvf_{ak(P,C)}$ berechnet wird. Der Flussgraph kann als Matrix dargestellt werden. Die Einträge der Matrix entsprechen den Kantengewichten und stellt somit TVM_{ak} dar.
Das indirekte erarbeitete Vertrauen von Peer C in Peer P kann aufgebaut werden, indem Peer P einem Peer A eine Leistung erbringt und Peer A wiederum Peer C eine Leistung erbringt. Das Vertrauen, das Peer C in Peer P setzt, kann nicht größer sein, als das Vertrauen, das von Peer A in Peer P gesetzt wird. Ein Peer kann nur soviel Vertrauen weiter geben, wie in ihn gesetzt wird. Entsprechend dem Krichhoffsche Flusserhaltungsgesetz 2.2.5 gilt, dass einfließende Vertrauen entspricht dem ausfließenden Vertrauen. In dem Beispiel des indirekten Vertrauen, fließt nur soviel Vertrauen von Peer C über Peer A in Peer P, wie es die Kantenkapazitäten zwischen den Knoten zulassen und entspricht dem minimalen Schnitt des Graphens.
Die Vertrauenswürdigkeit, die Peer C in einen potentiellen Provider setzt wird gemessen, indem der maximale Fluss von Peer C nach Peer P über den durch die Matrix TVM_{ak} definierten Graphen berechnet wird. Dieser maximale Fluss entspricht dem Vertrauen, das Peer C in Peer P setzten kann und ist gleichzusetzen mit den Gesamtleistungen, die Peer P dem Betrachter Peer C direkt oder indirekt erbracht hat. Dieses Vertrauen entspricht

der Reputation die Peer P aus der Sicht von Peer C besitzt.

$$rep_{pz(C,P)} = mf(C \rightarrow P, TVM_{ak}) \tag{3.19}$$

Wird die Reputation für den Verwendungszweck der Consumerauswahl berechnet, ist diese abhängig von den erhaltenen und erbrachten Leistungen des Consumers. Ein potentieller Consumer erhält eine hohe Reputation, wenn er viel Leistung erbracht hat, aber selbst nur wenig Leistung erhalten hat. Umgekehrt erhalten potentielle Consumer eine geringe Reputation, wenn sie bereits mehr Leistungen in Anspruch genommen haben als sie erbracht haben.
Umgesetzt wird dies in dem Modell des Flussgraphen, indem zusätzlich zur Messung des maximalen Flusses vom Betrachter, dem Provider P zum Consumer C im Kontext der Leistungsansprüche, auch der maximale Fluss von P nach C im Kontext der Leistungsverpflichtungen gemessen wird.
Der erste Fluss beschreibt die Leistungen, die ein potentieller Consumer dem Betrachter direkt oder indirekt erbracht hat. Dies entspricht den Leistungsansprüchen, die C gegenüber P besitzt und wird notiert als $mf(P \rightarrow C, TVM_{ak})$.
Der zweite Fluss entspricht den Leistung, die der potentielle Consumer bereits vom Betrachter erhalten hat, d.h der Consumer besitzt Leistungsverpflichtungen gegenüber dem Betrachter. Dieser Fluss wird mittels der Funktion $mf(P \rightarrow C, TVM_{vk})$ bestimmt.
Die eigentliche Reputation wird nun berechnet, indem die Leistungsverpflichtungen von den Leistungsansprüchen subtrahiert werden. Es gilt, um so größer die Leistungsansprüche und um so geringer die Leistungsverpflichtungen, desto größer ist die Reputation.
Um eine negativen Reputationswert zu verhindern, wird das Ergebnis der Subtraktion mit 0 maximiert.

$$rep_{cz(C,P)} = max(mf(P \rightarrow C, TVM_{ak}) - mf(P \rightarrow C, TVM_{vk}), 0) \tag{3.20}$$

Die Berechnung des Reputationswertes im Kontext der Consumerauswahl ähnelt stark dem in 2.3.5 vorgestellten einfachen Algorithmus 2.8. Sie unterscheiden sich lediglich darin, dass 2.8 eine relative Wahrscheinlichkeit berechnet, mit der ein Peer als Consumer ausgewählt wird. Die Funktion 3.20 berechnet hingegen einen absoluten Reputationswert. Beide Verfahren können Sybil-Attacken der Version A[22] verhindern, jedoch nicht Sybil-Attacken der Version B. Leider lässt sich die in 2.3.5 vorgestellte erweiterte Version des Algorithmuses, die diese Angriffe verhindert, nicht auf diesen Anwendungsfall übertragen. In 2.3.5 wird davon ausgegangen, dass die Teilnehmer des Netzwerks eine homogene Bandbreite besitzen und dass die Wahrscheinlichkeit, dass zwei Teilnehmer miteinander interagieren, für alle Teilnehmer identisch ist. Beide Eigenschaften treffen jedoch nicht

[22]Die Sybil-Attacken der Version A und B werden in 2.3.5 vorgestellt

für diese Arbeit zu. Zum einen besitzen Peers verschiedene Bandbreiten, diese werden in Abschnitt 4.3.2 definiert und die Wahrscheinlichkeit, inwieweit zwei Peers miteinander interagieren, ist stark davon abhängig, in welchen Clustern sie aktiv sind und welche Angebotsbreite sie besitzen.
Andere Möglichkeiten, wie die Sybil-Attacken der Version B verhindert werden könnten, werden in Kapitel 6 diskutiert.

Ein weiteres Problem liegt im minimalen Schnitt des Flussgraphen, der eine obere Schranke für den Reputationswert bestimmt. Ein Peer kann sich maximal eine Reputation in Höhe des minimalen Schnittes erarbeiten. Somit ist es für die Reputationsberechnung völlig unerheblich ob ein Peer deutlich mehr Leistungen geliefert hat als ein anderer Peer, solange beide mehr geliefert haben als die Kapazität des minimalen Schnittes beträgt.[23] Dieses Problem tritt vor allem bei Betrachtern auf, die erst wenige Transaktionen durchgeführt haben und somit über wenige eingehende und ausgehenden Kanten verfügen. Diese Kanten sind dann oft der Flaschenhals, der den maximalen Fluss begrenzt und somit bei vielen Peers zu gleichen Reputationswerten führt, unabhängig davon welcher Peer mehr Leistungen geliefert oder erhalten hat.

Zum Abschluss dieses Verfahren soll ein Beispiel die Berechnung der Reputationswerte verdeutlichen. Im Plausibilätsprüfungsschritt wird der Einfachheit halber das naive Verfahren verwendet und es wird davon ausgegangen, dass der Betrachter über folgende Tokenmenge verfügt:

$$T_A = \left\{ \begin{array}{l} t_{(A,B)} = (A, B, 0, 3, 1, 0), \\ t_{(B,C)} = (B, C, 0, 2, 1, 0), \\ t_{(B,D)} = (B, D, 0, 4, 1, 0), \\ t_{(D,E)} = (D, E, 0, 1, 1, 0), \\ t_{(E,B)} = (E, B, 0, 3, 1, 0), \\ t_{(F,B)} = (F, B, 0, 4, 1, 0), \\ t_{(D,G)} = (F, B, 0, 2, 1, 0), \\ t_{(E,G)} = (F, B, 0, 3, 1, 0) \end{array} \right\} \tag{3.21}$$

(3.22)

Daraus ergibt sich folgender Flussgraph:
folgende Reputationen im Kontext der Providerauswahl für den Betrachter Peer A:

[23] Vorausgesetzt für beide Peers ergibt sich der gleiche minimale Schnitt.

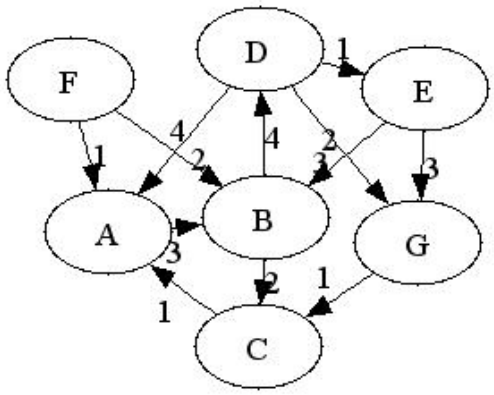

Abbildung 3.2.: Flussgraphen für die Consumerauswahl

$$rep_{cz(A,B)} = mf(A \rightarrow B, TVM_{ak}) = 3$$
$$rep_{cz(A,C)} = mf(A \rightarrow C, TVM_{ak}) = 3$$
$$rep_{cz(A,D)} = mf(A \rightarrow D, TVM_{ak}) = 3$$
$$rep_{cz(A,E)} = mf(A \rightarrow E, TVM_{ak}) = 1$$
$$rep_{cz(A,F)} = mf(A \rightarrow F, TVM_{ak}) = 0$$
$$rep_{cz(A,G)} = mf(A \rightarrow G, TVM_{ak}) = 3$$

und folgende Reputationen im Kontext der Consumerauswahl für den Betrachter Peer A:

$$rep_{pz(A,B)} = max(mf(A \rightarrow B, TVM_{ak}) - mf(A \rightarrow B, TVM_{vk}), 0) = 0$$
$$rep_{pz(A,C)} = max(mf(A \rightarrow C, TVM_{ak}) - mf(A \rightarrow C, TVM_{vk}, 0) = 2$$
$$rep_{pz(A,D)} = max(mf(A \rightarrow D, TVM_{ak}) - mf(A \rightarrow D, TVM_{vk}), 0) = 0$$
$$rep_{pz(A,E)} = max(mf(A \rightarrow E, TVM_{ak}) - mf(A \rightarrow E, TVM_{vk}), 0) = 0$$
$$rep_{pz(A,F)} = max(mf(A \rightarrow F, TVM_{ak}) - mf(A \rightarrow F, TVM_{vk}), 0) = 0$$
$$rep_{pz(A,G)} = max(mf(A \rightarrow G, TVM_{ak}) - mf(A \rightarrow G, TVM_{vk}), 0) = 2$$

3.3. Die Providerauswahl

Zusätzlich zu den gerade vorgestellten verschiedenen Möglichkeiten die Reputation zu berechnen, gibt es eine Vielzahl von Möglichkeiten mithilfe der berechneten Reputation einen Provider auszuwählen. An dieser Stelle sollen zwei Verfahren vorgestellt werden.

3.3.1. Peer mit höchster Reputation als Provider

Bei dieser Auswahlstrategie wählt ein Consumer immer den Provider mit der höchsten Reputation aus, in der Hoffnung bei ihm am schnellsten qualitativ hochwertige Leistung zu bekommen.

3.3.2. Probabilistische Providerauswahl

Bei genauer Betrachtung stellt die Wahl des Providers mit der höchsten Reputation, keine adequate Lösung für die Providerauswahl dar, denn wenn jeder Consumer bei dem besten Provider eine Anfrage stellt, ist dieser sehr schnell überlastet und Provider mit etwas geringerer Reputation bekommen keine Anfragen. Zusätzlich tritt ein selbstverstärkender Effekt ein. Es bekommen nur Peers, die bereits über eine hohe Reputation verfügen, Anfragen und können durch deren Abarbeitung ihre Reputation verbessern. Peers mit geringer Reputation bekommen keine Anfragen und können somit ihre Reputation auch nicht verbessern.
Bei der probabilistischen Providerauswahl wird ein Provider zufällig ausgewählt. Dies soll eine Lastverteilung bewirken, um dennoch die Reputation in die Entscheidung mit einfließen zu lassen, werden die Wahrscheinlichkeiten proportional zu ihrer Reputation gewichtet. Damit auch Peers mit unbekannten Reputation ausgewählten werden, wird zu jeder Wahrscheinlichkeit ein Mindestmaß addiert, dazu werden die Reputationen auf eins normalisiert, d.h. alle Reputationen werden durch den höchsten Reputationswert dividiert. Danach wird auf jeden Reputationswert der Mindestwert von eins durch Anzahl der Provider addiert.

3.4. Die Consumerauswahl

Bei der Consumerauswahl wird sich die Untersuchung auf eine Strategie beschränken. Es wird immer der Peer mit der höchsten Reputation als Consumer gewählt. Im Vergleich zur Providerauswahl wird eine zufällige Consumerauswahl nicht als sinnvoll erachtet, da dies nicht zu einer Lastverteilung führen würde, sondern lediglich zu einer ungerechteren Leistungsverteilung. Durch den Erhalt von Leistungen verschlechtert sich der Reputationswert eines Consumers. Deshalb tritt kein selbstverstärkender Effekt ein.
Falls mehrere Peers über die gleiche höchste Reputation verfügen, wird einer dieser Peers mit gleicher Wahrscheinlichkeit zufällig ausgewählt.

3.5. Die Tokenverteilung

Bei der Verteilung der Token im Netzwerk gibt es eine große Anzahl von Strategien. Unter dem Aspekt möglichst wenig Overhead durch das versenden der Token zu erzeugen, aber trotzdem andere Peers sinnvoll mit Informationen zu versorgen, stellen sich bei der Tokenverteilung folgende Fragen:

- **Warum soll ein Peer überhaupt Token verteilen?** Ein Peer wird nur solche Token im Netzwerk verteilen, die ihm einen gewissen Vorteil gegenüber anderen

Peers einbringen. Dies sind in erster Linie Token, die seine eigene Reputation steigern, aber auch Token die andere negativ belasten und ihn wiederum, in Relation gesehen, in ein besseres Licht stellen.

- **Wie soll die Verteilung ablaufen?** Man kann in erster Linie zwischen zwei Methoden unterscheiden, der Push- und der Pull-Methode.
  Bei der Pull-Methode erbittet ein Peer, dass ein anderer Peer ihm Token zusendet. Der Tokenzusteller würde dadurch profitieren, dass er den Tokenaustausch selbst in Rechnung stellen kann und dafür einen neuen Token generiert, der seine Leistung honoriert. Dies ist besonders sinnvoll für Peers die neu im Netzwerk sind und noch über keine Token verfügen.
  Bei der Push-Methode hingegen, verteilt ein Peer unaufgefordet die Token im Netzwerk. Dies spart den bei der Pull-Methode zusätzlich anfallenden Overhead ein, der durch die Anfrage entstehen würde. Jedoch werden dabei die Token sehr gestreut verteilt und es bleibt die Frage offen, ob der Tokenempfänger Verwendung für den Token findet.
  Unabhängig davon könnte man sich überlegen, ob nur Token versendet werden, deren Verteiler auch die Tokenerzeuger sind, oder ob auch Token, deren Tokenerzeuger sich vom Versender unterscheiden, versendet werden. Ersteres hätte den Nachteil, dass für die Verteilung eines Tokens alleine die zwei Erzeuger zuständig sind. Würden zusätzlich andere Peers diese Token mitverteilen, könnte man eine gewisse Lastverteilung erreichen, jedoch besteht dann die Gefahr, dass Peers Token mehrfach erhalten, da die Versender nicht wissen, ob dieser Peer den Token bereits von einem anderen Peer erhalten hat oder nicht.

- **Wann sollen Token verschickt werden** Ein Token kann sofort nach der Erstellung Versand werden oder erst zu einem späteren Zeitpunkt. Für einen Peer wäre es sinnvoll, den Token relativ zeitnahe zu versenden, um möglichst schnell davon zu profitieren. Jedoch steht der dadurch anfallenden Datentransfer in direkter Konkurrenz mit dem Datentransfer der eigentlichen Nutzdaten. Deshalb wäre es in manchen Situationen sinnvoll, die Token erst später zu verteilen, um eine zeitliche Lastverteilung zu erreichen.

- **Wie viele Token sollen verteilt werden?** Die Antwort auf diese Frage wird beeinflusst durch die Faktoren wie groß das Netzwerk ist, die Strukturen des Netzwerks, wie hoch die Auslastung eines Peers ist und welchen Nutzen es für einen Peer hätte, weiter Token zu verschicken. Des Weiteren könnte man sich vorstellen, dass die Anzahl der verschickten Token nach einer gewissen Zeit abnimmt, da mehr Kenntnisse über die Netzwerkstruktur gesammelt werden und deshalb die Token gezielter versendet werden könnten.

- **Wem soll der Token zugestellt werden?** Aus kollektiver Sicht sollte demjenigen der Token zugestellt werden, der dadurch am meisten Kenntnisse gewinnt. Jedoch mit dem Hintergrund der Spieltheorie und der subjektiven Sicht des Zustellers, sollte ein Token demjenigen zugesandt werden, bei dem der Zusteller am meisten profitiert. Diese beiden Gesichtspunkte müssen sich nicht zwangsläufig widersprechen.

Auf diese Fragen wird sich keine für alle P2P Netzwerke allgemeingültige Lösung finden lassen. Aufgrund der Vielzahl an Möglichkeiten können in dieser Arbeit nicht alle untersucht werden, deshalb wird ein Schwerpunkt auf die Frage, wem die Token zugeschickt

werden sollten, gelegt.
Auf die Frage, wie und wann die Token versendet werden, wird von folgendem Verhalten ausgegangen. Die Token werden mithilfe der Pushmethode im Netzwerk verteilt, dabei versenden nur die Erzeuger ihre Token, d.h. die Token werden immer aus erster Hand und nicht über zweite zugestellt. Es versendet immer nur einer der beiden Tokenerzeuger den Token und zwar derjenigen, der aus dem Token mehr Ansprüche als Verpflichtungen gelten machen kann, d.h. es versendet nur der Tokenerzeuger den Token, der dadurch profitieren kann. Für den anderen Tokenerzeuger würde sich das Versenden negativ auswirken, da aus dem Token mehr Verpflichtungen als Ansprüche für ihn hervorgehen.
Der Zeitpunkt der Verteilung ist direkt nach der Erzeugung, bzw. Modifikation eines Tokens, d.h. direkt nach einem erfolgten Datentransfer ohne Zeitverzögerung und zu keinem anderen Zeitpunkt.
Die Anzahl der verteilten Token wird während der gesamten Simulationszeit konstant bleiben und für alle Peers identisch sein.

3.6. Tokenempfängerwahl

In diesem Abschnitt werden verschiedene Strategien zur Tokenempfängerwahl vorgestellt. Diese Strategien bestehen aus einer Kernstrategie und verschiedenen ergänzenden Optimierungsverfahren. Durch die Analyse der Netzwerkstruktur versuchen die Kernstrategien, Token gezielt an Peers zu versenden, die ein hohes Interesse an den Tokeninformation besitzen könnten.
Die Optimierunsgverfahren versuchen die Ergebnisse der Kernstrategie durch Betrachtung weiterer Gesichtspunkte zu verbessern.
Als Ergebnis liefert die Kernstrategie einen Vektor $s\vec{w}v$[24], der mittels komponentenweiser Multiplikation mit den Ergebnisvektoren der Optimierungsmethoden $o\vec{w}v$[25] verrechnet wird, um den eigentlichen Reputationsvektor $re\vec{p}_{tz}$ für die Tokenempfängerwahl zu erhalten. Peers mit der höchsten Reputation erhalten den Token.

$$re\vec{p}_{tz(i)} = s\vec{w}v_i .*^{26} o\vec{w}v_i \tag{3.23}$$

3.6.1. Kernstrategien für die Tokenempfängerwahl

In den zwei folgenden Abschnitten werden Kernstrategien zur Tokenempfängerwahl erläutert, die sich die Strukturen des Netzwerks zu nutze machen. Im Anschluss daran, werden zwei weitere Verteilungsstrategien vorgestellt, die als Referenzverfahren dienen, um den Nutzen der Strukturanalyseverfahren aufzuzeigen.

3.6.1.1. Nachbarschaftsverteilung

Die Nachbarschaft eines Peers besteht aus den Peers, mit denen er in der Vergangenheit Transaktionen durchgeführt hat. Es besteht eine erhöhte Wahrscheinlichkeit, dass die Nachbarn untereinander Transaktionen durchführen, hier sei auf die eingangs erwähnte

[24] structure weighting vector
[25] optimizing weighting vector
[26] Die Operation ".∗" entspricht dem komponentenweisen Produkt.

Small World Network Eigenschaft 2.2.1.2 des P2P Netzwerks verwiesen.
Um die unterschiedlichen Transaktionsintensität zwischen dem Betrachter i und seinen Nachbarn zu berücksichtigen, wird jeder Nachbar mit dieser Intensität gewichtet. Sie entspricht dem Tokenvolumen zwischen dem Betrachter und seinem Nachbarn.

$$\vec{swv}_i = \vec{tvv}_{tk(i)} \tag{3.24}$$

3.6.1.2. Tokenverteilung mittels Ähnlichkeitsbewertung

Die Tokenverteilung kann als eine Art Empfehlungssystem betrachtet werden, bei dem Peers andere Peers empfehlen, mit den sie in der Vergangenheit gute Erfahrungen gemacht haben, bzw. von den sie abraten, weil sie mit ihnen schlechte Erfahrung gemacht haben.
Bei herkömmlichen Recommender-Verfahren werden den Usern Empfehlungen gemacht, die ein ähnliches Konsumverhalten haben, wie derjenige der die Empfehlung ausspricht. Im Gegensatz dazu, wird in 2.3.6 die Ähnlichkeit zweier Peers über die Ähnlichkeit ihrer bereitgestellten Daten ermittelt, d.h. in diesem Fall sind Peers zu einander ähnlich, falls sie ein ähnliches Angebotsverhalten besitzen.
Die Kombination von Konsumverhaltens und Angebotsverhalten entspricht dem Transaktionsverhalten eines Peers, da eine Transaktion sowohl ein Leistungsangebot als auch Leistungsverbrauch des Peers darstellen kann. Überträgt man dies auf die Tokenverteilung, entspricht die Ähnlichkeit der Ähnlichkeit des Transaktionsvehaltens der Tokenerzeuger und einem potentiellen Tokenempfänger.
Diese Ähnlichkeit kann berechnet werden, indem jeweils die Ähnlichkeit zwischen den beiden Tokenerzeugern und einem potentiellen Empfänger gebildet wird und beide Ergebnisse miteinander multipliziert werden. Diese Multiplikation bewirkt, dass ein Peer mit beiden Tokenerzeugern eine hohe Ähnlichkeit aufweisen muss, um als Tokenempfänger in Frage zu kommen. Würde er mit nur einem der beiden Erzeuger eine Ähnlichkeit aufweisen, so besteht eine gewisse Wahrscheinlichkeit, dass dieser Peer nicht an dem Token interessiert sein könnte, da er keine Ähnlichkeit und somit auch keine Beziehung zu dem anderen Erzeuger besitzt.
Um nun die Ähnlichkeit zweier Peers berechnen zu können muss zuvor eine Ähnlichkeitsmetrik festgelegt werden, diese wird im Folgenden als $vSim$[27] bezeichnet. Es hat sich durch verschiedene Vergleichssimulationen herausgestellt, dass das einfache Skalarprodukt ein gutes Ergebnis liefert. Als Eingabevektoren für das Skalarprodukt dienen die beiden Tokenbewertungsvektoren der betrachteten Peer. Dabei werden die Token im Kontext des Tokenvolumens betrachtet. Der Tokenbewertungsvektoren entspricht somit dem Transaktionsverhalten der Peers. Dadurch ergibt sich für $vSim$ ein großes Ergebnis, sobald die beiden betrachteten Peers eine hohe Ähnlichkeit in ihrem Transaktionsverhalten aufweisen, da in diesem Fall das Skalarprodukt der beiden Vektoren entsprechend groß ist.

$$vSim_{(m,k)} = < \vec{tvv}_{tk(m)}, \vec{tvv}_{tk(k)} > \tag{3.25}$$

Schließlich muss das Produkt der beiden Ähnlichkeiten zwischen dem potentiellen Tokenempfänger Peer k und den Tokenerzeugern Peer i und Peer j gebildet werden. Dieses Ergebnis wird als Token-Peer-Ähnlichkeit $tps(i,j,k)$[28] bezeichnet, diese entspricht dem

[27] Voting Similarity
[28] Token-Peer-Similarity

k-ten Eintrag des $\vec{swv}_i$, sowie der Ähnlichkeit zwischen den Tokenerzeugern des Tokens $t_{(i,j)}$ und dem Peer k.
Es es ist zu beachten, dass Peer i sowohl Tokenerzeuger als auch Betrachter bzw. Tokenversender ist. Diese Eigenschaft ist nur zu lässig, da in Abschnitt 3.5 festgelegt wurde, dass die Tokenerzeuger nur ihre eigenen Token verteilen. Würde ein Tokenverteiler fremde Token verteilen, müsste die Formel 3.26 entsprechend angepasst werden.

$$\vec{swv}_{i(k)} = tpsv_{(i,j,k)} = vSim_{(i,k)} * vSim_{(j,k)} \tag{3.26}$$

An dieser Stelle wird eine kleine Schönheitskorrektur eingeführt und zwar wird, falls einer der beiden Multiplikatoren 0 ist, dieser um einen relativ kleinen Wert erhöht. Dieses Vorgehen liegt darin begründet, dass das Interesse eines potentiellen Tokenempfänger an einem Token größer ist, wenn er eine gewisse Ähnlichkeit mit wenigstens einem Tokenerzeuger besitzt, als wenn er gar keine Ähnlichkeit mit beiden Erzeugern besitzen würde. Dies sollte dann auch entsprechend bei der Berechnung berücksichtigt werden. Deshalb wird $vSim_{s(m,k)}$ mit 0.01 maximiert.
Bei dieser Tokenverteilmethodik sind die Reputationswerte im Vergleich zur Nachbarschaftsverteilung nicht nur alleine abhängig von dem Betrachter und den zu bewertenden Peer, sondern auch von dem Token, der im Netzwerk verschickt werden soll. Dies ermöglicht eine für jeden Token individualisierte Verteilung im Netzwerk.
Mithilfe dieser Ähnlichkeitsbestimmung wird der Token nur an Peers geschickt, die im gleichen Transaktionscluster wie die Erzeuger liegen. Dadurch wird, im Vergleich zur Nachbarschaftsverteilung, die Streuung der Tokenverteilung weiter reduziert und es bekommen nur die Peers einen Token, die eine erhöhte Wahrscheinlichkeit besitzen mit beiden Tokenerzeugern zu interagieren. Bei der Nachbarschaftsverteilung ist dies nichts zwangsläufig gegeben, so kann der Nachbar eines Tokenerzeugers in einem anderen Cluster aktiv sein als der andere Tokenerzeuger.

3.6.1.3. Uneingeschränkte Verteilung

Bei der uneingeschränkten Verteilung verschickt ein Peer seine erzeugten Token an alle User im Netzwerk. Dadurch ist jeder Peer über jede Transaktion informiert, vergleichbar mit einer globalen allwissenden Informationsquelle, auf die jeder Peer zugreifen kann. Diese Tokenverteilung stellt die optimale Informationsgrundlage dar, um eine Entscheidung zu treffen und dient somit als Referenzmethode für die bestmögliche Tokenverteilung.
In der Praxis wäre dieser Ansatz jedoch offensichtlich schnell zum scheitern verurteilt, da er nicht skaliert und somit für etwas größere P2P Netzwerke völlig ungeeignet ist.

3.6.1.4. Keine Verteilung

Bei diesem Verfahren werden keine Token im Netzwerk verbreitet. Es findet somit kein Informationsaustausch zwischen den Peers statt. Jeder Peer trifft seine Entscheidungen lediglich anhand seiner persönlich gesammelten Erfahrungen. Dieses Verfahren stellt das schlechteste hier aufgeführte Verfahren dar.

3.6.1.5. Zufällige Verteilung

Ein weiterer Ansatz ist die zufällige Verteilung, bei der die Token an zufällige Peers im Netzwerk verschickt werden. Hier kann der Nutzen des Versendens des Tokens verschwindend gering ausfallen, da der Empfänger nur mit einer geringen Wahrscheinlichkeit mit einem der beiden Tokenerzeugern kommunizieren wird und sich deswegen auch keine Meinung über diese Peers bilden muss.

3.6.2. Optimierungsverfahren der Tokenverteilung

Die Tokenverteilung nur in abhängig von der Netzwerkstruktur zu betrachten, ist ein wenig kurzsichtig. Es bieten sich eine Vielzahl von weiteren Möglichkeiten an, die Tokenverteilung zu optimieren. In dieser Arbeit sollen zwei solcher Verfahren vorgestellt werden, die Versionsgewichtung des Tokens und die eigennützige Tokenverteilung. Beide Optimierungsverfahren lassen sich mit den Kernstrategien kombinieren.

3.6.2.1. Versionsgewichtung des Tokens

Bei diesem Verfahren wird der zeitliche Aspekt bzw. die Version, der bereits verschickten Token, berücksichtigt.
Nach jeder Transaktion zwischen den selben beiden Transaktionspartnern entsteht eine neue Version eines Tokens. Je höher die Version eines Tokens ist, desto geringer wird sich der Tokeninhalt zwischen zwei aufeinanderfolgenden Versionen ändern. In einem konkreten Beispiel sieht dies wie folgt aus. Die beiden Transaktionspartner Peer P und Peer C führen mehrere Transaktion hintereinander durch und erzeugen nach jeder Transaktion eine neue Tokenversion. Dabei ist P Provider und C entsprechend Consumer. Der Informationszuwachs zwischen der ersten und zweiten Tokenversion beträgt 100%, hingegen zwischen der zehnten und elften Tokenversion nur 10%.
Dies sollte ein Tokenversender bei der Auswahl der Tokenempfänger beachten. Hat der Tokenversender bereits ein ältere Version eines Tokens an potentielle Empfänger versendet, so gilt es den Informationszuwachs zwischen der alten und neuen Tokenversion zu gewichten und entsprechend eher einen Empfänger auszuwählen, der eine weitaus ältere Tokenversion besitzt, als ein Peer mit einer neueren Tokenversion.
Um diesen Aspekt in die Berechnung des optimalen Tokenempfänger einfließen zu lassen, wird die Tokenversionsgewichtung eingeführt. Die Tokenversionsgewichtung entspricht Eins dividiert durch die letzte Versionsnummer des Tokens. Die Tokenversionsgewichtung können zu einem Tokenversionsgewichtungsvektor $t\vec{v}v$[29] zusammengefasst werden. Dieser Vektor entspricht dem Optimierungsgewichtungsvektor $o\vec{v}w'$.
Zur Formalisierung des Algorithmuses wird die Tokenversionsfunktion $tv_{(t)}$ benötigt. Diese erhält als Parameter einen Token und gibt dessen Tokenversion zurück. In folgender Definition entspricht t_a dem aktuellen Token, der im Netzwerk verteilt werden soll und t_j, dem Token der zuletzt an Peer j versendet wurde und dessen Tokenerzeuger identisch mit t_a sind.

$$o\vec{w}v'_{i(j)} = t\vec{v}v_{i(j)} = \begin{cases} \frac{tv_{(t_a)}}{tv_{(t_i)}}, & \text{falls } t_i \neq null \\ 1 & \text{sonst} \end{cases} \tag{3.27}$$

[29] Token Version Weighting Vector

3.6.2.2. Eigennützige Tokenverteilung

Übertragt man die Kernaussage der Spieltheorie auf ein P2P Netzwerks, so kann davon ausgegangen werden, dass jeder Peer in erster Linie an seinem eigenen Wohlergehen interessiert ist und weniger am Allgemeinwohl. Ein Peer nimmt unter Umständen in Kauf, Entscheidungen zu treffen, die schädlich für das Allgemeinwohl sein können, ihn selbst aber einen gewissen Vorteil verschaffen könnten.
Jeder Peer möchte, dass seine Anfragen möglichst schnell vom Provider bearbeiten werden, dies erreicht er, indem er eine gute Reputation beim Provider besitzt, diese wiederum erhält er, indem der Provider gut über die erbrachten Leistungen des potentiellen Consumers informiert ist.
In diesem Sinne senden eigennützige Peers, ihre Token gezielt an die Peers, an die sie bereits eine Anfrage gestellt haben, um so eine vermeintlich höher Chance zu erzielen, von ihnen ausgewählt zu werden. Erreicht wird dies indem das Ergebnisse der Kernstrategie mit 0.001 multipliziert wird, falls der Tokenverteiler keine unbearbeitete Anfrage an den jeweilige Peer gestellt hat. Im Gegenzug bleiben die Ergebnisse der Kernstrategie für Peers, bei denen der Tokenverteiler eine Anfrage gestellt hat, unverändert. R_i entspricht in folgender Formalisierung der Menge der Peers, an die der Betrachter Peer i eine Anfrage gestellt hat.

$$\vec{owv''}_{i(j)} = \begin{cases} 1 & \text{falls } P_i \in R_i \\ 0.0001 & \text{sonst} \end{cases} \tag{3.28}$$

4. Simulator

4.1. Architektur

Für die Evaluierung, der im vorherigen Kapitel vorgestellten Strategien, ist es notwendig, verschiedene Simulationen durchzuführen. In diesem Kapitel soll die Funktionsweise des dazu verwendeten Simulators beschrieben werden. Zunächst soll in Abschnitt 4.1.1 bis 4.2 der Simulator unter programmatischen Gesichtspunkten betrachtet werden. In Abschnitt 4.3 liegt der Schwerpunkt auf der Implementierung der eigentlichen Simulationslogik. Dort wird der Simulationsablauf beschrieben.
In 4.4 wird erläutert, wie die während der Simulation entstandenen Daten vom Simulator ausgewertet werden. Abgeschlossen wird dieses Kapitel mit dem Abschnitt der Qualitätssicherung 4.5.

4.1.1. Anforderungen

Die Anforderungen an den Simulator sind sehr vielschichtig. Zum einen muss er den Austausch verschiedener Entscheidungsstrategien ermöglichen und zum anderen die Möglichkeit gewährleisten Simulationen unter verschiedenen Rahmenbedingungen durchzuführen. Diese unterschiedlichen Rahmenbedingungen werden im Folgenden als Szenario bezeichnet. Als Szenario kommen verschiedene Netzwerkstrukturen in Frage, die sich unter anderem in der Anzahl ihrer Peers und der Anzahl ihrer Cluster unterscheiden.
Des Weiteren spielt die Simulationszeit und der Speicherbedarf eine elementare Rolle bei der Entwicklung des Simulators. Die Rechenzeit und der Speicherbedarf stehen in einer direkter Konkurrenz zueinander, man denke hier z.B. an die Möglichkeit des Cachings. Es muss ein Kompromiss gefunden werden und je nach Simulationsbedingung sollten verschiedene Speicherverwaltungsstrategien umgesetzt werden können, dies setzt wiederum einen gewissen Grad an Modularität und Flexibilität des Simulators voraus.
Um eine nachvollziehbare Simulation durchzuführen, müssen die einzelnen Simulationen wiederholbar durchführbar sein und dabei, bei gleichen Simulationsbedingungen, die gleichen Ergebnisse liefern. Es ist zu beachten, dass der verwendete Zufallszahlengenerator, bei Mehrfachausführung identische Ergebnisse liefert, um eine Reproduzierbarkeit der Messergebnisse zu garantieren.

4.1.2. Architekturkonzept

Um diesen Anforderungen gerecht zu werden, ist es notwendig, ein möglichst flexibles Architekturkonzept zu wählen. Solch ein Konzept ist das “Model View Controller” (MVC) Konzept [20]. Dieses hat den Vorteil, dass die drei Komponenten Datenmodell, Visualisierung und Steuerung für sich gekapselt sind, somit relativ unabhängig voneinander entwickelt werden können und je nach Simulationsbedingungen ausgetauscht werden können.
In den folgenden drei Abschnitten sollen diese Komponenten genauer betrachtet werden.

Im Abschnitten 4.2 werden die einzelnen Datenobjekte erläutert. Der Abschnitten 4.3 befasst sich mit der Konfiguration und Steuerung des Simulators und in Auswertung 4.4 wird auf die statistische Auswertung der Simulationsergebnisse eingegangen und entspricht somit der Visualisierung im MVC Konzept.

4.2. Das Datenmodell

Das Datenmodell muss anfallende Daten verwalten, d.h. speichern und wieder abrufbar machen. Die Daten werden als Attribute in Objekten gesammelt und über entsprechende Getter- und Setter-Methoden zugänglich gemacht.
Des Weiteren hat das Datenmodell dafür zu sorgen, berechnete Daten zu cachen, um Rechenzeit einzusparen. Dabei muss darauf geachtet werden, dass die gecachten Werte konsistent bleiben, d.h. sie müssen gelöscht werden, sobald sich Daten verändern, aus denen der gecachte Wert berechnet wurde.
Die einzelnen Datenobjekte werden unterteilt in atomare und in komplexere. Letztere dienen als Container der atomaren Objekte. Die atomaren Klassen befinden sich im Package *model*, die der komplexeren Objekte in *model.container*

4.2.1. Das Peer-Objekt

Zentrale Klasse des Datenmodells stellt die atomare Peer-Klasse dar. Neben ihren primitiven Eigenschaften, die zunächst erläutert werden sollen, verwaltet ein Peer auch die komplexeren Datenstrukturen Token-Container, Warteschlange und die Peer-Cluster-Relation, die in den entsprechenden Unterkapiteln genauer betrachtet werden. In Diagramm 4.1 werden diese Beziehung dargestellt.
Sie besitzt die beiden Eigenschaften *avgNumberOfUploads* und *avgNumberOfDownloads*. Diese beschreiben die Aktivität eines Peers. *avgNumberOfUploads* steht für die durchschnittliche Anzahl, wie oft ein Peer während der gesamten Laufzeit als Provider agiert und entsprechend steht *avgNumberOfDownloads* für die Anzahl, wie oft er als Consumer eine Anfrage an einen Provider stellt.
Die *maxRange* Eigenschaft beschreibt die Angebotsbreite eines Peers. Je höher sie ist, desto wahrscheinlicher ist es, dass dieser Peer eine Anfrage eines Consumer bekommt.
Außerdem besitzt jeder Peer eine Kennung, den Index, über die jeder Peer eindeutig Identifizierbar ist, dieser wird bei der Netzwerkgenerierung definiert.

4.2.1.1. Der Token-Container

Der Token-Container dient der Verwaltung der einzelnen Token eines Peers, dies entspricht der Tokenmenge T_i, die im vorherigen Kapitel vorgestellt wurde. Das Hauptproblem, liegt in der Vielzahl der Token, die für jeden Peer gespeichert werden muss.
Die Komplexität dieses Problems wird deutlich, wenn man versucht dieses naiv zu lösen. Geht man von einem Netzwerk mit 10.000 Peers aus, so können in diesem Netzwerk 99.990.000 verschiedene Token erzeugt werden. Speichert man nun die Referenzen auf diese Token in einer Matrixstruktur mit vordefinierten Größe und angenommen jede Referenz belegt ein Byte, so muss für die Tokenmenge eines Peers ein Speicher von rund 100 MB reserviert werden und da nicht nur ein Peer sondern 10.000 Peers simuliert werden, wird ein Speicherplatz von einem TB benötigt. Dies entspricht einer Komplexität für den Speicherbedarf von n^3.

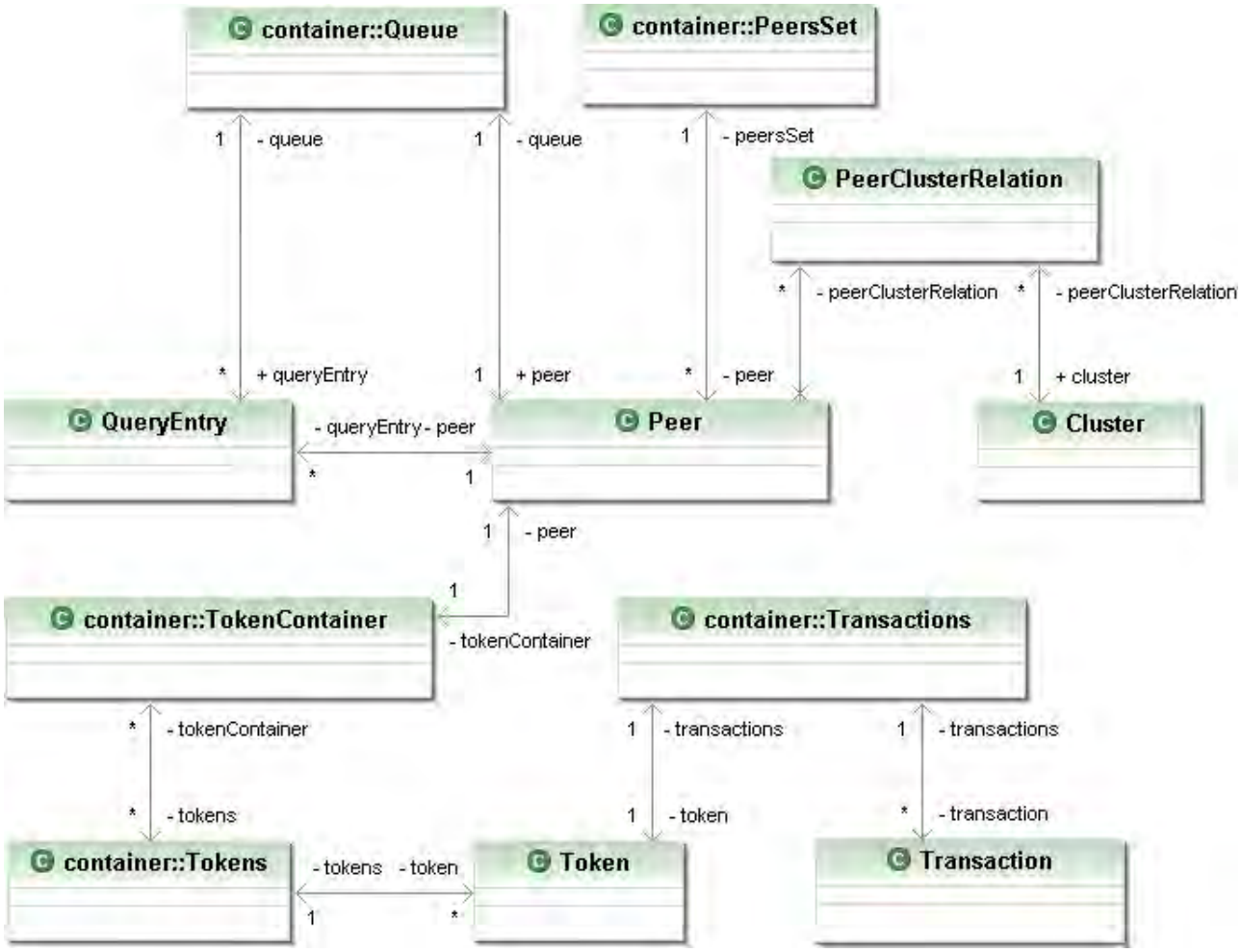

Abbildung 4.1.: Beziehung des Peer-Objekts

Versucht man nun Speicherplatz zu sparen, indem die Token nicht in einer Matrix mit vordefinierter Größe gespeichert werden, sondern in einer Liste mit dynamischer Länge, so wird sehr schnell ersichtlich, dass dies zu einem Laufzeitproblem führt. Spätestens wenn ein Peer mehrere tausend Token besitzt und aufgrund seiner Berechnungen über diese Tokenliste mehrfach iterieren muss, um einzelne Token zu finden, wird diese Tokenliste sehr schnell zu einem Flaschenhals für den gesamten Simulator.
Wie aus diesen Beispielen ersichtlich wird, muss der Token-Container zum einen speichereffizient zum anderen geschwindigkeitseffizient sein und beides steht in einem Gegensatz zueinander. Zwischen diesen beiden Extrema muss ein Kompromiss gefunden werden. Die im Simulator implementierte Lösung verwendet ein Array mit der Länge n. Jedes Feld dieses Arrays speichert eine Liste von Token. Wird nun eine Tokenreferenz in dieser Struktur abgelegt, so werden die beiden Indizes der Tokenerzeuger als Index für das Array verwenden und die Referenz in beiden zugehörigen Listen gespeichert. Dies hat zwar den Nachteil, dass jede Referenz zweimal in der Struktur abgelegt werden muss, jedoch lässt sich so ein Token sehr schnell auffinden und trotzdem durch die dynamische Länge der Liste Speicherplatz sparen. Dies setzt voraus, dass ein Peer nur einen ausgewählten Teil an Token erhält, um so die Länge der Listen minimal zu halten. Dies erfordert wiederum eine effiziente Tokenverteilung. Somit beeinflusst eine effiziente Tokenverteilung nicht nur den Erfolg des Reputationssystems, sondern auch die Laufzeit der Simulation.

4.2.1.2. Die Warteschlange

Die Peers, die bei dem aktuellen Peer eine Anfrage gestellt haben, werden bei ihm in die Warteschlange eingereiht. Diese Warteschlange ist im Vergleich zu einer realen Warteschlange, keine sortierte Liste von Objekten sondern lediglich eine Menge von unsortierten Objekten. Erst wenn ein Peer seine Warteschlange abarbeiten will, bewertet er die enthaltenen Peers je nach seiner Consumerauswahlstrategie und wählt den am höchsten bewerteten Peer als Consumer aus.
Der Grund für diese Implementationsvariante liegt darin, dass die Berechnung der Bewertung der einzelnen Peers in der Warteschlange durch viele Faktoren bestimmt wird. Diese Faktoren ändern sich in der Regel sehr oft, so reicht allein das Erhalten eines einzelnen Tokens aus, dass sich die Bewertungen der Peers in der Warteschlange grundlegend verändert. Deshalb ist das Verwalten einer sortierten Liste nicht praktikabel.
Eine Mehrfacheintragung eines Peers in die Warteliste ist möglich. Der Parameter *queueLength* gibt an, wie viele Einträge in der Warteschlange gespeichert werden können. Diese Länge wird jedoch erst überprüft, wenn der Provider eine Anfrage abarbeiten will. Stellt er dabei fest, dass in seiner Warteschlange mehr Peers eingetragen sind als zulässig, werden für alle Peers die Reputationen berechnet und die schlechtesten Peers aus der Warteschlange entfernt.

4.2.1.3. Die Peer-Cluster-Relation

Die Peer-Cluster-Relation gibt an, wie aktiv ein Peer in einem Cluster ist. Dabei wird die Beziehung als prozentualer Wert angegeben. Die Beziehung zu einem Cluster in dem ein Peer nicht aktiv ist, hat den Wert 0. Besitzt ein Peer eine Beziehung von 100 zu einem Cluster, dann ist er nur in diesem Cluster aktiv. Die Summe aller Relationen beträgt 100.

4.2.2. Gruppierung der Peers durch Peer-Sets

Peers, die gleiche Strategien verfolgen, werden zu sogenannten Peer-Sets zusammengefasst. Dies ist eine Menge von Peers, die zur Strukturierung der Simulation dient. Ein Peer-Set ist eine Gruppe von Peers, die einzeln analysiert und miteinander verglichen werden können. So kann z.B. die Wechselwirkung zweier Peer-Sets, die jeweils eine unterschiedliche Strategie verfolgen, verglichen werden.

4.2.3. Die Token

Das Tokenobjekt symbolisiert den Token, der im Netzwerk versendet wird. Um einen möglichst flexiblen Simulator zu gewährleisten, müssen alle vergangenen Transaktionen im Token archiviert werden. Dies hat den Vorteil, dass jeder Peer die Tokenbewertung, je nach seiner gewählten Strategie, berechnen kann. Jedoch ist dieser Vorgang entsprechend rechen- und speicherintensiv.
Die Verwaltung der verschiedenen Versionen eines Tokens erfolgt, indem für jede Tokenversion ein neues Tokenobjekt erzeugt wird.

4.2.4. Das Environment

Ein Environment-Objekt verwaltet alle relevanten Umgebungsvariablen eines Szenarios. Hierzu zählt das *Network*-Objekt und das *Rounds*-Objekt. Das *Network*-Objekt verwaltet die Strukturen des Netzwerks. Das *Rounds*-Objekt ist eine Liste von *Round*-Objekten. Jedes *Round*-Objekt zeichnet die in einer Runde entstandenen Transaktionen auf. Anhand dieser Aufzeichnungen findet die Auswertung des Szenarios statt. Diese Beziehungen werden in Grafik 4.2 illustriert.
Damit der Vergleich von mehreren Szenarien während einer Simulation möglich ist, wird für jedes Szenario ein eigenes Environment angelegt.

4.2.5. Der Simulator

Die Simulator-Klasse steuert die sequentielle Simulation der einzelnen Szenarien. Somit verwaltet diese Klasse auch alle Environment-Instanzen einer Simulation. Der Hintergedanke, der hinter dieser sequentiellen Ausführung der Szenarien während einer Simulation steht, ist, dass die Messergebnisse der Szenarien bei Simulationsende miteinander verglichen werden können, ohne dass ein zusätzlicher Arbeitsgang nötig wäre.

4.3. Konfiguration und Steuerung

Dieser Abschnitt beschäftigt sich mit der eigentlich Logik des Simulators. Alle hier beschrieben Klassen befinden sich im Package *control* und dessen Unterpaketen.

4.3.1. Die Konfiguration

Die Konfiguration des Simulators erfolgt zum einen mittels einer XML-Datei, die beim Starten des Simulators als Parameter angegeben wird und zum anderen über die sogenannte Strategy-Klassen. In Grafik 4.3.1.1 wird der Aufbau dieser Konfigurationsdatei

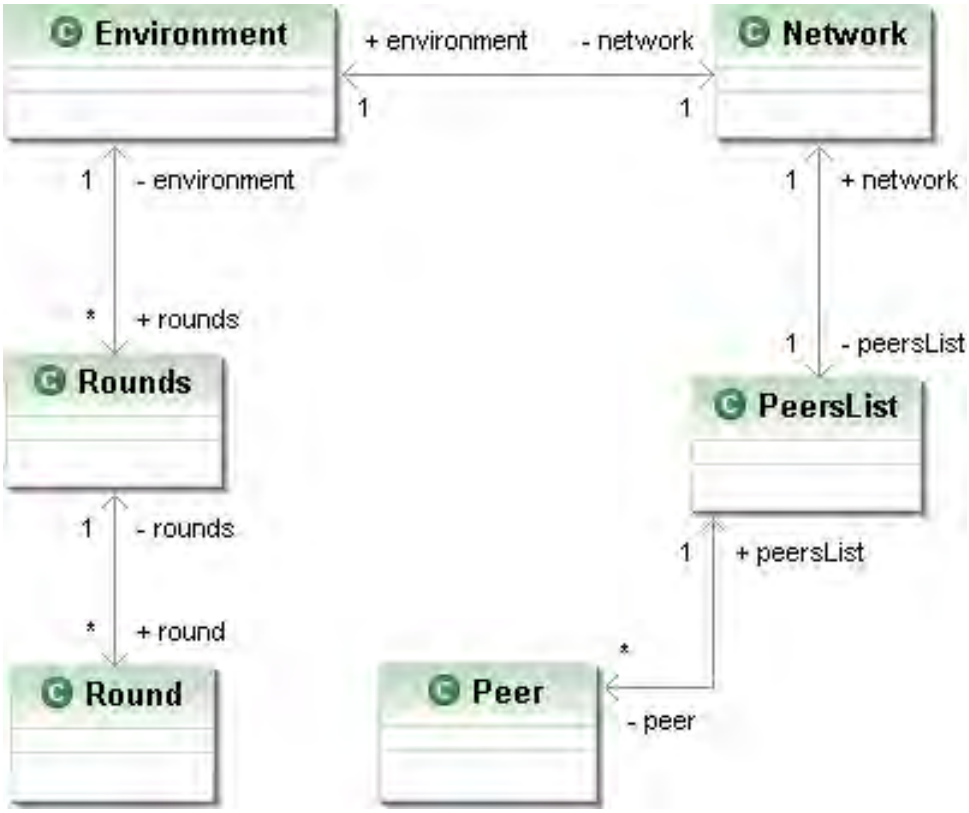

Abbildung 4.2.: Beziehung des Environment-Objekts

genauer erläutert.
In der XML-Datei können primitive Parameter definiert werden. Diese bestehen aus Integer und Double Werten. Übersicht 4.3.1.2 listet diese auf. Des Weiteren werden in der XML-Datei die Klassennamen, der zu ladenen Strategy-Klassen, definiert. Diese Strategy-Klassen erlauben es, Simulationsabläufe mit einer komplexen Logik zu definieren. Sie beschreiben das Verhalten des Simulators und der Peers in bestimmten Situationen. In Abschnitt 4.3.1.3 werden sie genauer erläutert. Alle in dieser Arbeit verwendeten Konfigurationsdateien befinden sich im Anhang A.5.

4.3.1.1. Aufbau der XML-Konfigurationsdatei

Im Codelisting 4.1 wird der Rumpfaufbau der Konfigurationsdatei aufgeführt. Hauptbestandteil der Konfiguration sind die drei ineinander verschachtelten Tags *<testSeries>*,*<scenario>* und *<peersSet>*. Dabei besteht der äußerste Rahmen aus dem Tag *<testSeries>*, dieser beinhaltet alle für eine Simulation relevanten Daten. Innerhalb dieses Tags können mehrere Szenarien mittels des *<scenario>*-Tag definiert werden. Dieser Tag wiederum kann aus mehreren *<peersSet>*-Tags bestehen. Jeder dieser Tags beschreibt eine Gruppe von Peers[1].
Für jeden dieser Tags kann der Parameter *name* gesetzt werden. Dieser dient lediglich der Identifikation und der Sortierung der Auswertungsdateien in Verzeichnissen.
Innerhalb der Tags können die im Folgenden genauer beschriebenen primitiven Parameter und die Strategy-Klassen definiert werden. Dabei erfolgt die Konfiguration rekursiv.

[1] siehe 4.2.2

Soll z.B. die Anzahl der Peers für das *peersSet1* im *scenarioName1* bestimmt werden, so wird zuerst innerhalb des *<peersSet name="peersSet1">*-Tag nach dem Tag *<numberOfPeers>* gesucht. Ist er dort nicht definiert, wird er innerhalb des *<scenario name="scenarioName1">* gesucht. Wird er dort ebenfalls nicht gefunden wird er im äußeren Tag *<testSeries name="simulationName">* gesucht. Falls er dort auch nicht zu finden ist werden Standardwerte verwendet, die im Folgenden definiert werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="simulationName">

  <scenario name="scenarioName1">
     <peersSet name="peersSet1">
    </peersSet>
     <peersSet name="peersSet2">
    </peersSet>
  </scenario>

  <scenario name="scenarioName2">
     <peersSet name="peersSet1">
    </peersSet>
     <peersSet name="peersSet2">
    </peersSet>
  </scenario>

</testSeries>
```

Listing 4.1: Rumpfaufbau der XML-Konfigurationsdatei

4.3.1.2. Die primitiven Parameter

In der folgenden Auflistung sind alle Konfigurationsparameter der XML-Datei aufgeführt. Die Angegebene Typbeschreibung in der ersten eckigen Klammer, definiert die zulässigen Eingabewerte und der angegebenen Wert in der zweiten Klammer entspricht dem Standardwert, der verwendet wird, falls kein anderer Wert definiert wird.

- **numberOfPeers [int][100]:**
  Anzahl der Peers pro Peer-Set
- **maxRange [int][10]:**
  Angebotsbreite eines Peers
- **meanDownStream [int][100]:**
  Anzahl der durchschnittlichen Downloads pro Peer
- **reliability [double][1.0]:**
  Zuverlässigkeit eines Peers
- **qualityMultiplier [double][1.0]:**
  Multiplikator bei der Qualitätsbewertung eines Services
- **numberOfDistributedTokensPerTransaction [int][10]:**
  Anzahl der Token, die pro Transaktion erzeugt werden

- **numberOfRounds [int][1000]:**
  Anzahl der Runden die simuliert werden
- **numberOfClusters [int][3]:**
  Anzahl der Cluster die simuliert werden
- **clusterTau [double][2.0]:**
  Tau-Parameter der bei Generierung des Netzwerks die Verteilung der Peers auf die Cluster regelt.
- **protectedCluster [int][-1]:**
  Geschütztes Cluster, das bei der Generierung des Netzwerks nicht belegt wird, außer es wird über den Parameter *defaultCluster* definiert. Falls kein Cluster geschützt werden soll, muss ein Wert von -1 angegeben werden.
- **defaultCluster [int][-1]:**
  Index eines Cluster, das ein Peer auf jeden Fall belegt. Falls ein Wert von -1 für den Parameter definiert wird, besitzt der Peer kein vordefiniertes Cluster.
- **queueLength [int][10]:**
  Maximale Anzahl an Anfragen, die ein Peer vorhält.
- **randomInitalisationValueDefaultValue [int][7]:**
  Initialisierungswert für den Zufallsgenerator
- **roundOfActivation [int][0]:**
  Die Runde in dem ein Peer aktiv wird, d.h. die Runde ab der ein Peer beginnt Anfragen zu stellen und entgegen zu nehmen.
- **dummyPeer [boolean][false]:**
  Falls dieser Wert auf true gesetzt wird, werden alle von diesem Peer durchgeführten Transaktionen nur als Scheintransaktionen gewertet. Diese Eigenschaft wird für die Sybils bei der Sybil-Attacke 5.2.4 benötigt.

4.3.1.3. Die Strategy-Klassen

Der Simulator muss in der Lage sein, verschiedene Entscheidungsmuster zu simulieren. Dies können z.B. verschiedene Reputationsberechnungsalgorithmen oder unterschiedliche Tokenverteilungsstrategien sein, die individuell für jeden Peer im Netzwerk simuliert werden müssen. Diese Strategien lassen sich gewissen Strategietypen zuordnen, wie z.B. den Tokenverteilungsstrategien oder den Vertrauenswürdigkeitsprüfungsverfahren. Um den flexiblen Austausch der verschiedenen Algorithmen während der Laufzeit gewährleisten zu können, wird das Strategy Pattern [20] angewendet. Jede Strategy-Klasse implementiert ein bestimmtes Entscheidungsmuster und erbt von einer abstrakten Strategy-Klasse, die einen Strategietyp repräsentiert. Solch eine abstrakte Strategy-Klasse ist z.B. die *ATokenDistributor* Klasse, von der alle Tokenverteilungsstrategien erben müssen.
Diese abstrakten Klassen wiederum implementieren das Interface IStrategy. Dieses Interface dient lediglich dazu, zu erkennen, das es sich bei der entsprechenden Klasse um eine Strategy-Klasse handelt.
Mithilfe des Erbens von der selben abstrakten Strategy-Klasse wird gewährleistet, dass jede konkrete Strategy-Klasse über die gleiche Signatur verfügt, auf die der Simulator

zugreifen kann, ohne dass er über die eigentliche Implementierung des Algorithmuses Kenntnis besitzen muss.
Diese Signatur besteht bei dem Beispiel der *ATokenDistributor* Klasse aus der Methode *getNextReceiver()*, die als Rückgabewert denjenigen Peer zurück gibt, der als nächstes einen Token erhalten soll. Die Abbildung 4.3 veranschaulicht dieses Beispiel.

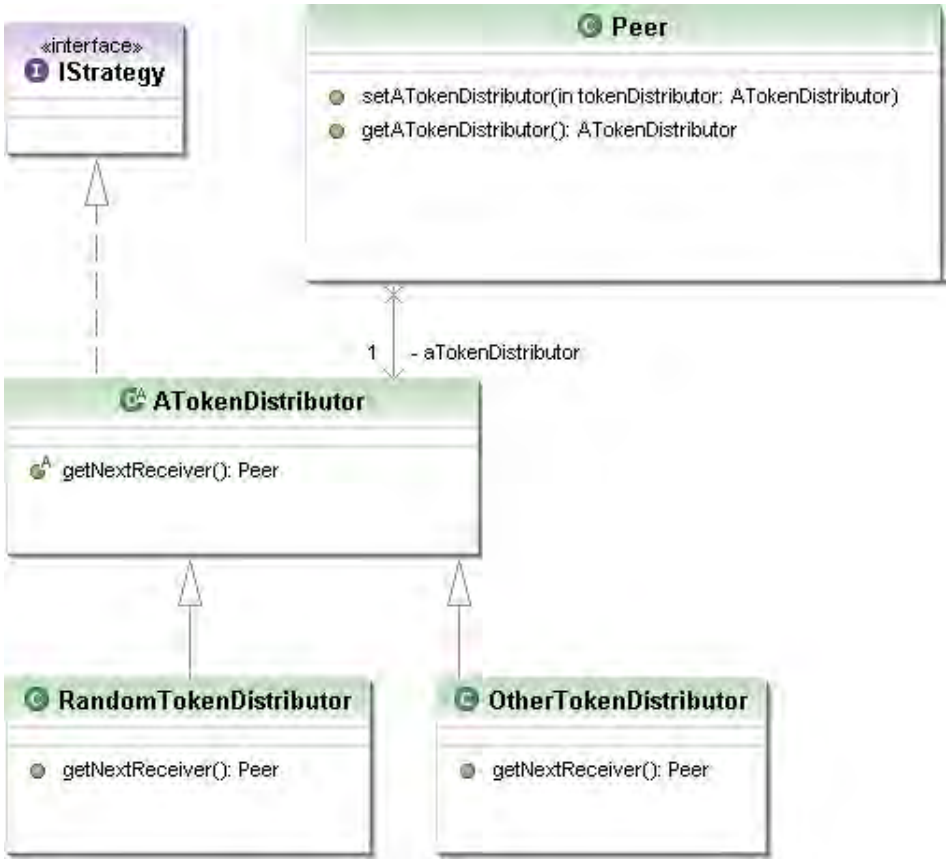

Abbildung 4.3.: Strategy Pattern am Beispiel des Token Distributors

Die Verwendung des Strategy-Patterns ermöglicht den Austausch einzelner Strategien, jedoch müssen diese zuvor vom Simulator instanziert werden. Diese Instanzierung erfordert einen speziellen Mechanismus, der in der Lage sein muss, auch nachträglich erstellte Strategien zu laden, ohne dass dabei Änderung im vorhanden Quellcode nötig sind. Um dies zu realisieren wird das Abstract Factory Pattern [20] angewendet. Die *AStrategyFactory* Klasse instanziert die jeweilige Strategy-Klasse. Dazu muss lediglich der Paket- und Klassenname an die Getter-Funktion der Factory-Klasse übergeben werden. Der Klassenname wird aus der Konfigurationsdatei gelesen und konkrete Factory-Klassen setzen den Paketnamen. Diese konkreten Factory-Klassen, wie z.B. die *TokenDistributorFactory* erben von der Abstrakten Factory-Klasse und adaptieren das Laden einer allgemeinen Strategie auf ihren konkreten Strategietyp, wie z.B. der *ATokenDistributor* Strategy-Klasse. In Abbildung 4.4 wird dieser Mechanismus illustriert.
Soll nun z.B. eine neue TokenDistributor-Strategy-Klasse implementiert werden so reicht es aus, diese im Paket *control.strategy.tokenDistributor* zu erstellen, und sie von der Klasse *ATokenDistributor* erben zu lassen. Sie kann danach geladen werden, indem in der Konfigurationsdatei ihr Klassenname eingetragen wird. Eine zusätzliche Änderung im Quellcode des Simulators ist nicht erforderlich.

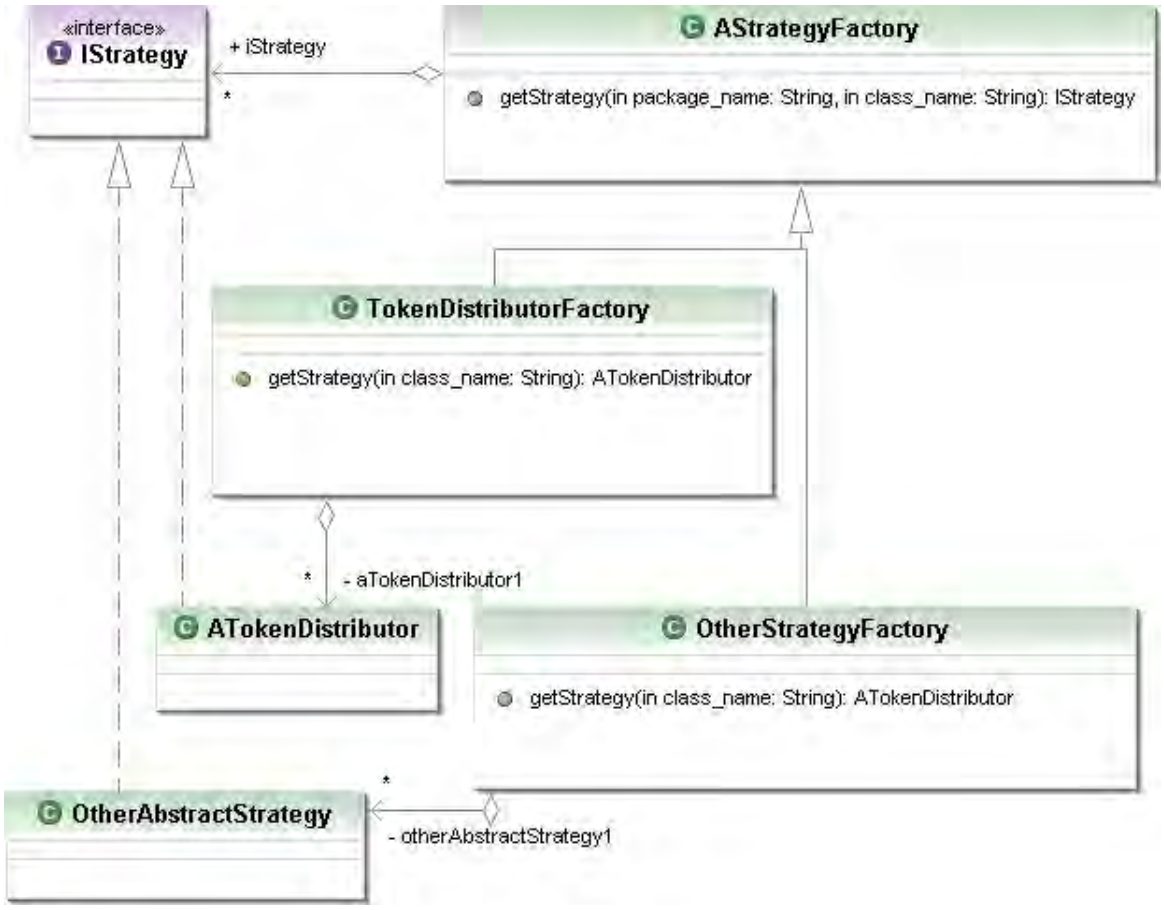

Abbildung 4.4.: Abstract Factory Pattern am Beispiel der Token Distributors Factory

In der folgenden Liste werden die verschiedene Strategietypen mit ihren konkreten Implementierungen vorgestellt. Es werden nur die Implementierungen aufgelistet, die für die Simulation von Bedeutung sind. Auf das Aufführen weitere Klassen, die z.B. für das Testen benötigt wurden, wird verzichtet. Die mit "*" markierten Strategien, sind die Standardstrategien, die verwendet werden, falls keine Strategie definiert wird.

- **clientSelector:** Wählt einen Consumer aus der Menge der Peers aus, die sich in der Warteschlange des Providers befinden.
  Verwendete Implementierungen:
  - *DefaultClientSelector*:* Implementiert die Standardreputationsberechnung 3.2 deren Teilschritte austauschbar sind und als Attributwert angegeben werden können.
  - *RandomClientSelector:* Wählt zufällig einen Consumer aus.
  - *ClientSelectorRankedByGlobalUploadDownloadDifference:* Wählt den Consumer mit der größten Differenz von Up- und Downloads aus. Dies entspricht einem allwissenden Betrachter und dient als Referenzstrategie.

  Zusätzlich können folgende Attribute gesetzt werden:
  - TrustabilityCalculator: Als Attributwert kann hier der Klassenname einer Strategie der Vertrauenswürdigkeitsprüfung 3.2.3 angegeben werden.
  - PlausibilityCalculator: Als Attributwert kann hier der Klassenname einer Strategie der Plausibilitätsprüfung 3.2.2 angegeben werden.

- **providerMarketGenerator:** Bestimmt die Menge der potentiellen Provider, aus der sich der Consumer einen Provider auswählen kann.
  Verwendete Implementierungen:
  - *ClusterOrientatedAndRangeWeightedProviderMarketGenerator*:* Berücksichtigt die Clusterverteilung und die *Range* der Peers bei der Bestimmung der Menge der potentiellen Provider.

- **providerReputationCalculator:** Berechnet die Reputation für jeden potentiellen Provider. **Verwendete Implementierungen:**
  - *DefaultProviderSelector*:* Implementiert die Standardreputationsberechnung 3.2 deren Teilstrategien über die Attributwert definiert werden können.

  Zusätzlich können folgende Attribute gesetzt werden:
  - TrustabilityCalculator: Als Attributwert kann hier der Klassenname einer Strategie der Vertrauenswürdigkeitsprüfung 3.2.3 angegeben werden.
  - PlausibilityCalculator: Als Attributwert kann hier der Klassenname einer Strategie der Plausibilitätsprüfung 3.2.2 angegeben werden.

- **providerSelector:** Wählt einen Provider anhand dessen Reputation aus.
  Verwendete Implementierungen:
  - *BestReputationSelector:* Wählt den Provider mit der höchsten Reputation aus (siehe 3.3.1).

  Zusätzlich können folgende Attribute gesetzt werden:

  - *ProbabilisticProviderSelector*:* Wählt einen Provider zufällig, gewichtet nach deren Reputation aus (siehe 3.3.2).

- **similarityMetric:** Definiert die zu verwendenden Ähnlichkeitsmetrik für die Plausibilitätsprüfung.
  Verwendete Implementierungen:
  - *DefaultMetric*:* Entspricht der in 3.2.2.4 vorgestellten Metrik.

- **tokenDistributor:** Wählt die Peers aus, die einen Token erhalten sollen.
  Verwendete Implementierungen:
  - *NeighborhoodTokenDistributor:* Verteilt die Token anhand der in 3.6.1.1 vorgestellten Reputationsberechnung.
  - *SimilarityTokenDistributor*:* Verteilt die Token anhand der in 3.6.1.2 vorgestellten Reputationsberechnung.
  - *RandomTokenDistributor:* Verteilt die Token zufällig im Netzwerk.
  - *EmptyTokenDistributor:* Verteilt keine Token im Netzwerk.

  Zusätzlich können folgende Attribute definiert werden:
  - *AgeWeighted:* Wenn dieser Wert auf *true** gesetzt ist, wird das Ergebnis zusätzlich mit dem Tokenversionsgewichtungsvektor multipliziert.
  - *Selfish:* Wenn dieser Wert auf *true** gesetzt ist, wird zusätzlich der Faktor durch die eigennützige Tokenverteilung mit eingerechnet.

- **tokenValueCalculator:** Implementiert die Tokenbewertungsfunktion.
  Verwendete Implementierungen:
  - *DefaultTokenValueCalculator*:* Bewertet die Token anhand der in 3.4 vorgestellten Tokenbewertungsfunktion.

- **transactionValidator:** Wertet aus, ob eine Transaktion erfolgreich war.
  Verwendete Implementierungen:
  - *DefaultTransactionValidator*:* Berechnet anhand der Zuverlässigkeit des Providers, ob eine Transaktion erfolgreich war.

4.3.2. Die Netzwerkgenerierung

Für die Generierung des Netzwerkmodells ist die Klasse *control.NetworkGenerator* zuständig. Die Aufgabe dieses Generators ist es, die spezifischen Eigenschaften der Peers und Netzwerkstruktur zu definieren. Nach welcher Methodik er dabei vorgeht, wird nun im einzelnen erläutert.

- **Up und Download Wahrscheinlichkeit:** Orientiert an den Messergebnissen in Abbildung 4.5 aus [2] wurde für den Simulator die in Abbildung 4.6 dargestellte Verteilung der Up- und Downstream verwendet.
  Dieser Verteilung kann mittels des Parameters *averageDownloadsPerPeer* entlang der x-Achse verschoben werden. In einem konstanten Verhältnis zu diesem Wert

steht der Parameter *averageUploadsPerPeer*. Dieses Verhältnis wird durch den Konfigurationsparameter *downUpStreamRatioPerPeer* bestimmt.

$$averageUploadsPerPeer = averageDownloadsPerPeer * downUpStreamRatioPerPeer \tag{4.1}$$

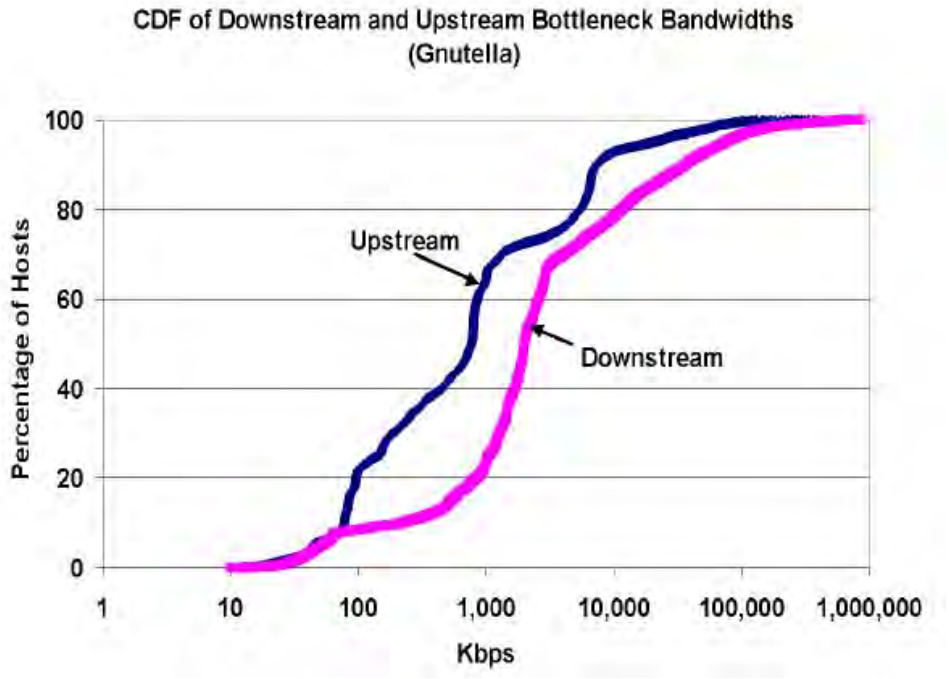

Abbildung 4.5.: Verteilung der Up- und Downloads in einem Gnutella-Netzwerk [2]

- **Range:** Dieser Parameter wird mittels der Pareto-Verteilung auf dem Intervall zwischen 1 und dem angegebenen *maxRange* Parameter in der XML-Konfigurationsdatei bestimmt. Um nun eine Pareto-verteilte Zufallszahl zu generieren wird die auf Seite 9 beschriebene Funktion 2.1 verwendet. Als τ Parameter wird der Wert 0.1 und für c der Wert 1 definiert. Falls die generierte Zufallszahl größer als *maxRange* sein sollte, wird diese Zufallszahl verworfen und es wird eine neue generiert.

- **Peer-Cluster-Beziehung:** Es wird angenommen, dass die Anzahl der Cluster, in denen ein Peer aktiv ist, ebenfalls Pareto-verteilt ist. Diese Annahme beruht auf den in 2.2.1.1 vorgestellten Zahlen. Es wird die auf Seite 9 beschriebene Funktion 2.1 zur Berechnung der Anzahl der aktiven Cluster gewählt. Der $ClusterTau$ Parameter dient als τ und $\sqrt{numberOfClusters}$ als c-Parameter. Um die generierten Zufallszahlen noch auf das Intervall $[1 \ldots numberOfClusters]$ abzubilden, wird von dem Ergebnis der Pareto-Funktion $\sqrt{numberOfClusters} - 1$ subtrahiert und falls das daraus resultierende Ergebnis größer als *numberOfClusters* sein sollte, wird eine neue Zufallszahl generiert.

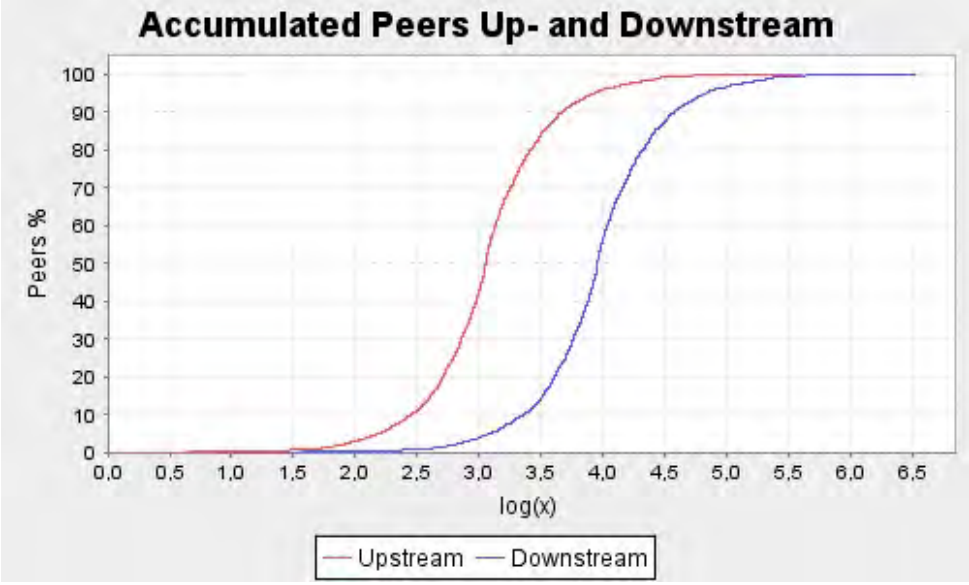

Abbildung 4.6.: Verwendete Up- und Downloadratenverteilung

4.3.3. Die Simulationsrunde

Die Simulation erfolgt rundenbasierend. Jede Runde unterteilt sich in einzelne Phasen. In Abbildung 4.7 wird der Rundenablauf skizziert. Für die Steuerung des Ablaufs einer Runde ist die Klasse *control.RoundController* zuständig und ist somit das Kernstück des Simulators.

4.3.3.1. Startup Phase

Zu Beginn einer Runde wird überprüft, welche Peers über die Konfiguration aktiviert werden, falls sich die Menge der aktiven Peers ändert, werden alle abhängigen Caches gelöscht.

4.3.3.2. Queueup Phase

Die eigentliche Rundensimulation beginnt mit der *Queueup-Phase* sie unterteilt sich in folgende vier Einzelabschnitte.

- **Consumer Selection:** Für jeden Peer wird anhand seiner Downloadwahrscheinlichkeit ermittelt, ob er in dieser Runde eine Anfrage an einen Provider stellt.
- **Provider Market Selection:** Danach wird für jeden Peer, der eine Anfrage stellt, die Menge seiner potentiellen Provider gesucht. Ein Peer ist ein potentieller Provider, sobald er im gleichen Cluster agiert. Von dieser Menge aller potentiellen Provider wird eine zufällige Teilmenge gebildet. Die über das Datenangebot gewichtet wird. Dies soll die Eigenschaft simulieren, dass nicht jeder Peer über die gleichen Daten verfügt auch wenn er im gleichen Cluster liegt. Ein Peer, der über ein großes Datenangebot verfügt, besitzt eine höhere Wahrscheinlichkeit in diese Menge aufgenommen zu werden.
- **Provider Reputation Calculation:** Für jeden Peer aus dieser Teilmenge wird die Reputation berechnet.

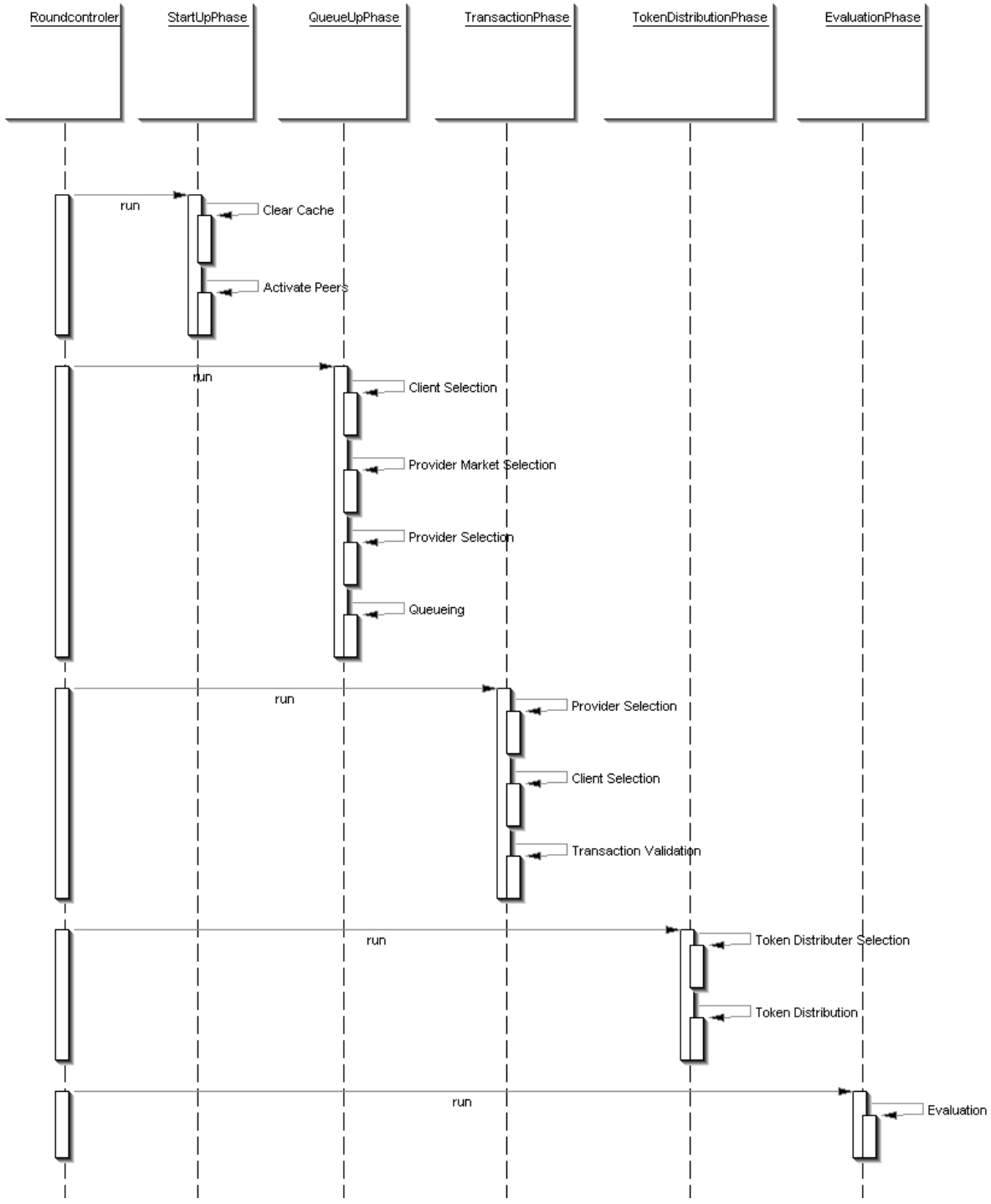

Abbildung 4.7.: Übersicht über die Aktionen innerhalb einer Runde

- **Provider Selection:** Der Consumer wählt anhand seiner Providerauswahlstrategie einen Provider aus.
- **Queueing:** Zum Abschluss der Queueup Phase findet das eigentliche eintragen in die Warteliste des Providers statt.

4.3.3.3. Transaktionsphase

Die Transaktionsphase unterteilt sich wiederum in drei Teilabschnitte.

- **Provider Selection:** Zuerst wird für jeden Peer anhand seiner Uploadwahrscheinlichkeit ermittelt, ob er in dieser Runde eine Transaktion als Provider abwickelt.
- **Consumer Selection:** Falls ein Peer in dieser Runde als Provider agiert, wählt er mittels seiner Consumerauswahlstrategie einen Peer aus seiner Warteschlange aus mit dem er in dieser Runde eine Transaktion durchführt.
- **Transaction Validation:** Anhand der Zuverlässigkeit des Providers wird ausgewertet, ob die Transaktion erfolgreich war oder nicht.

4.3.3.4. Token Distribution Phase

Dieser Phase unterteilt sich in folgende drei Teilabschnitte.

- **Token Distributor Selection:** Aus der Menge der Transaktionspartner die in dieser Runde aktiv waren, wird ermittelt, welche Peers einen Nutzen davon ziehen würden, wenn sie, den in dieser Runde erzeugten Token an Dritte weiter schicken würden. Dies wird genau ein Peer pro Transaktion sein.[2]
- **Token Receiver Selection:** Anhand seiner Tokenverteilungsstrategie wählt ein Peer die Tokenempfänger aus und schickt ihnen den Token.
- **Token Distribution:** Im letztem Teilschritt verarbeitet der Tokenempfänger den erhaltenen Token.

4.3.3.5. Evaluation Phase

In der abschließenden Phase werden die Ergebnisse der aktuellen Runde evaluiert. Abschnitt 4.4 geht auf die verschiedenen Auswertungsverfahren ein.

4.3.4. Der Zufallsgenerator

Neben der elementaren Aufgabe der Generierung von Zufallszahlen, wird an den Generator die Anforderung gestellt, dass er für jeden Durchlauf eines Szenarios immer das gleiche Netzwerk generiert. Damit ein Vergleich verschiedener Strategien stattfinden kann, ohne dass die Messergebnisse durch unterschiedliche Strukturen des Netzwerks beeinflusst werden.
Diese Eigenschaften gewährleistet der *control.Randomizer*. Dieser Randomizer ist eine statische Klasse, die die Klasse *java.util.Random* vor Beginn jedes Szenariodurchlaufs instanziert und alle Zufallszahlen mithilfe dieser Instanz generiert.

[2] Siehe 3.5

4.3.5. Starten des Simulators

Zum Starten des Simulators wird die Klasse *control.Starter* mit dem Java Interpreter ab Version 1.6 ausgeführt. Als Parameter muss der Name der XML-Konfigurationsdatei angegeben werden, die sich im Verzeichnis *./scenarios* befinden muss.
Zuvor muss sichergestellt werden, dass alle Bibliotheken im Verzeichnis *./libs* im Classpath gesetzt sind.
Für den Java Interpreter kann als Option *-ea* angegeben werden, um die Simulation mit Assertion-Tests durchzuführen. Zusätzlich ist darauf zu achten, dass der dem Interpreter zur Verfügung gestellte Arbeitsspeicher mit der Option *-Xmx <size>* erhöht wird. Folgendes Beispiel soll eine Vorstellung des benötigten Speichers vermitteln. Eine Simulation mit 3000 Peers, die einen durchschnittliche Downloadrate von 500 Downloadanfragen pro Simulation besitzen und 20 Token pro Transaktion verschicken, benötigt ca. 700 MB Speicher.
Das Beispielkommando 4.2 simuliert, die in der *./scenarios/test.xml* Datei definierten Szenarien aus.

```
1 java -ea -XMX1000m control.Starter test.xml
```

Listing 4.2: Starten des Simulators

4.4. Die Auswertung

In diesem Abschnitt soll lediglich auf die formale Berechnung und Bedeutung der einzelnen Auswertungen eingegangen werden und nicht auf deren programmatischen Umsetzung.

4.4.1. Versuchsreihen

Eine Simulation ist abhängig von vielen zufällig generierten Parametern, deswegen ist die Aussagekraft einer einzelnen Simulation sehr gering. Es bieten sich mehrere Möglichkeiten an, den Einfluss der Zufallsparameter zu verringern. Eine Möglichkeit besteht darin mehrere Simulationen mit gleichen Parametern durchzuführen. Dabei wird lediglich der Initialisator für die Zufallszahlen des Simulators variiert und über die Messergebnisse gemittelt. Auf diese Variante wird bewusst verzichtet, da sie mehrere Nachteile besitzt. Einerseits gehen durch die Zusammenfassung verschiedener Simulationen Detailaussagen verloren und es lässt sich keine Aussage über einzelne Peers machen. Andererseits können die Messergebnisse der einzelnen Simulationen stark vom Mittelwert variieren, dies würde unter Umständen unentdeckt bleiben. Dabei ist diese Erkenntnis besonders wichtig, um die Verlässlichkeit eines Reputationssystem beurteilen zu können.
Bei einer anderen Möglichkeit, die in dieser Arbeit Verwendung findet, werden ebenfalls Vergleichssimulationen, bei denen nur der Zufallswert verändert wird, stichprobenartig durchgeführt. Es wird jedoch dabei auf die Bildung des Mittelwertes verzichtet. Lediglich eine Simulation wird aus dieser Versuchsreihe detailliert betrachtet, falls die Ergebnisse der verschiedenen Simulationen relativ dicht beisammen liegen. Weichen hingegen die Ergebnisse der einzelnen Simulationen stark von einander ab, so ist das verwendete Reputationssystem unzuverlässig und eine genaue Betrachtung überflüssig, außer man will den Ursachen der Unzuverlässigkeit auf die Spur kommen.
Des Weiteren ist es kein konkretes Ziel der Simulation äußerst genaue Messergebnisse zu

liefern, sondern es interessieren mehr die besonderen Charakteristiken einzelner Reputationssysteme im Zusammenhang von der Netzwerkstruktur. Daher ist die Bildung von Mittelwerten überflüssig. Diese Versuchsreihen werden im Anhang A.1 aufgeführt.
Eine weitere Möglichkeit die Aussagekraft einer Simulation zu erhöhen, ist es, die Netzwerkgröße zu erhöhen, da so die Auswirkungen der Entscheidungen der einzelnen Peers verringert wird. Jedoch beeinflusst die Netzwerkgröße zugleich die Laufzeit der Simulation und lange Laufzeiten verhindern, dass weitere Vergleichssimulationen durchgeführt werden können. Damit steht die Netzwerkgröße in direkter Konkurrenz zur Anzahl der Vergleichssimulationen und es muss entsprechend zwischen diesen beiden Parametern abgewogen werden.

4.4.2. Metriken

In diesem Abschnitt werden diverse Metriken vorgestellt, mit deren Hilfe die einzelnen Simulationen ausgewertet werden.

4.4.2.1. Abweichung von Uploads und Downloads

Bei dieser Bewertung wird die durchschnittliche Abweichung zwischen Up- und Downloads pro Transaktion jedes einzelnen Peers berechnet. Sie wird als $udd_{(x)}$[3] bezeichnet und gibt Aufschluss, wie gerecht die Verteilung der Leistungen zwischen den Peers ist. Für die Formalisierung werden die beiden Funktionen $uploads_{(i,x)}$ und $downloads_{(i,x)}$ benötigt. Sie geben an, wie viel Uploads bzw. Downloads Peer i bis zur Runde x insgesamt getätigt hat.
Ein $udd = 0$ würde bedeuten, dass die Anzahl der Uploads und Downloads bei allen Peers ausgeglichen ist. Ein Wert von 1 würde bedeuteten, dass jeder Peer entweder nur Downloads oder nur Uploads getätigt hat. Ein Wert von $0,5$, dass die Abweichungen der Uploads von den Downloads 50% aller Transaktionen ausmacht.

$$udd_{(x)} = \frac{\sum_i |uploads_{(i,x)} - downloads_{(i,x)}|}{\sum_i uploads_{(i,x)} + downloads_{(i,x)}} \tag{4.2}$$

4.4.2.2. Anfragen pro Download

Diese Funktion gibt an wie viele Anfragen durchschnittliche pro Download gestellt wurden. Die Hilfsfunktion $requests_{(i,x)}$ beschreibt, wie viele Anfragen von Peer i bis zur Runde x gestellt wurden.

$$rpd_{(x)} = \frac{\sum_i \frac{requests_{(i,x)}}{downloads_{(i,x)}}}{n} \tag{4.3}$$

4.4.2.3. Uploads pro Downloads

Diese Funktion gibt an, wie viele Uploads durchschnittlich pro Download innerhalb einer Peergruppe getätigt wurden. Fällt dieser Wert für eine Peergruppe unter 1, so erhält sie mehr Leistungen als sie selbst liefert. Umgekehrt liefert sie mehr Leistungen als sie erhält,

[3] Upload Download Difference

falls dieser Wert über 1 steigt.

$$upd_{(x)} = \frac{\sum_i \frac{uploads_{(i,x)}}{downloads_{(i,x)}}}{n} \tag{4.4}$$

4.4.2.4. Prozentualer Fehleranteil

Durch diese Funktion lässt sich darstellen, wie groß der Anteil fehlerhafter Transaktion innerhalb eines Peer-Sets ist. Hier werden als Hilfsfunktionen $transf_{(i,x)}$ und $trans_{(i,x)}$ benötigt. Die erste gibt an, wie viele fehlerhafte Transaktionen Peer i bis zum Zeitpunkt x getätigt hat und die zweite, die Anzahl aller Transaktionen die Peer i bis zum Zeitpunkt x durchgeführt hat.

$$ft_{(x)} = \frac{\sum_i transf_{(i,x)}}{\sum_i trans_{(i,x)}} \tag{4.5}$$

4.5. Qualitätssicherung

An dieser Stelle sollen verschiedene Verfahren vorgestellt werden, die verwendet wurden, um möglichst verlässliche Messergebnisse zu garantieren. Es wird zwischen automatischen Tests, manuellen Tests und der Analyse der Messergebnisse unterschieden werden.

4.5.1. Automatische Tests

Zu den automatischen Tests gehören die Assertions und die Unittests. Sie testen die Programmcode auf unterster Ebene.
Die Assertions überprüfen zur Laufzeit, ob Teilergebnisse innerhalb des Wertebereichs liegen. So stellt z.B. eine Assertion sicher, dass alle berechneten Reputationswerte nicht kleiner Null sind. Da sie oftmals die Laufzeit des Simulators verlängern, sollten sie bei größeren Simulationen nicht aktiviert werden.[4]
Die Unittests überprüfen einzelne Methoden. Hierfür werden oftmals kleine Simulationen generiert und anhand kleiner Beispiele die einzelnen Methoden auf Korrektheit überprüft. Zum erzeugen der Unittests wird Junit verwendet. Die Tests befinden sich im Package *tests*. Abbildung 4.8 zeigt das erfolgreiche Ausführen der Unittests innerhalb der Eclipse Umgebung.

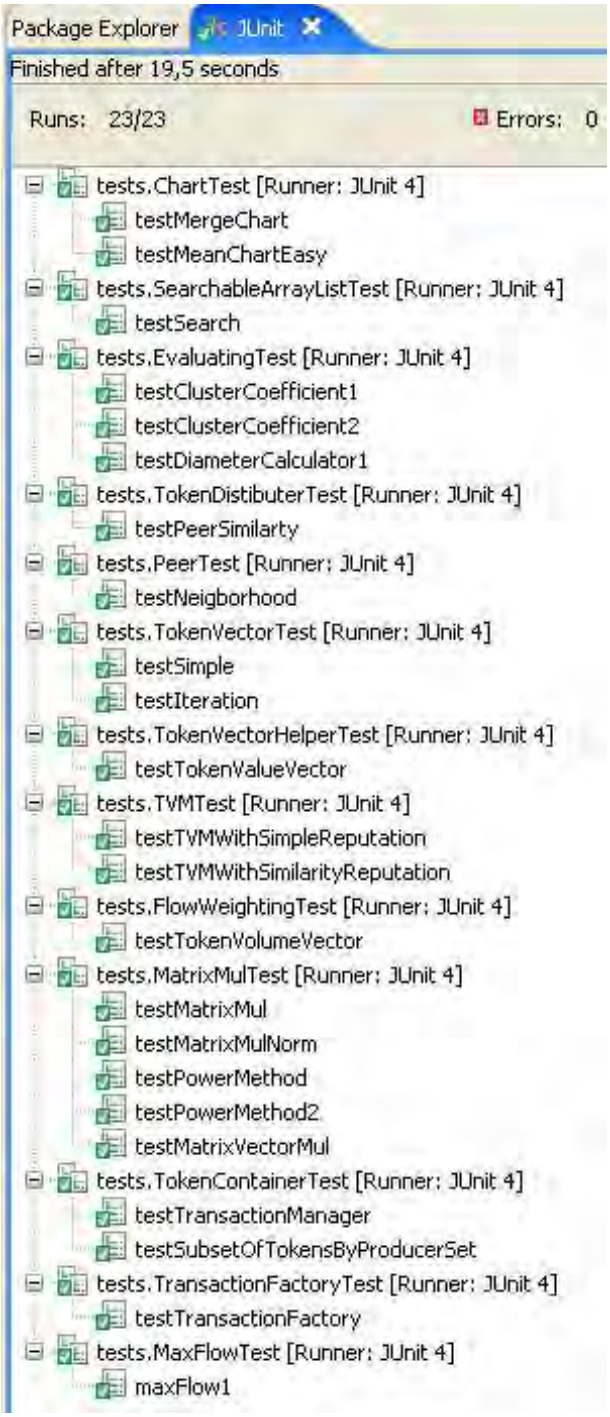

Abbildung 4.8.: Unittests

4.5.2. Manuellen Tests

Die manuellen Tests wurden während der Programmentwicklung des Simulators durchgeführt. Sie bestehen einmal aus dem Debuggen des Prgrammcodes und der Verwendung von Profiling Tools.
Aufgrund der großen Anzahl von erzeugten Objekten und Variablen, war die Verwendung des Debuggers nur eingeschränkt

[4] siehe Abschnitt 4.3.5

möglich. Er wurde meist nur auf kleinere Simulationen mit wenigen Peers angewendet. Hingegen wurde das Profiling Tool nicht nur zur Performance-Optimierung eingesetzt, sondern auch zur Überprüfung der Anzahl erzeugter Instanzen bzw. der Anzahl Aufrufe einzelner Methoden und deren Laufzeit. Gab es z.B. mehr Instanzen der Transaction-Klasse als von der Token-Klasse, so war dies ein deutliches Indiz für einen Fehler in der Implementierung.

4.5.3. Analyse der Messergebnisse

Als letzte Maßnahme der Qualitätssicherung dient die Analyse der Messergebnisse. Hierzu werden zum einen Testsimulationen erzeugt und zum anderen die Messergebnisse verschiedener Simulationen miteinander verglichen und auf Stimmigkeit überprüft.
Bei den Testsimulationen werden Simulationen durchgeführt, deren Ergebnis im Vorfeld bekannt sind. Ein Beispiel für eine Testsimulation ist der Test der Tokenverteilung. Hier werden verschiedene Tokenverteilungsstrategien miteinander verglichen, bei einer maximalen Anzahl verteilter Token. Dabei müssen für jedes Szenario die selben Messergebnisse erzielt werden, da jeder Peer über alle Token verfügt, unabhängig davon, welche Tokenverteilungsstrategie verwendet wurde. Eine andere Möglichkeit ist es, bei der Simulation verschiedene Strategy-Klassen zu verwenden, deren Logik identisch sein soll, sie sich aber in ihrer Implementierung unterscheiden. Die Konfigurationsdateien für diese Testszenarien wurden im Verzeichnis *./scenarios/testscenarios* abgelegt. Einen Auszug aus den Messergebnissen der Testszenarien befindet sich im Anhang A.2 .
Das Testverfahren des Vergleichs verschiedener Simulationsergebnisse verschwimmt mit der eigentlichen Auswertung der Simulation. Treten bei der Simulation unerwartete Ergebnisse auf, so kann dies an einer fehlerhaften Implementierung, an einem Konzeptionsfehler im Algorithmus liegen oder ein wirkliches Messergebnis sein, welches als besonders wertvoll gilt, da es zu neuen Erkenntnisse verhilft. Um die Ursache herauszufinden, ist es notwendig weitere Simulationen durchzuführen, mit deren Hilfe eine Eingrenzung der Ursache erzielt werden soll.

5. Evaluation

Die verschiedenen vorgestellten Tokenverteilungsstrategien und Reputationsberechnungsalgorithmen sollen in diesem Kapitel unter verschiedenen Gesichtspunkten analysiert werden. Es soll überprüft werden, wie die einzelnen Strategien einen fairen Austausch von Daten gewährleisten, wie sie in Wechselwirkung miteinander stehen und wie robust sie gegenüber Angriffen und Manipulationsversuchen sind. Des Weiteren soll untersucht werden, inwiefern sich die einzelnen Strategien auf die Clusterstruktur auswirken.
Leider kann Aufgrund der Vielzahl von Netzwerkparametern und Strategien nur ein kleiner Teil der möglichen Szenarien in diesem Kapitel besprochen werden.

5.1. Simulation unter Idealbedingungen

In dieser Versuchsreihe sollen verschiedene Tokenverteilungsverfahren und Vertrauenswürdigkeitsprüfungsverfahren unter Idealbedingungen untersucht werden.
Unter Idealbedingungen wird verstanden, dass alle Transaktionen erfolgreich ausgeführt werden, d.h. beide Transaktionspartner werden zufrieden gestellt und alle Peers geben eine objektive Bewertung ab. Deshalb fallen alle Bewertungen positiv aus. Somit muss das Reputationssystem in diesem Szenario keinem Angriff standhalten, sondern lediglich für eine gerechte Leistungsverteilung sorgen.
Zunächst soll mit der Simulation in 5.1.1 ein allgemeiner Überblick über die Auswirkung der verschiedenen Strategien vermittelt werden.
In 5.1.2 wird die Simulation mit unterschiedlichen Clusterstrukturen wiederholt und untersucht, inwiefern sich dies auf die Messergebnisse auswirkt.
In dem Experiment 5.1.3 sollen die Auswirkungen aufgezeigt werden, die durch den Versand einer unterschiedlichen Anzahl Token entsteht.
In 5.1.4 wird versucht die beste Tokenverteilungsstrategie aus den hier vorgestellten Strategien zu ermitteln und die Frage zu beantwortet, ob die allgemein fairste Strategie auch für den einzelnen die besten Ergebnisse liefert.
Schließlich wird im letzten Abschnitt 5.1.5 kurz auf die Auswirkungen der Optimierungsverfahren der Tokenverteilungsstrategien eingegangen.

5.1.1. Die Ausgangssimulation

Diese für den weiteren Verlauf der Versuchsreihe als Ausgangssimulation dienende Simulation besteht aus acht verschieden einzelnen Szenarien. Diese unterscheiden sich in den verwendeten Tokenverteilungsstrategien und den Vertrauenswürdigkeitsprüfungsverfahren. Für alle Szenarien wird die Plausibilität mittels der Ähnlichkeitsbewertung bestimmt.
Bei allen Tokenverteilungsstrategien werden beide vorgestellten Optimierungsverfahren

verwendet.[1] Es wird jeweils ein Netzwerk mit 3000 Peers in 30 Clustern über einen Zeitraum von 20000 Runden simuliert. Die Anzahl der verteilten Token pro Transaktion beträgt 25 Token. Die Konfigurationsdaten lassen sich A.5.1.1 entnehmen.

Die Szenarien:

1. SimilarityFlow:
   - tokenDistributor: SimilarityTokenDistributor
   - TrustabilityCalculator: FlowTrustability
2.
3. NeighborhoodFlow:
   - tokenDistributor: NeighborhoodTokenDistributor
   - TrustabilityCalculator: FlowTrustability
4. RandomSimple:
   - tokenDistributor: RandomTokenDistributor
   - TrustabilityCalculator: SimpleTrustability
5. SimilaritySimple;
   - tokenDistributor: SimilarityTokenDistributor
   - TrustabilityCalculator: SimpleTrustability
6. RandomFlow:
   - tokenDistributor: RandomTokenDistributor
   - TrustabilityCalculator: FlowTrustability
7. EmptySimple:
   - tokenDistributor: EmptyTokenDistributor
   - TrustabilityCalculator: SimpleTrustability
8.
9. NeighborhoodSimple:
   - tokenDistributor: NeighborhoodTokenDistributor
   - TrustabilityCalculator: SimpleTrustability
10. Total:
    - tokenDistributor: EmptyTokenDistributor
    - clientSelector: ClientSelectorRankedByGlobalUploadDownloadDifference

[1] Dies gilt im Besonderen für die zufällige Tokenverteilung, die dadurch die Token nur bedingt zufällig im Netzwerk verteilt.

5.1.1.1. Ergebnisse

Die Grafik 5.1 zeigt die durchschnittliche Differenz zwischen Uploads und Downloads pro Transaktion und wird mit der in 4.4.2.1 vorgestellten Metrik berechnet.
Die beiden Graphen 5.2 und 5.3 zeigen, wie erfolgreich Peers sind, die in mehreren Clustern aktiv sind, im Vergleich zu Peers, die nur in einem Cluster aktiv sind. Die Werteskala entspricht der durchschnittlichen prozentualen Abweichung von der Anzahl Uploads zu der Anzahl Downloads, die ein Peer getätigt hat.
Die Anzahl aller getätigter Transaktionen beträgt durchschnittlich ca. 233000 pro Szenario. Dabei beträgt die Abweichung der einzelnen Szenarien maximal $0,5\%$ und ist somit relativ unerheblich für die weitere Betrachtung.

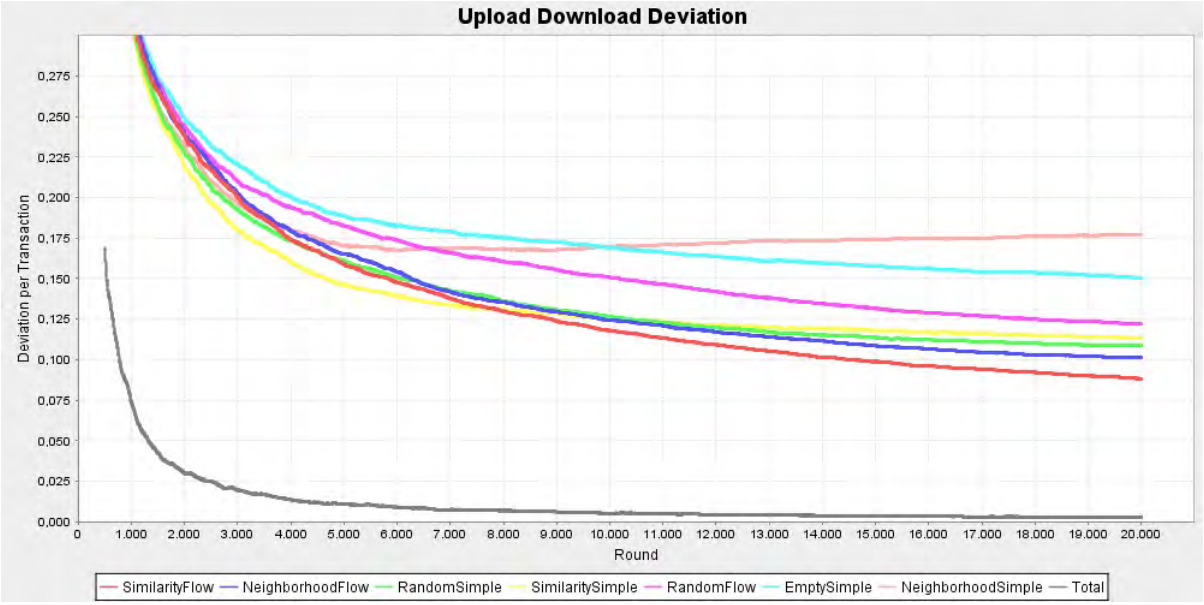

Abbildung 5.1.: Abweichung von Uploads und Downloads

5.1.1.2. Diskussion

Aus der Grafik 5.2 wird ersichtlich, dass die Szenarien mit einfacher Vertrauenswürdigkeitsprüfung zuerst relativ gute Ergebnisse liefern, doch dann sehr schnell schlechter abschneiden, als die Vertrauenswürdigkeitsprüfung mittels maximalen Flusses. Davon war zunächst nicht auszugehen, da die Vertrauenswürdigkeitsprüfung mittels maximalen Flusses primär dafür entwickelt wurde, das System robuster gegenüber Angriffen zu machen und nicht um eine gerechtere Leistungsverteilung zu ermöglichen.
Die anfängliche Schwäche des Vertrauenswürdigkeitsprüfungsverfahren mittels maximalen Flusses lässt sich damit erklären, dass sich ein Peer mithilfe der Token erst ein gewisses Wissen aufbauen muss, bevor es das Prüfungsverfahren effektiv anwenden kann. Im Besonderen muss ein Peer selbst bereits Leistungen geliefert und erhalten haben, um überhaupt einen Fluss berechnen zu können.

Besondere Beachtung gilt der Nachbarschaftsverteilung mit einfacher Vertrauenswürdig-

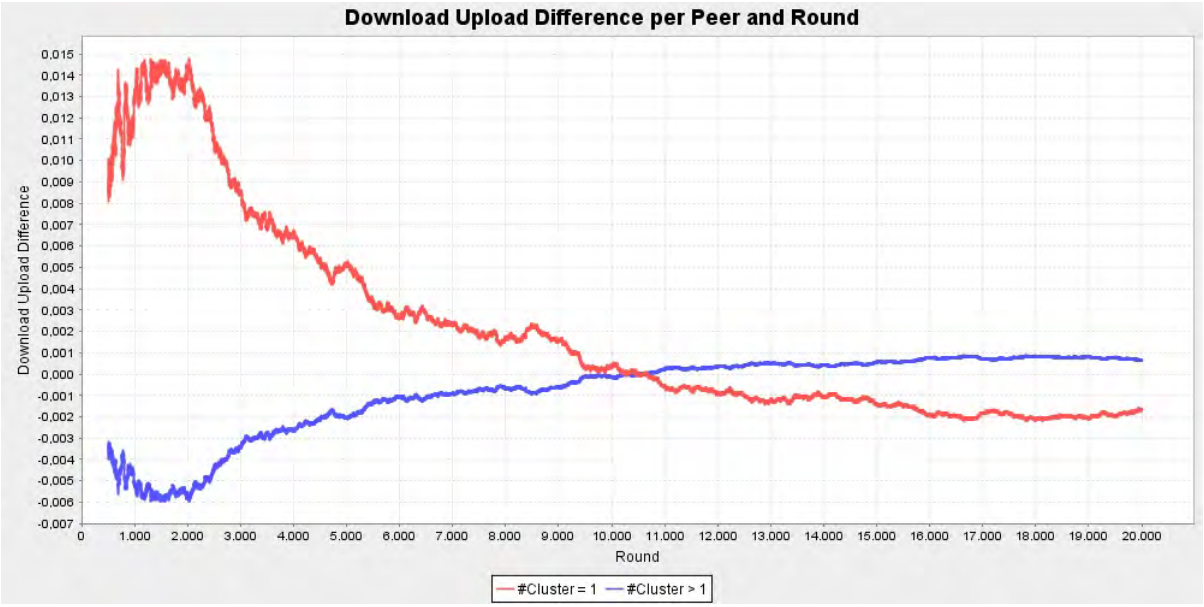

Abbildung 5.2.: Differenz von Downloads und Uploads beim Szenario SimilarityFlow

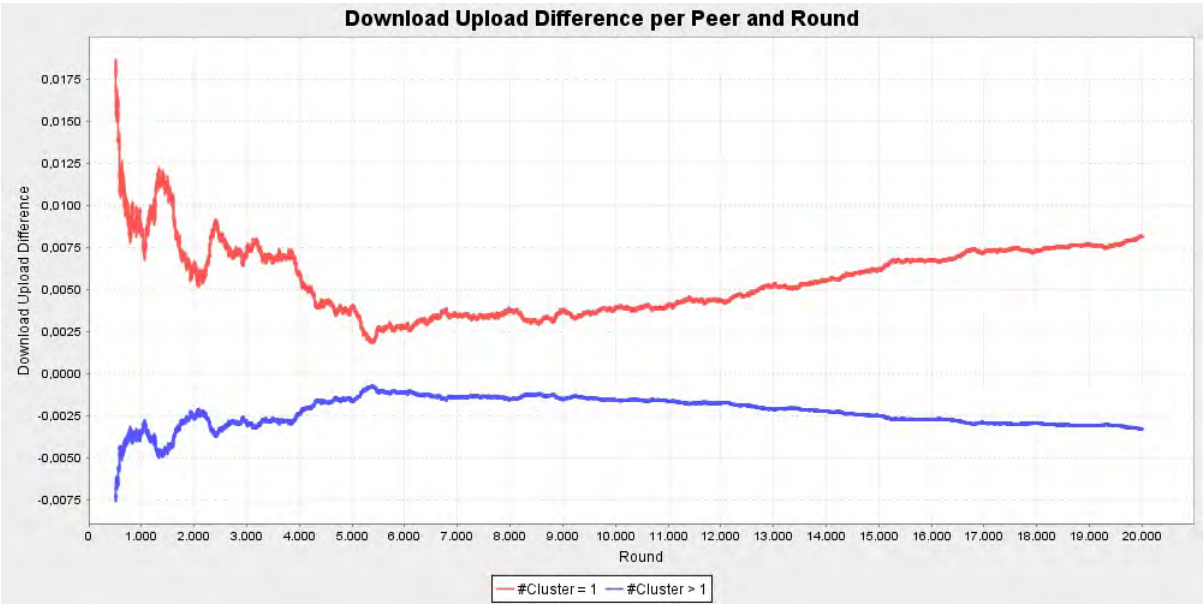

Abbildung 5.3.: Differenz von Downloads und Uploads beim Szenario NeighborhoodFlow

keitsprüfung. Sie schneidet schlechter ab, als ein Verfahren ohne Tokenverteilung und macht somit ein tokenbasiertes Reputationssystem mit solch einer Strategie überflüssig. Die Hauptursache für das Versagen dieser Strategie liegt vermutlich darin begründet, dass Peers nur die Token an Nachbarn schicken, die sie selbst positiv bewerten und zugleich den anderen Tokenerzeuger negativ belasten. Somit werden alle Tokenempfänger vor dem anderen Tokenerzeuger abgeschreckt und haben kein Interesse mit ihm in Kontakt zu treten. Dies soll an folgendem Beispiel verdeutlicht werden. Peer A liefert Peer B eine korrekte Leistung, den daraus erstellten Token schickt Peer A an Peer C, der Nachbar von Peer A ist, nicht aber von Peer B. Dadurch besitzt Peer B bei Peer C eine schlechte Reputation und kommt somit für Peer C weder als Provider noch als Client in Frage, solange genügend Alternativen vorhanden sind. Wenn nun Peer B Peer A im weiteren Verlauf der Simulation mehr Leistungen liefert als umgekehrt, so müsste sich Peer B eine bessere Reputation als Peer A erarbeiten. Jedoch erfährt dies nicht Peer C, da er kein Nachbar von Peer B ist und deshalb auch keine Token von Peer B erhält. Er erhält nur die Token von seinem Nachbarn Peer A, die für Peer A von Vorteil sind und ihn somit immer besser bewerten als Peer B. Bei der Vertrauenswürdigkeitsprüfung mittels des maximalen Flusses tritt dieses Problem nicht auf, da in diesem Fall kein Fluss von Peer C nach Peer B entsteht und somit die Bewertungen von Peer B nicht ins Gewicht fallen.

Vergleicht man die beiden Szenarien SimilarityFlow und NeighborhoodFlow, die sich lediglich in der Tokenverteilung unterscheiden, so führt die Tokenverteilung an ähnliche Peers zu einem besseren Ergebnis, als die Nachbarschaftsverteilung.
Betrachtet man bei diesen beiden Szenarien den Erfolg der Peers, die in mehr als einem Cluster aktiv sind, im Vergleich zu den Peers, die nur in einem Cluster aktiv sind, so kann man aus den Grafiken 5.2 und 5.3 entnehmen, dass die Peers, die in mehreren Clustern aktiv sind, zunächst in beiden Szenarien schlechter abschneiden. Der Grund für dieses zuerst schlechtere Abschneiden lässt sich leicht erklären. Das Umfeld der Peers, die in mehreren Clustern aktiv sind, ist deutlich größer als bei den Peers, die nur in einem Cluster aktiv sind. Deshalb ist die Anzahl ihrer potentiellen Tokenempfänger deutlich größer und somit auch die Anzahl der potentiellen Tokenempfänger, die den Token nicht erhalten.
Beim SimilarityFlow Szenario können nun im Gegensatz zum NeighborhoodFlow Szenario, die Peers, die in mehreren Clustern aktiv sind, ihren Rückstand ausgleichen. Dies ist ein Indiz dafür, dass die Tokenverteilung an ähnliche Peers die Token gezielt im Netzwerk verteilt. Die Peers benötigen zuerst eine Lernphase in denen sie die Clusterstrukturen des Netzwerks kennen lernen und je mehr Wissen sie sich angeeignet haben, desto gezielter versenden sie ihre Token im Netzwerk bzw. in dem Cluster aus dem auch ihr Transaktionspartner stammt.

5.1.2. Verhalten bei unterschiedlichen Clusterstrukturen

In dieser Simulation soll sich der Untersuchung der Auswirkung unterschiedlicher Clusterstrukturen gewidmet werden. Hierzu werden zwei weiteren Simulationen durchgeführt. Sie unterscheiden sich zur Ausgangssimulation lediglich in ihrem Cluster-τ-Parameter, der die Stärke des Content-Clustering definiert. Neben dem Cluster-τ-Wert von $2,0$, der in der Ausgangssimulation verwendet wird, wird für die anderen Simulationen ein Wert von $1,25$ und $2,75$ definiert. Dies entspricht der in der Grafik A.7 dargestellten Verteilungen. Die entsprechenden Konfigurationsdateien sind A.5.1.2 und A.5.1.3.

5.1.2.1. Ergebnisse

Die Abbildungen 5.4 und 5.5 zeigen die durchschnittliche Abweichung zwischen Uploads und Downloads.
In der Grafik 5.6 sind die durchschnittlichen Clusterkoeffizienten der Szenarien bei entsprechendem Cluster-τ angegeben. Der Clusterkoeffizient wurde in Abschnitt 2.2.1.2 näher erläutert. Er beschreibt wie stark die Nachbarn eines Peers untereinander in Beziehung stehen. Die detaillierten Verteilungen der einzelnen Clusterkoeffizienten werden im Anhang in den Graphen A.4, A.5 und A.5 abgebildet.

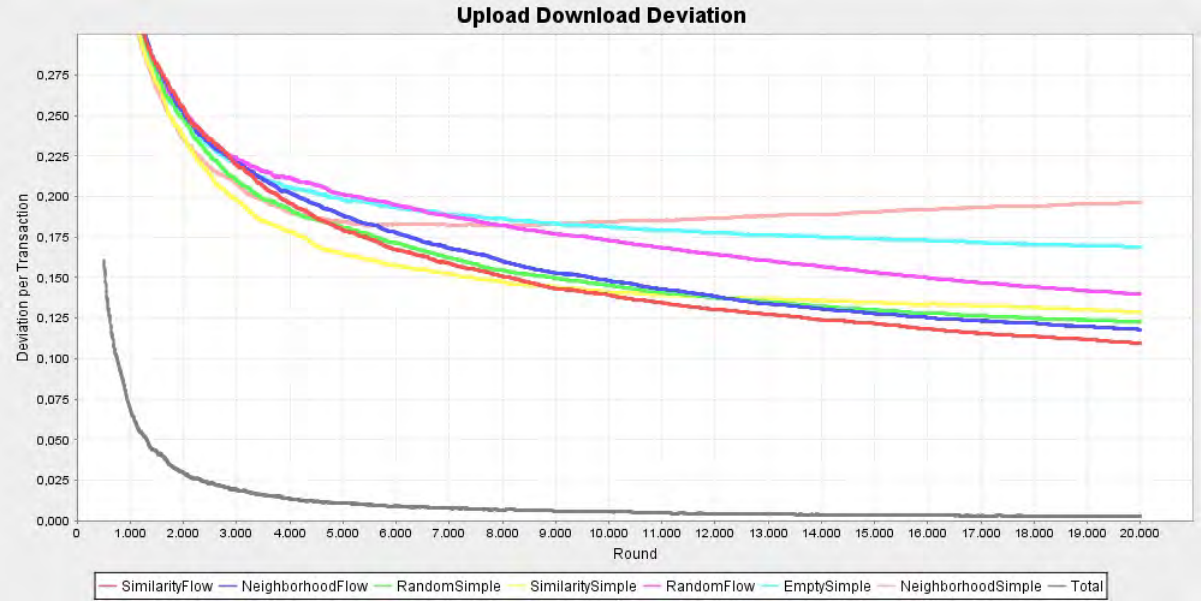

Abbildung 5.4.: Abweichung von Uploads und Downloads bei einem Cluster-τ von 1,25

5.1.2.2. Diskussion

Zunächst lässt sich bei allen Szenarien feststellen, dass mit ansteigenden Cluster-τ die Clusterkoeffizienten ansteigen. Dies bedeutet, dass ein höheres Content-Clustering zu einem höheren Transaktions-Clustering führt.
Ebenfalls steigt mit zunehmenden Cluster-τ die Gerechtigkeit im Netzwerk, unabhängig von der gewählten Tokenverteilung und dem Vertrauenswürdigkeitsprüfungsverfahren. Dabei ist zu beachten, dass die Verbesserung der Gerechtigkeit bei steigendem Cluster-τ für jedes Szenario unterschiedlich stark ausfällt, d.h. es gibt Strategien, die die Clustereigenschaften besser ausnutzen können als andere. Besonders gut gelingt dies dem SimilarityFlow Szenario, welches die Fehlerquote von 11 % bei einem Cluster-τ von 1,25 auf 7,5 % bei einem Cluster-τ von 2,75 senken kann, hingegen kann das NeighborhoordFlow Szenario die Fehlerquote nur von 11,8 % auf 9,3 % senken und das EmptySimple Szenario von 16,8 % auf 14.1 %.

Vergleicht man die Clusterkoeffizienten der einzelnen verwendeten Strategien miteinander,

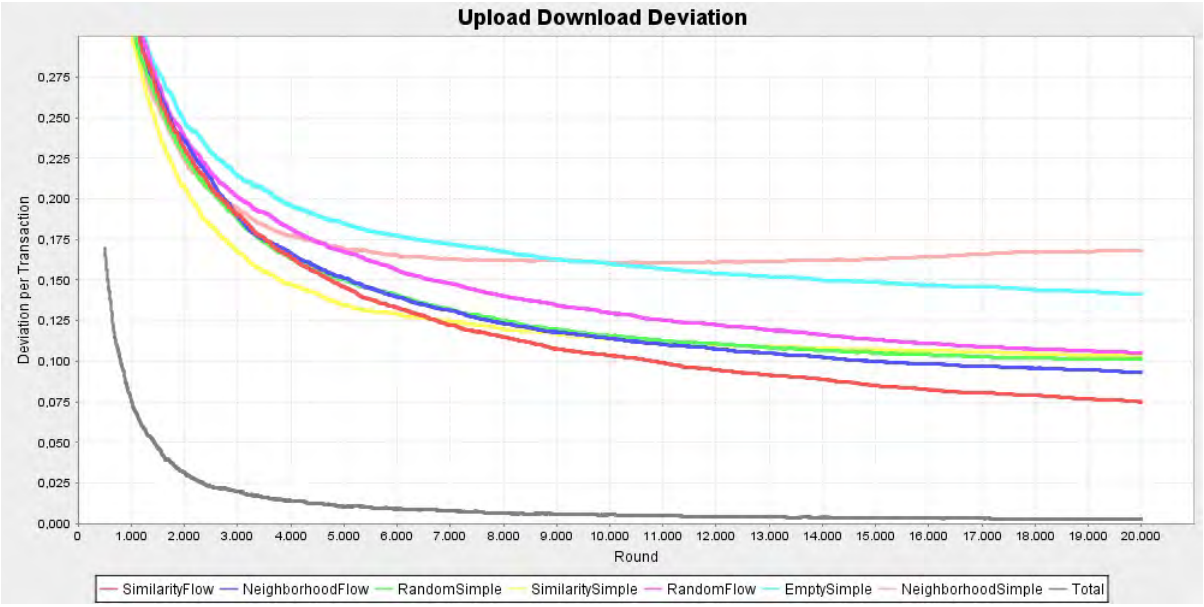

Abbildung 5.5.: Abweichung von Uploads und Downloads bei einem Cluster-τ von 2,75

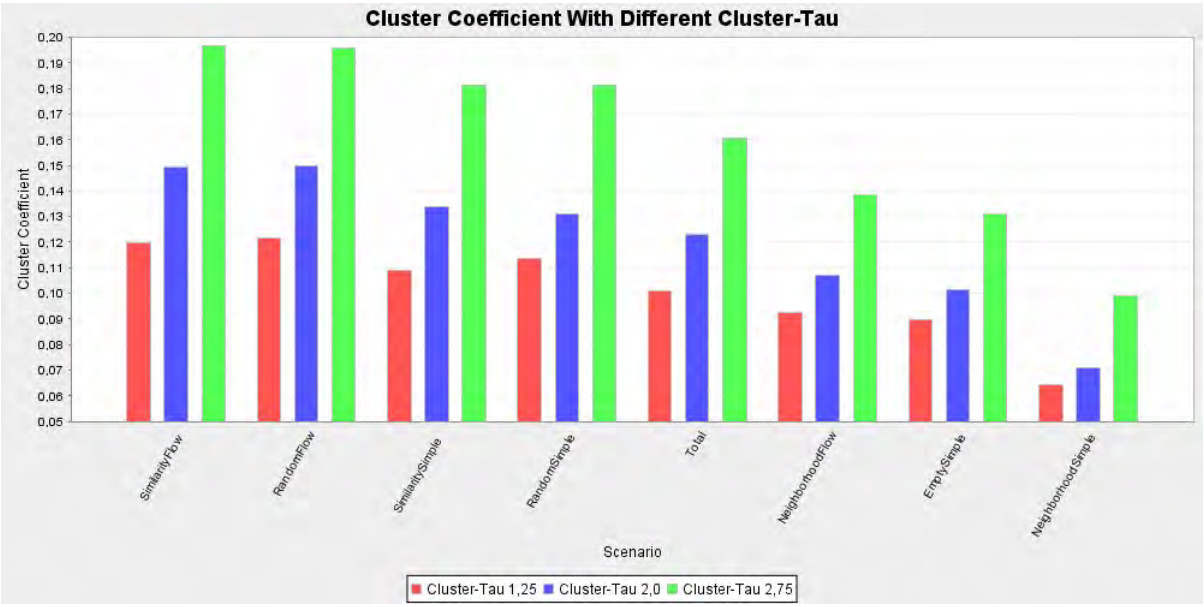

Abbildung 5.6.: Clusterkoeffizient bei unterschiedlichem Cluster-τ

so lässt sich feststellen, dass sowohl die gewählte Tokenverteilungsstrategie als auch das Vertrauenswürdigkeitsprüfungsverfahren das Transaktions-Clustering stark beeinflussen. So fällt der Clusterkoeffizient bei der Verwendung des Vertrauenswürdigkeitsprüfungsverfahren mittels des maximalen Flusses immer größer aus, als die einfache Vertrauenswürdigkeitsprüfung und eine Tokenverteilung an ähnliche Peers fällt immer höher aus, als die Tokenverteilung an die Nachbarn.
Ein Szenario mit einem sehr großen Clusterkoeffizienten ist das SimilarityFlow Szenario, dieses schafft, abgesehen vom Total Szenario, die gerechteste Leistungsverteilung. Den kleinsten Clusterkoeffizienten liefert das NeighborhoodSimple Szenario, welches auch die ungerechteste Leistungsverteilung besitzt.
Aus diesen Ergebnissen könnte man nun die Behauptung aufstellen, dass ein höherer Clusterkoeffizient auch zu einer höheren Gerechtigkeit führt. Jedoch gilt dies nur bedingt. Als Gegenbeispiel, dient hier das RandomFlow Szenario, welches einen ähnlich hohen Clusterkoeffizienten besitzt, wie das SimilarityFlow Szenario, jedoch zu einer deutlich niedrigeren Gerechtigkeit führt.
Ein hoher Clusterkoeffizient fördert nur dann eine gerechtere Leistungsverteilung, wenn die Nachbarn eines Peers, Informationen über dessen Transaktionen erhalten und diese nicht wahllos im Netzwerk verteilt werden, wie es bei der zufälligen Tokenverteilung der Fall ist.

5.1.3. Variation der Anzahl versandter Token

In diesem Experiment soll untersucht werden, inwiefern sich eine unterschiedliche Anzahl versandter Token auf die allgemeine Gerechtigkeit auswirkt und wie sie die Clusterstruktur beeinflusst. Dazu wird neben der Ausgangssimulation, bei der 25 Token pro Transaktion erzeugt werden, eine Simulation mit 15 und eine Simulation mit 35 Token generiert. Die Konfigurationsdateien sind in A.5.1.4 und A.5.1.5 zu finden.

5.1.3.1. Ergebnisse

Die Abbildungen 5.7 und 5.8 zeigen die durchschnittliche Abweichung zwischen Uploads und Downloads und die Grafik 5.9 einen Überblick über die durchschnittlichen Clusterkoeffizienten der Szenarien.

5.1.3.2. Diskussion

Generell lässt sich erkennen, dass die allgemeine Gerechtigkeit mit steigender Anzahl verteilter Token steigt. Vergleicht man das NeighborhoodFlow mit dem SimilarityFlow Szenario, so lässt sich erkennen, dass das SimilarityFlow Szenario eine größere Anzahl Token weit aus besser nutzen kann. Die Gerechtigkeit steigt bei diesem Szenario stärker an als beim NeighborhoodFlow Szenario.
Betrachtet man die Clusterkoeffizienten aller Szenarien, so lässt sich erkennen, dass lediglich bei den Tokenverteilungen an ähnliche Peers eine Steigerung des Koeffizienten festzustellen ist, bei steigender Anzahl verteilter Token. Dies lässt darauf schließen, dass nur die Verteilung an ähnliche Peers das Reputation-Clustering fördert, welches wiederum scheinbar eine bessere Reputationsberechnung ermöglicht.

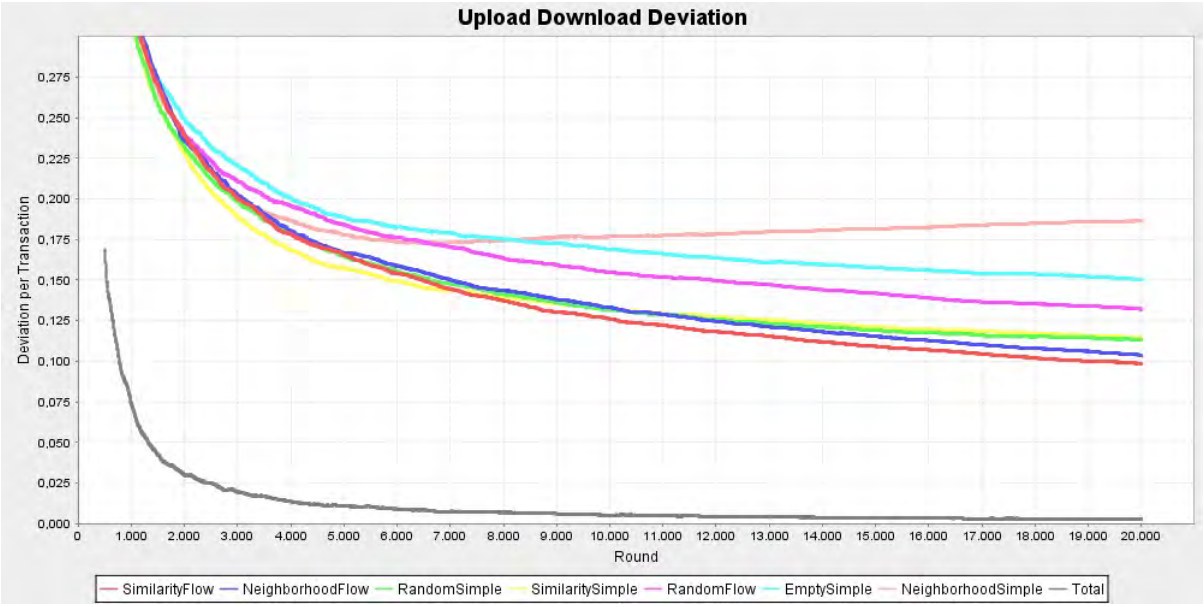

Abbildung 5.7.: Abweichung von Uploads und Downloads bei 15 versandten Token

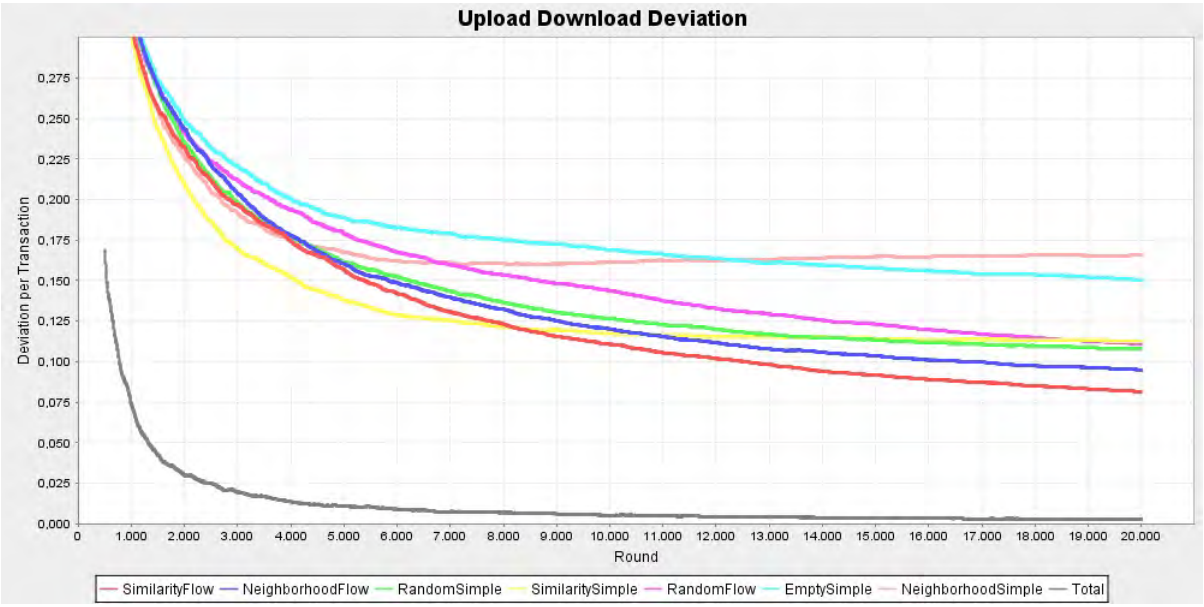

Abbildung 5.8.: Abweichung von Uploads und Downloads bei 35 versandten Token

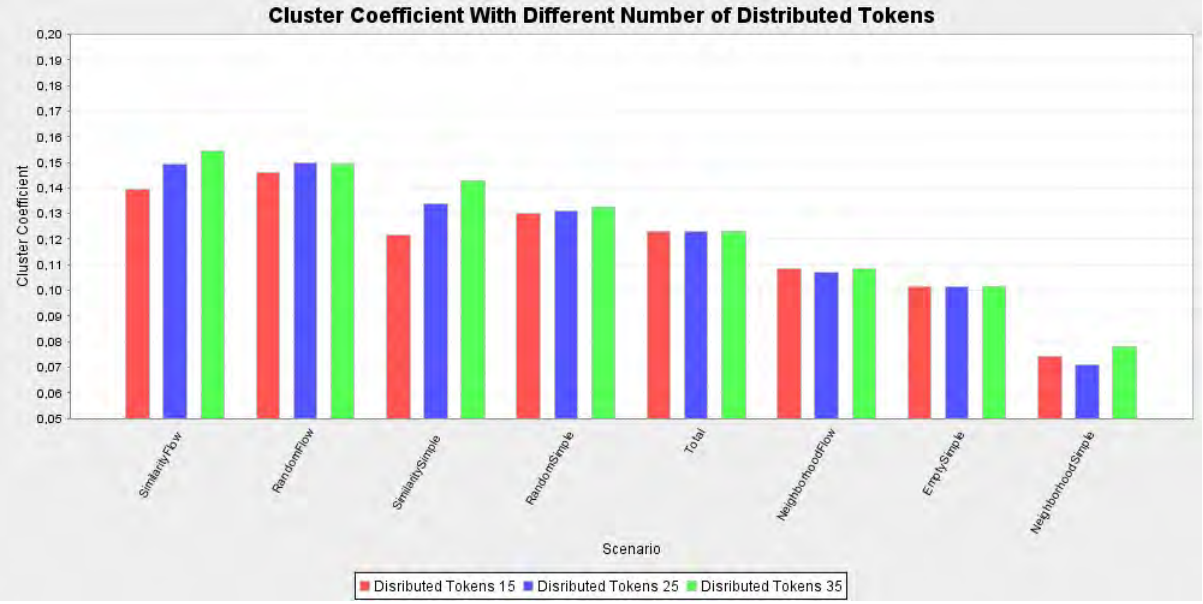

Abbildung 5.9.: Clusterkoeffizient bei einer unterschiedlichen Anzahl versandter Token

5.1.4. Tokenverteilungsstrategien im direkten Vergleich

In der Ausgangssimulation hat sich die Tokenverteilung an ähnliche Peers bereits als gerechteste Tokenverteilung herausgestellt. In dieser Simulation, soll untersucht werden, ob diese Verteilungsstrategie auch für den einzelnen Peers die besten Ergebnisse liefert.
Um dies festzustellen, werden die beiden Verteilungsstrategien, die Nachbarschaftsverteilung und die Verteilung an ähnliche Peers, in einem direkten Vergleich untersucht. Hierfür wird eine Simulation durchgeführt, bei der die eine Hälfte der Peers die Strategie der Nachbarschaftsverteilung verfolgt und die andere, eine Verteilung an ähnliche Peers. Um eine repräsentative Aussage treffen zu können, werden vier verschiedene Zufallszahlenreihen für die Simulation verwendet. Um die Vorteile, die eine Gruppe von Peers durch die zufällig generierte Netzwerkstruktur haben könnte, zu neutralisieren, wird jede Zufallszahlenreihe zweimal verwendet und jeweils die verwendete Strategie der beiden Peergruppen untereinander ausgetauscht. Die acht daraus resultierenden Szenarien werden von 1 bis 4 durchnummeriert, je nachdem welche Zufallszahlenreihe verwendet wird. Der Austausch der Strategien zwischen den Peergruppen wird mit den Buchstaben “a” und “b” versehen. Detaillierte Konfigurationsinformationen lassen sich A.5.1.6 entnehmen.

5.1.4.1. Ergebnisse

Die Grafik 5.10 zeigt den Verlauf der durchschnittlichen Anzahl Downloads pro Upload[2] an. In der Tabelle 5.1 sind detaillierte Angaben der Downloads pro Upload für jedes einzelne Szenario zusammengefasst.

[2]Diese Metrik wird in 4.4.2.3 beschrieben.

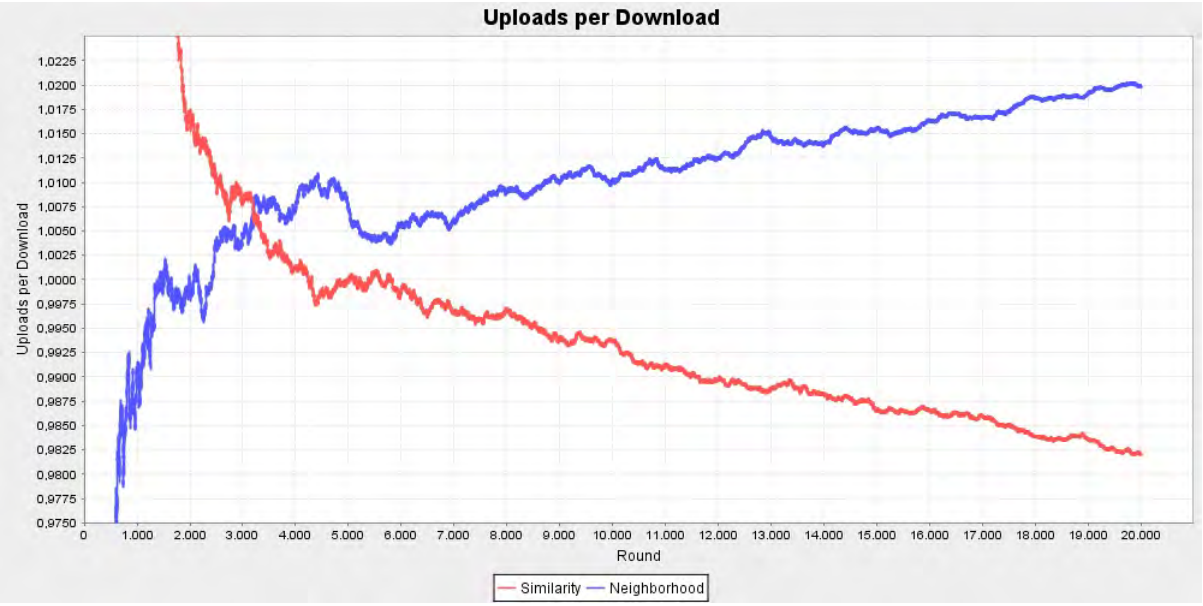

Abbildung 5.10.: Durchschnittliche Uploads pro Download

Runde	5000	10000	15000	20000
SimilarityNeighborhood1a	-0,2 %	2,1 %	3,9 %	4,5%
SimilarityNeighborhood1b	-0,5 %	-2,5 %	0,3 %	2,5 %
SimilarityNeighborhood2a	0,1 %	-0,3 %	1,1 %	2,5 %
SimilarityNeighborhood2b	2,8 %	1,1 %	1,2 %	2,4 %
SimilarityNeighborhood3a	-1 %	2,6 %	3,9 %	4,6 %
SimilarityNeighborhood3b	4,8 %	3,8 %	4,2 %	5,1 %
SimilarityNeighborhood4a	-1,8 %	2,1 %	4,9 %	5 %
SimilarityNeighborhood4b	0,8 %	3,8 %	4,2 %	4,5 %

Tabelle 5.1.: Abweichung von Uploads pro Download im Detail

5.1.4.2. Diskussion

Wie aus der Grafik 5.10 ersichtlich ist die Tokenverteilung an ähnliche Peers die bessere Variante im direkten Vergleich zur Tokenverteilung an die Nachbarschaft.
Zu Beginn aller acht Simulationen schwanken die Messwerte, jedoch können sie sich zu Gunsten der Tokenverteilung an ähnliche Peers einpendeln. Die Erfolgsquote liegt bei ca. $3,75\%$ im Vergleich zu der Nachbarschaftsverteilung und mit einer Steigerung bei längerer Laufzeit ist zu rechnen.
Das starke Schwanken zu Beginn der Simulation, lässt sich damit erklären, dass zu Beginn der Simulation noch kein Wissen über die Netzwerkstruktur vorhanden ist und deshalb die jeweiligen Tokenverteilungsstrategien nicht effektiv angewandt werden können. Die Token werden zu Beginn mehr oder minder nur zufällig im Netzwerk verteilt. Des Weiteren müssen die beiden Verteilungsmethoden eine gewissen Anzahl an Token verteilt haben, um repräsentative Ergebnisse liefern zu können.

5.1.5. Auswirkungen der Optimierungsverfahren

In den vorangegangenen Simulationen wurden die beiden Optimierungsmethoden für das Tokenverteilungsverfahren angewandt. In dieser Simulation soll untersucht werden, welche Auswirkungen diese auf die Messergebnisse haben. Hierfür wird zum einen das aus der Ausgangssimulation bekannten Szenario SimilarityFlow simuliert.
Des weiteren wird das RandomFlow Szenario genauer betrachtet. Hier stellt sich vor allem die Frage, wie stark beeinflussen die Optimierungsverfahren die zufällige Verteilung der Token. Kann man überhaupt noch von einer zufälligen Verteilung sprechen, oder beeinflussen die Optimierungsverfahren die Verteilung zu stark?
Die Konfigurationsdatei für diese Simulation befindet sich in A.5.1.7.

5.1.5.1. Ergebnisse

Die Abbildungen 5.11 zeigt die durchschnittliche Abweichung zwischen Uploads und Downloads bei der Verwendung unterschiedlicher Optimierungsverfahren.

5.1.5.2. Diskussion

Durch beide Optimierungsverfahren wird eine gerechte Leistungsverteilung im Netzwerk gewährleistet. Vor allem durch die eigennützige Tokenverteilung wird ein deutlich fairerer Austausch der Leistungen erreicht. Dies ist besonders erstaunlich, da dieses Optimierungsverfahren in erster Linie dafür entwickelt wurde, den persönlichen Gewinn eines Peers zu erhöhen und nicht um eine allgemein gerechtere Leistungsverteilung zu schaffen.
Nun stellt sich die Frage, warum führt die eigennützige Tokenverteilung zu einer deutlich besseren Fairness?
Um diese Frage zu klären muss man das Wissen betrachten, aufgrund dessen die Tokenverteilungsstrategien ihre Entscheidungen treffen. Die Kernstrategien ziehen ihr Wissen aus vorhandenen Token, d.h. ihr Wissen stammt aus vergangenen Transaktionen. Bei der eigennützigen Verteilung hingegen, werden die Token an diejenigen Peers versandt, bei denen der Versender in der Warteschlange steht. Dies sind genau die Peers mit denen der Versender in naher Zukunft eine Transaktion tätigen möchte, d.h. bei dieser Optimierung werden Vorhersagen über zukünftige Transaktionen gemacht. Dieses daraus entstehende

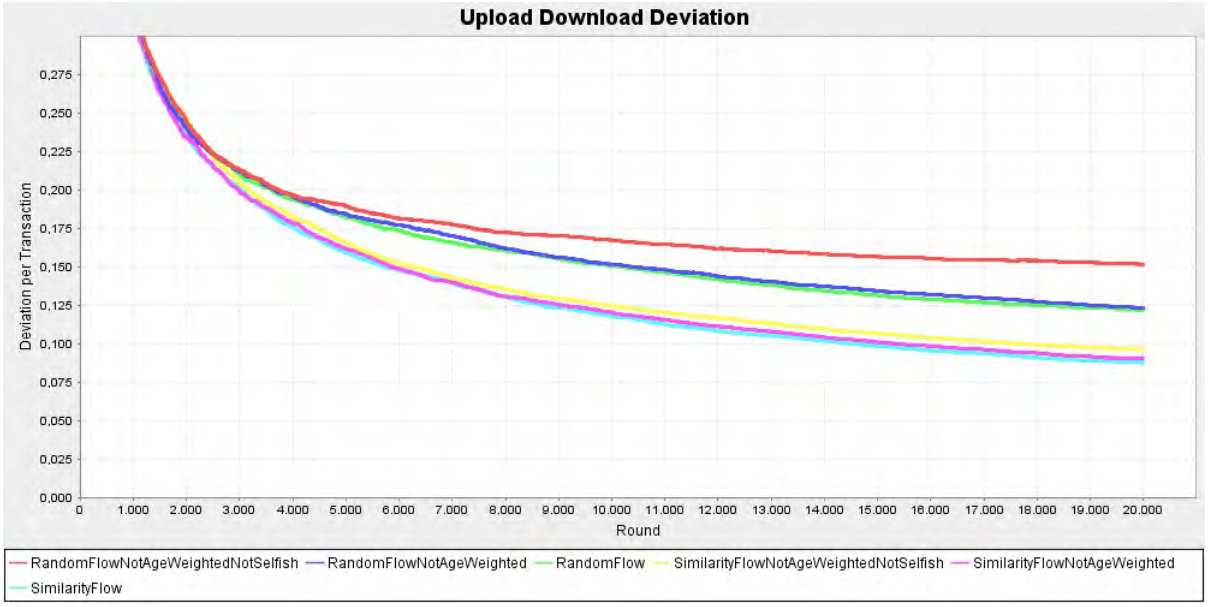

Abbildung 5.11.: Abweichung von Uploads und Downloads bei unterschiedlichen Optimierungsverfahren

zusätzliche Wissen über die Strukturen des Netzwerks ermöglicht eine gezieltere Verteilung der Token im Netzwerk und somit auch eine Steigerung des Informationsgrad mit der daraus resultierenden faireren Consumerauswahl.
Besonders stark wirkt sich die eigennützige Optimierung auf das RandomFlow Szenario aus, hier wird eine deutlich Steigerung der Fairness im Netzwerk erreicht. Somit kann bei diesem Szenario eigentlich nicht mehr von einer rein zufälligen Tokenverteilung gesprochen werden.

5.2. Angriffe auf das Reputationssystem

In der vorangegangen Versuchsreihe wurde das Reputationssystem nur unter Idealbedingungen betrachtet. In dieser Versuchsreihe sollen nun untersucht werden, wie Robust die einzelnen Strategien gegenüber Angriffen sind. Dies gilt im Besonderen für die Plausibilitätsprüfungsverfahren, die noch keiner genaueren Untersuchung unterzogen wurden.
Es werden vier verschiedene Angriffsszenarien simuliert. Die ersten beiden Angriffe, Freerider 5.2.1 und Foul Dealer 5.2.2 zählen zu den eher klassischen Angriffen auf das Reputationssystem. Bei diesen Szenarien geben die Peers objektiv und korrekte Bewertungen ab. Hingegen wird bei der Bad Voter Attacke 5.2.3 simuliert, dass die Angreifer durch verfälschte Bewertungen versuchen ihre eigene Reputation zu steigern.
Schließlich generiert bei der Sybil Attacke 5.2.4 der Angreifer imaginäre Peers, die den Angreifer positiv bewerten, um ihm somit zu einer guter Reputation verhelfen.

5.2.1. Freerider

Die sogenannten Freerider sind Peers, die selber keine Leistungen zur Verfügung stellen, aber dennoch selbst möglichst viel Leistung konsumieren möchten. Ein Reputationssy-

stem muss dafür sorgen die Leistungen, die die Freerider erhalten, einzuschränken.
Streng genommen handelt es sich hierbei nicht um einen gezielten Angriff auf das Reputationssystem, sondern eher um eine Alltagssituation, die dem 'Angreifer' keine bösartige Grundhaltung unterstellt.
Für diesen Versuch werden vier Szenarien simuliert, die sich nur in dem verwendeten Plausibilitätsprüfungsverfahren unterscheiden. Es werden insgesamt 3000 Peers in 30 Clustern simuliert. Von diesen Peers gehören 300 Peers der Angreifergruppe und 300 Peers einer Kontrollgruppe an. Beide Gruppen starten nach 3000 Runden, damit die restlichen Peers Zeit haben, sich kennen zu lernen und Vertrauen aufzubauen. Die Konfigurationsdatei lässt sich unter A.5.1.8 nachschlagen.

Die Szenarien:

1. Simple:
   - PlausibilityCalculator: SimplePlausibilityCalculator
2. Transitive:
   - PlausibilityCalculator: TransitivePlausibilityCalculator
3. NTransitive:
   - PlausibilityCalculator: NTransitivePlausibilityCalculator
4. Similarity:
   - PlausibilityCalculator: SimilarityPlausibilityCalculator

5.2.1.1. Ergebnisse

Gemessen wird der Erfolg der Reputationssysteme gegenüber den Freeridern, indem die Anzahl der Anfragen pro Download gemessen wird.[3] Der Übersicht halber wurden die Messergebnisse, der vier hier simulierten Szenarien, in den Graphen 5.12 für die Angreifer und 5.13 für die Kontrollgruppe zusammengefasst. In diesen Graphen ist auch der Referenzwert von 8[4] abgebildet.
Zu beachten gilt, dass die Szenarien Simple und Transitive die selben Messergebnisse liefern und deshalb nur jeweils drei Graphen zu erkennen sind.
In der Abbildung 5.14 werden die durchschnittlichen Clusterkoeffizienten der Simulation dargestellt.

5.2.1.2. Diskussion

Wie bereits erwähnt, liefert das Transitive Szenario und das Simple Szenario die selben Messergebnisse. Die Ursachen hierfür sollen in diesem Abschnitt erläutert werden, jedoch soll sich zunächst dem Vergleich der drei Szenarien Simple, Similarity und NTransitive gewidmet werden.
Betrachtet man den Erfolg der Angreifer, so schneidet das Simple Szenario deutlich schlechter ab, als die beiden anderen Szenarien. Bei diesem Szenario müssen die Freerider

[3] Die entsprechende Metrik wird in 4.4.2.2 beschrieben.
[4] Dies entspricht acht Anfragen pro Download und geht aus dem Verhältnis zwischen Upload- und Downloadrate eines Peers hervor.

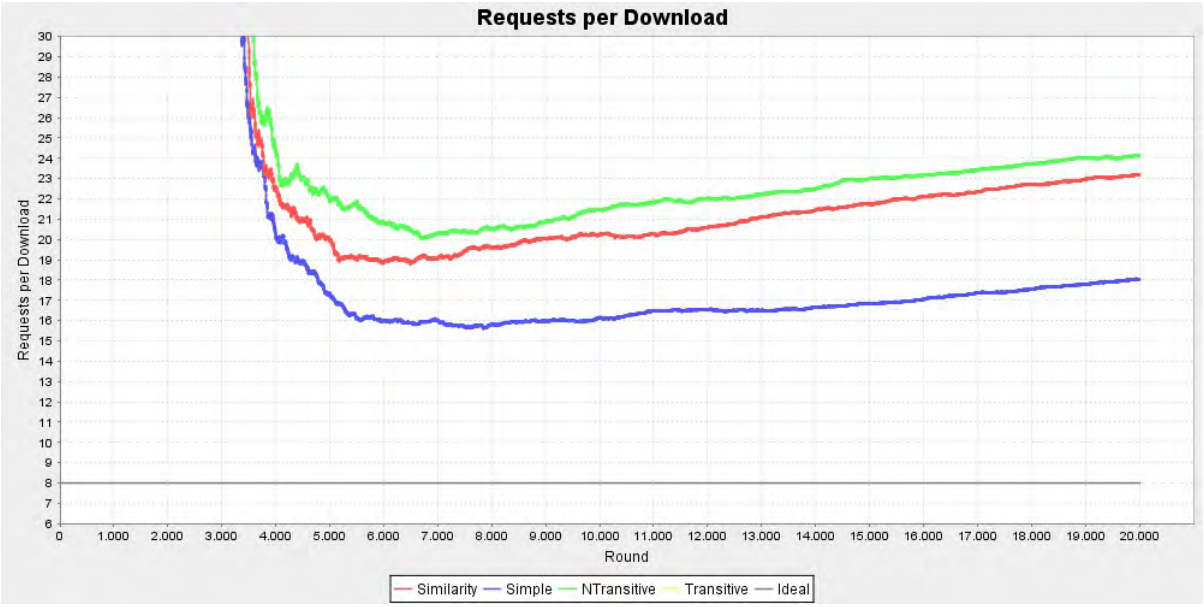

Abbildung 5.12.: Anfragen pro Download für die Angreifer

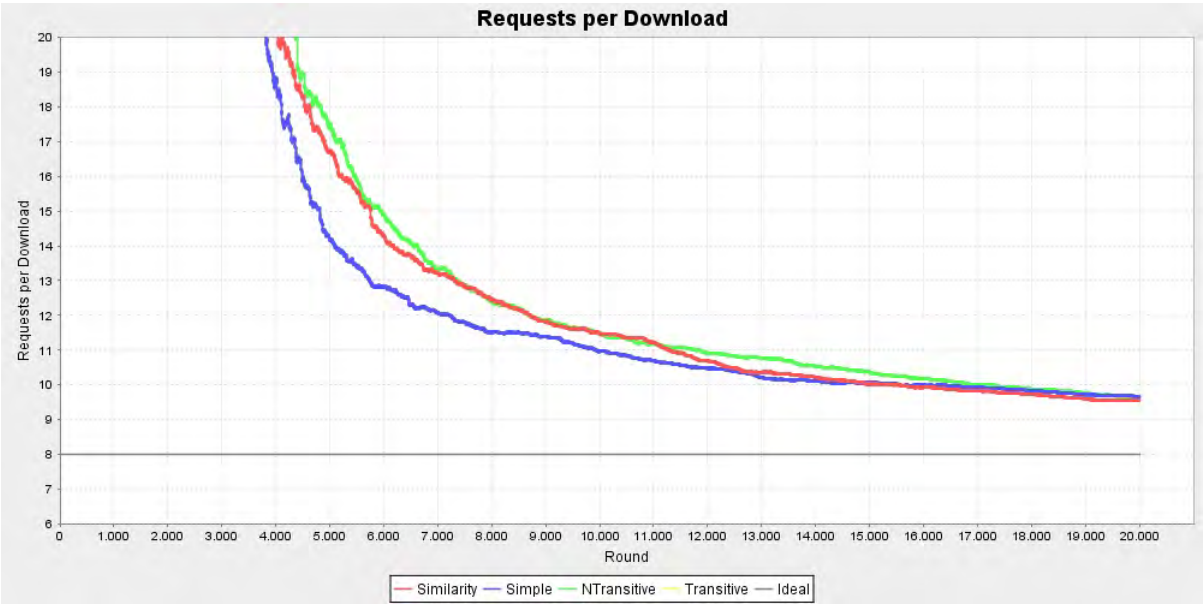

Abbildung 5.13.: Anfragen pro Download für die Kontrollgruppe

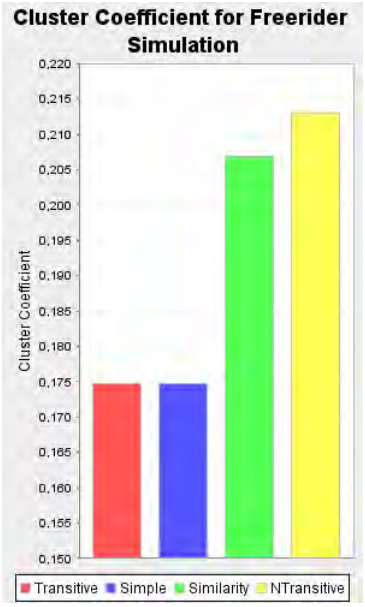

Abbildung 5.14.: Clusterkoeffizienten der einzelnen Freerider Szenarien

nur 18 Anfragen stellen, um eine Leistungseinheit zu erhalten. Beim Similarity Szenario müssen sie hingegen 23 Anfragen stellen und beim NTransitive Szenario sogar 24.
Betrachtet man die Zeit, die die Peers aus der Kontrollgruppe benötigen, bis sie sich eine gewisse Reputation erarbeiten haben, so stellt man fest, dass bei allen 3 Szenarien der Erfolg dieser Peers nach 20000 Runden nahezu identisch ist. Jedoch erreichen die Peers beim Simple Szenario deutlich schneller diese Leistungen als bei den anderen beiden Szenarien. Der Vorsprung im Simple Szenario liegt bei ca. 1000 Runden über einen Großteil der Simulationszeit. Dies bedeutet, das sich bei dem Simple Szenario die Peers aus der Kontrollgruppe deutlich schneller im Netzwerk etablieren können, als bei den anderen beiden Szenarien.
Als Ursache für dieses Verhalten lassen sich die unterschiedlichen Betrachtungsweisen der verwendeten Plausibilitätsprüfungsverfahren nennen.
Beim Simple Szenario wird eine globale Betrachtungsweise angewendet, bei den anderen beiden eine lokale. Die lokale Betrachtungsweise führt zu einem stärkeren Clustering, da der Betrachter eher Peers aus seinem eigenen Umfeld auswählt. Diese lokale Betrachtungsweise, lässt sich wie folgt für die einzelnen Szenarien begründen.
Beim NTransitiven Verfahren, können nur Bewertungen als plausibel angesehen werden von Peers, die direkte Nachbarn vom Betrachter sind. Somit können nur Peers in direkter Nachbarschaft zum Betrachter eine gute Reputation erhalten.
Beim Verfahren mittels der Ähnlichkeit können nur Peers als plausibel bewertet werden, wenn sie und der Betrachter den gleichen Peer ähnlich bewertet haben. Dies wiederum setzt voraus, dass beide Peers vom gleichen Peer Leistungen erhalten haben und somit besitzen sie ebenfalls eine lokale Nähe.
Das daraus folgende erhöhte Clustering lässt sich mittels den in Grafik 5.14 abgebildeten Clusterkoeffizienten nachweisen. Je höher dieses Clustering ist, desto stärker Vertrauen sich die Peers und umso schwerer ist es für außenstehende Peers Leistungen von ihnen zu erhalten.

Die Ursache für das identische Abschneiden des Simple und des Transitive Szenarien liegt darin, dass beide verwendeten Plausibilitätsprüfungsverfahren genau die Bewertungen der Peers als plausibel ansehen, die für die Berechnung der Vertrauenswürdigkeit mittels maximalem Fluss eine Relevanz besitzen.

Nun kann man behaupten, dass beide Verfahren die gleichen Plausiblitätsvektoren liefern. Jedoch hat eine Vergleichssimulation mittels des einfachen Vertrauenswürdigkeitsprüfungsverfahren ergeben, dass beide Szenarien unterschiedliche Ergebnisse liefern. Somit müssen sich die beiden Plausiblitätsvektoren unterscheiden.
Das Charakteristik des Plausibilitätsvektor für das Simple Szenario ist per Definition klar - alle Einträge besitzen einen Wert von eins.
Beim Transitive Szenario hingegen, werden manche Peers als nicht plausibel angesehen. Dies können nur Peers sein, die bei der Flussberechnung keine Relevanz besitzen, weil sonst unterschiedliche Messergebnisse berechnet werden würden. Untersucht man das Verhalten des transitiven Plausibilitätsprüfungsverfahren, so bewertet ein Betrachter genau die Peers als plausibel, die ihm direkt oder indirekt bzw. transitiv korrekte Leistungen erbracht haben. Soll nun der maximale Fluss für die Providerauswahl berechnet werden, so entsteht ein Fluss über die Peers, die dem Betrachter direkt oder indirekt Leistungen erbracht haben und dies sind genau die Peers, deren Bewertung als plausibel angesehen werden. Somit unterscheiden sich die Providerauswahl des transitive Verfahren nicht von der des naiven Verfahren für die spezielle Konfiguration dieser Simulation.
Bei der Berechnung des maximalen Flusses für die Consumerauswahl spielt die Plausibilität ebenfalls keine Rolle. Da nur korrekte Leistungen erzeugt werden, wird jeder Token im Kontext der Leistungsverpflichtungen voll angerechnet, unabhängig davon ob die Bewertung als plausibel angesehen wird oder nicht.
Lediglich fehlerhafte Leistungen bei plausibler Bewertung würden zu niedrigeren Tokenbewertungen führen. Diese sind jedoch bei diesen Simulationsbedingungen nicht vorhanden.

5.2.2. Foul Dealer Attacken

Bei diesem Experiment soll simuliert werden, wie effektiv die einzelnen Plausibilitätsprüfungsverfahren sogenannte Foul Dealer aufspüren können. Foul Dealer sind Peers, die korrupte Leistungen liefern, um damit die Funktionsfähigkeit des P2P Netzwerks einzuschränken.
Für diesen Versuch werden wie beim Freerider Experiment 5.2.1 vier Szenarien simuliert und die Peers in drei Gruppen aufgeteilt. Die genaue Konfiguration lässt sich A.5.1.9 entnehmen.
Im Unterschied zum Freerider Experiment, werden die Angreifer und die Peers der Kontrollgruppe keine Leistungen entgegennehmen, sondern nur Leistungen liefern.

5.2.2.1. Ergebnisse

Der Erfolg des verwendeten Plausibilitätsprüfungsverfahren wird gemessen, indem die Differenz der Uploads der Kontrollgruppe von den Uploads der Angreifer bestimmt wird. Diese Differenz wird in Grafik 5.15 pro Peer über den gesamten Zeitraum der Simulation angegeben.

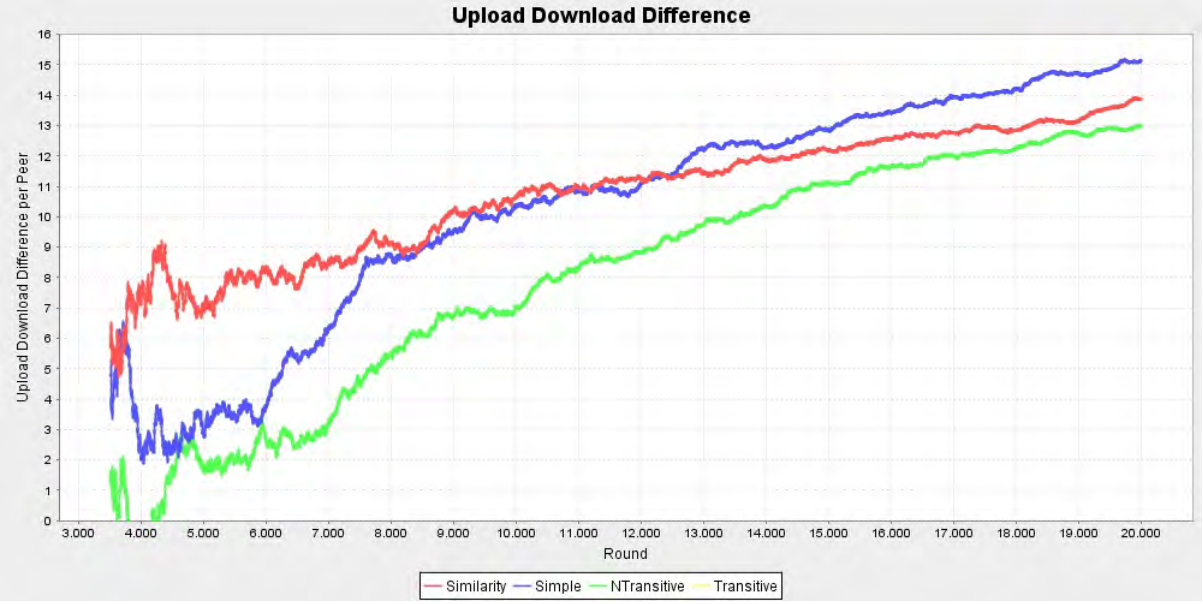

Abbildung 5.15.: Differenz der Uploads der Kontrollgruppe und der Angreifer

5.2.2.2. Diskussion

Zunächst lässt sich wieder feststellen, dass das simple Verfahren und das transitive Verfahren gleiche Messergebnisse liefern. Eine Begründung hierfür wird im weiteren Verlauf der Versuchsauswertung gegeben, jedoch sollen zuvor die unterschiedlichen Messergebnisse miteinander verglichen werden.
Bei allen Szenarien steigt die Differenz der Uploads zwischen Kontrollgruppe und den Angreifern. Dabei sind die einzelnen Szenarien unterschiedlich erfolgreich. Dieser Erfolg entsteht in den ersten Runden und ist wohl auf rein zufällige Ereignisse zurückzuführen, da im weiteren Verlaufen ihre Graphen annähernd parallel verlaufen und somit ein ähnliches Verhalten der Verfahren zu erkennen ist.
Für eine aussagekräftigere Analyse bedarf es der Simulation eines heterogeneren Netzwerks, in dem Peers nicht nur objektive Bewertungen, sondern auch subjektiv verfälschte Bewertungen abgeben. Unter solchen Rahmenbedingungen, die auch der Praxis näher kommen, ist zu erwarten, dass das Plausibilitätsprüfungsverfahren mittels der Ähnlichkeit der abgegebenen Bewertungen einen deutlich größeren Erfolg erzielen wird, als es für die anderen Verfahren zu erwarten ist. Erste Messergebnisse, die diese Aussage stützen, werden in Versuch 5.2.3 vorgestellt.
Es werden jedoch weitere Simulationen nötig sein, um repräsentative Messergebnisse zu erhalten. Diese Simulationen müssen in weiterführenden Arbeiten durchgeführt werden.

Das identische Abschneiden des transitiven und des simplen Verfahren lässt sich ähnlich begründen wie in der Freerider Simulation. Beim transitiven Verfahren werden nur die Bewertungen der Peers als plausibel angesehen, von denen der Betrachter direkt oder indirekt bzw. transitiv korrekte Leistungen erhalten hat. Wie bereits beim Freerider Szenario gezeigt, sind Peers, von denen der Betrachter direkt oder indirekt keine Leistungen erhalten hat, unerheblich für die Flussberechnung. Somit muss einzig das Verhalten der Angreifer überprüft werden, inwieweit dieses die Berechnung des maximalen Flusses be-

einflussen.

Bei der Providerauswahl sind die abgegebenen Bewertungen der Angreifer unabhängig davon, ob sie als plausibel eingestuft werden oder nicht. Die Leistungen der Angreifer werden von allen Peers als negativ bewertet. Entsprechend besitzen sie keine Leistungsansprüche und somit können sie keine Ansprüche weitergeben bzw. besitzen sie keine eingehenden Kanten über die ein Fluss entstehen könnte. Deshalb sind sie für die Flussberechnung bei der Providerauswahl unwichtig.
Ähnliches gilt für die Consumerauswahl, da sie selbst keine Leistungen konsumieren bewerten sie keine anderen Peers. Somit besitzen sie über keine ausgehenden Kanten und deshalb beeinflussen sie auch bei der Consumerauswahl den maximalen Fluss nicht.

5.2.3. Bad-Voter-Attacken

Die verschiedenen Plausibilitätsprüfungsverfahren sollen in dieser Simulation auf ihre Robustheit gegenüber den sogenannten "Bad-Voter-Attacken" überprüft werden. Bei diesen Angriffen versucht der Angreifer, durch die Abgabe von schlechten Bewertungen bei der Inanspruchnahme von Dienstleistungen, sich selber eine bessere Reputation zu verschaffen und gleichzeitig die Reputation des Provider zu verschlechtern.
Es werden die bereits bekannten vier Szenarien simuliert. Bei der Simulation werden die Peers in drei Gruppen unterteilt. Dies sind die objektiv bewertende Peers, die Angreifer, die alle erhaltenen Leistungen schlecht bewerten und einer Kontrollgruppe, die ebenfalls objektiv bewertet. Die Anzahl der Angreifer und der Kontrollgruppe beträgt wie im vorherigen Szenario 200 Peers. Sie starten mit ihren Aktivitäten nach 3000 Runden. Weitere Details der Konfiguration lassen sich A.5.1.10 entnehmen.

5.2.3.1. Ergebnisse

Die Ergebnisse der vier Szenarien werden in dem Graphen 5.16 zusammengefasst. Er zeigt die durchschnittlichen Uploads pro Download der Angreifergruppe für den Verlauf der Simulation. Zusätzlich wurde der Idealwert von 1[5] eingezeichnet.

5.2.3.2. Diskussion

Das einzige Plausibilitätsprüfungsverfahren, das diesen Angriff abwehren kann, ist das Verfahren mittels der Ähnlichkeitsbewertung die beiden transitiven und das naive Verfahren scheitern. Darüber hinaus schafft das Verfahren mittels Ähnlichkeitsbewertung, dass die Angreifer sogar benachteiligt werden, da sie schlechter abschneiden als der Idealwert. Die Hintergründe für dieses Abschneiden wurden bereits in 3.2.2.4 geschildert.

5.2.4. Sybil-Attacken

Bereits in 2.3.5 wurden die Sybil Attacken beschrieben. In dieser Simulation soll gezeigt werden, dass Vertrauenswürdigkeitsprüfung mittels des maximalen Flusses, im Gegensatz zum einfachen Vertrauenswürdigkeitsprüfungsverfahren, in der Lage ist, die Sybil Attacke der Version A zu verhindern. Hierzu wird ein Szenario erstellt, welches aus insgesamt 2250

[5] d.h ein Upload pro Download

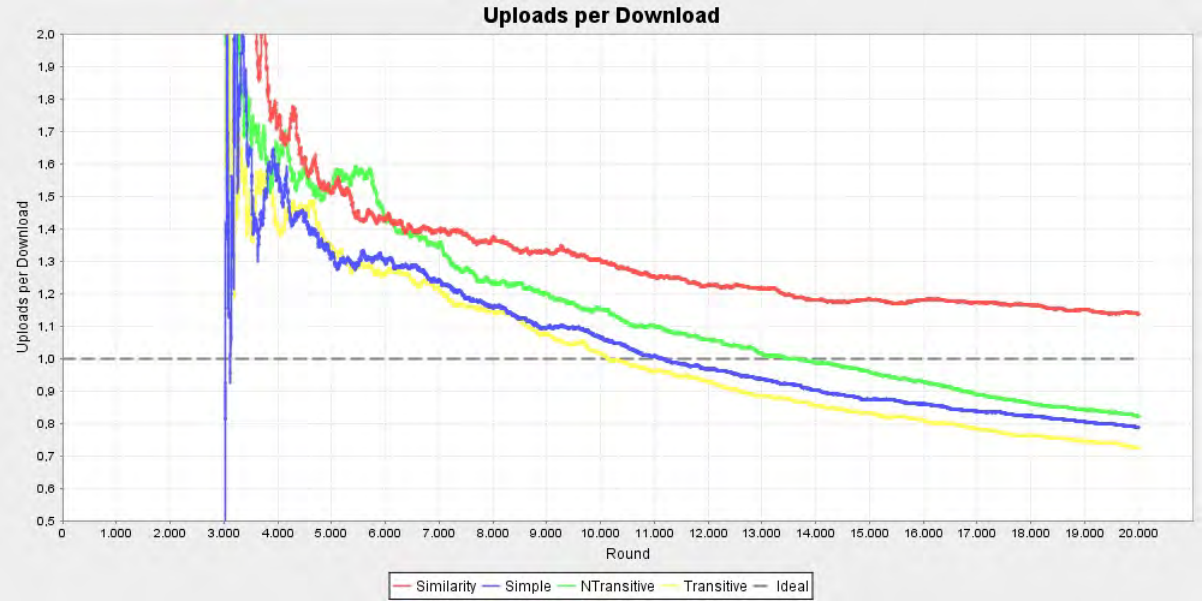

Abbildung 5.16.: Uploads pro Download Für die Angreifer

Peers besteht. Zunächst sind jedoch nur 1500 Peers aktiv, dies ist die Gruppe der kooperativen Peers. Ab Runde 3000, nachdem die kooperativen Peers Zeit hatten sich kennen zu lernen, beginnen drei weitere Gruppen aktiv zu werden, die jeweils aus 250 Peers bestehen. Dieser Gruppen sind die Angreifergruppe, die Sybilgruppe und die Kontrollgruppe. Die Kontrollgruppe verfolgt eine kooperative Strategie und wird benötigt, um Vergleichswerte zu produzieren. Die Sybilpeers sind sogenannte "Dummypeers". Alle Anfragen die diese Peers an einen Provider stellen, können von diesem sofort bearbeitet werden. Somit erhält der Provider Token, die ihn gut bewerten, ohne dass er dafür eine reguläre Transaktion durchgeführt hat.
Die Sybilpeers agieren nur in einem zuvor definierten Cluster. Die kooperative Peers sowie die Kontrollgruppe halten sich in allen anderen Clustern auf, somit gibt es keine Clusterüberschneidungen zwischen der Sybilgruppe und den kooperativen Peers bzw. der Kontrollgruppe und deshalb können diese Gruppen auch nicht in direkte Interaktion mit den Sybil Peers treten, abgesehen vom Tokenaustausch. Die Angreifer wiederum agieren in allen Clustern, somit können sie zum einen mit der Sybilgruppe und zum anderen mit den restlichen Peers interagieren. Die Konfigurationsdatei für diese Simulation ist A.5.1.11.

5.2.4.1. Ergebnisse

In den beiden Graphen 5.17 und 5.18 werden die durchschnittlichen Uploads pro Download für jede Gruppe dargestellt.

5.2.4.2. Diskussion

Wie man aus den Messergebnissen entnehmen kann, ist die einfache Vertrauenswürdigkeitsprüfung gegenüber den Sybil Attacken der Version A machtlos. Im Gegensatz dazu schafft es die Vetrauenswürdigkeitsprüfung mittels maximalen Fluss den Angriff abzuwehren. Die Hintergründe hierfür wurden bereits in 3.2.3 diskutiert.

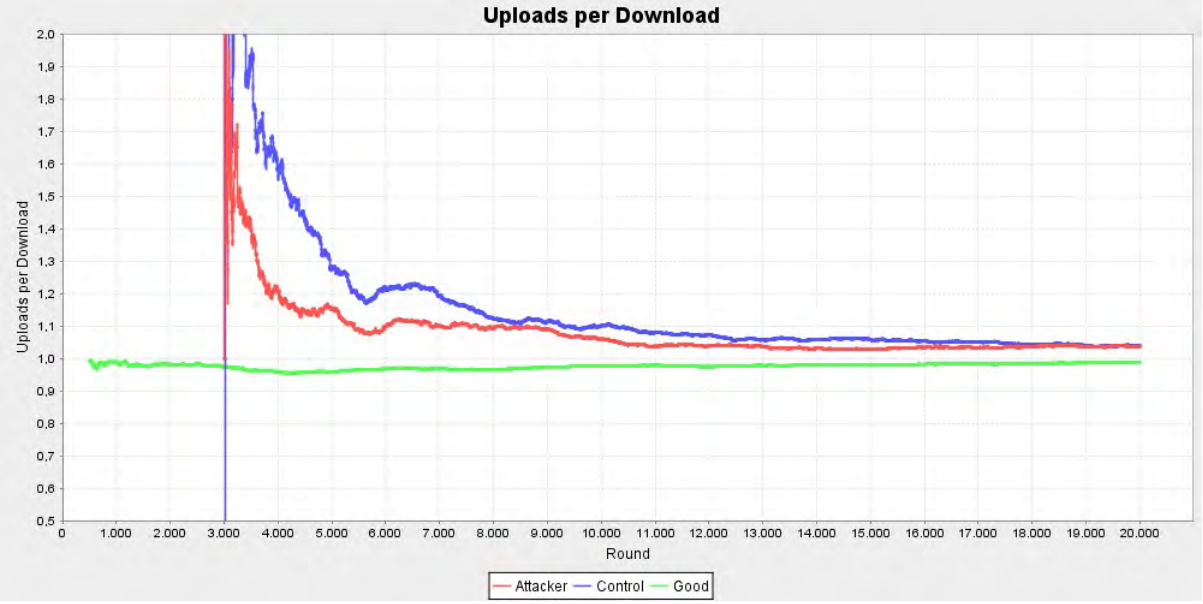

Abbildung 5.17.: Uploads pro Download beim Vertrauenswürdigkeitsprüfungsverfahren mit Hilfe des maximalen Flusses

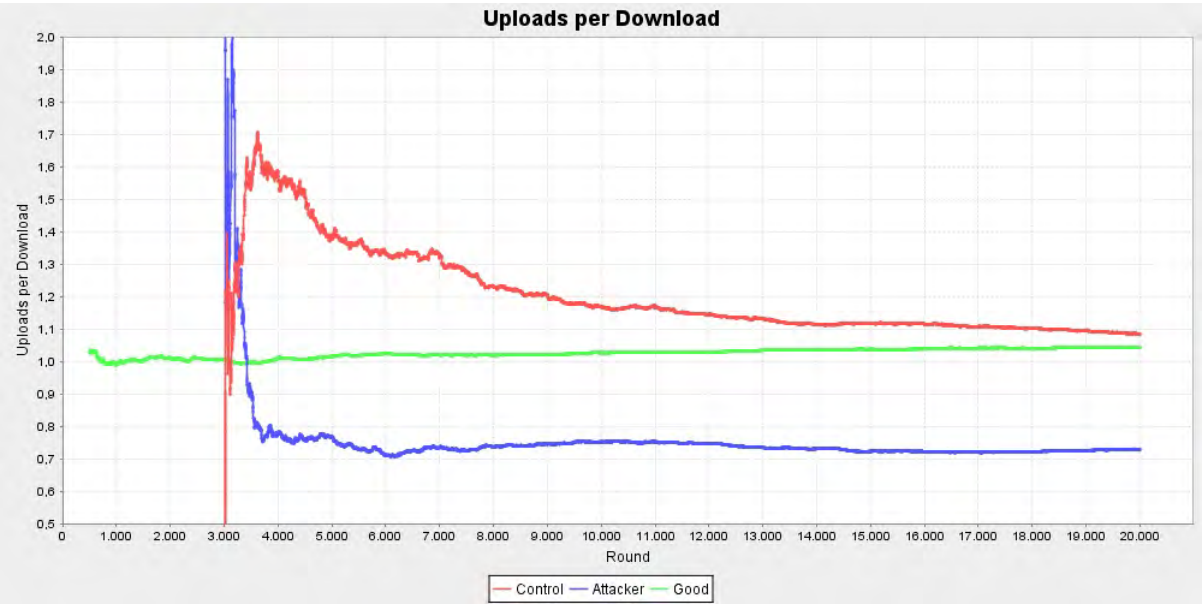

Abbildung 5.18.: Uploads pro Download beim einfachen Vertrauenswürdigkeitsprüfungsverfahren

6. Ausblick

Das vorgestellte tokenbasierte Reputationssystem mit seiner qualitativen und quantitativen Komponente bietet eine Vielzahl an Möglichkeiten, um die Leistungsverteilung gerechter und sicherer zu machen. Alleine die Thematik der Tokenverteilung lässt einen großen Spielraum für Optimierungsansätze. Diese wurden Ansatzweise in 3.5 diskutiert und bedürfen einer genaueren Untersuchung.

Das im Rahmen dieser Arbeit entwickelte Verfahren der Tokenverteilung an ähnliche Peers 3.6.1.2 ist, wie aus den verschiedenen Simulationen hervorgeht, eine effektive Methode um Token gezielt im Netzwerk zu verteilen und sollte deshalb elementarer Bestandteil einer optimierten Tokenverteilung sein.

Die qualitative Analyse in Form der Plausibiltätsprüfung mit Hilfe der Ähnlichkeit der Peers, die ebenfalls im Rahmen dieser Arbeit entwickelt wurde, scheint ein erfolgreicher Ansatz zu sein, um die qualitativen Leistungen von Peers korrekt einzuschätzen. Es bedarf jedoch einer genaueren Analyse, inwieweit dieses Verfahren in der Lage ist, subjektiv beeinflusste Bewertung richtig einzuschätzen. Die transitiven Verfahren, in der Form wie sie hier vorgestellt wurden, sind unzureichend, um in der Praxis Verwendung zu finden. Dies wurde unter anderem aus dem Bad Voter 5.2.3 Experiment deutlich.

Die quantitative Auswertung in Form der Vertrauenswürdigkeitsprüfung scheint ein weitaus problematischeres Feld zu sein. Zwar bietet die Vertrauenswürdigkeit mittels maximalen Fluss eine Schutz gegenüber Angriffen, wie der Sybil-Attacke der Version A, jedoch nicht der Version B. Um diese Probleme in den Griff zu bekommen, müssen weitere Ansätze entwickelt werden. Eine denkbare Möglichkeit wäre es, Peers aufzuspüren, deren Transaktionsverhalten stark von anderen Peers abweicht. Hierfür muss die Ähnlichkeit zwischen dem Transaktionsverhalten von Peers bestimmt werden. Für diese Ähnlichkeitsbestimmung wird eine Ähnlichkeitsmetrik benötigt, die verschiedene Gesichtspunkte, wie Transferraten und unterschiedliche Clusteraktivitäten[1] der Peers, berücksichtigt.

Die Clusterstrukturen innerhalb eines P2P Netzwerks haben nicht nur den negativen Effekt, dass sie die Problemstellung komplexer machen, sie können auch zum Vorteil genutzt werden, um Token gezielt im Netzwerk zu verteilen und so den Informationsgrad zu erhöhen. Aus dem Beispiel der Tokenverteilung an Nachbarn mit einfacherer Vertrauenswürdigkeitsprüfung 5.1.1 wurde deutlich, dass zunächst viel versprechende Lösungen kontraproduktiv sein können. Deshalb sind bei der Entwicklung solcher Verfahren Testsimulationen unablässig.
Die stärke des Clustering hat großen Einfluss auf den Erfolg des verteilten Reputationssystem. Je stärker das Clustering, desto präziser kann ein Peer in seinem Umfeld bewertet werden. Der Nachteil des hohen Clusterings ist, dass Peers außerhalb des eigenen Umfel-

[1] Warum sollte ein Peer nicht vertrauenswürdig sein, nur weil er teilweise in anderen Clustern aktiv ist als der Betrachter?

des deutlich schlechter eingeschätzt werden können.
Das Reputationssystem selbst hat die Möglichkeiten das Clustering durch unterschiedliche Tokenverteilungsstrategien, Provider- und Consumerauswahlstrategien zu beeinflussen. Weitere Forschungen können an dieser Stelle Ansätzen und intelligente Reputationssysteme entwickeln, die je nach Bedarf die Clusterstrukturen beeinflussen, um so ein möglichst ideales Clustering zu erreichen, welches zum einen groß genug sein muss, um eine sichere Aussage über Peers im eigenen Umfeld machen zu können, aber dennoch das eigene Umfeld nicht zu sehr von der Außenwelt abkapselen, so dass keine Aussagen mehr über Peers außerhalb des eignen Umfeldes gemacht werden können.

A. Anhang

A.1. Abweichungen der Messergebnisse

Dieses Kapitel soll einen Eindruck vermitteln, wie groß die einzelnen Abweichungen der Simulationen bei gleichen Simulationsparametern jedoch unterschiedlichen Zufallszahlen.

A.1.1. Similarity-Flow-Abweichung

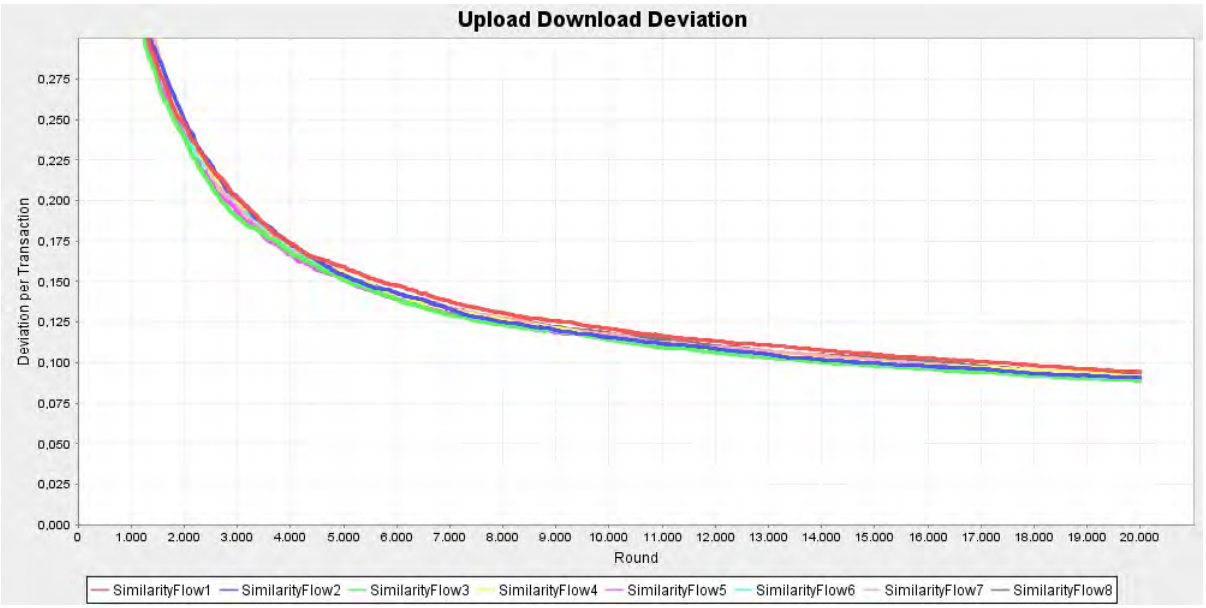

Abbildung A.1.: Abweichung der Messergebnisse bei unterschiedlichen Zufallszahlen

A.2. Testsimulationen

In diesem Kapitel werden die einzelnen Testsimulationen kurz erläutert.

A.2.1. Empty-Trustability-Test

Bei diesem Test werden beide Vertrauenswürdigkeitsprüfungsverfahren getestet. Mit einer Simulation bei der keine Token im Netzwerk verteilt werden. Somit kann jeder Peer nur auf sein selbst gesammeltes wissen zurückgreifen und es macht bei der Auswertung der Vertrauenswürdigkeit keinen unterschied, ob das einfache Verfahren oder das maximale Fluss-Verfahren verwendet wird. Entsprechend müssen beide Simulationen identische Ergebnisse liefern. Beispielhaft wird in A.2 der Graph der beiden Clusterkoeffizienten dargestellt, um die Übereinstimmung zu zeigen.

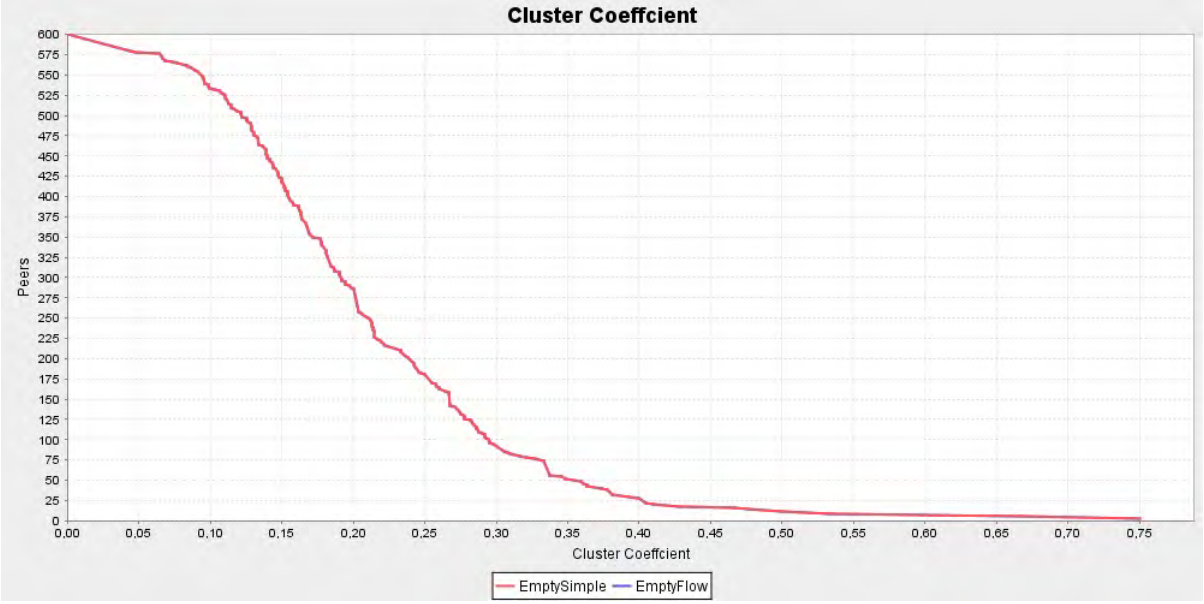

Abbildung A.2.: Clusterkoeffizient des Empty-Trustability-Tests

A.2.2. Token-Distribution-Test

Bei diesem Test werden an alle Peers im Netzwerk Token verschickt. Dies macht es unerheblich welches Tokenverteilverfahren verwendet wird, lediglich das verwendete Vertrauenswürdigkeitsprüfungsverfahren verändert das Ergebnis. In der Graphik werden die drei Tokenverteilverfahren Nachbarschaftsverteilung, Verteilung an ähnliche Peer und zufällige Verteilung mit den beiden Vertrauenswürdigkeitsprüfungsverfahren kombiniert. Als Ergebnis sollten sich zwei Graphen ergeben, einer für die einfache Vertrauenswürdigkeitsprüfung und einer für dir Vertrauenswürdigkeitsprüfung mittels maximalen Flusses.

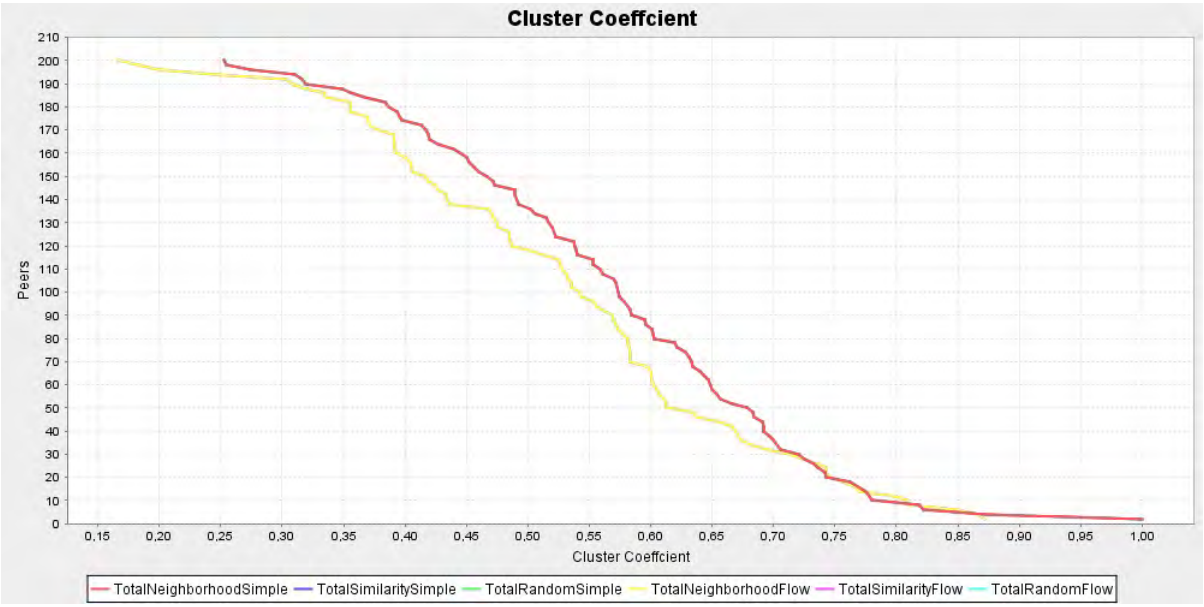

Abbildung A.3.: Clusterkoeffizient des Token-Distribution-Tests

A.3. Weitere Messergebnisse

In diesem Abschnitt werden zusätzlich zu den im Evaluationskapitel vorgestellten Simulationsergebnisse ergänzende Messergebnisse aufgeführt.

A.3.1. Simulation unter Idealbedingungen

A.3.1.1. Verhalten bei unterschiedlichen Clusterstrukturen

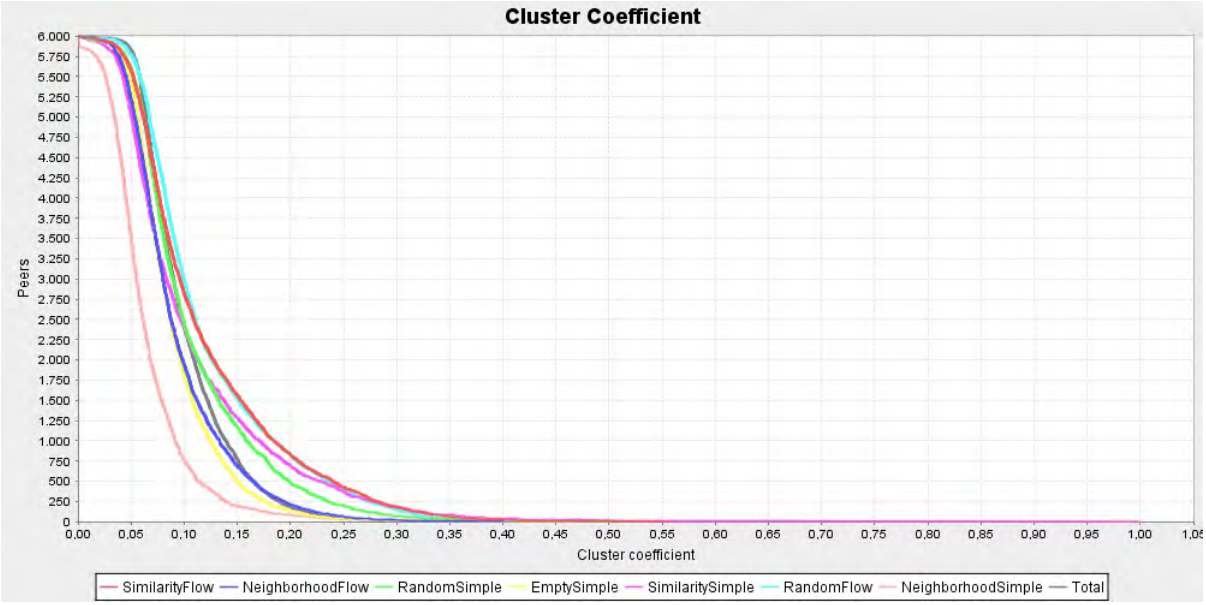

Abbildung A.4.: Clusterkoeffizientenverteilung bei einem Cluster-τ von 1,25

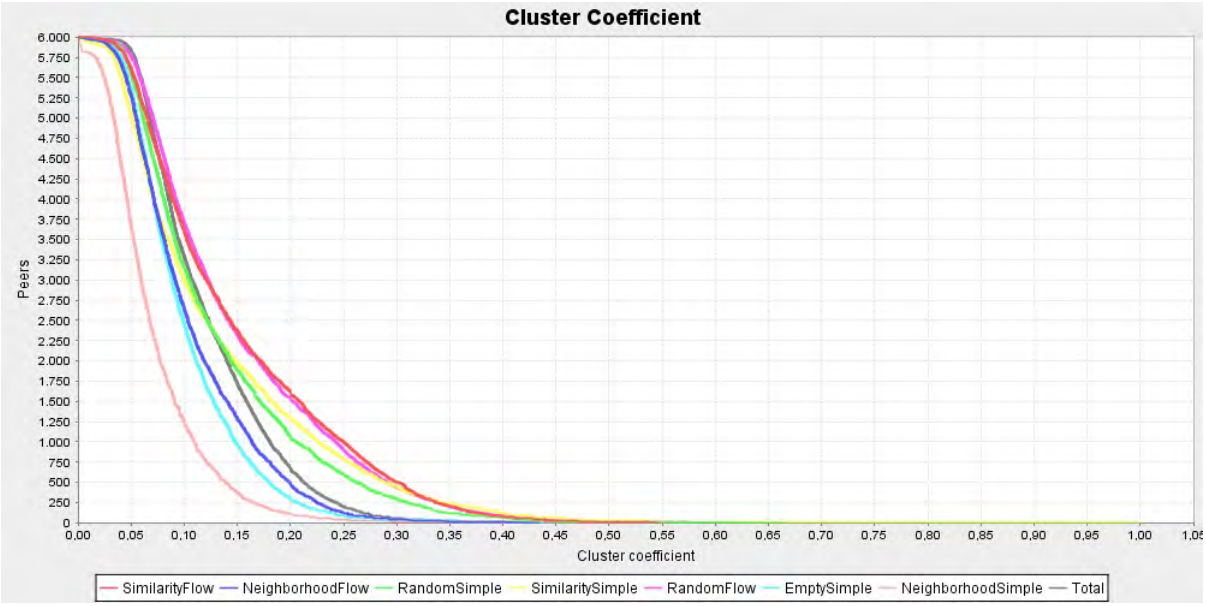

Abbildung A.5.: Clusterkoeffizientenverteilung bei einem Cluster-τ von 2,00

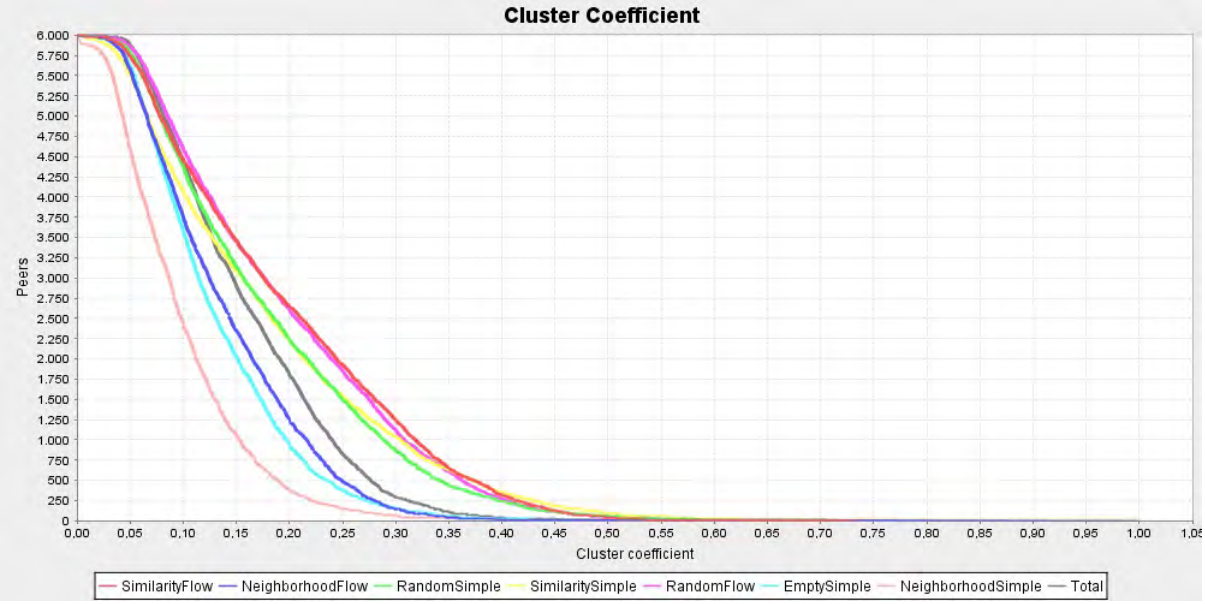

Abbildung A.6.: Clusterkoeffizientenverteilung bei einem Cluster-τ von 2,75

A.4. Nezwerkstrukturen

In diesem Abschnitt werden Daten über die generierten Netzwerkstrukturen aufgelistet.

A.4.1. Simulation unter Idealbedingungen

A.4.1.1. Verhalten bei unterschiedlichen Clusterstrukturen

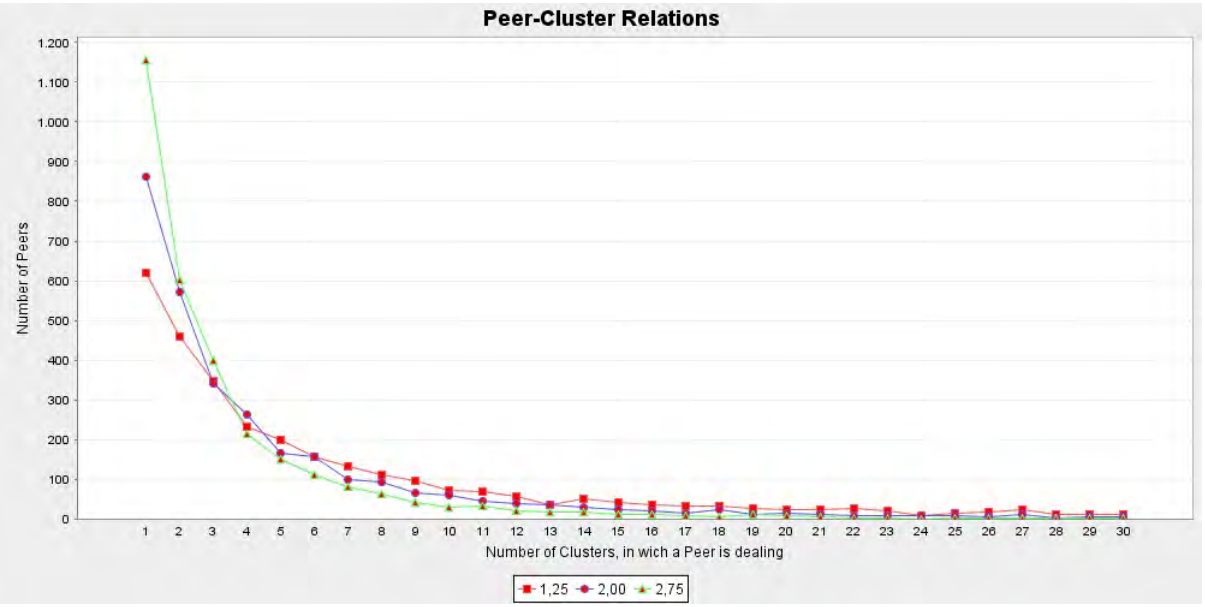

Abbildung A.7.: Anzahl der Peers die in x Clustern aktiv sind, bei entsprechendem Cluster-τ

A.5. Konfigurationsdateien

A.5.1. Evaluierung

A.5.1.1. Ausgangssimulation

```
<testSeries name="TokenDis200">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    25
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfPeers>3000</numberOfPeers>
  <numberOfClusters>30</numberOfClusters>
  <meanDownStream>650</meanDownStream>
  <clusterTau>2.0</clusterTau>
  <scenario name="SimilarityFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="NeighborhoodFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="NeighborhoodSimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="RandomFlow">
    <peersSet name="good" AgeWeighted="true" Selfish="true">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
```

```
  <scenario name="EmptySimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor Selfish="true">
        EmptyTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="SimilaritySimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="Total">
    <peersSet name="good">
      <clientSelector>
        ClientSelectorRankedByGlobalUploadDownloadDifference
      </clientSelector>
      <tokenDistributor>EmptyTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="RandomSimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.1.2. Simulation mit Cluster-τ 1,25

```
<testSeries name="TokenDis125">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    25
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfPeers>3000</numberOfPeers>
  <numberOfClusters>30</numberOfClusters>
  <meanDownStream>650</meanDownStream>
  <clusterTau>1.25</clusterTau>
```

```
105   <scenario name="SimilarityFlow">
106     <providerReputationCalculator
107       TrustabilityCalculator="FlowTrustability"
108       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
109     <clientSelector TrustabilityCalculator="FlowTrustability"
110       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
111     <peersSet name="good">
112       <tokenDistributor AgeWeighted="true" Selfish="true">
113         SimilarityTokenDistributor
114       </tokenDistributor>
115     </peersSet>
116   </scenario>
117   <scenario name="NeighborhoodFlow">
118     <providerReputationCalculator
119       TrustabilityCalculator="FlowTrustability"
120       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
121     <clientSelector TrustabilityCalculator="FlowTrustability"
122       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
123     <peersSet name="good">
124       <tokenDistributor AgeWeighted="true" Selfish="true">
125         NeighborhoodTokenDistributor
126       </tokenDistributor>
127     </peersSet>
128   </scenario>
129   <scenario name="NeighborhoodSimple">
130     <providerReputationCalculator
131       TrustabilityCalculator="SimpleTrustability"
132       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
133     <clientSelector TrustabilityCalculator="SimpleTrustability"
134       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
135     <peersSet name="good">
136       <tokenDistributor Selfish="true">
137         NeighborhoodTokenDistributor
138       </tokenDistributor>
139     </peersSet>
140   </scenario>
141   <scenario name="RandomFlow">
142     <peersSet name="good" AgeWeighted="true" Selfish="true">
143       <tokenDistributor>RandomTokenDistributor</tokenDistributor>
144     </peersSet>
145   </scenario>
146   <scenario name="EmptySimple">
147     <providerReputationCalculator
148       TrustabilityCalculator="SimpleTrustability"
149       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
150     <clientSelector TrustabilityCalculator="SimpleTrustability"
151       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
152     <peersSet name="good">
153       <tokenDistributor Selfish="true">
154         EmptyTokenDistributor
155       </tokenDistributor>
156     </peersSet>
157   </scenario>
158   <scenario name="SimilaritySimple">
159     <providerReputationCalculator
160       TrustabilityCalculator="SimpleTrustability"
161       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
162     <clientSelector TrustabilityCalculator="SimpleTrustability"
```

```
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="Total">
    <peersSet name="good">
      <clientSelector>
        ClientSelectorRankedByGlobalUploadDownloadDifference
      </clientSelector>
      <tokenDistributor>EmptyTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="RandomSimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.1.3. Simulation mit Cluster-τ 2,75

```
<testSeries name="TokenDis275">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    25
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfPeers>3000</numberOfPeers>
  <numberOfClusters>30</numberOfClusters>
  <meanDownStream>650</meanDownStream>
  <clusterTau>2.75</clusterTau>

  <scenario name="SimilarityFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="NeighborhoodFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
```

```
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="NeighborhoodSimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="RandomFlow">
    <peersSet name="good" AgeWeighted="true" Selfish="true">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="EmptySimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor Selfish="true">
        EmptyTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="SimilaritySimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="Total">
    <peersSet name="good">
      <clientSelector>
        ClientSelectorRankedByGlobalUploadDownloadDifference
      </clientSelector>
      <tokenDistributor>EmptyTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="RandomSimple">
```

```
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.1.4. Simulation mit 15 verteilten Token

```
<testSeries name="TokenDis15">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    25
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfPeers>3000</numberOfPeers>
  <numberOfClusters>30</numberOfClusters>
  <meanDownStream>650</meanDownStream>
  <clusterTau>2.0</clusterTau>
  <scenario name="SimilarityFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="NeighborhoodFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="NeighborhoodSimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor Selfish="true">
        NeighborhoodTokenDistributor
```

```
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="RandomFlow">
    <peersSet name="good" AgeWeighted="true" Selfish="true">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="EmptySimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor Selfish="true">
        EmptyTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="SimilaritySimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="Total">
    <peersSet name="good">
      <clientSelector>
        ClientSelectorRankedByGlobalUploadDownloadDifference
      </clientSelector>
      <tokenDistributor>EmptyTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="RandomSimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.1.5. Simulation mit 35 verteilten Token

```
<testSeries name="TokenDis35">
  <numberOfRounds>20000</numberOfRounds>
```

```
<numberOfDistributedTokensPerTransaction>
  35
</numberOfDistributedTokensPerTransaction>
<maxRange>10</maxRange>
<numberOfPeers>3000</numberOfPeers>
<numberOfClusters>30</numberOfClusters>
<meanDownStream>650</meanDownStream>
<clusterTau>2.0</clusterTau>
<scenario name="SimilarityFlow">
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <tokenDistributor AgeWeighted="true" Selfish="true">
      SimilarityTokenDistributor
    </tokenDistributor>
  </peersSet>
</scenario>
<scenario name="NeighborhoodFlow">
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <tokenDistributor AgeWeighted="true" Selfish="true">
      NeighborhoodTokenDistributor
    </tokenDistributor>
  </peersSet>
</scenario>
<scenario name="NeighborhoodSimple">
  <providerReputationCalculator
    TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <tokenDistributor Selfish="true">
      NeighborhoodTokenDistributor
    </tokenDistributor>
  </peersSet>
</scenario>
<scenario name="RandomFlow">
  <peersSet name="good" AgeWeighted="true" Selfish="true">
    <tokenDistributor>RandomTokenDistributor</tokenDistributor>
  </peersSet>
</scenario>
<scenario name="EmptySimple">
  <providerReputationCalculator
    TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <tokenDistributor Selfish="true">
      EmptyTokenDistributor
```

```
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="SimilaritySimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="Total">
    <peersSet name="good">
      <clientSelector>
        ClientSelectorRankedByGlobalUploadDownloadDifference
      </clientSelector>
      <tokenDistributor>EmptyTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
  <scenario name="RandomSimple">
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.1.6. Tokenverteilungsstrategien im direkten Vergleich

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="SimilarityNeighborhood">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    25
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <numberOfPeers>1500</numberOfPeers>
  <numberOfClusters>30</numberOfClusters>
  <meanDownStream>300</meanDownStream>
  <clusterTau>2.0</clusterTau>


  <scenario name="SimilarityNeighborhood1a">
```

```
    <randomInitalisationValue>23</randomInitalisationValue>
    <peersSet name="neigh">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
    <peersSet name="sim">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityNeighborhood2a">
    <randomInitalisationValue>87</randomInitalisationValue>
    <peersSet name="neigh">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>

    <peersSet name="sim">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityNeighborhood3a">
    <randomInitalisationValue>3</randomInitalisationValue>
    <peersSet name="neigh">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>

    <peersSet name="sim">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityNeighborhood4a">
    <randomInitalisationValue>67</randomInitalisationValue>
    <peersSet name="neigh">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>

    <peersSet name="sim">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

```

```
  <scenario name="SimilarityNeighborhood1b">
    <randomInitalisationValue>23</randomInitalisationValue>
    <peersSet name="sim">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <peersSet name="neigh">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityNeighborhood2b">
    <randomInitalisationValue>87</randomInitalisationValue>
    <peersSet name="sim">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <peersSet name="neigh">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityNeighborhood3b">
    <randomInitalisationValue>3</randomInitalisationValue>
    <peersSet name="sim">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <peersSet name="neigh">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityNeighborhood4b">
    <randomInitalisationValue>67</randomInitalisationValue>
    <peersSet name="sim">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <peersSet name="neigh">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        NeighborhoodTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

</testSeries>
```

A.5.1.7. Auswirkungen der Optimierungsverfahren

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="Optimierungsverfahren">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    25
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <numberOfPeers>3000</numberOfPeers>
  <numberOfClusters>30</numberOfClusters>
  <meanDownStream>650</meanDownStream>
  <clusterTau>2.0</clusterTau>

  <scenario name="RandomFlowNotAgeWeightedNotSelfish">
    <peersSet name="good" AgeWeighted="true" Selfish="true">
      <tokenDistributor AgeWeighted="false" Selfish="false">
        RandomTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="RandomFlowNotAgeWeighted">
    <peersSet name="good" AgeWeighted="true" Selfish="true">
      <tokenDistributor AgeWeighted="false" Selfish="true">
        RandomTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="RandomFlow">
    <peersSet name="good" AgeWeighted="true" Selfish="true">
      <tokenDistributor>RandomTokenDistributor</tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityFlow">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="false" Selfish="false">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityFlowNotAgeWeighted">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="false" Selfish="false">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="SimilarityFlowNotAgeWeightedNotSelfish">
```

```
    <peersSet name="good">
      <tokenDistributor AgeWeighted="false" Selfish="false">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>


</testSeries>
```

A.5.1.8. Freerider

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="Freerider">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    25
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfPeers>2400</numberOfPeers>
  <tokenDistributor>SimilarityTokenDistributor</tokenDistributor>
  <meanDownStream>650</meanDownStream>
  <scenario name="Similarity">
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <qualityMultiplier>1.0</qualityMultiplier>
    </peersSet>
    <peersSet name="Attacker">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
      <downUpStreamRatioPerPeer>0</downUpStreamRatioPerPeer>
    </peersSet>
    <peersSet name="control">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
    </peersSet>
  </scenario>
  <scenario name="Simple">
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimplePlausibilityCalculator" />
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimplePlausibilityCalculator" />
    <tokenDistributor>SimilarityTokenDistributor</tokenDistributor>
    <peersSet name="good">
      <qualityMultiplier>1.0</qualityMultiplier>
    </peersSet>
    <peersSet name="Attacker">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
      <downUpStreamRatioPerPeer>0</downUpStreamRatioPerPeer>
    </peersSet>
```

```
    <peersSet name="control">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
    </peersSet>
  </scenario>
  <scenario name="NTransitive">
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="NTransitivePlausibilityCalculator" />
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="NTransitivePlausibilityCalculator" />
    <peersSet name="good">
      <qualityMultiplier>1.0</qualityMultiplier>
    </peersSet>
    <peersSet name="Attacker">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
      <downUpStreamRatioPerPeer>0</downUpStreamRatioPerPeer>
    </peersSet>
    <peersSet name="control">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
    </peersSet>
  </scenario>
  <scenario name="Transitive">
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="TransitivePlausibilityCalculator" />
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="TransitivePlausibilityCalculator" />
    <peersSet name="good">
      <qualityMultiplier>1.0</qualityMultiplier>
    </peersSet>
    <peersSet name="Attacker">
      <roundOfActivation>3000</roundOfActivation>
      <numberOfPeers>300</numberOfPeers>
      <downUpStreamRatioPerPeer>0</downUpStreamRatioPerPeer>
    </peersSet>
    <peersSet name="control">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.1.9. Foul Dealer Attacke

```
<testSeries name="FoulDealer">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    25
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfPeers>2400</numberOfPeers>
  <numberOfClusters>30</numberOfClusters>
  <meanDownStream>400</meanDownStream>
```

```
570   <downUpStreamRatioPerPeer>2</downUpStreamRatioPerPeer>
571   <scenario name="Similarity">
572     <clientSelector TrustabilityCalculator="FlowTrustability"
573       PlausibilityCalculator="SimilarityPlausibilityCalculator">
574       DefaultClientSelector
575     </clientSelector>
576     <providerReputationCalculator
577       TrustabilityCalculator="FlowTrustability"
578       PlausibilityCalculator="SimilarityPlausibilityCalculator" />
579     <peersSet name="good">
580       <reliability>1</reliability>
581       <qualityMultiplier>1.0</qualityMultiplier>
582     </peersSet>
583     <peersSet name="Attacker">
584       <numberOfPeers>300</numberOfPeers>
585       <reliability>0.0</reliability>
586       <roundOfActivation>3000</roundOfActivation>
587       <providerSelector>EmptyProviderSelector</providerSelector>
588     </peersSet>
589     <peersSet name="control">
590       <numberOfPeers>300</numberOfPeers>
591       <roundOfActivation>3000</roundOfActivation>
592       <reliability>1</reliability>
593       <providerSelector>EmptyProviderSelector</providerSelector>
594     </peersSet>
595   </scenario>
596   <scenario name="Simple">
597     <clientSelector TrustabilityCalculator="FlowTrustability"
598       PlausibilityCalculator="SimplePlausibilityCalculator">
599       DefaultClientSelector
600     </clientSelector>
601     <providerReputationCalculator
602       TrustabilityCalculator="FlowTrustability"
603       PlausibilityCalculator="SimplePlausibilityCalculator" />
604     <peersSet name="good">
605       <reliability>1</reliability>
606     </peersSet>
607     <peersSet name="Attacker">
608       <numberOfPeers>300</numberOfPeers>
609       <reliability>0.0</reliability>
610       <roundOfActivation>3000</roundOfActivation>
611       <providerSelector>EmptyProviderSelector</providerSelector>
612     </peersSet>
613     <peersSet name="control">
614       <numberOfPeers>300</numberOfPeers>
615       <roundOfActivation>3000</roundOfActivation>
616       <reliability>1</reliability>
617       <providerSelector>EmptyProviderSelector</providerSelector>
618     </peersSet>
619   </scenario>
620   <scenario name="NTransitive">
621     <clientSelector TrustabilityCalculator="FlowTrustability"
622       PlausibilityCalculator="NTransitivePlausibilityCalculator">
623       DefaultClientSelector
624     </clientSelector>
625     <providerReputationCalculator
626       TrustabilityCalculator="FlowTrustability"
627       PlausibilityCalculator="NTransitivePlausibilityCalculator" />
```

```
    <peersSet name="good">
      <reliability>1</reliability>
    </peersSet>
    <peersSet name="Attacker">
      <numberOfPeers>300</numberOfPeers>
      <reliability>0.0</reliability>
      <roundOfActivation>3000</roundOfActivation>
      <providerSelector>EmptyProviderSelector</providerSelector>
    </peersSet>
    <peersSet name="control">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
      <reliability>1</reliability>
      <providerSelector>EmptyProviderSelector</providerSelector>
    </peersSet>
  </scenario>
  <scenario name="Transitive">
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="TransitivePlausibilityCalculator">
      DefaultClientSelector
    </clientSelector>
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="TransitivePlausibilityCalculator" />
    <peersSet name="good">
      <reliability>1</reliability>
    </peersSet>
    <peersSet name="Attacker">
      <numberOfPeers>300</numberOfPeers>
      <reliability>0.0</reliability>
      <roundOfActivation>3000</roundOfActivation>
      <providerSelector>EmptyProviderSelector</providerSelector>
    </peersSet>
    <peersSet name="control">
      <numberOfPeers>300</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
      <reliability>1</reliability>
      <providerSelector>EmptyProviderSelector</providerSelector>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.1.10. Bad-Voter-Attacke

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="BadVoterAttack">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    20
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfClusters>20</numberOfClusters>
  <numberOfPeers>1600</numberOfPeers>
  <tokenDistributor>SimilarityTokenDistributor</tokenDistributor>
  <meanDownStream>400</meanDownStream>

```

```
<scenario name="NTransitive">
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="NTransitivePlausibilityCalculator">
    DefaultClientSelector
  </clientSelector>
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="NTransitivePlausibilityCalculator" />
  <peersSet name="good">
    <qualityMultiplier>1.0</qualityMultiplier>
  </peersSet>
  <peersSet name="Attacker">
    <roundOfActivation>3000</roundOfActivation>
    <qualityMultiplier>0</qualityMultiplier>
    <numberOfPeers>200</numberOfPeers>
  </peersSet>
  <peersSet name="control">
    <numberOfPeers>200</numberOfPeers>
    <roundOfActivation>3000</roundOfActivation>
    <qualityMultiplier>1.0</qualityMultiplier>
  </peersSet>
</scenario>

<scenario name="Transitive">
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="TransitivePlausibilityCalculator">
    DefaultClientSelector
  </clientSelector>
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="TransitivePlausibilityCalculator" />
  <peersSet name="good">
    <qualityMultiplier>1.0</qualityMultiplier>
  </peersSet>
  <peersSet name="Attacker">
    <roundOfActivation>3000</roundOfActivation>
    <qualityMultiplier>0</qualityMultiplier>
    <numberOfPeers>200</numberOfPeers>
  </peersSet>
  <peersSet name="control">
    <numberOfPeers>200</numberOfPeers>
    <roundOfActivation>3000</roundOfActivation>
    <qualityMultiplier>1.0</qualityMultiplier>
  </peersSet>
</scenario>

<scenario name="Similarity">
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator">
    DefaultClientSelector
  </clientSelector>
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <qualityMultiplier>1.0</qualityMultiplier>
  </peersSet>
  <peersSet name="Attacker">
```

```
      <roundOfActivation>3000</roundOfActivation>
      <qualityMultiplier>0</qualityMultiplier>
      <numberOfPeers>200</numberOfPeers>
    </peersSet>
    <peersSet name="control">
      <numberOfPeers>200</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
      <qualityMultiplier>1.0</qualityMultiplier>
    </peersSet>
  </scenario>

  <scenario name="Simple">
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimplePlausibilityCalculator">
      DefaultClientSelector
    </clientSelector>
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimplePlausibilityCalculator" />
    <peersSet name="good">
      <qualityMultiplier>1.0</qualityMultiplier>
    </peersSet>
    <peersSet name="Attacker">
      <roundOfActivation>3000</roundOfActivation>
      <qualityMultiplier>0</qualityMultiplier>
      <numberOfPeers>200</numberOfPeers>
    </peersSet>
    <peersSet name="control">
      <numberOfPeers>200</numberOfPeers>
      <roundOfActivation>3000</roundOfActivation>
      <qualityMultiplier>1.0</qualityMultiplier>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.1.11. Sybil-Attacke

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="SybilAttack">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    20
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfClusters>20</numberOfClusters>
  <numberOfPeers>1500</numberOfPeers>
  <protectedCluster>5</protectedCluster>
  <meanDownStream>500</meanDownStream>

  <scenario name="Simple">
    <clientSelector TrustabilityCalculator="SimpleTrustability"
      PlausibilityCalculator="SimplePlausibilityCalculator">
      DefaultClientSelector
    </clientSelector>
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustability"
```

```
689       PlausibilityCalculator="SimplePlausibilityCalculator" />
690
691     <peersSet name="Good">
692       <reliability>1</reliability>
693       <qualityMultiplier>1.0</qualityMultiplier>
694       <tokenDistributor>
695         SimilarityTokenDistributor
696       </tokenDistributor>
697     </peersSet>
698     <peersSet name="Control">
699       <roundOfActivation>3000</roundOfActivation>
700       <numberOfPeers>250</numberOfPeers>
701       <reliability>1</reliability>
702       <qualityMultiplier>1.0</qualityMultiplier>
703       <tokenDistributor>
704         SimilarityTokenDistributor
705       </tokenDistributor>
706     </peersSet>
707     <peersSet name="Attacker">
708       <roundOfActivation>3000</roundOfActivation>
709       <numberOfPeers>250</numberOfPeers>
710       <reliability>1</reliability>
711       <qualityMultiplier>1.0</qualityMultiplier>
712       <tokenDistributor>
713         SimilarityTokenDistributor
714       </tokenDistributor>
715       <defaultCluster>5</defaultCluster>
716       <clusterTau>0.1</clusterTau>
717     </peersSet>
718     <peersSet name="Sybils">
719       <roundOfActivation>3000</roundOfActivation>
720       <clusterTau>max</clusterTau>
721       <defaultCluster>5</defaultCluster>
722       <dummyPeer>true</dummyPeer>
723       <numberOfPeers>250</numberOfPeers>
724       <reliability>1</reliability>
725       <qualityMultiplier>1</qualityMultiplier>
726       <providerReputationCalculator
727         TrustabilityCalculator="SimpleTrustability"
728         PlausibilityCalculator="SimplePlausibilityCalculator">
729         RandomProviderReputationCalculator
730       </providerReputationCalculator>
731       <tokenDistributor Selfish="false">
732         RandomTokenDistributor
733       </tokenDistributor>
734       <meanDownStream>500</meanDownStream>
735     </peersSet>
736   </scenario>
737
738   <scenario name="flowSimi">
739     <clientSelector TrustabilityCalculator="FlowTrustability"
740       PlausibilityCalculator="SimplePlausibilityCalculator">
741       DefaultClientSelector
742     </clientSelector>
743     <providerReputationCalculator
744       TrustabilityCalculator="FlowTrustability"
745       PlausibilityCalculator="SimplePlausibilityCalculator" />
746     <peersSet name="Good">
```

```
      <reliability>1</reliability>
      <qualityMultiplier>1.0</qualityMultiplier>
      <tokenDistributor>
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <peersSet name="Control">
      <roundOfActivation>3000</roundOfActivation>
      <numberOfPeers>250</numberOfPeers>
      <reliability>1</reliability>
      <qualityMultiplier>1.0</qualityMultiplier>
      <tokenDistributor>
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <peersSet name="Attacker">
      <roundOfActivation>3000</roundOfActivation>
      <numberOfPeers>250</numberOfPeers>
      <reliability>1</reliability>
      <qualityMultiplier>1.0</qualityMultiplier>
      <tokenDistributor>
        SimilarityTokenDistributor
      </tokenDistributor>
      <defaultCluster>5</defaultCluster>
    </peersSet>
    <peersSet name="Sybils">
      <roundOfActivation>3000</roundOfActivation>
      <clusterTau>max</clusterTau>
      <defaultCluster>5</defaultCluster>
      <dummyPeer>true</dummyPeer>
      <numberOfPeers>100</numberOfPeers>
      <reliability>1</reliability>
      <qualityMultiplier>1</qualityMultiplier>
      <providerReputationCalculator
        TrustabilityCalculator="SimpleTrustability"
        PlausibilityCalculator="SimplePlausibilityCalculator">
        RandomProviderReputationCalculator
      </providerReputationCalculator>
      <providerReputationCalculator>
        RandomProviderReputationCalculator
      </providerReputationCalculator>
      <tokenDistributor Selfish="false">
        RandomTokenDistributor
      </tokenDistributor>
      <meanDownStream>500</meanDownStream>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.2. Abweichungen der Messergebnisse

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="SimilairtyFlowDeviation">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    15
```

```
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <providerMarketGenerator>
    ClusterOrientatedAndRangeWeightedProviderMarketGenerator
  </providerMarketGenerator>
  <transactionValidator>
    DefaultTransactionValidator
  </transactionValidator>
  <tokenValueCalculator restricted="true">
    DefaultTokenValueCalculator
  </tokenValueCalculator>
  <similarityCalculator>DefaultMetric</similarityCalculator>
  <numberOfPeers>3000</numberOfPeers>
  <numberOfClusters>30</numberOfClusters>
  <providerSelector>ProbabilisticProviderSelector</providerSelector>
  <meanDownStream>500</meanDownStream>
  <clusterTau>2.0</clusterTau>
  <scenario name="SimilarityFlow1">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>23</randomInitalisationValue>
  </scenario>

  <scenario name="SimilarityFlow2">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>3</randomInitalisationValue>
  </scenario>
  <scenario name="SimilarityFlow3">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>13</randomInitalisationValue>
  </scenario>
  <scenario name="SimilarityFlow4">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>16</randomInitalisationValue>
  </scenario>
  <scenario name="SimilarityFlow5">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
```

```
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>42</randomInitalisationValue>
  </scenario>
  <scenario name="SimilarityFlow6">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>17</randomInitalisationValue>
  </scenario>
  <scenario name="SimilarityFlow7">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>53</randomInitalisationValue>
  </scenario>
  <scenario name="SimilarityFlow8">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>98</randomInitalisationValue>
  </scenario>
  <scenario name="SimilarityFlow9">
    <peersSet name="good">
      <tokenDistributor AgeWeighted="true" Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
    <randomInitalisationValue>37</randomInitalisationValue>
  </scenario>
</testSeries>
```

A.5.3. Testsimulationen

A.5.3.1. Empty-Trustability-Test

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="EmptyTrustabilityTest">
  <numberOfRounds>20000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    20
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <numberOfPeers>300</numberOfPeers>
  <numberOfClusters>3</numberOfClusters>
  <meanDownStream>300</meanDownStream>
  <clusterTau>2.00</clusterTau>

  <scenario name="EmptySimple">
```

```
    <providerReputationCalculator
      TrustabilityCalculator="SimpleTrustabilityEmptyTest"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector
      TrustabilityCalculator="SimpleTrustabilityEmptyTest"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />

    <peersSet name="good">
      <tokenDistributor Selfish="true">
        EmptyTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>


  <scenario name="EmptyFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustabilityEmptyTest"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector
      TrustabilityCalculator="FlowTrustabilityEmptyTest"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />

    <peersSet name="good">
      <tokenDistributor Selfish="true">
        EmptyTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>
</testSeries>
```

A.5.3.2. Token-Distribution-Test

```
<?xml version="1.0" encoding="UTF-8"?>
<testSeries name="TotalDistributionTest">
  <numberOfRounds>5000</numberOfRounds>
  <numberOfDistributedTokensPerTransaction>
    98
  </numberOfDistributedTokensPerTransaction>
  <maxRange>10</maxRange>
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <providerMarketGenerator>
    ClusterOrientatedAndRangeWeightedProviderMarketGenerator
  </providerMarketGenerator>
  <transactionValidator>
    DefaultTransactionValidator
  </transactionValidator>
  <tokenValueCalculator restricted="true">
    DefaultTokenValueCalculator
  </tokenValueCalculator>
  <similarityCalculator>DefaultMetric</similarityCalculator>
  <numberOfPeers>100</numberOfPeers>
```

```
<numberOfClusters>3</numberOfClusters>
<providerSelector>ProbabilisticProviderSelector</providerSelector>
<meanDownStream>300</meanDownStream>
<clusterTau>2.00</clusterTau>

<scenario name="TotalNeighborhoodSimple">
  <providerReputationCalculator
    TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <tokenDistributor Selfish="true">
      NeighborhoodTokenDistributor
    </tokenDistributor>
  </peersSet>
</scenario>

<scenario name="TotalSimilaritySimple">
  <providerReputationCalculator
    TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <tokenDistributor Selfish="true">
      SimilarityTokenDistributor
    </tokenDistributor>
  </peersSet>
</scenario>

<scenario name="TotalRandomSimple">
  <providerReputationCalculator
    TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="SimpleTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <tokenDistributor Selfish="true">
      RandomTokenDistributor
    </tokenDistributor>
  </peersSet>
</scenario>

<scenario name="TotalNeighborhoodFlow">
  <providerReputationCalculator
    TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <clientSelector TrustabilityCalculator="FlowTrustability"
    PlausibilityCalculator="SimilarityPlausibilityCalculator" />
  <peersSet name="good">
    <tokenDistributor Selfish="true">
      NeighborhoodTokenDistributor
    </tokenDistributor>
  </peersSet>
</scenario>


```

```
  <scenario name="TotalSimilarityFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor Selfish="true">
        SimilarityTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

  <scenario name="TotalRandomFlow">
    <providerReputationCalculator
      TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <clientSelector TrustabilityCalculator="FlowTrustability"
      PlausibilityCalculator="SimilarityPlausibilityCalculator" />
    <peersSet name="good">
      <tokenDistributor Selfish="true">
        RandomTokenDistributor
      </tokenDistributor>
    </peersSet>
  </scenario>

</testSeries>
```

Literaturverzeichnis

[1] Adriana Iamnitchi, Matei Ripeanu, and Ian T. Foster. Small-world file-sharing communities. *CoRR*, cs.DC/0307036, 2003. informal publication.

[2] Stefan Saroiu, P. Krishna Gummadi, , and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking Conference*, 1 2002.

[3] Matteo Dell'Amico. Neighbourhood maps: decentralised ranking in small-world p2p networks. In *IPDPS*. IEEE, 2006.

[4] Qiao Lian, Yu Peng, Mao Yang, Zheng Zhang, Yafei Dai, and Xiaoming Li. Robust incentives via multi-level tit-for-tat. *Concurrency and Computation: Practice and Experience*, 20(2):167–178, 2008.

[5] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, computer science department, Stanford University, 2002.

[6] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[7] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small world' networks. *Nature*, 393, 1998.

[8] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small world' networks. *Nature*, 393, 1998.

[9] Theodore Hong. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter Performance, pages 203–241. O'Reilly, 1999.

[10] Lada Adamic and Bernardo Huberman. Zipf´s law and the internet. *Glottometrics*, 3:143–150, 2002.

[11] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[12] Yoram Kulbak and Danny Bickson. The emule protocol specifikation, January 20, 2005.

[13] Bram Cohen. Incentives build robustness in bittorrent, 2003.

[14] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. Searching the web. *ACM Trans. Internet Techn.*, 1(1):2–43, 2001.

[15] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.

[16] Ivan Martinovic, Christof Leng, Frank A. Zdarsky, Andreas Mauthe, Ralf Steinmetz, and Jens B. Schmitt. Self-Protection in P2P networks: Choosing the right neighbourhood. In *Proceedings of the International Workshop on Self-Organizing Systems (IWSOS 2006), Passau, Germany*. Springer LNCS, September 2006.

[17] Elias C. Efstathiou, Pantelis A. Frangoudis, and George C. Polyzos. Stimulating participation in wireless community networks. In *INFOCOM*. IEEE, 2006.

[18] Giancarlo Ruffo and Rossano Schifanella. Evaluating peer-to-peer recommender systems that exploit spontaneous affinities. In Yookun Cho, Roger L. Wainwright, Hisham Haddad, Sung Y. Shin, and Yong Wan Koo, editors, *SAC*, pages 1574–1578. ACM, 2007.

[19] Johannes Buchmann. *Einführung in die Kryptographie*. Springer, Berlin, 2003.

[20] Steven John Metsker. *Design patterns Java workbook*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[21] Andreas Schlosser and Marco Voss. Simulating data dissemination techniques for local reputation systems. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1173–1174, New York, NY, USA, 2005. ACM.

[22] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[23] N. Boccara. *Modeling Complex Systems*. Springer-Verlag, 2003.

[24] Christof Leng. Konzeption und Implementierung eines Trust-Protokolls in einem Peer-to-Peer-Netzwerk, 2004.

[25] Andreas Schlosser. Simulation von Reputationsberechnungsverfahren in globalen und lokalen Reputations-Systemen, 2004.