Anonymität und Sicherheit in Peer-to-Peer-Systemen

Wolfgang Müller

wolfgang.mueller@wiai.uni-bamberg.de

LS Medieninformatik, Otto-Friedrich-Universität Bamberg

27. Dezember 2005

Peer-to-Peer-(P2P)-Netze ermöglichen unabhängig voneinander administrierten, gleichberechtigten Rechnern, große Aufgaben verteilt gemeinsam zu bearbeiten. Ihre Popularität verdanken P2P-Netze dem Umstand, dass ein Peer sehr einfach zu administrieren ist. Eine wichtige Anwendung von P2P-Netzen sind verteilte Datenspeicher, in denen die Peers ihren Plattenplatz zur Verfügung stellen, und das P2P-System Such- und Zugriffsfunktionalität bereit stellt.

Während im WWW eine Linkstruktur zur Verfügung steht, die die im Netz gespeicherten Dokumente miteinander verbindet, existiert eine derartige Struktur in P2P-Netzen üblicherweise nicht. Um die im P2P-Netz vorliegenden Daten nutzen zu können, wird also eine effiziente, mächtige Suchfunktionalität benötigt. Diese Suchfunktionalität bereitzustellen, ist ein aktuelles Forschungsgebiet. Die Herausforderung ist hierbei insbesondere, dass die Daten verteilt vorliegen und die beteiligten Rechner (Peers) nicht ausfallsicher sind.

Forschungsarbeiten über Suche in P2P-Netzen betrachten meist die eigentliche Effizienz der Algorithmen. Sicherheitsbetrachtungen zu den Algorithmen werden jedoch eher selten angestellt. Dies ist problematisch, da sich mit der Verteilung und dem verteilten Zugriff auf Ressourcen auch neue Angriffsmöglichkeiten eröffnen.

Der vorliegende Artikel gibt eine Einführung in die Herausforderungen, die sich in den Gebieten Anonymität und Sicherheit bei Entwurf und Erstellung von P2P-Systemen ergeben.

1 Einführung

Das Aufkommen des File-Sharing-Werkzeugs Napster hat einerseits Veränderungen im Copyright ins Rollen gebracht und andererseits war Napster eine überzeugende Demonstration des Potentials verteilter Anwendungen, deren Erfolg den heutigen Boom bei verteilten Anwendungen und deren Erforschung nach sich gezogen hat.[1]

[1] [19] enthält eine journalistische, weniger technische Abhandlung über die Geschichte von Napster.

Napster war die Kombination einer zentralen Index-Komponente mit einem verteilten Datenspeicher. Die zentrale Index-Komponente indexierte effizient *Dateinamen* der verteilt gespeicherten Musikdateien. Mittels einfacher Suchmöglichkeiten auf Dateinamen konnten Napster-Nutzer verteilt vorliegende Dateien in dem zentralen Index suchen. Gefundene Dateien wurden dann direkt (also Peer-to-Peer) ohne Zutun des zentralen Napster-Servers bei dem Anbieter der Datei heruntergeladen. Da auf diese Weise die Bandbreiten-intensive Verteilung der Daten nicht von Napster, sondern von den Napster-Teilnehmern, den Peers, vorgenommen wurde, konnte Napster mit sehr viel weniger Aufwand betrieben werden als eine vergleichbare reine Client/Server-Anwendung benötigt hätte. Gleichzeitig zeigte Napster, dass es möglich ist, unter Verwendung von unzuverlässigen Komponenten einen attraktiven Dienst aufzubauen.

Die Suche auf verteilt vorliegenden Daten ist hingegen ein komplexes, forderndes Problem, das in Napster mittels des zentralen Indexes elegant umgangen wurde. Jedoch wurde im Verlaufe der juristischen Auseinandersetzung zwischen Napster und seinen Gegnern deutlich, dass ein so genannter *Single Point of Failure* Nachteile hat. Erstens konnte durch Abschalten des Napster-Servers (bzw. der Napster Server-Farm) das gesamte Napster-Netz ausgeschaltet werden (die Peers waren ohne Suchfunktion sinnlos), zweitens (und aus Sicht der Sicherung der Privatsphäre schwerwiegender) konnten an einem zentralen Punkt — eben wieder dem Napster-Server — Nutzerdaten entnommen werden.

Napster hat aufgrund seiner Stärken, aber auch aufgrund seiner Schwächen, die Forschungsgebiete des P2P entscheidend motiviert. Viele Arbeitsgruppen beschäftigen sich damit, Peer-to-Peer-Netze zu entwerfen, die komplexe Dienste ohne Verwendung zentraler Komponenten anbieten.

Diese P2P-Netze sind dadurch gekennzeichnet, dass ihre Komponenten sehr unzuverlässig sind. Typische File-Sharing Szenarien gehen davon aus, dass von 100 Peers, die in einem Netzwerk zu einem gegebenen Zeitpunkt online waren, nach einer Stunde ca. 50 Peers das Netzwerk wieder verlassen haben. Peers treten üblicherweise einige wenige Male pro Tag einem Netz bei. Peers sind für das Netz also größenordnungsmäßig nur 10% der Zeit verfügbar, also um sehr viel weniger als übliche Server. Hieraus folgt, dass sich in P2P-Netzen ständig neue Peers mit dem Netz verbinden, und ständig Peers aus dem Netz ausscheiden, zumeist ohne Ankündigung. Es ist offensichtlich, dass es unter diesen Umständen eine Herausforderung ist, eine gute Dienstqualität zu bieten. Ferner ist deutlich, dass es schwer möglich ist, Angreifern den Zutritt zum Netz zu verwehren.

Eine große Gruppe von Wissenschaftlern befasst sich mit verschiedenen Arten von Suche in P2P-Netzen, die alle in ihrer Mächtigkeit weit über das von Napster Gebotene hinausgehen. Eine Basistechnologie hierfür sind verteilte Hashtabellen (*Distributed Hash Tables*, DHTs, siehe z. B. [26, 21, 22]), die die effiziente Verwaltung von Schlüssel/Wert-Paaren erlauben. Sie erlauben jedoch meist nur die exakte Suche nach Schlüsseln. Die verwendeten Schlüssel tragen üblicherweise keine Semantik. Nähe im Schlüsselraum bedeutet in DHTs also keine semantische Nähe. Die verteilte Verarbeitung von Datenbankabfragen setzt üblicherweise DHTs ein [14, 15].

Verschiedene Ansätze gibt es beim P2P-Information-Retrieval (P2P-IR), also der P2P-Ähnlichkeitssuche auf Texten: Der in [28] vorgestellte Ansatz basiert auf DHTs, die in [3, 20] geschilderten Ansätze folgen den Ideen des klassischen verteilten Information Retrieval (Zusammenfassungen der Daten in den Peers werden innerhalb des Netzes verteilt und zur Auswahl von Datenquellen verwendet), weitere Ansätze versuchen, in unstrukturierten Netzen die Linkstruktur schrittweise zu verbessern. Mit all die-

sen Techniken ist schlüsselwortbasierte Suche ähnlich wie bei Internet-Suchmaschinen möglich.

Es fällt jedoch auf, dass ein Großteil der heutzutage im Bereich P2P-Datenhaltung publizierten Arbeiten die Sicherheit der verwendeten Verfahren außer Acht lässt. Die weit verbreitete Meinung ist, dass die Forschung sich zunächst damit beschäftigen sollte, unter Annahme von größtenteils gutwilligen Peers brauchbare Verfahren zu erarbeiten und aus diesen dann unter anwendungsspezifischen Sicherheitsaspekten auszuwählen.

Der hier vorliegende Artikel verfolgt eine andere Zielsetzung: Es sollen für die Datenhaltung in P2P-Netzwerken relevante Aspekte der Anonymität und Sicherheit herausgearbeitet werden. Hierbei wird deutlich, wie leicht angreifbar P2P-Netze sind, und welche Abwehrmöglichkeiten existieren. P2P-Netze sind einfach angreifbar, weil jeder Teilnehmer eines Netzes ja gleichzeitig einen Dienst zur Verfügung stellt und Dienste nachfragt, und somit keine klare Grenze zwischen vertrauenswürdigen, zu sichernden Anbietern/Servern und potentiell angreifenden Nachfragern/Clients gezogen werden kann.

Im Einzelnen werden die folgenden Themen behandelt:

- *Anonymität:* Dingledine *et al.* [8] geben eine Einteilung, in der verschiedene Arten der Anonymität, die im Zusammenhang mit der Publikation von Daten wichtig sind, benannt und beschrieben werden. Diese Einteilung wird zunächst geschildert, dann werden zwei Arten von Anonymität gesondert und tiefer behandelt. Hier werden insbesondere die so genannte *Publisher/Reader*-Anonymität und die Anonymität von Anfragen behandelt.
- *Sicherheit:* Hier werden im Wesentlichen die folgenden Aspekte betrachtet:
  - *Sicheres Routing:* Dabei geht es darum, DHTs gegen Angriffe auf die *Suchqualität* zu immunisieren.
  - *Reputationssysteme :* Reputationssysteme bewerten die Qualität der Mitarbeit einzelner Peers und machen diese Information anderen Peers zugänglich. Eine Möglichkeit zur Nutzung dieser Information ist z.B., die Interaktion mit Peers geringer Reputation so weit wie möglich zu meiden. In unserer Beschreibung von P2P-Reputationssystemen wird insbesondere auch auf Schwierigkeiten von Reputationssystemen verwiesen, die im Zusammenhang mit der Ähnlichkeitssuche auftreten können.

2 Anonymität

Im täglichen Leben geht man natürlich davon aus, dass ein Großteil der Kommunikation weitestgehend anonym verläuft. Geht man z. B. in ein Geschäft, um sich über ein Produkt zu erkundigen, wird man nur in seltenen Fällen seinen Namen und seine Adresse hinterlassen. Ebenso wird man bei den meisten Gesprächen auf offener Straße davon ausgehen, dass für die überwiegende Mehrheit der an der Kommunikation nicht Beteiligten die Kommunikation anonym abläuft.

Im Internet wird zwar häufig von den Nutzern so verfahren, als sei die Kommunikation anonym, es ist jedoch bekannt, dass vielerorten Internet-Adressen gespeichert werden und Nutzer entgegen ihrem Empfinden, anonym zu sein, sehr schnell und einfach ausfindig gemacht werden können. Die Spur zu verschleiern, die jeder im Internet hinterlässt, ist ein Ziel der Systeme, die in diesem Abschnitt vorgestellt werden.

Im Kontext der Suche in P2P-Netzen wird das Thema Anonymität dadurch dringender, dass jede Anfrage, die an das Netz gestellt wird, an mehrere Peers weitergeleitet wird. Deren Vertrauenswürdigkeit ist in den allermeisten P2P-Protokollen ungeprüft. Es ist also für angreifende Peers recht einfach, sich in ein Netz einzunisten, Teile des Datenverkehrs mitzuschneiden und eventuell Vorteile daraus zu ziehen.

Nun werden zunächst der Begriff der Anonymität näher definiert und Angriffe auf die Anonymität beschrieben. Schließlich werden Systeme vorgestellt, die Anonymität implementieren. Diese Systeme nutzen die Tatsache, dass in P2P-Systemen Anfragen über mehrere Ebenen weitergeleitet werden, bevor sie abschließend bearbeitet werden können.

2.1 Arten von Anonymität:

In [8] diskutieren Dingledine *et al.* Free Haven, ein System zur anonymen Publikation. Bevor sie die Eigenschaften von Free Haven näher beschreiben, gehen sie zunächst allgemein auf verschiedene Facetten von Anonymität ein:

1. *Publisher Anonymity/Anonymität des Veröffentlichers:* Ein System ermöglicht Anonymität des Veröffentlichers, falls ein eventueller Angreifer daran gehindert wird, das Dokument seinem Veröffentlicher zuzuordnen. Somit kann also kein Netzteilnehmer an der freien Meinungsäußerung gehindert werden.

2. *Reader Anonymity/Anonymität des Lesers:* Ein System bietet Anonymität der Leser, falls es Angreifern unmöglich gemacht wird, Dokumente ihren Lesern zuzuordnen. Somit wird die Lektüre unliebsamer Meinungen ermöglicht. Ferner ist es unmöglich, von einzelnen Nutzern ohne deren Zustimmung und Mitarbeit Zugriffsprofile zu erstellen.

3. *Server Anonymity/Server-Anonymität:* Für ein gegebenes Dokument ist es dem Angreifer unmöglich herauszufinden, auf welchen Servern es gespeichert ist. Somit ist es dem Angreifer nicht möglich, für ein gegebenes Dokument zunächst die Server ausfindig zu machen, und diese dann vom Netz zu nehmen.

4. *Document Anonymity/Dokument-Anonymität:* Hier weiß der Server nicht, welche Dokumente er speichert. Somit hat er die Wahl, entweder am Netz teilzunehmen (und so zu riskieren, dass er Daten hostet, mit deren Inhalten er nicht einverstanden ist), oder die Mitarbeit am Netz gänzlich zu vermeiden.

5. *Query Anonymity/Anfrage-Anonymität:* Hier weiß der Server nicht, welche Anfrage er gerade bearbeitet. Es ist ihm somit nicht möglich, unliebsame Anfragen zu filtern.

2.2 Angriffe auf Anonymität

Freedman und Morris, die Autoren von Tarzan [11], sehen die folgenden Angriffsszenarien auf Anonymität:

- Einzelne oder Gruppen, die aus reiner Neugier Datenverkehr mitschneiden.
- Kriminelle, die in fremde Maschinen einbrechen, um mit einer großen Gruppe von Rechnern Datenverkehr abzuhören.

- Große Organisationen, die den Datenverkehr der Internet Backbones sowie großer Provider abhören.

Diesen Angriffen ist nicht mithilfe einzelner Komponenten zu begegnen. Ein möglicher Ansatz mit nur einer Komponente Kommunikation zu anonymisieren wäre die Einrichtung eines Proxy Servers, der ankommende Pakete so modifiziert und weiterleitet, dass es für Außenstehende so aussieht, als wäre der Proxy die Quelle der Requests. Hält der Proxy seine Kommunikation geheim, so bietet dies ein gewisses Maß an Anonymität. Wenn man jedoch versucht, mittels eines einzelnen Proxy Servers Verbindungen zu anonymisieren, vertraut man *erstens* darauf, dass diese eine Komponente die Anonymität respektiert und den Datenverkehr nicht abhört. *Zweitens* wird angenommen, dass niemand auf der IP-Route zu und vom Proxy diesen Datenverkehr abhört und ankommende/ausgehende Pakete miteinander korreliert.

P2P-Netze bestehen per Definition aus sehr vielen Komponenten. Alle Peers in einem P2P-Netzwerk effektiv zu überwachen ist also sehr viel schwieriger als bei einem einzelnen Proxy. Die Peers sind jedoch nicht vertrauenswürdig. Die Herausforderung bei der Erstellung von Netzwerken zur anonymen Kommunikation und Publikation ist nun der Entwurf von Protokollen, die eine gewisse Anzahl von Angreifern im P2P-Netz verkraften, ohne die Anonymität preiszugeben.

Die im Folgenden kurz beschriebenen P2P-Systeme implementieren jeweils eine Teilmenge der oben geschilderten Arten von Anonymität. Tarzan ist eine P2P-Lösung zur Herstellung von *Reader Anonymity*, Freenet ist ein System zur Anonymen Publikation, dessen Ziel es ist, *Publisher* und *Reader Anonymity* herzustellen.

2.3 Tarzan

Tarzan [11] ist ein P2P-System zur Anonymisierung von IP-Paketen. Es ist also nicht nur innerhalb von P2P-Netzen, sondern für jegliche Kommunikation einsetzbar. Im Folgenden wird jedoch nur die Kommunikation zwischen Peers betrachtet. Jeder Peer in einem Tarzan-Netz kann in einer Kommunikation Datenquelle, Datensenke oder anonymisierender Peer sein.

Tarzan versucht, auf unter Ebene *Sender/Recipient Anonymity* herzustellen. Mit anderen Worten soll für eine gegebenes Paket nicht herausgefunden werden können, wer der Absender und wer der Empfänger ist. Auf Basis von *Sender/Recipient Anonymity* lässt sich direkt *Reader Anonymity* realisieren: Der Leser wird auf jeden Dienst, und insbesondere auch auf die innerhalb des P2P-Systems publizierten Dokumente des P2P-Netzes mittels Tarzan zugreifen. Jedoch muss hierzu erst einmal der Ort, an dem das Dokument zu finden ist, bekannt sein, *Publisher Anonymity* ist also ohne weiteres Zutun hier nicht gegeben.

Tarzan erreicht das Ziel der *Publisher Anonymity* dadurch, dass ähnlich wie in *Onion Routing* [27] jedes Paket sowie sein Adressat bei jedem Hop[2] verschlüsselt beziehungsweise entschlüsselt werden. Will Peer *Alice* Peer *Dave* ein Paket schicken, so findet *Alice* zunächst eine Route heraus, die von *Alice* zu *Dave* führt. Auf dieser Route wird ein so genannter Tunnel aufgebaut. Nehmen wir an, die Peers entlang dieses Tunnels seien *Bob* und *Charlie*. Beim Aufbau des Tunnels werden *Flow Tags* vergeben. Beispielsweise wird die Verbindung zwischen *Alice* und *Bob* mit dem Tag 21, zwischen *Bob* und *Charlie* mit 33 und die zwischen *Charlie* und *Dave* mit dem Tag 12 versehen.

[2] Ein *Hop* (Hüpfer) bezeichnet die Weitergabe einer Botschaft über eine Stufe.

Bob weiß nun, dass ein Paket, das bei ihm mit Tag 21 ankommt, verarbeitet, und dann mit dem Tag 33 versehen an *Charlie* weitergeleitet werden muss. Ebenso weiss *Charlie* dass ein Paket, das mit dem Tag 33 bei ihm ankommt, verarbeitet, und mit dem Tag 12 versehen an *Dave* weitergeleitet werden muss. Der Aufbau des Tunnels erfolgt so, dass *Bob* und *Charlie* nur jeweils ihre direkten Nachbarn im Tunnel (und die entsprechenden *Flow Tags*) kennen, jedoch weiß *Bob* nicht von *Dave*, und *Charlie* nicht von *Alice*.

Beim Aufbau des Tunnels wird *Alice* die Session-Schlüssel von *Bob*, *Charlie* und *Dave* erzeugen und an sie weiterleiten. So kennt *Alice* alle Session-Schlüssel, und *Bob*, *Charlie* und *Dave* kennen jeweils ihre eigenen Schlüssel.

Vor dem Versand wird nun *Alice* das eigentliche zu versendende Paket "schichtweise" verschlüsseln, wie in Abb. 1 zu sehen und im Folgenden beschrieben. Zunächst wird das eigentliche Paket (im Folgenden p genannt), mit *D*aves Schlüssel k_D verschlüsselt. Wir schreiben $\text{enc}(k_D, p)$ für das resultierende Paket. $\text{enc}(k_D, p)$ wird von einer weiteren Schicht umgeben, nämlich wird sie für *Charlie* verschlüsselt, unter Verwendung dessen Schlüssels: $\text{enc}(k_C, \text{enc}(k_D, p))$. Ebenso wird mit *Bob*s Schlüssel verfahren: $\text{enc}(k_B, \text{enc}(k_C, \text{enc}(k_D, p)))$. Dies ist schließlich die Botschaft, die *Alice* an *Bob* verschickt, erweitert mit dem Flow Tag 21. Bei Empfang wird *Bob* eine Schicht Verschlüsselung entfernen, also $\text{enc}(k_C, \text{enc}(k_D, p))$ berechnen und unter Verwendung des Flow Tags 33 an *Charlie* weiterreichen. Dieser entschlüsselt die Botschaft, erhält $\text{enc}(k_B, p)$ und leitet diese Botschaft mit Flow Tag 12 an *Dave* weiter. Dieser erhält durch Entschlüsselung schließlich p.

Auf dem Rückweg wird die Antwort p' wieder mehrfach verschlüsselt. *Dave* gibt p' an *Charlie* zurück, jedoch erst nach Verschlüsselung: $\text{enc}(k_D, p')$. *Charlie* verfährt ähnlich und sendet folgendes an *Bob*: $\text{enc}(k_C, \text{enc}(k_D, p'))$. *Bob* schließlich versendet $\text{enc}(k_B, \text{enc}(k_C, \text{enc}(k_D, p')))$ an *Alice*. *Alice* muss nun $\text{enc}(k_B, \text{enc}(k_C, \text{enc}(k_D, p')))$ in drei Schritten, unter Verwendung von k_B, dann k_C, dann k_D entschlüsseln, um p' zu erhalten.

Durch die schrittweise Verschlüsselung kennt keiner außer *Alice* und *Dave* p bzw. p'. Die Verwendung von Verschlüsselung und *Flow Tags* stellt sicher, dass *Bob* nicht erfährt, dass *Dave* der Adressat von p' ist. Ebenso weiß *Charlie* nicht, dass *Alice* der Ausgangspunkt der Botschaft war. Denial-of-Service Angriffe gegen Tarzan werden dadurch schwierig, dass *Alice* sowohl das ausgehende als auch das eingehende Paket je dreimal ver- beziehungsweise entschlüsseln muss. Die inneren Peers des Tunnels, *Bob* und *Charlie*, sowie der Adressat *Dave* müssen die Pakete p und p' jeweils nur einmal ent- beziehungsweise verschlüsseln. Beim Versuch *Bob*, *Charlie* oder *Dave* mittels anonymisierter Pakete zu überlasten, wird also eher *Alice* selbst überlastet.

Das in Tarzan verwandte Verfahren wird wegen der aus Verschlüsselungs-Schichten aufgebauten Verpackung des zu versendenden Pakets *Onion Routing* genannt. Tarzans Beitrag hierzu ist die Portierung des Verfahrens auf P2P-Systeme, ein Algorithmus zum effizienten Aufbau von Tunneln, sowie die Kombination mit Mechanismen, die Tarzan-Peers die Entdeckung anderer Tarzan-Peers ermöglichen. Gleichzeitig wird es trotz des flexiblen Aufbaus von Tunnels ermöglicht, dass Peers zur Verschleierung des Nutzdatenverkehrs so genannten *Cover Traffic* austauschen, der es unmöglich macht, herauszufinden, wann Peers Nutzdaten versenden. Ohne Cover Traffic ist es mächtigen Angreifern möglich, über die zeitliche Korrelation des Datenverkehrs innerhalb des Netzes darauf zu schließen, welche Peers an welchen Tunnels teilnehmen.

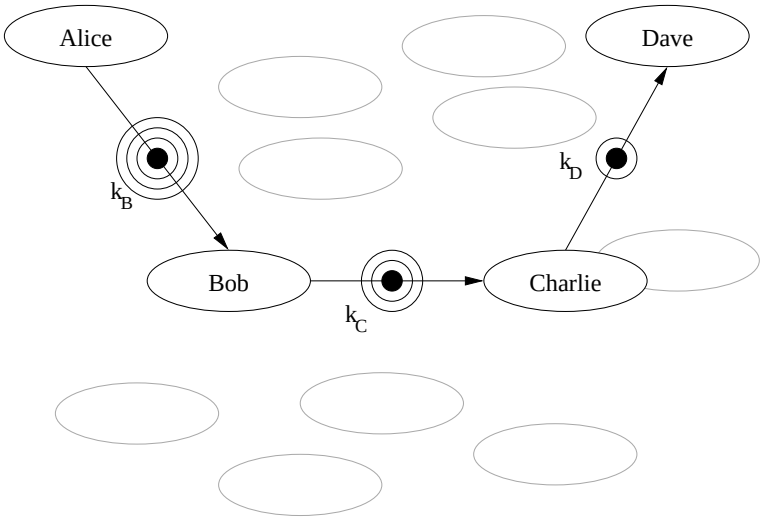

Abbildung 1 — Versand einer Botschaft mittels Tarzan von *Alice* zu *Dave*.

2.4 Freenet

Freenet [6] hat zum Ziel, ein unzensierbarer, sicherer, globaler Informationsspeicher zu sein. Die Unzensierbarkeit folgt aus der Anonymität, die Freenet zur verfügung stellt. Freenet speichert Daten als BLOBs (Binary Large OBjects), auf die mithilfe eines ganzzahligen Schlüssels zugegriffen werden kann. Diese Speicherung wird nicht zugesichert, sondern Freenet verhält sich wie ein Cache. Freenet ist thematisch fokussierter als Tarzan (Tarzan ist eine Anonymisierungsschicht für beliebige Anwendungen) und hat gleichzeitig sehr viel breitere Zielsetzungen bezüglich der gewünschten Anonymität.

In [6] wird Freenet dicht beschrieben und ausführlich motiviert. Freenet bietet *Publisher/Reader Anonymity* und in einem gewissen Maße auch *Server Anonymity* und *Document Anonymity*. *Query Anonymity* ist nur insoweit gegeben, als ein Server keine Kenntnis der *Semantik* einer Anfrage hat.

Wie eine DHT speichert Freenet Dokumente als Schlüssel/Wert-Paare und ermöglicht die Suche nach ganzzahligen Schlüsseln. Wie in einer DHT werden Peers durch Schlüssel identifiziert. Im Unterschied zu einer DHT gibt es jedoch in Freenet keine im Vorhinein bekannte Zuordnung von Dokumentschlüsseln zu Peerschlüsseln. Die (mögliche) Spezialisierung von Peers auf einen Teil des Schlüsselraums findet erst bei der Benutzung des Peers und in Abhängigkeit von den gestellten Anfragen statt.

Freenet-Peers agieren hierbei als *Cache*. Die langfristige Publikation von Dokumenten wird nicht zugesichert. Unpopuläre, d.h. selten nachgefragte Dokumente verschwinden aus dem Cache und müssen eventuell neu publiziert werden. Populäre Dokumente werden jedoch unabhängig von ihrem Inhalt auf viele Knoten im Netz verteilt.

Es gibt zwei Arten von global eindeutigen Dokumentschlüsseln (GUIDs) in Freenet, *Content-Hash Keys* (CHKs) und *Signed-Subspace Keys* (SSKs).

CHKs werden durch Berechnung eines SHA-1 Hashes [24] über den Inhalt des Dokumentes erzeugt. Die SHA-1 Hashfunktion setzt beliebige Bitfolgen in eine 160 Bit lange Zahl um. [3] Dass zwei Dokumente zufällig denselben Hashwert haben ist also extrem

[3] 2^{160}, also die Zahl der möglichen Hashwerte, entspricht grob gesagt ungefähr der Zahl der

unwahrscheinlich. Die Erstellung eines Dokumentes, dessen Inhalt einen gegebenen Hashwert hat, ist außerdem schwierig. Der Hash kann also für die meisten Zwecke zur Identifikation des Dokumentinhalts verwendet werden. CHKs tragen jedoch keinerlei Semantik. Um den CHK eines Dokumentes zu erfahren, müsste man das Dokument schon im eigenen Besitz haben, um wieder den CHK berechnen zu können. Um bei nur teilweiser Kenntnis eines Dokuments dieses auch finden zu können, gibt es die SSKs.

Signed Subspace Keys ermöglichen es, Dateien zu benennen, so dass jeder die entsprechend benannte Datei lesen, aber nur der Autor des SSKs sie ändern kann. Um einen SSK zu erstellen, benötigt der Autor zunächst ein zufälliges public/private Schlüsselpaar, das zur Identifikation des Subspaces benötigt wird. Zusätzlich benötigt er einen Namen für jede Datei. Dieser Name sollte für Menschen lesbar sein und Semantik tragen. Clarke *et al.* [6] wählen `politics/us/pentagon-papers` als Beispiel. Der SSK wird dann dadurch berechnet, dass sowohl die beschreibende Zeichenkette, als auch der öffentliche Schlüssel mit SHA-1 gehasht werden. Die resultierenden Hashes werden konkateniert, und die sich hieraus ergebende Bitfolge wird dann nochmals gehasht und mittels des eben erzeugten privaten Schlüssels signiert. Der Signierte SSK wird dann verwendet, wenn das Dokument *eingefügt wird*. Für diese Signatur benötigt man den *private Key*. Um nun ein Dokument des Subspaces zu *lesen*, benötigt der Leser nur den für Menschen lesbaren Namen, sowie den *public Key* des Subspaces.

SSKs werden hauptsächlich verwendet, um Dokumente mit Listen von CHKs in Freenet einzustellen. In der Sicht von Clarke und Kollegen entspricht der SSK dem Dateinamen in dem Dateisystem einer Festplatte und ein CHK/Dokument-Paar dem einzelnen Block auf der Platte: Dateien sind auf für Menschen lesbare Art benannt, Blöcke nicht.

Ähnlich wie in Tarzan wird nun Anonymität dadurch hergestellt, dass Anfragen und Antworten jeweils über mehrere Peers weitergegeben werden und jeder Schritt der Kommunikation selbst verschlüsselt wird. Auch hier weiß jeder Knoten, der eine Botschaft weitergibt, nur den Nachbarn, von dem er die Botschaft erhalten hat, und den Nachbarn, der die Botschaft erhält. Kein einziger Knoten, der eine Botschaft weiterleitet, kann sagen, wer Anfangs- und wer Endpunkt der Botschaft ist. Kein Knoten kann sagen, welcher Knoten eine Anfrage erfolgreich beantwortet hat. Im Gegensatz zu Onion Routing jedoch wird die Botschaft nicht vorher mehrfach verschlüsselt, und dann auf dem Pfad wieder schrittweise entschlüsselt, denn jeder Knoten soll ja die Anfrage erfahren und gegebenenfalls auf sie reagieren.

Zum Auffinden von Daten nutzt Freenet die Small-World-Eigenschaften [1] seines Netz-Graphen: Da Knoten, die viele Nachbarn haben, mit höherer Wahrscheinlichkeit neue Nachbarn bekommen (*Preferential Attachment*), ist Freenet ein zufälliger Graph aus Knoten (Peers) und Kanten (Verbindungen), in dem die Eingangs- und Ausgangsgrade der Knoten Zipf-verteilt sind[4]. Derartige Graphen haben die so genannte Small-World Eigenschaft, die besagt, dass die kürzesten Pfade zwischen zwei beliebigen Knoten nur logarithmisch mit der Netzgröße wachsen[5]. Es muss einem nur gelingen, die jeweils besten Pfade zwischen zwei Knoten auszuwählen. Zur Suche nutzt Freenet ein Gradientenaufstiegs-Verfahren: Jeder Knoten schickt eine Anfrage an jeweils denjenigen Knoten weiter, der dem Ziel am nächsten ist. Als Distanzmaß dient hierzu die

Wasserstoff-Atome, die in den Erdball passen. Mit Zahlen im irdischen Maßstab Hash-Kollisionen zu bekommen ist also extrem unwahrscheinlich. Man kann somit davon ausgehen, dass zwei unabhängig voneinander berechnete Bitfolgen unterschiedliche Hashwerte haben.

[4] Dies bedeutet, dass die Wahrscheinlichkeit, einen Peer mit x Nachbarn anzutreffen, $\frac{1}{x}^{\alpha}$ ist. α liegt hierbei in realen P2P-Netzwerken nahe 2.

[5] Eine Einführung in die Statistische Mechanik komplexer Netzwerke, die dieses und andere Phänomene behandelt, findet sich in [1].

einfache Differenz zwischen Hashwerten. Hierzu hält jeder Freenet-Knoten als Routingtabelle eine Liste von anderen Peers, und diesen zugeordnet jeweils eine Liste von GUIDs, die diese Peers wahrscheinlich enthalten.

Erhält nun ein Peer die Anfrage nach einer GUID, so wird er zunächst prüfen, ob er selbst die der GUID zugehörige Datei gespeichert hat. Wenn ja, wird er die Datei zurücksenden zusammen mit einem *Tag*, das ihn als Datenquelle für diese GUID ausweist. Falls er jedoch diese Datei nicht enthält, so wird er die Anfrage nach der GUID unter Verschleierung des Knotens, von dem er die Anfrage erhalten hat, weitergeben. Der Knoten, an den die Nachricht weitergereicht wird, wird danach ausgewählt, ob er ähnliche GUIDs enthält. Ist diese weitergeleitete Anfrage erfolgreich, so wird die Datei von der Datenquelle aus schrittweise an alle Knoten weitergereicht, die auf dem Weg von der Datenquelle zum Anfragesteller liegen. Knoten, die auf diesem Weg liegen, können sich entscheiden, eine Kopie dieser Datei lokal zu cachen. Ferner können Knoten die Angabe der Datenquelle in der weiterzuleitenden Botschaft verändern und sich selbst als Datenquelle eintragen. Da diese Knoten selbst wissen, wer die ursprüngliche Datenquelle war, wird die Erreichbarkeit der Daten hierdurch nicht beeinträchtigt.

Die Anfrage schlägt fehl, wenn die Zahl der Hops einen vorher gesetzten Maximalwert, die *Time To Live* (TTL), übersteigt. Da man über die TTL und die Zahl der Hops einen Rückschluss auf den Ursprung der Anfrage ziehen könnte, ermöglicht Freenet auch, mittels eines Anonymisierungsschritts, der ähnlich wie Tarzan funktioniert, den Ursprung der Anfrage weiter zu verschleiern.

Die Routingtabelle der einzelnen Knoten enthält Information, die durch die Interaktion mit dem Netz gewonnen wird. Beim Weiterleiten von Dokumenten wird jeweils ihre Datenquelle notiert. Ferner wird die Routingtabelle nie an Nachbarn weitergegeben. Somit besteht auch nicht die Möglichkeit für Angreifer, die Routingtabelle von Nachbarn durch gezielte Weitergabe falscher Routingtabellen zu zerstören.

Das Einfügen von Dokumenten geschieht wie in den meisten Datenstrukturen: das neue Dokument wird dahin platziert, wo eine Anfrage es anhand seines SSK oder CHK suchen würde.

Im Gegensatz zu DHTs, in denen Knoten sofort nach dem Betreten des Netzes optimale Leistung erreichen, müssen Peers in Freenet erst Informationen über andere Peers im Netz sammeln. Die Effizienz von Freenet verbessert sich also mit der Zeit.

Aus den hier vorgestellten Algorithmen von Freenet folgen seine wichtigsten Eigenschaften: In seiner Reinform verschleiert Freenet die Quelle der Information, sowie den Nachfrager einer Information. Wir haben also *Publisher/Reader Anonymity*. Dadurch, dass Dokumente in Freenet bei Nachfrage selbstständig gecachet werden und die ursprüngliche Datenquelle nicht von einem Cache unterschieden werden kann, haben wir bei nachgefragten Dokumenten auch *Server Anonymity*: Sind einige Anfragen für ein Dokument bereits durchgeführt worden, so ist es nicht möglich herauszufinden, welche Peers das Dokument gecachet haben. In der Urform von Freenet werden jedoch Dokumente unverschlüsselt gespeichert, und somit ist keine *Document Anonymity* gegeben.

Document Anonymity bezeichnet den Umstand, dass ein Server den Inhalt der Dokumente nicht kennt, die er speichert, oder (in seiner schwächeren Form) zumindest abstreiten kann, dass er den Inhalt kennt. Dies kann in einem weiteren Verschlüsselungsschritt erzielt werden. Es werden also nicht mehr GUID/Dokument-Paare sondern GUID/verschlüsseltes Dokument-Paare verschickt. Ein Knoten, der sich nicht aktiv um Schlüssel zur Entschlüsselung der Dokumente bemüht, wird also den Inhalt der Dokumente nicht erfahren. Als Mittel, den Schlüssel zugänglich zu machen, werden

SSKs und die zugehörigen Dokumente empfohlen, in denen neben CHKs auch die entsprechenden Schlüssel zum Entschlüsseln der Dokumente zugänglich gemacht werden können.

Query Anonymity ist jedoch in Freenet nie gegeben. Unter geeigneten Umständen (beispielsweise in einem Überwachungsstaat) kann also den Knoten abverlangt werden, Anfragen zu gewissen Schlüsseln nicht zu beantworten.

Ein schwerwiegenderer Nachteil von Freenet ist, dass eine inhaltsbasierte Suche mit den vorliegenden Verfahren nur schwer zu implementieren ist. Clarke *et al.* [6] schlagen vor, spezielle SSKs zu verwenden, zu denen alle beitragen können. Unter diesen SSKs könnten dann beispielsweise GUIDs von Dokumenten gespeichert werden, die gewisse Suchterme enthalten. Ein anderer Ansatz ist der FASD-Ansatz von Kronfol [17], in dem statt der CHKs für das Routing inhaltsbasierte Schlüssel verwendet werden, die im Dokument vorhandene Terme als einen Bitvektor, eine Signatur, kodieren. Jedoch widersprechen beide Ansätze der eigentlichen Zielsetzung von Freenet. In dem Moment, in dem inhaltsbasierte Suche möglich ist, kann von den einzelnen Peers verlangt werden, Dokumente zu blockieren, deren Signatur bestimmte Worte enthält. Die Freiheit, Beliebiges im Netz zu publizieren, wird also damit eingeschränkt.

3 Sicherheit

Während die oben beschriebenen Publikationen ihren Fokus auf Anonymität in unstrukturierten Netzwerken gerichtet haben, ist ein großer Teil der Forschung über P2P-Netzwerke auf die effiziente Suche gerichtet. Hier liegt der Schwerpunkt weniger auf der Anonymität als auf der Qualität der Daten und der Effektivität der Verwaltung. Offensichtlich hängt der hierbei erzielbare Nutzen der Datenverwaltung von der Zahl und der Effizienz der Angreifer ab.

In diesem Artikel werden zwei grundlegend verschiedene Ansätze zur Abwehr von Angriffen betrachtet: Sicheres Routing und Reputationssysteme. Zunächst jedoch sei auf die Problematik sicherer Peer-Identifikatoren hingewiesen.

3.1 Sichere Peer-IDs

In P2P-Netzen wird zur Abwehr von Angriffen auf die Suchqualität und auch zur Handhabung der normalen Mitgliederfluktuation im P2P-Netz Redundanz eingesetzt. Aufgaben, die von einzelnen Peers effizient zu bewältigen wären, werden redundant mithilfe mehrerer Peers gelöst, um so die Wahrscheinlichkeit eines Fehlers zu reduzieren.

Diese Redundanz ist nützlich, solange es einzelnen Netzteilnehmern nicht möglich ist, nach Belieben neue Identitäten anzunehmen. In verteilten Hashtabellen werden beispielsweise Peer-Identifikatoren (Peer-IDs) verwendet, um die Strukturinvariante der DHTs aufrecht zu erhalten. Ein Angreifer kann bei DHTs also voraussehen, welche ID welcher Funktion im Netzwerk entsprechen würde und gleichzeitig verschiedene Identitäten annehmen, um so dem Netzwerk maximalen Schaden oder gezielt den gewünschten Schaden zuzufügen.

Peers die Auswahl der eigenen ID zuzugestehen ist also ein Sicherheitsrisiko. Wie von Douceur in [9] nachgewiesen, ist es ohne eine logisch zentrale Instanz nicht möglich,

in großen P2P-Netzwerken eine sichere Verwaltung von Peer-IDs durchzuführen. Alternative Ansätze, wie z. B. der Ansatz, neue Peers zunächst ein rechenintensives Rätsel lösen zu lassen, scheitern daran, dass die Rechenleistung der Peers über mehrere Größenordnungen variiert. Ein Rätsel, das dem Durchschnittspeer Stunden Rechenleistung abverlangt, kann für einen interessierten, mächtigen Angreifer in Sekunden lösbar sein.

Wirklich sichere Verfahren benötigen direkt oder indirekt also eine zentrale Instanz zur Identitätsverwaltung. Das Vorhandensein einer solchen Identitätsverwaltung wird im Folgenden angenommen.

3.2 Sicheres Routing

Sicheres Routing hat zum Ziel, trotz einer Anzahl nicht vertrauenswürdiger Peers im Netz Daten korrekt in DHTs einzufügen und sie ebenso korrekt auszulesen.

Sit und Morris geben in [25] eine Liste von Entwurfs-Regeln (Prinzipien) an, die beim Bau sicherer DHTs zu beachten sind (siehe Tab. 1). Die Grundidee ist hierbei, die Struktur der DHT nicht nur zu verwenden, um Daten aufzufinden, sondern auch um zu prüfen, ob die Daten auf korrekte Art und Weise gefunden wurden. Offensichtlich läuft dies der Zielsetzung anonymer Netzwerke zuwider, deren Ziel ist, den einzelnen an der Bereitstellung der Daten beteiligten Peers zu ermöglichen, eben gerade ihre Mitarbeit an der Bereitstellung möglichst weitgehend abzustreiten.

1. Define verifable system invariants (and verify them!)
2. Allow the querier to observe lookup progress.
3. Assign keys to nodes in a verifiable way.
4. Server selection in routing may be abused.
5. Cross-check routing tables using random queries.
6. Avoid single points of responsibility.

Tabelle 1 — Entwurfs-Prinzipien für sichere DHTs (entnommen aus [25])

Sit und Morris beschreiben ferner verschiedene DHT-spezifische Angriffe, die mit den in Tab. 1 gegebenen Entwurfs-Regeln abgewehrt werden können. Im Wesentlichen werden hier drei Klassen von Angriffen unterschieden: Angriffe auf das *Routing* von Anfragen, Angriffe auf das *Speichern und Auslesen* von Daten und andere Angriffe, die nicht in diese beiden Kategorien fallen.

3.2.1 Angriffe auf das Routing

Die Suchalgorithmen in allen DHTs sind so konzipiert, dass jeder Routing-Schritt die Anfrage dem Zielknoten näher bringt.

In Pastry [22] z. B. ist jeder Knoten durch eine Bitfolge identifiziert. Die Schlüssel/Wert-Paare werden jeweils in denjenigen Knoten gespeichert, deren Identifikator-Bitfolge (ID) das längste Präfix mit dem Schlüssel des Schlüssel/Wert-Paars (dem Einfügeschlüssel) gemeinsam hat. Bei jedem Routing-Schritt werden nun Knoten erreicht, deren ID ein längeres Präfix mit dem Einfügeschlüssel teilen. Nach einer logarithmischen Anzahl von Schritten erreicht dieses Verfahren den Ziel-Peer.

Ob nun ein Routing-Schritt erfolgreich war, lässt sich leicht feststellen, wenn der eigentliche Anfragesteller über die einzelnen Routingschritte informiert wird. DHT-Protokolle lassen sich bei Bedarf leicht so umstellen, dass jeder Routing-Schritt von der Quelle der Anfrage initiiert wird, also so genanntes *iteratives Routing* durchgeführt wird. Als Folge dieses Vorgehens kann der Anfragesteller jeden Schritt kontrollieren (siehe Abb. 2 und 3). Damit sind die Prinzipien 1 und 2 aus Tab. 1 beachtet.

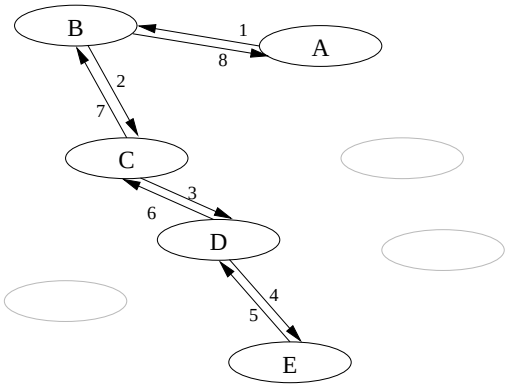

Abbildung 2 — Rekursives Routing

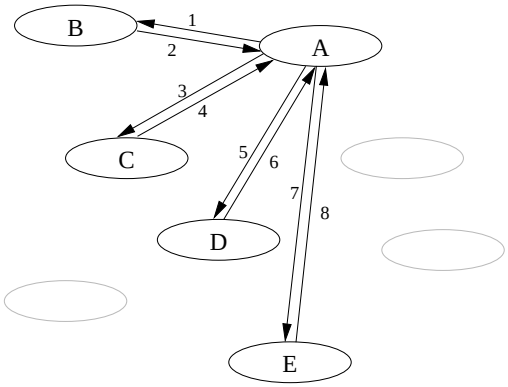

Abbildung 3 — Iteratives Routing: In jedem Schritt erfragt A, welcher Peer Adressat des nächsten Pakets werden soll.

Eine andere Angriffsmöglichkeit ist der Versand gefälschter Routingtabellen. Um korrekt verbunden zu bleiben, auch wenn ständig Peers dem Netz beitreten und es wieder verlassen (also bei *Churn*), tauschen Peers in einer DHT Routinginformationen mit ihren Nachbarn aus. Diese Routinginformation kann gefälscht werden, wenn das P2P-Protokoll die Freiheiten hierzu gibt. Hier muss der Entwickler eines P2P-Netzwerkes also abwägen, ob er der Robustheit und Effizienz oder eher der Sicherheit gegen Angriffe den Vorzug gibt. Die Robustheit und Effizienz lassen sich steigern, wenn man flexiblere Routing-Tabellen erlaubt, da man dann beispielsweise unter einer relativ großen Menge von Peers diejenigen Peers auswählen kann, die besonders gut erreichbar sind (Server Selection, Prinzip 4). Ist die Menge der Peers, die in einer Routingtabelle vorkommen dürfen, jedoch relativ klein, so lässt sich diese Routingtabelle nur unter größeren Schwierigkeiten fälschen, beispielsweise mit Peers bevölkern, die alle unter Kontrolle des Angreifers stehen.

Ein anderer, trickreicher Angriff ist es, einen neu in ein Netz eintretenden Peer in ein

Schattennetz zu entführen, ein so genannter Partitionsangriff: Will ein Knoten einem beliebigen P2P-Netz beitreten, so muss er einen Knoten kontaktieren, der bereits Mitglied dieses Netzes ist. Dieser Knoten muß vertrauenswürdig sein, ansonsten hat dieser neben den bereits beschriebenen Angriffen auf die Routing-Qualität die Möglichkeit, den neuen Peer in ein eigenes Netz zu entführen, z. B. in ein kleines Netz, das nur aus Angreifern besteht. Sit und Morris schlagen als Abwehr vor, ständig zufällige Anfragen abzusetzen und die Ergebnisse hiervon mit denen zufälliger Anfragen bekannter, vertrauenswürdiger Peers zu vergleichen (Prinzipien 5 und 6).

Abb. 4 zeigt den Vorgang: *Alice* tritt dem Netz bei, *Bob* ist Mitglied des von *Alice* gewünschten korrekten Netzwerks, bewerkstelligt für *Alice* aber den Beitritt in das Schattennetzwerk, dessen Mitglied er ebenfalls ist. Dieses Netz verwendet das gleiche Protokoll wie das korrekte Netzwerk, ist jedoch von anderen Peers bevölkert, von denen einige Peers Angreifer (schwarz markiert), und andere Peers eventuell arglose Angriffsopfer sind.

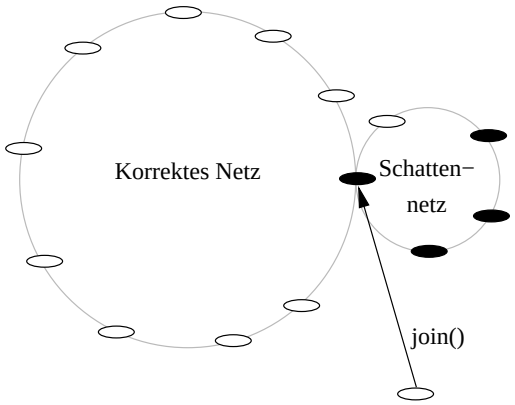

Abbildung 4 — Partitionsangriff: *Alice* will über *Bob* dem korrekten Netz beitreten, *Bob* entführt *Alice* in das Schattennetz.

3.2.2 Angriffe auf Speicherung und Auslesen

Die Grundidee dieser Angriffe ist, dass ein Peer Daten zwar zur Speicherung annimmt, auf Anfrage dann jedoch meldet, diese Daten nicht zu besitzen.

Die Abwehr dieser Angriffe ist in Prinzip 6 geschildert. Ein *Single Point of Responsibility*, also ein Peer, der für gewisse Daten allein verantwortlich ist, kann durch Replikate vermieden werden. Gleichzeitig muss jedoch gesichert sein, dass *erstens* die Verteilung dieser Replikate nicht einem einzelnen Peer obliegt und *zweitens*, dass immer mehrere Peers wissen, wo sich die Replikate befinden und wie man auf sie zugreifen kann. Der Anfragesteller muss ferner bei einer Suche durch Zugriff auf mehrere Replikate die Integrität der gefundenen Daten prüfen.

Die oben beschriebenen Angriffe sind schwieriger zu entdecken, wenn sie nicht immer, sondern nur manchmal durchgeführt werden.

Die weiteren, von Sit und Morris geschilderten Angriffe werden der Kürze halber hier nicht wiedergegeben.

3.2.3 Sicheres Routing nach Castro et al.

Castro *et al.* untersuchen in [4] sicheres Routing in DHTs. Sie stellen hierzu zunächst ein generisches DHT-Modell vor und argumentieren auf Basis dieses Modells. In weiten Teilen folgt ihre Argumentation der von Sit und Morris, jedoch wird in wichtigen Punkten von der dort gegebenen Argumentation abgewichen. Die in [4] vorgestellten Ansätze wurden dann als Modifikationen in Pastry [22] eingearbeitet und experimentell getestet.

Die augenfälligste Abweichung ist die Abweichung von Prinzip 2, das dem iterativen Routing gegenüber rekursivem Routing den Vorzug gibt.

Castro *et al.* schlagen stattdessen vor, eine Anfrage zunächst rekursiv und effizient zu routen und dann das Resultat auf Korrektheit zu testen. In Pastry wird jedes Schlüssel/Wert-Paar der so genannten Wurzel R zugeordnet, also dem Knoten, dessen ID dem Einfügeschlüssel am nächsten ist. Ferner wird es noch weitere $2r$ Mal repliziert, und zwar auf die Knoten, deren ID der von R am nächsten ist. Diese Knoten werden Replikenwurzeln genannt. Von den Peer-IDs der Replikenwurzeln sind r IDs kleiner, und r größer als die ID der Wurzel. All diese Eigenschaften lassen sich einfach nachprüfen. Komplizierter ist ein weiterer Test, der auf die IDs der Replikenwurzeln angewandt wird.

Hier wird die genaue Verteilung der IDs überprüft. Hierbei wird angenommen, dass die Peer-IDs gleichverteilt sind. Dies wird schon bei der Konstruktion von Peer-IDs gefordert, um balancierte DHT-Strukturen zu erhalten. Ferner wird angenommen, dass die Durchschnittsdichte von Peer-IDs pro Teilvolumen des Schlüsselraums kleiner ist als die Durchschnittsdichte von Peers IDs eventueller Angreifer. Wenn nun ein Angreifer eine gefälschte Wurzel und mehrere gefälschte Replikenwurzeln als Resultat des Routings zurückgibt, dann wird die Dichte der IDs der gefälschten Replikenwurzeln von der normalen ID-Dichte abweichen.

Der Anfragesteller wird also nach dem Routing die Dichte von Peer-IDs in seiner Umgebung mit der Dichte von Peer-IDs der Replikenwurzeln des gesuchten Schlüssels vergleichen. Schlägt dieser Vergleich fehl, so wird das Ergebnis des effizienten Routings verworfen und dieselbe Anfrage mittels redundanten Routings durchgeführt.

Eine genaue Beschreibung des redundanten Routings würde hier zu weit führen. Es sei jedoch gesagt, dass auch hier zur Bearbeitung einer Anfrage bei jedem Hop mehrere Peers kontaktiert werden. Es wird jedoch mittels kryptographischer Techniken in einem iterativen Prozess sichergestellt, dass alle Replikenwurzeln erreicht werden und bestätigen, dass sie erreicht wurden. Das Verfahren funktioniert in mehr als $99,9\%$ der Fälle, falls 30% der Knoten des Netzwerks oder weniger Angreifer sind.

3.3 Reputation

In großen P2P-Netzen interagieren Peers nur mit einem kleinen Anteil der anderen Peers direkt. In sehr vielen P2P-Systemen beschränkt sich die protokollgemäße Interaktion der Peers auf die Nachbarn im strukturierten oder unstrukturierten Netzwerk sowie auf das Herunterladen von Dateien, die bei einer Suche als *best Matches* angegeben wurden. Gerade dieses Herunterladen findet in vielen P2P-Systemen direkt statt: Der Nachfragende lädt die Daten bei dem Halter einer Datei herunter. Bei großen P2P-Netzwerken von hunderttausenden von Peers ist es unwahrscheinlich, dass man

häufig die gleiche Datenquelle verwendet. Die meisten kontaktierten Peers sind dem Nachfragenden also gänzlich unbekannt.

Dieses Szenario entspricht dem von eBay. Während eBay mit zentralen Servern realisiert wird, stehen die einzelnen Verkäufer vor einem recht ähnlichen Problem, wie Peers, die Daten im Gnutella-Netz [7] nachfragen. Ein Einkäufer sucht nach Gütern, die ihm gefallen und muss dann entscheiden, ob er bei einer Versteigerung mitbietet. Der Einkäufer wird seinen Verkäufer nur selten kennen. Als Entscheidungshilfe hat eBay ein Reputationssystem eingeführt. Die Marktteilnehmer bewerten jeweils Transaktionen, an denen sie teilgenommen haben. Diese Bewertungen werden von eBay zentral verwaltet und sind öffentlich zugänglich. Sie werden zu einer Gesamtnote aggregiert. Potentielle Käufer können nun anhand der Bewertungen entscheiden, ob sie an einer gegebenen Versteigerung teilnehmen oder nicht.

P2P-Systeme versuchen Ähnliches durch verteilte Berechnung zu erzielen. Sie wollen auf diese Weise Netzteilnehmern helfen, gute Server als Datenquellen auszuwählen und schlechte durch Nichtbeachtung aus dem Netz zu drängen. Ziel ist dabei eine möglichst hohe Effizienz in dem Sinne, dass böswillige Peers starke Nachteile erleiden, während gleichzeitig der Verwaltungsaufwand gutwillige Peers nur wenig bremsen soll.

Marti und Garcia-Molina geben in [18] einen Überblick über die verschiedenen Dimensionen in denen Entwurfsentscheidungen getroffen werden können. Sie sehen im Wesentlichen drei Dimensionen, in denen sich P2P-Reputationssysteme unterscheiden: Die Art der Informationsaquise, die Aggregation der Information zu einem Score oder einem Rang sowie die Reaktion des Netzes auf Peers mit niedriger Reputation.

3.3.1 Informationsaquise in P2P-Reputationssystemen

Auswahl der Informationsquelle: Die Informationsaquise ist gekennzeichnet durch die Auswahl der Informationsquelle. Hier besteht die Wahl zwischen wenig Information hoher Qualität und viel Information niedriger Qualität. Beispielsweise kann ein Peer nur seine eigenen Erfahrungen als vertrauenswürdig einstufen, oder sich nur auf Peers verlassen, deren Benutzer seinem Betreiber persönlich bekannt sind. Schließlich lässt sich die Menge der Information über andere Peers dadurch vergrößern, dass man vertrauenswürdige Peers nach ihren Erfahrungen fragt, die dann eventuell wieder die ihnen vertrauenswürdig erscheinenden Peers fragen. Mit jeder Stufe nimmt hier die Zahl der Peers zu, über die man etwas erfährt, und gleichzeitig nimmt die Zahl der unerkannten Angreifer zu, die fehlerhafte Daten zu dem Gesamtergebnis beisteuern und so das Gesamtergebnis stören können. Systeme wie EigenTrust [16] schließlich berechnen Vertrauenswerte über alle Peers unter Verwendung der Daten aller Peers. Marti und Garcia-Molina nennen solche Systeme *Global History*-Systeme, weil sie sämtliche Interaktionen des Netzwerkes in ihre Berechnungen einbeziehen.

Reaktion auf fremde Peers: Mit der Informationsaquise verbunden ist die Reaktion auf fremde Peers. Hier besteht die Wahl, entweder dem Reputationssystem zu vertrauen (also den Aussagen einer mehr oder weniger großen Menge vertrauenswürdiger Peers), oder aber nur auf die eigenen Erfahrungen zu bauen.

Bestehen keine solchen Erfahrungen, oder hat kein einziger Peer mit dem neuen Peer interagiert, so besteht die Wahl, unbekannten Peers zunächst einen Vertrauensvorschuss zu geben, oder Peers zu ignorieren, bis sie sich als vertrauenswürdig erwiesen haben. Diese Wahl kann dadurch unterstützt werden, dass das Reputationssystem das

Durchschnittsverhalten fremder Peers berechnet, und so eine Aussage ermöglicht, wie risikoreich Vertrauen auf die Korrektheit fremder Peers im Durchschnitt ist.

3.3.2 Bewertung und Ranking von Peers

Welche Daten sollen gesammelt werden? Idealerweise wird man versuchen, korrekte und fehlerhafte Transaktionen festzuhalten. Häufig wird man jedoch nicht erfahren können, ob eine Transaktion fehlerhaft war. In unstrukturierten Netzen beispielsweise, ist nicht von außen zu beurteilen, ob gewisse Daten in einem Peer vorliegen oder nicht. Stellt ein Peer diese Daten zur Verfügung, so kann dies als positive Transaktion notiert werden. Jedoch kann in unstrukturierten Netzwerken nur in seltenen Fällen festgestellt werden, dass ein Peer zwar Daten enthält, diese aber Anderen trotzdem nicht zur Verfügung stellt. Eben dieses Faktum macht das so genannte *Free Riding*, die Teilnahme an P2P-Netzen ohne einen eigenen Beitrag zu leisten, so einfach.

Ist es nur möglich, Information über Beiträge zu sammeln, nicht jedoch über Fehler, so ist es nützlich, auch die Menge der Transaktionen zu vermerken. Systeme, die Verhaltensänderungen von Peers berücksichtigen wollen, müssen auch die Zeiten von Transaktionen vermerken und Informationen über ältere Transaktionen schwächer in die Reputationsberechnung mit einbeziehen als Informationen über neuere Transaktionen.

Welche Eigenschaften sollen berechnet werden? Es wird vorgeschlagen, nicht nur einen, sondern mehrere Trust-Werte für jeden Peer zu berechnen. Das TRELLIS-System [12] beispielsweise bewertet Peers bezüglich ihres Verhaltens in Transaktionen sowie hinsichtlich ihrer Bewertung anderer Peers.

Wie sollen die Daten verwendet werden? Hier ist zu empfehlen, die Bewertung eines Peers mit einem Schwellwert zu vergleichen. Enthalten jedoch viele vertrauenswürdige Peers die gewünschten Daten, dann sind diejenigen Peers vorzuziehen, die am besten bewertet sind. In ihrem Papier über EigenTrust schlagen Kamvar *et al.* vor, nicht deterministisch den Peer mit der besten Bewertung auszuwählen, um zu vermeiden, dass sich Last und Reputation extrem unregelmäßig im Netz verteilen.

3.3.3 Konsequenzen aus guter Bewertung

Bisher ist davon auszugehen, dass ein Peer, der eine gute Bewertung hat, eine höhere Last hat. Es liegt also im Interesse eines Reputationssystems, Peers anderweitig zu motivieren, eine gute Bewertung anzustreben. Hier kann man mit positiver und mit negativer Motivation arbeiten.

Mögliche Motivation ist aus verbesserter Geschwindigkeit bei der Abarbeitung von Anfragen zu ziehen. Bittorrent, z. B. gibt Peers die Bandbreite, die sie anderen Peers zur Verfügung stellen. Ähnlich könnte man die Qualität oder Menge übertragener Bilddaten der Bewertung anpassen, oder schließlich Peers sich ihre Dienste gegenseitig bezahlen lassen [13].

3.3.4 Konsequenzen aus schlechter Bewertung

Als Konsequenz aus negativem Benehmen wird vorgeschlagen, dass Peers mit Peers, die zu schlecht bewertet sind, nicht mehr kommunizieren. P2P Netzwerke mit einem Bezahlsystem [13] können natürlich auch Geldstrafen an Angreifer verhängen.

4 Anonymität und Reputation im Zusammenhang mit Ähnlichkeitssuche

Die oben vorgestellten Verfahren eignen sich insbesondere für exakte Suche nach Schlüsselwerten. Systeme wie Freenet gehen ferner davon aus, dass die Schlüssel keinen semantischen Bezug zu den gespeicherten Werten (Dokumenten) haben.

Im Folgenden wird diskutiert, welche Auswirkungen die Erweiterung der Suchfunktionalität von rein Schlüsselbasierter suche hin zu inhaltsbasierter Suche auf die hier vorgestellten Verfahren hat.

4.1 Anonymität vs. Suchbarkeit:

Freenet ist für die anonyme Suche nach Inhalten ungeeignet, da zur effizienten Suche die Schlüssel Semantik tragen müssten, und es somit den einzelnen Peers möglich wäre, aufgrund der Semantik der Schlüssel gewisse Daten von der Speicherung und Weiterleitung auszuschließen. Die Forschung nach Alternativen ist nach Kenntnisstand des Autors bisher noch völlig offen. Eine Möglichkeit könnte z. B. sein, *Private Information Retrieval*-Techniken [5] anzuwenden, die dem einzelnen Peer volle *Query Anonymity* garantieren und diese Techniken mit neuen Verfahren zur Suche in unstrukturierten Netzen zu kombinieren [23]. Private Information Retrieval ist jedoch ein weites, aktives Feld der Forschung und hier nicht Gegenstand der Diskussion. Ein anderer, bereits untersuchter Ansatz [2] ist es, Peers Gruppen formen zu lassen, die gemeinsam für einen Datenbestand zuständig sind, und die Suche so zu gestalten, dass jeder Peer abstreiten kann, das Suchresultat geliefert zu haben. Außerhalb der Peergruppe ist nur bekannt, dass *mindestens ein* Peer der Gruppe die inkriminierenden Daten hält.

4.2 Reputation und Suche

In der bisherigen Diskussion (Abschnitt 3.3) wurde nur recht abstrakt über die Qualität und die Bewertung von Transaktionen gesprochen. Es wurde auch darauf hingewiesen, dass ein Angriff in einer Transaktion nicht immer erkannt werden kann.

Bei der Ähnlichkeitssuche wiegt dieses Problem besonders schwer. Während in einer DHT noch recht einfach entschieden werden kann, ob ein Peer ein Datum enthalten müßte oder nicht, ist diese Frage bei einigen Verfahren zur P2P-Ähnlichkeitssuche nicht so einfach zu beantworten.

Am meisten Erfolg versprechen hier Verfahren, in denen Peers alle ihre Indexdaten in eine DHT-basierte verteilte Indexstruktur publizieren (beispielsweise [28]). Die DHT kann hier mittels sicheren Routings geschützt werden. In diese gesicherte DHT publizieren Peers jeweils Paare, die eindeutigen Dokumentschlüsseln Indexdaten zuordnen. Wird ein Dokument bei der Ähnlichkeitssuche als besonders ähnlich bewertet und

heruntergeladen, so kann der Nachfragende selbst Indexdaten aus dem Dokument extrahieren und dann überprüfen, ob die publizierten Indexdaten mit den von ihm selbst bestimmten Indexdaten des Dokuments übereinstimmen. Es besteht dann also eine klare Bewertungsgrundlage für die Ähnlichkeitssuche: Bewertet wird, ob ein Peer Indexdaten korrekt publiziert. Die Korrektheit der Suchresultate erwächst dann direkt aus den Eigenschaften der gesicherten verteilten Hashtabelle sowie aus den Eigenschaften der Indexdaten.

Schwieriger wird solch ein Vorgehen bei zusammenfassungsbasierten Verfahren. Zusammenfassungsbasierte Verfahren gehen davon aus, dass es entweder zu teuer oder aber nicht wünschenswert ist, die Indexdaten einzelner Peers komplett zu publizieren und auf andere Rechner zu replizieren. Um eine effiziente Suche zu erzielen, werden jedoch Zusammenfassungen der Peerdaten publiziert, die sehr viel kleiner sind als die gesamten Indexdaten, und die es trotzdem ermöglichen, für eine gegebene Anfrage die besten Peers auszuwählen ([2, 3, 20] sind beispielsweise solche Verfahren). Ohne Erweiterung ist es in diesen Verfahren selten möglich, die Qualität einer Suchanfrage abschließend negativ zu beurteilen[6]: Wenn ein Peer keine ähnlichen Resultate liefert, obwohl seine Zusamenfassung anderes erwarten lässt, dann wird dies zumeist an dem Faktum liegen, dass es sich nur um eine Kurzzusammenfassung handelt, die kürzer und weniger präzise ist als die echten Indexdaten ([2] nutzt explizit die Unschärfe der über Peer-Gruppen berechneten Zusammenfassung zur Verschleierung der Identität des wahren Datenhalters). Wenn ein Angreifer geschickt vorgeht, so kann er häufig Anfragen negativ bescheiden, ohne seiner Zusammenfassung nachweisbar zu widersprechen.

Diesem Angriff kann man durch Stichproben begegnen, oder dadurch, dass man hybrid vorgeht und einen Teil der Indexdaten publiziert, während ein anderer Teil geheim gehalten wird. Nach Kenntnisstand des Autors gibt es hierzu noch keinerlei Arbeiten.

5 Zusammenfassung

In diesem Artikel wurden einige wesentliche Resultate über anonyme Publikation und Sicherheit in P2P-Netzen vorgestellt. Diese Resulate sind sehr nützlich, jedoch lassen sie viel Raum für weitere Arbeiten. Zusätzlich zu der Konzeption und Implementierung von Sicherheitsmerkmalen an sich, beispielweise von sicheren DHTs, anonymen Publikationssystemen, Reputationssystemen oder Bezahlsystemen gibt es Herausforderungen, die in der *Kombination* von Sicherheit und Anonymität mit der Suche nach Inhalten in P2P-Datenbanken bzw. P2P-IR-Systemen liegen. Hier sind als Herausforderungen insbesondere die Auflösung des Widerstreits zwischen Suchbarkeit und Anonymität sowie die Bewertung von Resultaten der Ähnlichkeitssuche für die Reputationsbestimmung zu nennen.

Literatur

[1] R. Z. Albert. *Statistical mechanics of complex networks.* PhD thesis, 2001. Director: Albert-Làszlò Barabasi.

[6] Positiv kann die Suchanfrage dann beurteilt werden, wenn der Peer ein sehr ähnliches Resultat liefert, das den Informationswunsch des Nutzers befriedigt.

[2] M. Bawa, R. J. B. Jr., and R. Agrawal. Privacy-preserving indexing of documents on the network. In *VLDB*, pages 922–933, 2003.

[3] M. Bender, S. Michel, G. Weikum, and C. Zimmer. The MINERVA Project: Database Selection in the Context of P2P Search. In G. Vossen, F. Leymann, P. C. Lockemann, and W. Stucky, editors, *BTW*, volume 65 of *LNI*, pages 125–144. GI, 2005.

[4] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, 2002.

[5] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[6] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[7] Clip2. The Gnutella Protocol Specification v0.4. URL: `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`, 2000.

[8] R. Dingledine, M. J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[9] J. R. Douceur. The Sybil Attack. In Druschel et al. [10], pages 251–260.

[10] P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors. *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*. Springer, 2002.

[11] M. J. Freedman and R. Morris. Tarzan: a peer-to-peer anonymizing network layer. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206, New York, NY, USA, 2002. ACM Press.

[12] Y. Gil and V. Ratnakar. Trusting information sources one citizen at a time. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 162–176, London, UK, 2002. Springer-Verlag.

[13] D. Hausheer, N. Liebau, A. Mauthe, R. Steinmetz, and B. Stiller. Token-based accounting and distributed pricing to introduce market mechanisms in a peer-to-peer file sharing scenario. In *Peer-to-Peer Computing*, pages 200–201. IEEE Computer Society, 2003.

[14] K. Hose, M. Karnstedt, K.-U. Sattler, and E.-A. Stehr. Adaptive routing filters for robust query processing in schema-based p2p systems. In *IDEAS*, pages 223–228. IEEE Computer Society, 2005.

[15] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *CIDR*, pages 28–43, 2005.

[16] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust algorithm for reputation management in p2p networks. In *WWW*, pages 640–651, 2003.

[17] A. Z. Kronfol. FASD: A fault-tolerant, adaptive, scalable, distributed search engine, 2002.

[18] S. Marti and H. Garcia-Molina. Taxonomy of trust: Categorizing p2p reputation systems.

[19] J. Menn. *all the rave, The Rise and Fall of Shawn Fanning's Napster*. Crown Business, New York, 2003.

[20] W. Müller, M. Eisenhardt, and A. Henrich. Scalable summary based retrieval in p2p networks. In O. Herzog, H.-J. Schek, N. Fuhr, A. Chowdhury, and W. Teiken, editors, *CIKM*, pages 586–593. ACM, 2005.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. Conf. on applications, technologies, architectures, and protocols for computer communications*, San Diego, CA, USA, 2001.

[22] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *Middleware*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[23] N. Sarshar, P. O. Boykin, and V. P. Roychowdhury. Percolation search in power law networks: Making unstructured peer-to-peer networks scalable. In *Peer-to-Peer Computing*, pages 2–9. IEEE Computer Society, 2004.

[24] Secure hash standard, 1995.

[25] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In Druschel et al. [10], pages 261–269.

[26] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. ACM SIGCOMM Conf.*, San Diego, CA, USA, 2001.

[27] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, 4–7 1997.

[28] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, 2002.