Technische Universität Berlin

Fakultät IV - Elektrotechnik und Informatik

Fachgebiet - Information Systems Engineering

Prof. Dr.-Ing. Stefan Tai

Bachelorarbeit

Entwicklung und Design eines sozialen partiell anonymen Filesharing Systems auf Basis von P2P und Tor

Sommersemester 2015

Tim Hinkes

Matrikel–Nummer xxxxxx

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbstständige und eigenhändige Anfertigung dieser Arbeit versichert an Eides statt:

Ort, Datum	Unterschrift

Abstract

Die Zensur-resistente anonyme Übertragung von größeren Datenmengen im Internet, stellt ein großes Problem dar. Die existierenden Systeme bieten entweder eine langsame anonyme Zensur-resistente, oder eine nicht anonyme schnelle Übertragung. Onion^{Core} (Onion^{Core}) soll diese Lücke, aufbauend auf dem BitTorrent System, schließen, indem es dort wo keine Anonymität nötig ist, eine schnelle Datenübertragung erlaubt, gleichzeitig aber auch weitere anonyme langsamere Verbindungen möglich sind. Um diese Zeile zu erreichen, werden Onion^{Core}-Teilnehmer in Gruppen eingeteilt. Mitglieder der eigenen Gruppen gelten dabei als Vertrauenswürdig, wobei dann keine Anonymität nötig ist. Alle anderen Teilnehmer werden als nicht vertrauenswürdig betrachtet. Verbindungen zu diesen Teilnehmern, sind nur über das Anonymisierungsnetzwerk The Onion Router (Tor) möglich. Die Evaluation vonOnion^{Core} hat ergeben, dass besonders bei Teilnehmern die geographisch weit voneinander entfernt sind, das gleichzeitige Teilen einer Datei sowohl mit anonymen Teilnehmern, als auch mit vertrauten Mitgliedern Vorteile bei der Geschwindigkeit ergeben kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im heutigen Internet spielt der Austausch von Dateien im globalen Kontext eine bedeutende Rolle. Die hierfür verwendeten Verfahren lassen sich grob in zwei Ansätze unterteilen.

Der erste Ansatz basiert auf dem sog. Client-Server-Modell. Hierbei gibt es einen zentralen Dienstanbieter (Server) welcher Speicherplatz und Netzwerkbandbreite bereitstellt, sodass ein Nutzer (Client) eine Datei zu diesem Dienstanbieter über das Internet hochladen kann. Danach können andere Nutzer diese Datei vom Dienstanbieter herunterladen. Dieses Konzept hat den Vorteil, dass die Internetverbindung des ursprünglichen Besitzers nur einmal belastet wird.

Der zweite weit verbreitete Ansatz basiert auf der Peer-to-Peer (P2P) Technik. Hierbei wird die Datei von Interessenten direkt vom ursprünglichen Besitzer heruntergeladen. Damit aber nicht jeder Interressent die Datei komplett von einem einzigen Nutzer herunterladen muss, wird sich zunutze gemacht, dass andere Nutzer die Datei auch schon besitzen, sodass Teile der Datei von verschiedenen Nutzern gleichzeitig heruntergeladen werden können. Das entlastet die Verbindung des ursprünglichen Nutzers, nachdem die Datei eine initiale Verbreitung erreicht hat.

Beide Konzepte haben ihre eigenen Vor- und Nachteile. Bei der Client-Server Technik gibt es das Problem der Zentralisierung, was dem Serverbetreiber die Möglichkeit gibt, die Verbreitung von unliebsamen Inhalten zu unterbinden oder Profile über die Nutzer anzulegen. Die P2P Technik hat zwar das Problem der Zentralisierung großteils umgangen, jedoch gibt ein Nutzer hier seine Identität (IP-Adresse), welche über den Internet Service Provider (ISP) auf den Anschlussinhaber abgebildet werden kann, an andere Nutzer preis.

Der Nachteil, dass durch die Nutzung von P2P Netzwerken die eigene Identität verraten wird, lässt sich allerdings durch eine komplette Anonymisierung umgehen. Dies ist z.B. möglich, indem der gesamte Netzwerkverkehr über das Tor-Netzwerk[1] geleitet wird, was aber zur Folge hat, dass der Datendurchsatz stark verringert wird[2]. Oftmals ist jedoch eine komplette Anonymisierung überhaupt nicht nötig, wenn z.B. Dateien sowohl mit vertrauenswürdigen Partnern als auch mit Unbekannten geteilt werden sollen. In diesem Fall würde es reichen, den Teil der Daten, welcher mit Unbekannten geteilt werden soll, über eine anonyme Verbindung zu leiten und trotzdem einen hohen Datendurchsatz zu den vertrauenswürdigen Partnern beizubehalten.

1.1 Wissenschaftliche Frage

Diese Arbeit soll genau einen Ansatz zu dem oben beschriebenen Problem liefern, sodass klassisches P2P Filesharing betrieben werden kann, dabei aber zwischen vertrauenswürdigen Peers und nicht vertrauenswürdigen Peers unterschieden wird. Ziel ist es, dass die Datenübertragung zwischen vertrauenswürdigen Peers mit voller Bandbreite gelingt, während eine Übertragung zwischen nicht

[1] Tor ist ein Programm welches den über es geleiteten Netzwerkverkehr anonymisiert. Siehe `https://www.torproject.org/`

[2] Hierzu gibt es aktuell keine verlässlichen Zahlen, da die meisten Projekte darauf abzielten den Durchsatz von Bittorrent Verkehr über das Tor-Netz ins normale Internet zu messen. Bei dieser Arbeit aber geht es um Verkehr der von HiddenServices ausgeht und damit das Tor-Netz garnicht verlässt.

vertrauenswürdigen Peers über eine anonyme (wie z.B. Tor) und damit langsame Verbindung, stattfindet. Bisher erreicht man etwas Ähnliches durch die Nutzung von zwei seperaten Programmen, welche jeweils ihre eigene Clientverwaltung haben und nicht miteinander kommunizieren. Diese Arbeit soll nun diese beiden getrennten Netzen miteinander verbinden, sodass ohne eine redundante Datenhaltung/-Verwaltung ein und dieselbe Datei gleichzeitig sowohl mit vertrauenswürdigen als auch mit nicht-vertrauenswürdigen Peers getauscht werden kann. Am Beispiel BitTorrent würde dies bedeuten nur die Struktur einer Torrentdatei zu nutzen während sie in beide Netze verteilt werden kann.

1.2 Stand der Technik

Es gibt eine Vielzahl an Techniken, welche es ermöglichen anonym Daten zu verschicken und zu erhalten. Dabei wird unterschieden zwischen Netzwerken, welche eine Netzwerk-Schicht-Anonymisierung anbieten wie z.B. Tor oder Invisible Internet Project (I2P), sowie Netzwerke, welche auch direkt eine Funktionalität zum Verteilen von Daten bereitstellen.

- **Freenet** ist ein in sich geschlossenes System auf P2P Basis. Hierbei werden alle Daten auf verschiedenen teilnehmenden Peers gespeichert. Ingesamt basiert die Anonymität bei Freenet darauf, dass zwei Peers nicht direkt miteinander kommunizieren, sondern die Daten verschlüsselt über andere Peers weiterleiten, bis sie ihr Ziel erreicht haben. Dadurch kennt kein Peer die Adressen von Empfänger **und** Absender. Im Freenet werden alle Dateien verteilt gespeichert. Das heißt, dass jeder Teilnehmer einen Teil seiner Speicherkapazität zur Verfügung stellt und dort kleine Teile von Dateien des Netzwerks abgelegt werden. Das gilt sowohl für einfache HTML-Seiten als auch für größere Dateien. Dies hat den Vorteil, dass der Uploader nicht mit dem Netzwerk verbunden sein muss, wenn andere seine Dateien herunterladen möchten.

  Diese Architektur hat aber gleichzeitig den Nachteil, dass wenig populäre Dateien langsam aus den Speichern der Teilnehmer verschwinden,deshalb nach einiger Zeit nicht mehr abrufbar sind und neu hochgeladen werden müssen. Einen sozialen Aspekt bzw. eine Unterscheidung zwischen vertrauenswürdigen und nicht vertrauenswürdigen Peers gibt es hier nur bei der Wahl des Peers, über den Zugang zum Netz erlangt wird, denn dieser kennt zwangsläufig die reale Adresse der Teilnehmer die über ihn Zugang erhalten.

- **BitBlender** ist im Gegensatz zu Freenet ausdrücklich auf BitTorrent-Filesharing ausgerichtet. Dabei wird auch hier für die Anonymisierung das grundlegende Konzept des Übertragens von Daten über mehrere Zwischenstationen angewendet. Hierbei gibt es aber die Besonderheit, dass die Peers, welche Daten herunterladen, nicht wissen ob die Daten über mehrere Zwischenstationen laufen oder direkt vom Kommunikationspartner bereitgestellt werden. Dadurch können sich alle teilnehmenden Peers auf die *glaubhafte Abstreitbarkeit* in Bezug auf den Ursprung der von ihnen übertragenen Daten berufen.

Als weitere Konzepte können hier noch Netzwerke gesehen werden, welche ein reine Transportschicht Anonymisierung bieten.

- **Tor** basiert für die Anonymisierung wie Freenet oder I2P auf dem Konzept, dass Daten über mehrere Zwischenstationen weitergeleitet werden, sodass keine Station jeweils Empfänger und Absender kennt. Dabei bietet das Tor-Netzwerk selbst keinerlei weitere Funktionalität an. Es kann daher sehr einfach von Programmen zur Anonymisierung der übertragenen Daten genutzt werden. Zum Finden von Peers über die Daten geleitet werden können, benutzt das Tor-Netzwerk zentralisierte Verzeichnis-Server.

- **I2P** nutzt zur Anonymisierung auch mehrere Zwischenstationen, sodass niemand gleichzeitig Absender und Empfänger kennt. I2P kommt hierbei aber ohne zentrale Verzeichnis-Server aus. Lediglich zum Finden von Eintrittspunkten kann ein öffentliches Verzeichnis genutzt werden. Im Gegensatz zum Tor-Netzwerk ist I2P aber als in sich geschlossenes Netz zu betrachten, welches nur sehr wenige Peers hat, die eine Verbindung ins offene Internet bereitstellen.

1.3 Beitrag und Ergebnisse

Ziel dieser Arbeit ist es einen Prototypen eines P2P-Filesharing Programmes zu entwickeln welcher den Folgenden Anforderungen genügt

Anonymes Filesharing : anonymes Anbieten und Herunterladen von Dateien.

Soziales Filesharing : nicht anonymes Filesharing innerhalb von sozialen Gruppen.

Hybrid Filesharing Tauschen ein und derselben Datei sowohl innerhalb einer Gruppe als auch anonym im Internet, sodass beim Download nicht nur private Peers sondern auch öffentliche Peers 11genutzt werden können.

Als Machbarkeitsstudie soll das System zeigen dass der Ansatz des Hybriden Filesharings praktikabel ist, und gleichzeitig als Prototyp dienen um die dafür notwendigen Konzepte zu studieren.

2 Grundlagen

Das in dieser Arbeit beschriebene System nutzt einige erprobte Konzepte und Systeme um die formulierten Ziele zu erreichen. Da das Verständnis für dieser unerlässlich ist, um zu verstehen wie Onion^{Core} funktioniert, und sie bei der Beschreibung der Funktionsweise nicht weiter erklärt werden, sondern ohne weitere Erklärung genutzt werden, folgt hier eine Einführung in die für Onion^{Core} wichtigsten Systeme und Konzepte.

2.1 BitTorrent

Das BitTorrent Protokoll beschreibt eine Verfahren zur effizienten dezentralen Verteilung von Daten. Um eine möglichst schnelle Verbreitung der Daten zu erreichen, werden diese nicht am Stück an einen einzigen Interessenten (Peer) geschickt, sondern Daten werden in Stücke (sog. *Chunks*) aufgeteilt, und diese *Chunks* an verschiedene Peers verteilt. Ein Peer der Daten anbietet wird *Seeder* genannt. Wenn ein Peer einen *Chunk* vollständig erhalten hat, bietet er diesen *Chunk* auch anderen Peers zum Download an. So muss ein Peer nicht darauf warten, dass der Besitzer der Datei ihm ein bestimmten *Chunk* der Datei schickt, sondern kann es von anderen Seedern herunterladen, die genau diesen gesuchten *Chunk* schon empfangen haben. Dies kann im Idealfall dafür sorgen, dass der Besitzer einer Datei diese insgesamt nur ein einziges mal übertragen muss, um einer großen Menge an Peers die Daten zugänglich zu machen.

Um eine bestimmte Datei über BitTorrent beziehen zu können, ist es also notwendig Metainformationen, wie z.B. die Anzahl, Größe und Hashwerte der einzelnen *Chunks* zu kennen vgl.[1]. Diese Metainformationen liegen in Form von sog. Torrent-Dateien vor. Torrent-Dateien selber enthalten keinen Teil des eigentlich gewünschten Inhaltes, sondern nur die Informationen die nötig sind um die eigentliche Datei beziehen zu können.

Diese Torrent-Dateien werden normalerweise, abseits des BitTorrent Netzwerkes, über eine herkömmliche Webseite zum Download zur Verfügung gestellt. Im herkömmlichen Betrieb enthalten diese Torrent-Dateien zusätzlich zu den Metadaten der Datei noch die Adresse eines oder mehrere zuständiger *Tracker*, welcher die einzelnen Peers koordiniert, indem diese dem Tracker melden, welche *Chunks* eines bestimmten Torrents sie schon besitzen, bzw. welche Teile sie anbieten oder noch benötigen. Der Tracker sammelt diese Informationen, um einem Peer, der einen bestimmten *Chunk* benötigt, die Adresse eines Seeders mitteilen zu können, welcher den gesuchten *Chunk* schon besitzt. Weiterhin enthalten diese Torrent-Dateien den sog. *info_hash*, welcher aus dem Hashwert der Metadaten der Torrent-Datei besteht. Dieser Wert wird genutzt um eine bestimmte Torrent-Datei gegenüber anderen zu identifizieren.

2.2 Verteilte Hashtabelle

Eine verteilte Hashtabelle (Distributed hash table (DHT)) ist eine verteilte Datenstruktur die fast[3] ohne zentrale Stelle (z.B. zentraler Datenbankserver) auskommt, und trotzdem Informationen vom Format *Schlüssel* <-> *Wert* speichern, und öffentlich verfügbar machen kann. Das System basiert darauf, dass eine Menge an Speicherknoten jeweils für einen bestimmten Schlüsselbereich zuständig ist. Die Schlüsselbereiche werden dabei aus dem Wertebereich einer Hashfunktion berechnet, sodass bei der Nutzung einer gleichverteilten Hashfunktion Schlüssel gleichmäßig in alle Bereiche abgebildet werden und so die Lastverteilung gleichmäßig bleibt. Da ein DHT, wie jedes andere verteilte

[3]Ein Client der am DHT teilnehmen will muss trotz der verteilten dezentralen Struktur, mindestens die Adresse eines anderen Knotens kennen. Für diesen Zweck gibt es oft einzelne feste oder zentrale Knoten die solche Initialisierungsdaten bereitstellen

System, darauf ausgelegt ist den Beitritt und das unerwartete Ausscheiden von Speicherknoten zu tolerieren, ist nicht nur ein Speicherknoten für einen Schlüsselbereich zuständig, sondern mehrere Knoten halten die Daten für einen Schlüsselbereich.

Durch die Aufteilung in Schlüsselbereiche kann eine Anfrage anhand des Hashwertes des Schlüssels effizient an einen der für diesen Bereich zuständigen Speicherknoten geleitet werden. Solch eine verteilte Hashtabelle unterstützt meist nur zwei Funktionen

einfügen(Schlüssel, Wert)

Dieser Aufruf fügt den Wert zum Schlüssel zum DHT hinzu.

suchen(Schlüssel)

Dieser Aufruf sucht den Wert zum Schlüssel im DHT und gibt ihn zurück.

2.3 Asymmetrische Kryptographie

Die asymmetrische Kryptographie setzt im Gegensatz zur symmetrischen Kryptographie, nicht nur einen Schlüssel zur Ver- und Entschlüsselung ein, sondern zwei unterschiedliche Schlüssel. Bei diesen Schlüsseln spricht man von einem öffentlichen und einem privaten Schlüssel. Beide Schlüssel hängen mathematisch voneinander ab, dabei ist jedoch kein Schlüssel vom jeweils anderen ableitbar[4]. Diese Eigenschaft wird genutzt, um das Problem der vertraulichen Übertragung des Schlüssels, welche beim Einsatz von symmetrischer Verschlüsselung nötig wäre, zu umgehen. Hierbei wird eine Nachricht mit dem weltweit bekannten öffentlichen Schlüssel des gewünschten Empfängers verschlüsselt. Eine derart verschlüsselte Nachricht kann nur mit dem zum öffentlichen Schlüssel gehörendem privaten Schlüssel entschlüsselt werden(vgl. [2]).

Dadurch dass der öffentliche Schlüssel die Nachricht nicht entschlüsselt, besteht keine Notwendigkeit den privaten Schlüssel über vertrauliche Kanäle austauschen zu müssen. Der private Schlüssel darf allerdings keinesfalls bekannt werden.

Eine weitere Verwendung für das System der zwei voneinander abhängigen Schlüssel neben der Sicherstellung der Vertraulichkeit, besteht in der Verifikation der Integrität einer Nachricht bzw. der Authentizität des Absenders. Hierbei werden die Schlüssel entgegengesetzt zur Verschlüsselung eingesetzt. Dabei bildet der Absender einer Nachricht den Hashwert der Nachricht, und verschlüsselt diesen Hashwert mit seinem privaten Schlüssel[5]. Dieser so verschlüsselte Hashwert wird als Signatur bezeichnet. Die Signatur wird der eigentlichen Nachricht angehängt. Der Empfänger prüft, ob er die Signatur mit dem bekannten öffentlichen Schlüssel des Absenders entschlüsseln kann. Stimmt der so entschlüsselte Hashwert mit dem Hashwert der erhaltenen Nachricht überein, ist sichergestellt, dass die Nachricht nicht verändert wurde und der Absender im Besitz des zum öffentlichen Schlüssel passenden privaten Schlüssels ist. Denn wäre die Nachricht durch eine dritte Partei verändert worden,

[4]Da beide Schlüssel Mathematisch aufeinander basieren besteht zumindest die theoretische Möglichkeit den privaten Schlüssel aus dem öffentlichen Schlüssel abzuleiten. Nach heutigen Maßstäben spielt diese Möglichkeit jedoch in der Praxis keine Rolle da der Aufwand zum Ableiten des privaten Schlüssels nicht in sinnvoller Zeit zu erbringen wäre.

[5]Theoretisch kann auch die gesamte Nachricht mit dem privaten Schlüssel verschlüsselt werden, allerdings wäre es einem Empfänger der den öffentlichen Schlüssel des Absenders nicht kennt, nicht möglich diese zu entschlüsseln.

würde der vom Absender verschlüsselte Hashwert der Nachricht nicht mehr zur Nachricht passen, oder wenn die Signatur verändert wurde, diese sich nicht mehr mit dem öffentlichen Schlüssel entschlüsseln lassen.

2.4 Tor

Ziel des Tor-Projekts ist es auf der Transportebene anonyme Kommunikation zu ermöglichen. Das heißt, dass Tor für andere Anwendungen als transparenter Proxy[6] auftreten kann. Dabei werden die normalen Verbindungen abgefangen und über das Tor-Netzwerk geleitet, anstatt direkt ins Internet.

Zur Anonymisierung setzt das Tor-Projekt auf eine Form des *Onion-Routing*. Beim *Onion-Routing* werden Daten, anstatt direkt zum Empfänger geschickt zu werden, vorher über mindestens drei Zwischenstationen umgeleitet. Das Ziel dieser Umleitung ist, dass keine der Stationen jeweils gleichzeitig Absender und Empfänger kennen darf. Um diese Vertraulichkeit sicherzustellen, werden Datenpakete, welche über das Tor-Netzwerk laufen, mehrfach verschlüsselt. Dabei kann jede Zwischenstation jeweils nur die Schicht entschlüsseln, welche die für sie notwendigen Daten enthält. Aus dieser Art der Verschlüsselung in einzelnen Schichten entsteht auch der Name *Onion-Routing*.

Die Zwischenstationen welche das Tor-Netz nutzt werden in folgende Arten eingeteilt.

Eintrittsknoten (Entry-Node) sind die Stationen, zu denen sich ein Tor-Nutzer initial verbindet. Solch ein Eintrittsknoten kennt die Identität des Nutzers, kann jedoch, dank der Schichtweisen Verschlüsselung, nicht auf den Inhalt der eigentlichen Nachricht zugreifen, sondern nur auf die äußere Schicht der Verschlüsselung, welche lediglich Informationen enthält, wie der Eintrittsknoten die Nachricht an den Weiterleitungsknoten schicken soll. Der Eintrittsknoten kennt damit ausschließlich Absender und den Weiterleitungsknoten.

Weiterleitungsknoten (Relay-Node) sind die Stationen, welche Nachrichten von Eintrittsknoten erhalten und weiterleiten. Diese Weiterleitungsknoten können nun die zweite Verschlüsselungsschicht der Nachricht entschlüsseln, welche die Information enthält, zu welchem Austrittsknoten die Nachricht weitergeleitet werden soll. Ein Weiterleitungsknoten kennt also den Eintrittsknoten und den Austrittsknoten einer Nachricht.

Austrittsknoten (Exit-Node) sind die Stationen, welche Nachrichten aus dem Tor-Netz empfangen und ins offene Internet weiterleiten. Dabei erhalten sie die Nachrichten von einem Weiterleitungsknoten und können die letzte Verschlüsselungsschicht entfernen, welche die Informationen zum endgültigen Ziel einer Nachricht sowie den eigentlichen Inhalt enthält. Wenn solch eine Nachricht an einen Dienst im Internet weitergeleitet wird, erscheint der Austrittsknoten für den Dienst als Absender der Nachricht, ohne dass der Dienst erfährt, dass der eigentliche Absender das Tor-Netz genutzt hat. Ein Austrittsknoten kennt damit nur den Weiterleitungsknoten sowie das endgültige Ziel einer Nachricht.

[6] Ein Proxy ist eine Anwendung die als Zwischenstation für Netzwerkverbindungen auftritt und diese Verbindungen bzw. die darüber laufenden Daten verändern, umleiten oder auch blocken kann

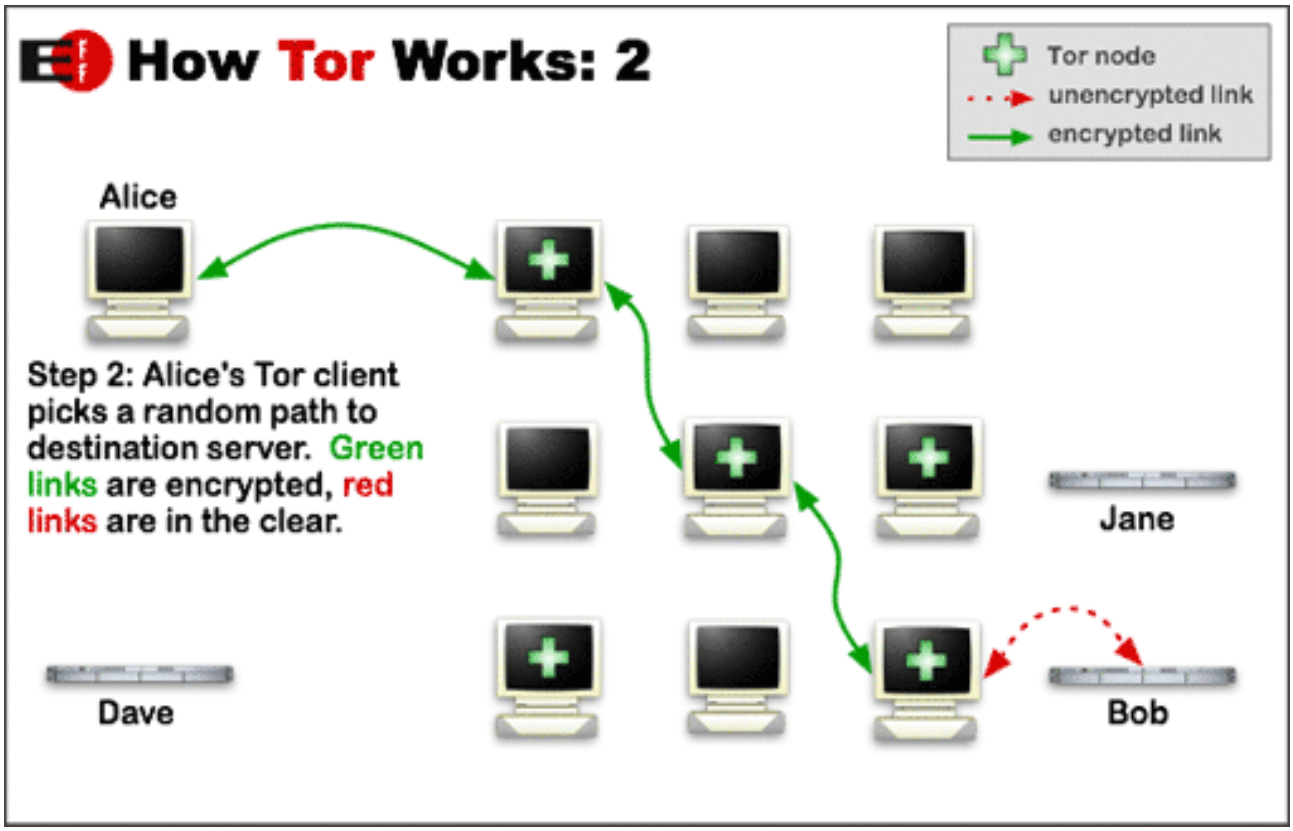

Abbildung 1: Funktionsweise einer Verbindung über das Tor-Netzwerk [3]

Bei der Nutzung des Tor-Systems muss aber beachtet werden, dass das Tor-Projekt nur die Zurückverfolgung des Weges, den Daten genommen haben, anonymisiert, die übertragenen Daten selbst jedoch nicht verändert oder anonymisiert werden. Das bedeutet dass Nutzer sich bewusst sein müssen, dass Anwendungen die Daten übertragen welche Informationen zur Identität des Absenders beinhalten (Login Namen, Adressen, Kontonummern etc.), nicht durch die Nutzung von Tor geschützt werden.

2.4.1 Versteckte Dienste

Das Tor-Projekt bietet neben der Anonymisierung von Datenverkehr ins normale Internet noch die weitere, für diese Arbeit wichtige Funktion, sog. versteckte Dienste anzubieten. Bei einem versteckten Dienst handelt es sich um einen Dienst, der nur von innerhalb des Tor-Netzes erreichbar ist. Datenverkehr von einem Client zu einem verstecken Dienst verlässt also niemals das Tor-Netzwerk. Bei einem versteckten Dienst erzeugt der Dienstanbieter (Server) zuerst ein Schlüsselpaar bestehend aus einem privaten und einem öffentlichen Schlüssel2.3. Aus diesem Schlüsselpaar leitet der Server eine *.onion*-Adresse ab[7]. Zusätzlich zum Schlüsselpaar wählt der Server einige Weiterleitungsknoten als Einführungspunkte aus. Danach werden die gewählten Eintrittspunkte zusammen mit der *.onion*-Adresse dem Tor-Netzwerk und der Dienst somit allen bekannt gemacht.

[7]Eine .onion-Adresse kann innerhalb des Tor-Netzes wie eine URL genutzt werden

Wenn nun ein Client sich mit einem verstecktem Dienst verbinden will, baut er eine Tor-Verbindung zu einem der bekanntgemachten Einführungsknoten auf. Dorthin sendet der Client Informationen zu einem von ihm gewählten Treffpunkt-Knoten. Der Einführungsknoten leitet diese Informationen nun an den versteckten Dienst weiter. Dieser baut nun seinerseits eine Tor-Verbindung zum erhaltenen Treffpunkt-Knoten auf. Danach können Client und Dienst miteinander kommunizieren. Solch eine Verbindung besteht aus insgesamt sechs Zwischenstationen, ohne dass dabei Verkehr über einen Austrittsknoten ins offene Internet weitergeleitet wird. Von diesen sechs Knoten wurden drei vom Client gewählt, die anderen 3 vom Dienst. Diese Strategie sorgt für starke Anonymität, da auf diese Weise keine bestimmte feste Gruppe von Knoten für die eigentliche Kommunikation zwischen Client und Dienst verantwortlich ist.

3 Design von Onion^{Core}

Onion^{Core} wurde mit als Machbarkeitsstudie entwickelt, wobei aber alle genutzten Konzepte darauf ausgelegt sind, in erweiterter Form, auch in einem etwaigen Produktivsystem zu funktionieren. Onion^{Core} ist in JavaSE 8 implementiert und greift zur Anonymisierung des Datenverkehrs zwischen zwei Teilnehmern auf die bereits existierende Lösung zur Transportschicht Anonymisierung Tor zurück. Als Bibliothek welche die Funktionalität bereitstellt um Verbindungen über das Tor-Netz aufzubauen wird *silvertunnel*[4] verwendet. Für den Austausch von Metadaten, beispielsweise zur Organisation von PeerGroups oder anderer nicht direkt auf das Übertragen von Dateien bezogene Kommunikation wird ein Webservice(HTTP-Service3.1) verwendet. Nur über diese Komponente können andere Teile des Systems (außer Torrents und DHT) mit anderen Teilnehmern kommunizieren. Das zur Koordination notwendige DHT ist für diese Arbeit nur simuliert, und wird durch einen zentralen Server mit einem entsprechenden Interface dargestellt. Für das Filesharing über das BitTorrent Protokoll wurde die freie BitTorrent Bibliothek *BitLet*[5] eingebunden.

3.1 Systemarchitektur

Grundsätzlich besteht Onion^{Core} aus den Komponenten 3.1

GroupManager ist verantwortlich für die Verwaltung von PeerGroups und deren Mitgliedern sowie die Korrdination von Gruppeneinladungen3.5.1.

HTTP-Service fungiert als Dienst, bei dem alle nicht Torrent bezogenen Anfragen an einen Onion^{Core} Client ankommen, verarbeitet und danach an die verantwortliche Komponente weitergeleitet werden3.5.3

TorManager ist verantwortlich für die Verwaltung und Generierung des nötigen Schlüsselpaares3.2, sowie für die Bereitstellung von `ServerSockets` für verstecke Dienste und den Aufbau von Verbindungen zu anderen versteckten Diensten über deren HiddenAddress.3.3

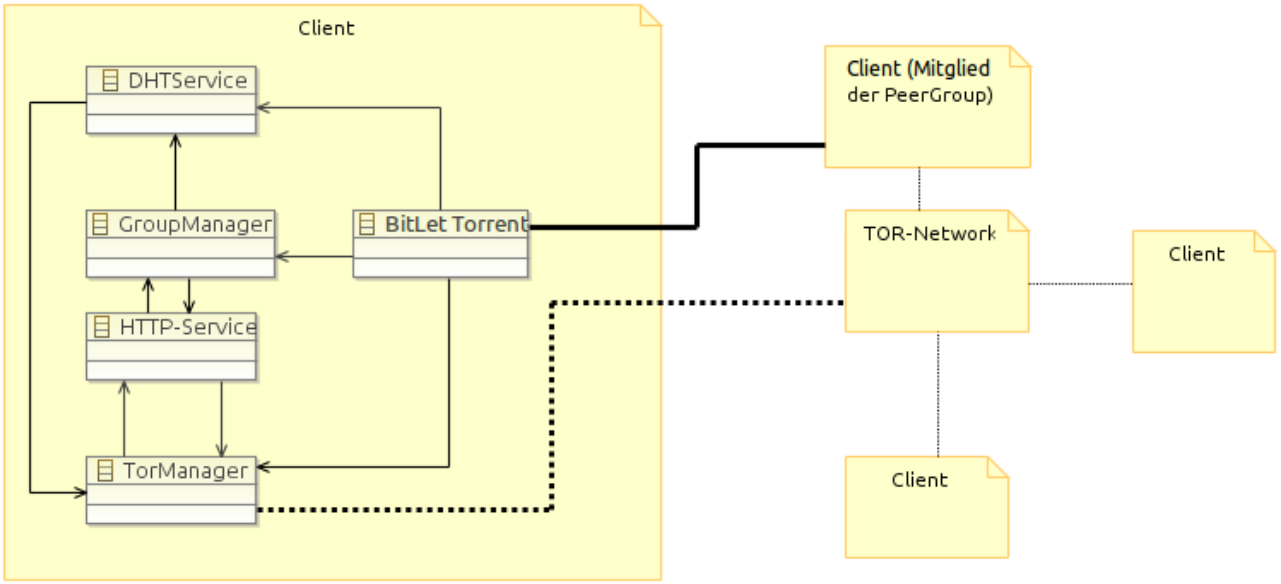

Abbildung 2: Architekturbild Onion^{Core}

BitLet ist die genutzte Torrent-Bibliothek welche alle Funktionen zur Kommunikation zwischen zwei Torrent Peers, zum Austausch von Dateien, bereitstellt.3.7

DHTService ist die Komponente welche Onion^{Core} vom unterliegenden DHT-Protokoll abstrahiert, und in diesem System ein globales DHT simuliert.3.4

3.2 Schlüsselverwaltung

Viele Onion^{Core} Abläufe verlassen sich stark auf Verfahren der asymmetrischen Kryptographie. Weiterhin sind auch für viele Funktionen des Tor-Netzes entsprechende Schlüsselpaare erforderlich. Dabei greift Onion^{Core} für alle Funktionen auf ein einziges Schlüsselpaar zu. Dieses Schlüsselpaar wird beim ersten Start des Systems automatisch generiert und dann zur Verwendung in späteren Sitzungen, in einer Datei abgelegt und gespeichert. Sollte so eine Datei beim Start von Onion^{Core} schon vorhanden sein, wird das Schlüsselpaar aus dieser Datei geladen und genutzt. Bei der Generierung des Schlüsselpaares wird auf die Tor-Bibliothek zurückgegriffen.

3.3 Tor Verbindungsverwaltung

Der Aufbau von Verbindungen über das Tor-Netz ist sehr Zeit- und Rechenintensiv, besonders Verbindungen zu versteckten Diensten aufzubauen ist aufgrund der Anzahl der Nachrichten, die zum Aufbau nötig sind, sowie die Menge an verschiedenen involvierten Zwischenstationen, sehr Zeitintensiv.

Zusätzlich zum AUfwand bei der Herstellung einer Verbindung, besteht auch noch das Problem dass die genutzte Tor-Bibliothek manchmal keine Verbindung zu einem versteckten Dienst aufbauen kann.

Um beide Probleme abzumildern, nutzt Onion^{Core} beim Aufbau von Verbindungen über das Tor-Netz einen Cache, welcher einmal bestehende Verbindungen zu einem Client für eine gewisse Zeit zwischenspeichert und bei Bedarf eine schon bestehende Verbindung herausgibt, anstatt eine neue Verbindung aufzubauen. Dadurch kann das Aufbauen von Tor-Verbindungen oft vermieden werden, und so die Anzahl an kritischen Phasen minimiert werden.

3.4 Vereinfachtes DHT

Als DHT kommt bei dieser Arbeit ein einfacher zentraler Key-Value-Store zum Einsatz, der über eine Tor Verbindung angesprochen werden kann. Da das DHT nicht der Schwerpunkt dieser Arbeit ist wurde bei diesem Teil auf eine Authentifizierung der Peers verzichtet. Unter Produktivbedingungen muss ein Peer nachweisen, dass die Adresse, die er zu einem bestimmten Schlüssel einträgt wirklich seiner Kontrolle unterliegt. Gleiches gilt für das Entfernen von Peers aus der Liste. Ansonsten könnten wenige *böse* Clients alle Peers aus der DHT entfernen und so das gesamte Netzwerk lahmlegen. Solch eine Authentifizierung ist in der offiziellen Spezifikation von *MainlineDHT* allerdings bereits vorgesehen.

3.5 PeerGroup

Eine PeerGroup ist die Einheit, in der Clients, die sich gegenseitig vertrauen, zusammengefasst werden. Jeder Client kann dabei Mitglied in beliebig vielen verschiedenen Gruppen sein. Diese PeerGroups stellen dabei die Trennung zwischen vertrauenswürdigen und nicht vertrauenswürdigen Clients dar, und dienen somit als Grundlage zur Entscheidung ob eine Verbindungsanfrage über eine nicht anonyme Verbindung für einen Torrent angenommen werden darf oder nicht. Solch eine Gruppe ist ein fester Zusammenschluss von Clients, deren Nutzer sich entweder persönlich kennen, oder sich zumindest gegenseitig soweit vertrauen, dass die Anonymität gegenüber den anderen Mitgliedern aufgegeben wird.

3.5.1 Aufbau einer PeerGroup

Eine PeerGroup besteht aus einer Liste aller Mitglieder sowie einer eindeutigen Kennung[8]. Ein Gruppenmitglied selbst besteht dabei aus dem öffentlichen Schlüssel des entsprechenden Clients sowie dessen HiddenAddress**??**. Zusätzlich versuchen die Gruppenmitglieder die IP-Adressen der anderen Gruppenmitglieder aktuell zu halten, indem sie die entsprechenden Mitglieder über ihre HiddenAddress kontaktieren und um die aktuellen IP-Adressen bitten.

Zur einfacheren Unterscheidung der einzelnen Gruppen durch den Nutzer kann noch ein beliebig wählbarer Name vergeben werden, dieser Name ist nur lokal bekannt und wird nicht an andere Mitglieder weitergegeben. Jede PeerGroup enthält zusätzlich zur Kennung ihrer Mitglieder noch eine

[8] Hierbei kommt `java.util.UUID` zum Einsatz um eine weltweit eindeutige Kennzeichnung zu generieren

Liste an ausgesprochenen gültigen noch offenen Beitrittseinladungen. Anfänglich besteht eine Gruppe nur aus dem Client, der diese auch erstellt hat.

Zwischen Mitgliedern einer Gruppe gibt es keine Unterscheidung in Bezug darauf welche Rolle sie darstellen oder welche Rechte diese haben. Diese Eigenschaft führt dazu dass, jedes Mitglied einer Gruppe andere Clients zur Gruppe einladen und hinzufügen kann. Das führt allerdings auch dazu dass die Nutzer einander wirklich vertrauen sollten, da ein einziger bösartiger Client, der es schafft Mitglied zu werden, die gesamte Gruppe deanonymisieren kann.

3.5.2 Aufnahme eines neuen Mitglieds

Als Mittelpunkt der gesamten Vertrauensverwaltung muss der Beitritt eines Clients zu einer Gruppe als besonders kritisch betrachtet werden. Denn sollte ein nicht Vertrauenswürdiger Client in die Gruppe aufgenommen werden, so könnte dieser leicht alle anderen Mitglieder deanonymisieren. Das gleiche gilt für den Client der einer Gruppe beitreten will, sollte dieser der falschen Gruppe beitreten so kann er deanonymisiert werden.

Zum Beitritt einer Gruppe bedarf es einer Einladung eines Mitglieds. Da besonders im Umfeld Sicherheit und Verschlüsselung der Mensch die größte Fehlerquelle ist, ist es hier sinnvoll die Interaktion mit dem Benutzer möglichst gering zu halten. Daher enthält eine Einladung auch keinerlei Informationen über die eingeladenen Client. Dies ermöglicht die Vereinfachung der Kommunikation zwischen den Benutzern, da so kein fehleranfälliger Austausch von HiddenAddresses notwendig ist, sondern nur eine einmalige Nachricht mit einem kurzzeitig einmal gültigem beliebig langem und komplexen Schlüssel.

Spricht ein Client eine Einladung aus, generiert er, wie im Sequenzdiagram 3.5.2 zu sehen ist, zuerst einen *nonce*. Daraus erstellt er ein Key-value Paar aus dem Hash des *nonce* als Schlüssel und der HiddenAddress, unter der er zu erreichen ist. Dieses Paar trägt er ins DHT ein. Den *nonce* muss der Benutzer, welcher die Einladung ausgesprochen hat, auf sichere Weise an den Bewerber weitergeben, der beitreten möchte. Dabei ist es wichtig, dass die Weitergabe des *nonce* auf sicherem Wege abläuft, denn der *nonce* berechtigt den Besitzer zum Gruppenbeitritt.

Wenn nun der Bewerber den *nonce* erhalten hat, kann sein Client im DHT nach dem Hashwert des *nonce* suchen und bekommt dann die HiddenAddress des Einladenden heraus. Da der Client jetzt Kenntnis der HiddenAddress hat, kann er eine Beitrittsanfrage an diese Adresse schicken. Dieser Anfrage fügt der Bewerber den nur ihm (und demjenigDiese Anfrage enthält zur Verifizierung der Berechtigung en der die Einladung ausgesprochen hat) bekannten *nonce* an. Der Einladende kann nun anhand dieser Anfrage überprüfen ob der Absender wirklich den *nonce* kannte, ist dies der Fall gilt die Anfrage als verifiziert und der Bewerber wird in die Gruppe aufgenommen indem dessen PublicKey als Mitglied gespeichert wird. Zusätzlich schickt der Einladende auch alle ihm bekannten Mitglieder der Gruppe an den Bewerber.

Dieser Ansatz hat allerdings zwei schwere Sicherheitsprobleme, die ausgenutzt werden können um den Bewerber sowie alle anderen Mitglieder der Gruppe zu identifizieren.

Da alle Einträge im DHT öffentlich einsehbar und veränderbar sind, ist es zwar für einen Angreifer nicht möglich, sich den Gruppenbeitritt zu erschleichen, denn der dafür nötige *nonce* ist ja nur als Hashwert sichtbar[9]. Allerdings wäre es für einen Angreifer möglich den Eintrag im DHT so zu verändern, dass er die originale HiddenAddress durch seine eigene ersetzt. Wenn dann der Bewerber seine Beitrittsanfrage abschicken würde, würde diese Nachricht beim Angreifer ankommen, welcher dann den *nonce* kennt und sich damit eine Mitgliedschaft in der ursprünglichen Gruppe erschleichen kann. Gleichzeitig wäre er auch in der Lage dem Bewerber gegenüber einen erfolgreichen Gruppenbeitritt zu simulieren und sich so gegenüber dem Bewerber als vertrauenswürdiges Mitglied ausgeben kann und damit den Bewerber zu identifizieren.

Um diese Art von Angriff zu verhindern, wird der DHT-Eintrag einer Einladung nun etwas verändert. Hier wird der Einladende nämlich nun als Wert des Eintrages nicht nur seine HiddenAddress eintragen, sondern zusätzlich auch noch eine verhashte Version seiner HiddenAddress mit dem Klartext *nonce*.

```
1 String securedAddress = hash(nonce+hiddenAddress)
DHT.put(hash(nonce),securedAddress+":"+hiddenAddress);
```

Listing 1: Gesicherter Gruppeneinladungs DHT Eintrag

Diese Art des Eintrages versetzt den Bewerber in die Lage zu überprüfen, ob der die HiddenAddress ins DHT eingetragende Client den *nonce* kannte oder nicht, und ob die eingetragene Adresse nicht nachträglich durch einen Angreifer verändert wurde.

Das ist möglich da ein Angreifer zwar den Eintrag dahingehend verändern könnte, dass die Klartextadresse auf ihn zeigt, jedoch würde der Bewerber dabei feststellen, dass die Klartextadresse zusammen mit dem Klartext *nonce* verhasht dann nicht mehr der *securedAddress* entspräche, und damit den Vorgang abbrechen kann bevor Schaden entsteht. Gleiches gilt für den Fall in dem der Angreifer den gesamten Eintrag so verändert dass dieser auf ihn zeigt, aber auch hier kann der Angreifer ohne Kenntniss des *nonce* den *securedAddress* Teil nicht gültig nachbilden da dann der *securedAddress* Hash zwar zur Klartext hiddenAddress passt, jedoch nicht zum Klartext *nonce* passt.

Solange der Bewerber also den DHT Eintrag verifiziert, kann ein Angreifer zwar den Beitritt verhindern, indem er alle Daten ungültig macht, aber er kann auf diese Weise keine Gefahr für die Anonymität der anderen Gruppenmitlgieder oder des Bewerbers darstellen. 3.5.2.

[9] Unter der Vorraussetzung, dass eine starke Hashfunktion genutzt wird und der *nonce* nicht durch Rainbowtables angreifbar ist.

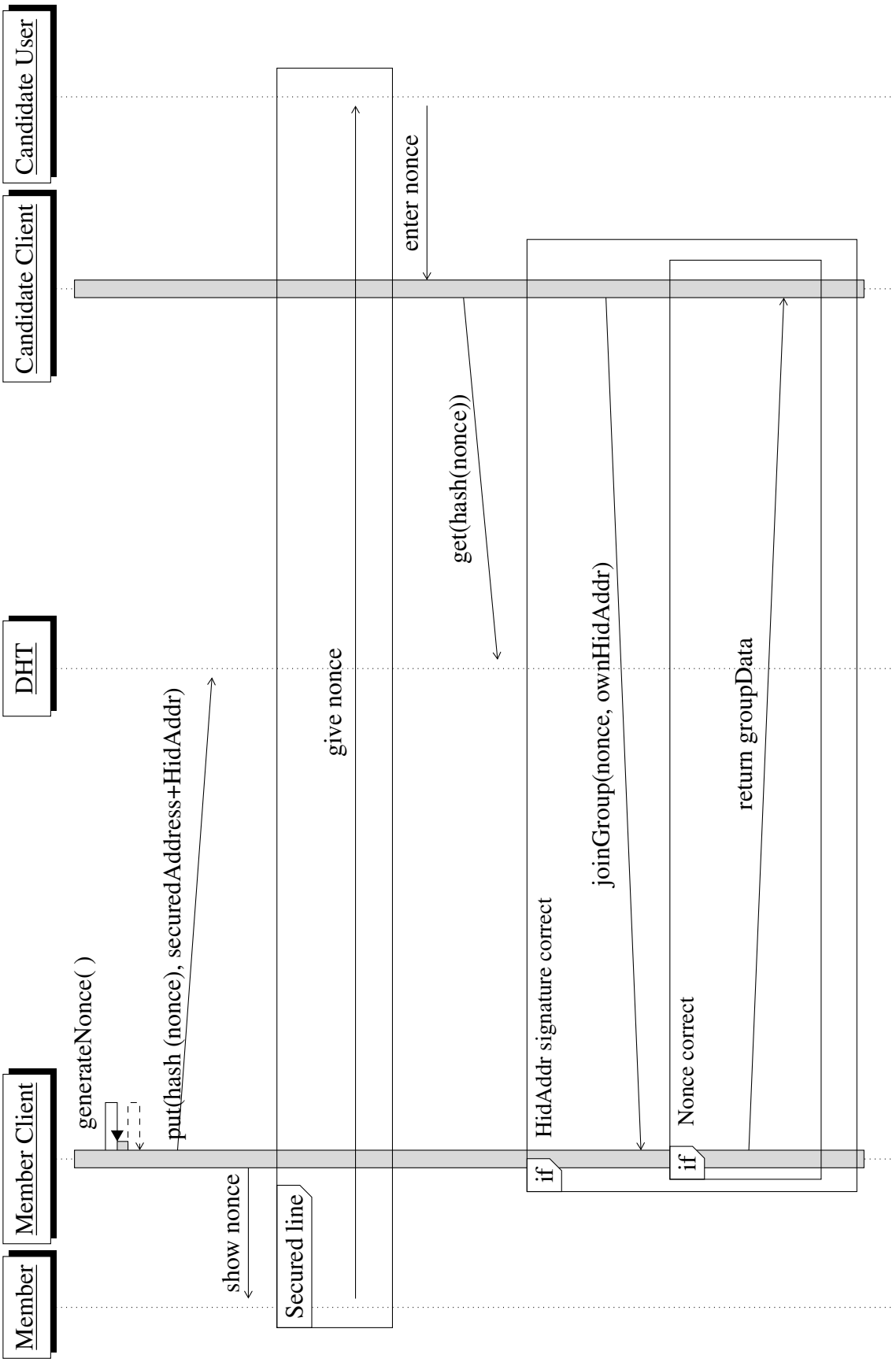

Abbildung 3: Sequenzdiagramm zur Aufnahme eines Mitgliedes in eine Gruppe.

3.5.3 Kommunikation und Koordination

Damit alle Mitglieder den möglichst aktuellen Zustand ihrer Gruppe kennen, ist ein gewisses Maß an Kommunikation nötig. Da permanent Gruppenmitglieder online und offline gehen bzw. ihre IP-Adressen ändern, erfolgen alle Anfragen bzgl. der Gruppenkommunikation über anonyme Tor-Verbindungen über die HiddenAddresses der Clients. Dies hat den Vorteil, dass es außer dem Tor-Netzwerk keinen zentralen Server geben muss bei dem alle Mitglieder ihre IP-Adresse hinterlegen um andere Clients erreichen zu können.

Innerhalb einer Gruppe kommunizieren die Mitglieder über ein polling System miteinander. Dabei fragen sie sich gegenseitig nach z.B. ihrer aktuellen IP anstatt von sich heraus bei einer Änderung den neuen Zustand an alle Mitglieder zu schicken. Unter der Annahme, dass sich die Mitglieder einer bestehenden Gruppe weniger oft ändern als die IPs, kann so die Häufigkeit der Kommunikation eingeschränkt werden, da ein Client die IP eines anderen nur dann abfragt wenn diese auch wirklich benötigt wird. Der Nachteil bei diesem polling System ist allerdings, dass nach der Aufnahme eines neuen Mitgliedes nur das Mitglied, welches die Einladung ausgesprochen hat, von dem Beitritt weiß, während die anderen Mitglieder erst bei ihrer nächsten Aktualisierungsanfrage davon erfahren. Eine einfache Lösung hierzu wäre z.B. die Einführung eines neuen Nachrichtentypens, mit welchem ein Mitglied ein anderes Mitglied entweder auffordern kann eine Aktualisierung durchzuführen oder möglicherweise den neuen Zustand direkt in die Nachricht schreibt.

Als weitere Einschränkung im Rahmen dieser Arbeit können Mitglieder einer Gruppe diese weder verlassen noch von anderen entfernt werden. Durch diese Entscheidung können bei Gruppenupdates keine Konflikte auftreten bzgl. welche Mitglieder die Gruppe verlassen haben bzw. hinzugekommen sind. Um die Mitgliederliste eines Clients zu aktualisieren schickt dieser eine Aktualisierungsanfrage an die HiddenAddress aller Mitglieder. Dabei antworten alle Clients mit der ihnen bekannten Mitgliederliste. Sollten sich dabei die empfangenen Mitgliederlisten unterscheiden vereinigt der Empfänger alle Listen, um so eine möglichst vollständige Liste zu erhalten.

Eine andere Art der Aktualisierung ist die Frage nach der aktuellen IP-Adresse eines Mitglieds im Fall dass diese für die Torrentverbindung gebraucht wird. Hierbei schickt das fragende Mitglied eine Aktualisierungsanfrage an die HiddenAddress eines gewünschten Client. Dieser antwortet daraufhin mit einer Nachricht welche die aktuelle IP-Adresse enthält.

3.5.4 Authentifizierung von Anfragen

Damit Clients nur Anfragen von vertrauten Stellen annehmen, muss eine Authentifizierung von Nachrichten bzw. deren Absendern sichergestellt sein. Auch hierbei kommt wieder asymmetrische Verschlüsselung zum Einsatz. Da jedes Gruppenmitglied die öffentlichen Schlüssel jedes anderen Mitgliedes kennt[10], können Nachrichten durch eine digitale Signatur des Absenders identifiziert werden. Im Rahmen dieses Projektes kommt eine etwas vereinfachte Version der Authentifizierung zum Einsatz. Hierbei signiert der Absender einer Nachricht nur die HiddenAddress des Empfängers mit sei-

[10] unter der Vorraussetzung dass alle Gruppenmitglieder die aktuellste Mitglieder Liste besitzen

nem privaten Schlüssel und hängt diese zusätzlich zu seiner eigenen HiddenAddress an die Nachricht an. Dabei überprüft der Empfänger, ob die in einer Anfrage enthaltene Signatur, seiner eigenen HiddenAddress zum öffentlichen Schlüssel des Absenders passt. Mit der angefügten HiddenAddress des Absenders wird dessen öffentlicher Schlüssel innerhalb der PeerGroups gesucht. Sofern die digitale Signatur der Nachricht zum, aus der PeerGroup bekannten, öffentlichen Schlüssel passt, gilt die Nachricht als authentisch und wird beantwortet. Diese vereinfachte Art der Authentifizierung hat den Nachteil, dass, sollte eine Nachricht eines bestimmten Absenders an einen bestimmten Empfänger aus welchen Gründen auch immer, abgefangen werden, kann sich der Angreifer daraufhin gegenüber dem ursprünglichen Empfänger als valides Gruppenmitglied ausgeben und diese eine Anfrage immer wiederholen. Diese Angriffsart wird als *Replay-Angriff* bezeichnet bei dem aufgezeichnete Daten wiederholt gesendet werden um eine falsche Identität vorzutäuschen.

Diese Einschränkung könnte umgangen werden indem entweder eine Art fortlaufende Variable mit in die digitale Signatur eingefügt wird z.B. der aktuelle Zeitstempel, sodass die aufgezeichnete digitale Signatur ihre Gültigkeit verloren hat, wenn der Angreifer sie versucht nochmals zu senden. Eine andere Möglichkeit wäre auch das verschlüsseln der Antwort mit dem öffentlichen Schlüssel des Empfängers, sodass ein Angreifer sich womöglich als gültiger Empfänger ausgeben könnte, dann aber die erhaltenen Daten nicht entschlüsseln könnte. Hierbei würde der Empfänger jedoch trotzdem nicht merken, dass er gerade mit einem Angreifer kommuniziert, anstatt mit dem eigentlichen Gruppenmitglied. Ein weiterer Weg, der auch diesen Nachteil nicht hätte, dafür aber mehrere *Round Trips* und damit die Wartezeit erhöht, wäre eine Art *Handshake Protokoll*, bei dem ein Client eine Handshake-Anfrage an ein anderes Gruppenmitglied schickt, welches daraufhin eine zufällige Zahl als Antwort schickt, und der Absender diese dann mit in die digitale Signatur einfügen muss. Dadurch könnte der Empfänger sicherstellen, dass er wirklich zur Zeit mit dem Besitzer eines gültigen privaten Schlüssel kommuniziert3.5.4.

Im Rahmen dieses Projektes wird bewusst auf eine extra Transportverschlüsselung für Nachrichten verzichtet die über das Tor-Netzwerk laufen, da alle Verbindungen innerhalb des Tor-Netzwerks über eine Transport Verschlüsselung vgl.Transport Layer Security (TLS) gesichert werden. Sollte jedoch eine explizite Transportverschlüsselung notwendig werden, so kann dafür die schon vorhandene Infrastruktur der bekannten öffentlichen Schlüssel der Mitglieder genutzt werden. *info_hash*

3.6 Torrent Verwaltung

Diese Arbeit basiert auf einem trackerlosen BitTorrent System welches sich über DHT organisiert. Dies hat den Vorteil dass es keine zentrale Stelle gibt deren Funktion für das Netzwerk notwendig wäre. Bei Onion^{Core} trägt jeder Peer seine HiddenAddress für jeden Torrent den er entweder bereitstellt, oder an dem er interessiert ist, ins globale DHT ein. Als Schlüssel wird dabei der eindeutige *info_hash* der Torrent-Datei genutzt. Onion^{Core} unterscheidet dabei nicht zwischen Torrents die nur innerhalb einer PeerGroup getauscht werden sollen, und solchen die öffentlich verteilt werden dürfen. Dabei wird sich darauf verlassen dass die Torrent-Datei der Daten welche nur innerhalb der Grup-

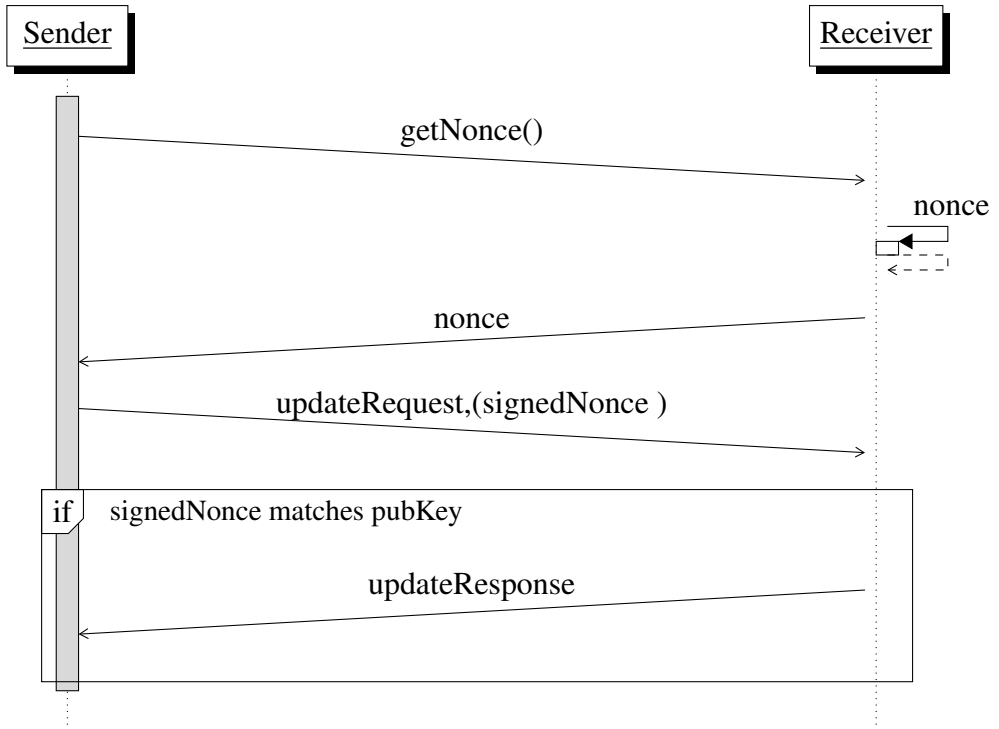

Abbildung 4: Sequenzdiagramm zum möglichen Ablauf von Gruppenkommunikation.

pe getauscht werden sollen, nicht an Gruppenexterne Peers weitergegeben wird. Diese Limitierung könnte allerdings dadurch umgangen werden, indem zusätzlich zur Liste aller Gruppenmitglieder noch eine Liste mit den *info_hash* der Torrents gespeichert wird, welche nur innerhalb der Gruppe getauscht werden sollen. Beim späteren Verbindungsaufbau zur eigentlichen Datenübertragung könnte dann der gesendete *info_hash* mit der Liste der privaten Torrents der Gruppe abgeglichen werden. Sollte dieser Torrent in der Liste der privaten Torrents enthalten sein und die Anfrage von einem nicht Gruppenmitglied stammen, würde der Download verweigert.

3.6.1 Torrent Findung

Um Dateien über BitTorrent zu beziehen, ist es notwendig die Torrent-Datei mit den Metainformationen über die eigentlichen Daten, zu besitzen. Beim herkömmlichen FileSharing wird dabei darauf zurückgegriffen diese Torrent-Dateien auf normalen Webseiten zur Verfügung zu stellen. Dies ist ohne Probleme möglich, da diese Dateien eben nur Metadaten enthalten, aber keinen Aufschluss über die Art der zugehörigen Daten beinhaltet.

Auch Onion^{Core} hat keinen Mechanismus zur Verteilung von Torrent-Dateien, und basiert darauf dass Torrent-Dateien auf externem Wege verteilt werden, beispielsweise per Email oder öffentlich auf einer Webseite.

Denkbar wäre aber eine Auflistung von Torrent-Dateien die den Mitgliedern einer PeerGroup bekannt sind. Eine Möglichkeit wäre das Hinzufügen einer weiteren Art von Kommunikation zwischen zwei Mitgliedern einer Gruppe. Dabei könnte ein Mitglied ein anderes nach allen Torrent-Dateien fragen, welche dieses Mitglied kennt. Wenn das angefragte Mitglied nun in seine Antwort eine Lis-

te aller Torrent-Dateien anhängt müssten zumindest Torrent-Dateien zwischen Gruppenmitgliedern nicht manuell vom Benutzer ausgetauscht werden.

Dieses Konzept würde aber nur innerhalb von PeerGroups funktionieren, jedoch nicht für Torrent-Dateien die an Gruppen externe Peers weitergegeben werden sollen, da diese externen Peers alle Teilnehmer des Onion^{Core}-Netzwerks nach ihren Torrent-Dateien fragen müsste, was sich einerseits durch den großen Aufwand schwierig gestalten würde, und andererseits nicht funktioniert, da es kein globales Verzeichnis aller Onion^{Core} Teilnehmer gibt.

3.6.2 Peer Findung

Onion^{Core} nutzt einen dezentralen Ansatz zur Peer Findung. Wie auch bei herkömmlichen trackerlosen BitTorrent Systemen kommt ein DHT zum Einsatz, um Seeder für einen bestimmten Torrent ausfindig zu machen.

Um Seeder für einen hinzugefügten Torrent zu finden, wird eine Suchanfrage ans DHT geschickt, als Schlüssel kommt dabei der *info_hash* der Torrent-Datei zum Einsatz. Da sich jeder Peer der einen bestimmten Torrent anbietet vorher mit der eigenen HiddenAddress für diesen Torrent ins DHT eingetragen hat, wird eine Suchanfrage zum Torrent die Adressen der anderen Peers zurückgeben. Dieses Konzept ist angelehnt an den Entwurf des MainlineDHT Protokolls vgl.[6].

Auf diese Weise können Peers gefunden werden die einen Torrent anbieten. Allerdings beinhaltet diese Liste nur HiddenAddresses der Peers, die zwar einen Download über das Tor-Netz erlauben, welcher aber verglichen mit dem Datendurchsatz einer direkten Verbindung zu einem PeerGroup Mitglied langsam ist.4.2 Damit sich ein Peer möglichst mit einem Mitglied einer seiner PeerGroups, über eine normale Verbindung, verbindet, anstatt das langsamere Tor-Netz zu nutzen, gleicht der Peer die Liste an erhaltenen HiddenAddresses mit den bekannten HiddenAddresses der Mitglieder seiner PeerGroups ab3.5.1. Sollte sich unter den Adressen die Adresse eines Mitglieds befinden, wird eine Aktualisierungsanfrage über den Weg der Gruppenkommunikation an das entsprechende Mitglied geschickt, und bei Erfolg, der Eintrag aus der Liste der möglichen Peers für den entsprechenden Torrent, ersetzt durch einen Eintrag welcher die echte IP-Adresse des Mitgliedes enthält. Die so modifizierte Liste wird verwendet um Verbindungen für den Datentransfer zu den jeweiligen Peers aufzubauen.3.7

Diese Art der Peer Findung hat das Problem, dass sich alle Mitglieder einer Gruppe für einen Torrent ins globale DHT Eintragen. Damit kann es für einen Angreifer möglich sein die zu einer Gruppe zusammengehörenden HiddenAddresses ausfindig zu machen, indem er im DHT nach mehreren Torrents sucht, die jeweils nur von einigen ganz bestimmten Peers angeboten werden. Das stellt zwar kein direktes Problem für die Anonymität der Mitglieder dar, aber es gibt trotzdem unnötig viele Daten über eine PeerGroup und deren Mitglieder preis. Als Gegenmaßnahme und um die Menge an öffentlichen Informationen zu minimieren, war im ersten Ansatz dieser Arbeit vorgesehen, dass sich Mitglieder eine Gruppe, für einen Torrent der nicht öffentlich geteilt werden soll, nicht ins globale DHT eintragen. Stattdessen hätte ein Mitglied einer Gruppe das einen Torrent beziehen möchte, welcher nur innerhalb der Gruppe verteilt wird, eine Anfrage an alle Gruppenmitglieder geschickt, ob

diese den gewünschten Torrent anbieten. Dieser Ansatz wurde allerdings verworfen, nachdem Instabilitäten in der genutzten Bibliothek, welche Verbindungen über das Tor-Netz zur Verfügung stellt, aufgetreten sind. Damit ein insgesamt funktionierendes System entsteht wurde versucht die Anzahl an notwendigen Verbindungen über das Tor-Netz zu reduzieren.

3.7 Dateitransfer

Nachdem der Leecher[11] eine Liste verfügbaren Seedern erhalten hat, baut der Leecher eine Verbindung zu den Seedern auf. Dabei ist zu beachten dass sich in der Liste der Seeder sowohl anonyme Peers als auch vertraute Peers befinden können. Zu anonymen Peers können nur Verbindungen über das Tor-Netz aufgebaut werden, während die Verbindunge zu vertrauten Peers über das normale Internet stattfinden sollte. Zu erkennen ist dies an der Form der Adresse eines Peers in der Liste. Bei anonymen Peers steht dort nur die HiddenAddress, bei vertrauten Peers wurde die Adresse bereits in einem früheren Schritt 3.6.2 durch die echte IP-Adresse ausgetauscht.

Die Verbindungsanfragen die beim einem der Seeder ankommen, werden von Onion^{Core} aufgeteilt in Verbindungen die über das Tor-Netz kommen, und Verbindungen die über das normale Internet eingehen. Verbindungen die über das Tor-Netz kommen werden dabei immer sofort akzeptiert, da hier keine weitere Prüfung des Vertrauens stattfinden muss, da eine solche Verbindung kein Risiko bzgl. der Anonymität des Seeders darstellt. Bei Verbindungen aus dem normalen Internet jedoch muss geprüft werden ob die Gegenstelle ein Mitglied einer der eigenen PeerGroups ist. Onion^{Core} überprüft dies indem die Absender IP-Adresse der Datenpakete mit den bekannten IP-Adressen aller PeerGroup Mitglieder verglichen wird. Sollte die Adresse einem Mitglied gehören, so gilt die Verbindung as vertrauenswürdig und wird zugelassen. Sollte die IP-Adresse allerdings keinem der Mitglieder zugeordnet werden können, kann dies entweder auf einen Angriff hindeuten, oder ein Zeichen dafür sein dass die Informationen über die Gruppenmitglieder veraltet sind.

Diese Überprüfung der Absender IP-Adresse aus den Verbindungsdaten heraus hat den Nachteil angreifbar zu sein. Durch sog. IP-Spoofing könnte es einem Angreifer möglich sein, die IP-Adresse für versendete Pakete zu fälschen und so dem Empfänger vorzuspielen, dass die Verbindungsanfrage von einem vertrauten Mitglied kommt. Als Gegenmaßnahme kann auch hier, wie schon bei der Authentifizierung von Gruppenkommunikation auf die vorhandene Infrastruktur der bekannten öffentlichen- und privaten Schlüsseln zurückgegriffen werden, um z.B. eine Transportsicherheitsschicht wie TLS[12] über die Verbindung zu legen. Dadurch wäre gesichert, dass direkte Verbindungen aus dem normalen Internet nur von vertrauten Peers zugelassen wird.

Sobald eine Verbindung aufgebaut und akzeptiert wurde, ist der weitere Ablauf für anonyme und nicht anonyme Verbindungen identisch, da die Verbindungen nachdem sie akzeptiert wurden nicht

[11] In diesem Abschnitt wird der Peer der eine Datei herunterladen will, zum besseren Verständniss, als *Leecher* bezeichnet, ein Peer der Daten anbietet wird auch als *Seeder* bezeichnet

[12] TLS ist die Abkürzung für *Transport Layer Security* (ehemals SSL), eine Protokoll auf Transportschichtebene zur Verschlüsselung und Authentisierung auf Basis von asymmetrischer Kryptographie

mehr voneinander zu unterscheiden sind. Nachem die Verbindung zustande gekommen ist, sendet der Leecher dem Seeder den *info_hash* des Torrents an dem er interessiert ist. Nachdem der Seeder den *info_hash* bestätigt hat, tauschen beide Peers eine *Bitmap*[13] aus, welche Informationen darüber enthält welche *Chunks* eines Torrents ein Peer bereits vollständig besitzt. Nach Erhalt dieser *Bitmap* weiß der Leecher welche *Chunks* der Seeder anbietet, und kann diese mit den *Chunks* die er selbst schon besitzt abgleichen. Wenn ein *Chunk* gefunden wird, welches dem Leecher noch fehlt, der Seeder dies aber anbietet, so wird eine Anfrage nach genau diesem *Chunk* an den Seeder gesendet. Dieser kann daraufhin dem Leecher den Angefragten *Chunk* senden.

Im Allgemeinen entspricht der Ablauf der Kommunikation nachdem eine Verbindung zwischen zwei Peers aufgebaut und akzeptiert wurde, dem im BitTorrent Protokoll spezifizierten Ablauf [7]

4 Experimentelle Evaluation

Nachdem das oben beschriebene System Implementiert wurde, muss es auch auf seine Funktionsfähigkeit getestet werden, ob es für den vorgesehenen Einsatzzweck verwendbar ist und was für Leistungen zu erwarten sind.

4.1 Experimentelles Vorgehen

Um die Funktionsfähigkeit des Filesharings zu evaluieren, wurden mehrere Instanzen von Onion$^{\text{Core}}$ auf, jeweils eigenen, virtuellen Server gestartet. Als Anbieter wurde Amazon Web Services (AWS) gewählt, da dieser eine einfache Verteilung von virtuellen Servern in verschiedene, geographisch voneinander getrennte, Region ermöglicht[14]. Diese Möglichkeit zur geographischen Trennung der Server ist nötig, um sinnvolle Werte bzgl. der Geschwindigkeit des Filesharings zwischen, weit voneinander entfernte, Peers erhalten zu können.

Amazon Web Services bietet eine Vielzahl an virtuellen Servern mit verschiedenen verfügbaren Leistungsstufen an. Um eine Beeinflussung der Ergebnisse, durch die, auf dem jeweiligen Server, zur Verfügung stehende Leistung zu verhindern, wurden Vorab Test mit den verschiedenen Leistungsstufen durchgeführt. Dabei wurden sowohl Instanzen vom Typ `m4.large` mit Onion$^{\text{Core}}$ getestet, als auch Instanzen mit weniger Leistung vom Typ `t2.micro`. Nachdem bei diesen Vorläufigen Tests keine Unterschiede in der Leistung von Onion$^{\text{Core}}$ festgestellt werden konnten, wurden alle weiteren Experimente aus virtuellen Servern vom Typ `t2.micro` durchgeführt.

Zusätzlich zu den Instanzen welche für die Experimente vorgesehen waren, wurde noch eine zusätzliche Instanz in der Region *EU-central-1* (Frankfurt)[15] genutzt, auf der sich der Dienst befand, welcher einen Eintrittspunkt des globalen DHTs simuliert. Dabei wurde darauf geachtet, dass auch

[13] Eine *Bitmap* ist eine Abfolge von einzelnen voneinander unabhängigen Bits mit gleicher Bedeutung

[14] AWS bietet virtuelle Server sowhol in Nord- und Südamerika als auch im Europäischen und Asiatisch- Pazifischem Raum an

[15]

dieser Service, wie es auch bei der Nutzung eines echten DHT wäre, nur über seine HiddenAddress erreichbar ist.

Zum testen der Geschwindigkeit der Datenübertragung beim verteilen einer Datei über das Implementierte BitTorrent System, kam eine extern erstelle Torrent-Datei zum Einsatz (`small.zip.torrent`). Die Instanzen welche als Seeder fungieren sollten wurden sowohl mit dieser Torrent-Datei, als auch mit der zugehörigen eigentlichen Datei ausgestattet.[16] Dabei wurde darauf geachtet, dass die zu Übertragende Datei eine entsprechende Größe hat, damit Initialisierungseffekte, welche beim Starten des Downloads auftreten, die Messung nicht beeinträchtigen. Zusätzlich dazu wird die Messung der Geschwindigkeit erst gestartet, nachdem die Verbindung zu allen Verfügbaren Peers aufgebaut wurde. Zum Messen der Geschwindigkeit wird die Startzeitpunkt der Messung mit dem Zeitpunkt des Abschlusses des Downloads verglichen und die Dauer berechnet. Danach wird die Menge an übertragenen Daten durch die Dauer in Sekunden geteilt, um die Geschwindigkeit in Form $\frac{Byte}{Sekunde}$ zu erhalten.

4.1.1 Tor Geschwindigkeit innerhalb einer Region

Der Test, zum ermitteln der erreichbaren Übertragungsraten, zwischen zwei Peers die sich in der gleichen Region befinden, wurde in der Region *EU-central-1* durchgeführt.

Als Torrent kam `small.zip.torrent` zum Einsatz. Die Messreihen wurden für Konfigurationen mit einem Leecher und einer variablen Anzahl von Seedern durchgeführt. Dabei wurde die Geschwindigkeit beim Einsatz von einem Seeder über zwei Seeder, bis hin zu vier Seedern gemessen. Auf Grund von schon oben erwähnten Instabilitäten beim Verbindungsaufbau über das Tor-Netz konnten kein vollständige Messung zur Konfiguration mit fünf, oder mehr, Seedern durchgeführt werden.

4.1.2 Tor Geschwindigkeit zwischen Regionen

Zur Messung der Übertragungsraten zwischen zwei Peers über das Tor-Netz, über geographisch voneinander getrennten Regionen, wurde die das Angebot von virtuellen Servern in verschiedenen Regionen von AWS genutzt. Dabei waren die Seeder in der Region *EU-central-1* angesiedelt, während der Leecher in *US-west-2* (Oregon,USA) lokalisiert war.

Da bei Verbindungen über verschiedene Geographische Regionen mit einer Einschränkung der Datenübertragungsrate gerechnet werden muss, sollte zusätzlich zur Übertragung über das Tor-Netz auch die Übertragung über nicht anonyme Verbindungen getestet werden, damit ein Vergleich der Veränderung der Datenrate zwischen der Nutzung innerhalb einer Region, und der Nutzung über mehrere Regionen hinweg, angestellt werden kann.

Auch hier sollten mehrere Messungen mit verschiedener Anzahl an Seedern durchgeführt werden. Allerdings gab es, wie bei früheren Tests, Probleme beim Verbindungsaufbau über das Tor-Netz.

[16] Sowohl die Torrent-Datei als auch die eigentliche Datei, können in den beigefügten Daten im Ordner `Evaluation/` gefunden werden

Daher gelang jeweils nur eine Messung für Übertragungen über das Tor-Netz und über eine nicht anonyme Verbindung. Dabei stellte sich die Übertragung über eine nicht anonyme Verbindung als sehr viel Problemanfälliger heraus, da hierfür die Nutzung der PeerGroup Funktionalität benötigt wird. Zur Koordination von PeerGroups sind aber mehr Verbindungen über das Tor-Netz notwendig, als bei der Nutzung ohne PeerGroups.

4.1.3 Hybrides Filesharing

Beim Test des Betriebs des Hybriden Filesharings, bei dem sowohl nicht anonyme Verbindungen, zu Gruppenmitgliedern, als auch anonyme Verbindungen über das Tor-Netz, verwendet werden, geht es weniger um den erreichbaren Datendurchsatz, als um das Verhalten des Systems insgesamt. Durch die Art der Datenübertragung in einzelnen Teilen, ist es denkbar, dass durch die geringere Latenz der nicht anonymen Verbindung, diese immer den Vorzug bekämen bei der Koordination welcher *Chunk* von welchem Seeder heruntergeladen werden soll.

Leider sind auch hier keine Messungen möglich gewesen, da durch den höheren Koordinationsaufwand der PeerGroup-Infrastruktur, durch die Instabilitäten beim Verbindungsaufbau über das Tor-Netz, kein Zustand erreicht wurde indem das System hätte getestet werden können.

4.2 Ergebnisse

4.2.1 Geschwindigkeit über Tor

Die Messungen zur Geschwindigkeit der Datenübertragung über das Tor-Netz ergaben eine durchschnittliche Geschwindigkeit von etwa 50 Kilobyte pro Sekunde (kb/s) pro verbundenem Seeder. Die bei den Messungen aufgetretene Streuung der Übertragungsraten, von bis zu 35kb/s pro Verbindung, sind aufgrund der Struktur von Tor-Verbindungen zu versteckten Diensten, welche über sechs Zwischenstationen laufen, zu erwarten gewesen. Dabei muss beachtet werden, dass die maximal erreichbare Übertragungsrate einer Verbindung, der Geschwindigkeit der Zwischenstation mit der geringsten Übertragungsgeschwindigkeit entspricht.

In der Grafik 4.2.1 ist eine Steigerung der gesamt Übertragungsrate, bei zusätzlichen Peers, zu erkennen. Dabei verdoppelt sich die Übertragungsrate im Durchschnitt mit jeder zusätzlichen Verbindung. Durch das Konzept, dass beim Verbindungsaufbau zu versteckten Diensten, keine bestimmte Zwischenstation des Tor-Netz für einen versteckten Dienst verantwortlich ist, war zu erwarten dass sich die Übertragungsrate mit jeder zusätzlichen Verbindung (welche mit hoher Wahrscheinlichkeit keine der Zwischenstationen zweimal nutzt) nahezu verdoppeln. Leider konnten die Experimente nicht soweit fortgesetzt werden um eine obere Grenze dieses Zuwachses zu bestimmen. Solch eine obere Grenze könnte existieren, da sowohl der versteckte Dienst, als auch der Leecher, nur eine gewisse Anzahl an Eintrittknoten nutzen. Ob nach simultaner Nutzung aller Eintrittsknoten Verbindungen zu neuen Eintrittsknoten aufgebaut werden, wird von der Implementation der unterliegenden Bibliothek abhängen, und müsste weiter untersucht werden.

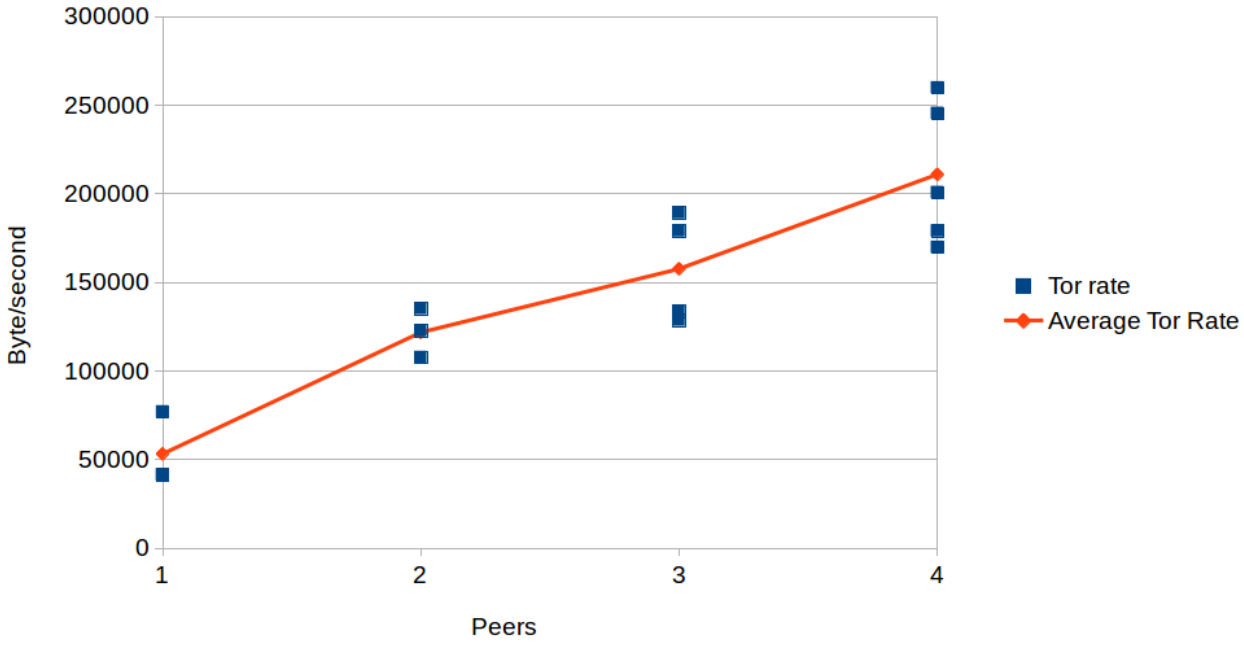

Abbildung 5: Diagramm der Übertragungsrate innerhalb Regionen

1 Seeder	2 Seeder	3 Seeder	4 Seeder
41424	107771	189369	200846
77068	122739	179133	169946
41554	135263	133897	259918
		128697	179133
			245478

Tabelle 1: Durchschnittliche Datenübertragungsgeschwindigkeit über Tor-Verbindungen in b/s, innerhalb einer Region

Die Messungen zur Übertragungsrate zwischen zwei Peers die sich gemeinsam in einer Peer-Group befinden, ergaben keine unerwarteten Ergebnisse. Es war zu erkennen, dass die Übertragungsgeschwindigkeit der Geschwindigkeit des Peers mit der geringsten Netzwerkanbindung entsprach. Daher wurden dazu keine weiteren Messungen durchgeführt.

4.2.2 Geschwindigkeit zwischen Regionen im Vergleich

Beim Messen der erreichbaren Übertragungsraten, wurde, wie gut im Diagramm 4.2.2 zu erkennen ist, die Erwartung, dass sich die Übertragungsraten bei geographisch verteilten Peers im Tor-Netz nicht signifikant verändern, erfüllt. Die Messung der Geschwindigkeit von direkt verbundenen Peers ergab jedoch dass die Geschwindigkeit bei dieser Art von Verbindung sehr durch die geographische Trennung beeinflusst wird. Bei einer direkten Verbindung zwischen zwei Peers, von dem sich einer in einem AWS Rechenzentrum in der Region *US-west-2*, und ein anderer in *EU-central-1* befand,

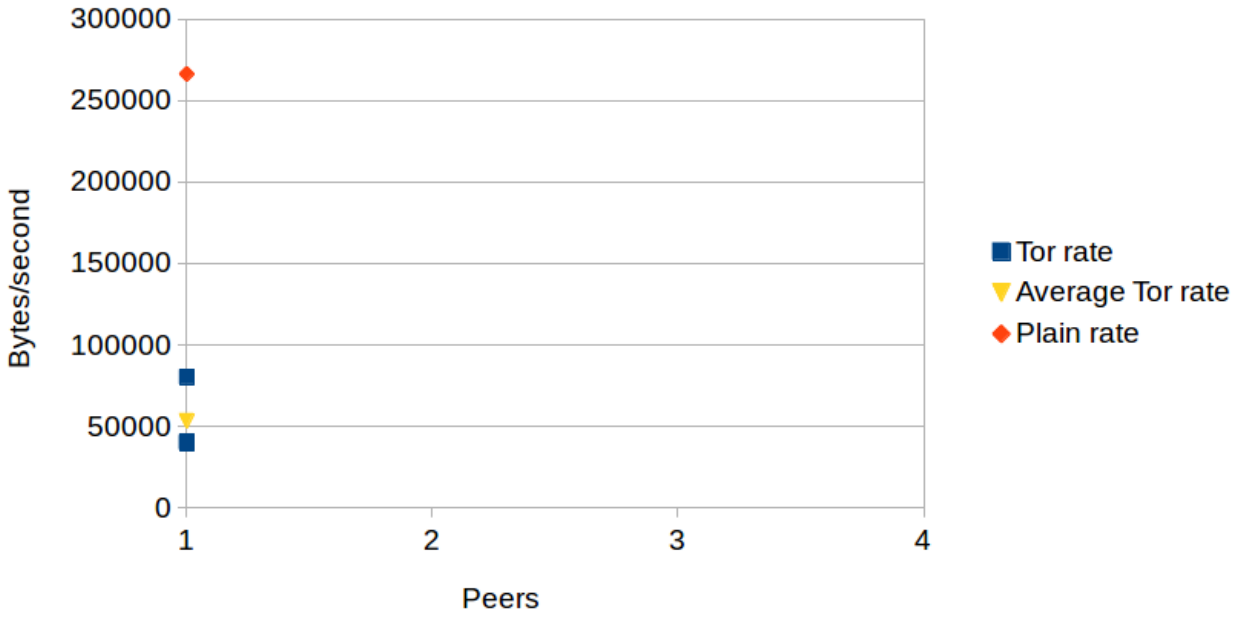

Abbildung 6: Diagramm der Übertragungsrate zwischen Regionen

1 Seeder	
Tor	Plain
40474	266240
80047	
39539	

Tabelle 2: Durchschnittliche Datenübertragungsgeschwindigkeit über Tor-Verbindungen im Vergleich zu nicht anonymen Verbindungen, in b/s, zwischen verschiedenen Regionen

konnte eine Übertragungsgeschwindigkeit von etwa 260kb/s bei Verbindung zu einem Peer gemessen werden. Die Übertragungsrate einer Verbindung über das Tor-Netz lag auch weiterhin bei etwa 50kb/s. Dadurch dass Tor-Verbindungen über sechs Zwischenstationen gerouted werden, welche sich auf der gesamten Welt befinden können, und eine Verteilung der genutzt Stationen, sowohl geographisch als auch in Bezug auf die Jurisdiktion in der sich eine Station befinden, durchaus gewünscht ist, ist erwartet worden, dass die Übertragungsgeschwindigkeit nicht wesentlich von der Region, in der sich die Endpunkte befinden, abhängt.

4.2.3 Praktikabilität des Hybriden Filesharings

Durch die Struktur der genutzen BitTorrent implementierung, die keine Unterscheidung zwischen Verbindungen über das Tor-Netz und direkten Verbindungen, vornehmen kann, sind im hybriden Betrieb die gleichen Übertragungsraten zu erwarten, welche sich aus den in 4.2.2 gemessenen Geschwindigkeit zusammensetzen. Diese These konnte jedoch auf Grund von Problemen beim Verbin-

Dateigröße	Anzahl vertrauter Peers	Anzahl anonymer Peers	Geschwindigkeit	Dauer
100MB	1	0	250kb/s	409 Sekunden
100MB	1	3	400kb/s	256 Sekunden

Tabelle 3: Beispielrechnung zur Veränderung der Downloadzeit beim hybriden Filesharing

dungsaufbau über das Tor-Netz nicht bestätigt werden. Es steht als einziges die Beobachtung zur Verfügung dass die Implementation des hybriden Filesharings, bei einem vorläufigen Test mindestens einmal wie erwartet funktioniert hat. Da dies bei einem vorläufigen Test beobachtet wurde, gibt es dazu leider keine Messdaten.

Durch die in vorherigen Tests gemessenen Geschwindigkeiten, und der Übereinstimmung der Ergebnisse mit den Erwartungen, kann die These aufgestellt werden, dass bei Übertragungen zwischen zwei Regionen, auch wenn direkte Verbindungen zu Mitgliedern der PeerGroup bestehen, es sinnvoll sein kann zusätzliche anonyme Verbindungen aufzubauen.

Wenn z.B. ein Onion$^{\text{Core}}$ Client in der Region *US-west-1* einen Torrent von etwa 100MB herunterladen möchte, und dafür die direkte Verbindung zu einem PeerGroup Mitglied aus der Region *EU-central-1* zur Verfügung steht, wird der Download (entsprechend der Ergebnisse aus 4.2.2) etwa sechs bis sieben Minuten Dauern. Stehen allerdings zusätzlich zu einem PeerGroup Mitglied, noch 3 weitere anonyme Peers zur Verfügung, würde sich die Zeit auf etwa 4 Minuten verkürzen (siehe Tabelle 4.2.3).

5 Verwandte Arbeiten

Es gibt viele Projekte die sich mit der Anonymisierung von Datenverkehr im Internet befassen. Dabei basieren die meisten Ansätze auf Abwandlungen des *onion routings*, bei dem Daten über mehrere Zwischenstationen gesendet werden, sodass es keine Stelle gibt, die sowohl Empfänger als auch Absender kennt. Ein sehr bekanntes Beispiel ist das Tor-Netzwerk. Dieses basiert komplett auf der Idee des *onion routings* und ist nur als eine weitere Netzwerkschicht gedacht, durch die anonymes Surfen im öffentlichen Internet realisiert wird vgl.[8].

Verwandt mit Tor ist das I2P Netzwerk welches *garlic routing*[17] nutzt. Allerdings geht I2P einen anderen Weg als Tor. I2P funktioniert als ein in sich geschlossenes Netzwerk, über das in der Regel nicht auf das öffentliche Internet zugegriffen werden kann (Ausnahmen hiervon bilden die sogenannten *outproxies* welche Verkehr aus dem I2P Netzwerk ins öffentliche Internet weiterleiten vgl. *exit-Nodes* im Tor-Netz). Sollen Dienste über dieses Netzwerk genutzt werden, dann müssen sie auch innerhalb des Netzwerks Angeboten werden (vgl. [9]).

Mehr Überschneidungen im Kontext von Anonymität und Filesharing hat das Freenet Projekt. Dort wird bezüglich des Konzepts auf I2P aufgesetzt und eine Funktionalität zum verteilten Speichern von Daten eingefügt. Dabei können Daten in das Netzwerk hochgeladen werden, welche dann auf verschiedene Peers verteilt werden. Hierbei besteht allerdings das Problem, dass die Peers nur

[17] Eine abgewandelte Form des *onion routing* bei dem mehrere Datenpakete in der innersten Schicht versendet werden

eine begrenzte Menge an Speicherplatz bereitstellen. Dies hat zur Folge, dass Daten die selten abgerufen werden mit der Zeit aus den Speichern der Peers verdrängt werden und irgendwann nicht mehr verfügbar sind (vgl. [10]).

Allerdings bieten all diese Lösungen keinen Ansatz für das hier besprochene Problem der variablen Anonymität anhand von Gruppenzugehörigkeiten. Denn die Geschwindkeitsverluste, die eine Anonymisierung mit sich bringt, sind teilweise überflüssig, wenn Verbindungen zwischen zwei vertrauten Peers genutzt werden.

Einen ersten Ansatz bietet hier das P^5-Projekt (P^5), bei dem der notwendige Grad an Anonymität innerhalb von Gruppen konfigurierbar ist vgl. [11]. P^5 bietet einen Schutz auch gegen passive Angreifer, welche den gesamten Datenverkehr aller Teilnehmer überwachen. Dabei setzt P^5 ein Verfahren ein, bei dem jeder Teilnehmer immer eine feste Anzahl von immer gleichgroßen Paketen versendet, unabhängig davon, ob gerade echte Informationen (Signal-Messages) übertragen werden sollen oder nicht. Wenn keine Daten versand werden, wird die fehlende Anzahl an Paketen durch das Versenden von Pakten ohne echten Inhalt (Noise-Messages) ersetzt, sodass die insgesamt Menge an Paketen konstant bleibt. Dabei werden alle Pakete über einen Broadcast-Kanal, welcher durch eine bestimmte Menge an Hops über andere Teilnehmer realisiert wird, an alle Teilnehmer gesendet. Durch die Wahl geeigneter Parameter für die Broadcast-Gruppe, wie z.B. die Anzahl an gewünschten Hops oder die Rate von Paketen, lässt sich die Anonymität an die eigenen Anforderungen anpassen.

Ein Projekt, was sich direkt auf die Anonymisierung von BitTorrent Verkehr konzentriert, ist das *BitBlender* Projekt. Auch hierbei wird zur Anonymisierung auf mehrere Zwischenstationen gesetzt. Allerdings gibt es zwei große Unterschiede zu z.b Tor oder I2P. Denn BitBlender setzt für die Verbindungen keine Verschlüsselung ein, da davon ausgegangen wird, dass nur öffentliche Dateien getauscht werden, sodass jeder Peer sehen darf, welche Daten er gerade überträgt. Dies hat zum Vorteil, dass Peers Daten cachen können und der Verschlüsselungs-overhead entfällt. Ein zweiter signifikanter Unterschied ist, dass Verbindungen bei BitBlender nicht zwangsläufig über mehrere Peers verlaufen, sondern theoretisch auch direkt vom Sender zum Empfänger gehen können. Durch die Struktur des Netzes ist dies aber weder für den Empfänger noch für den Absender erkennbar und der Sender kann glaubhaft Abstreiten, dass die Daten von ihm direkt stammen (vgl. [12]).

6 Fazit

Anonyme Zensur-resistente Datenübertragung im Internet hat meist den Nachteil, dass die erreichbaren Übertragungsraten, zu jeder Art von Teilnehmer, gering sind. Dieser Nachteil ist aber nicht in jedem Fall nötig, bei Übertragung von Daten an Teilnehmer denen vertraut werden kann, muss keine anonyme Verbindung genutzt werden. Wenn die gleichen Daten aber auch öffentlich anonym zur Verfügung gestellt werden sollen, werden heute meist zwei voneinander getrennte Systeme genutzt. Das in dieser Arbeit entwickelte $\text{Onion}^{\text{Core}}$ hat zum Ziel diese beiden voneinander getrennten Systeme zu vereinen. Dabei werden vertrauenswürdige Teilnehmer in PeerGroups zusammengefasst. Dabei können neue Teilnehmer, mit sehr wenig menschlicher Interaktion, zu einer PeerGroup einge-

laden werden. Dabei läuft der Datenaustausch unter den Mitgliedern einer Gruppe nicht anonym ab, sondern über direkte schnelle Verbindungen. Für die Kommunikation mit allen anderen Onion^{Core}-Teilnehmern, werden diese ausschließlich als versteckte Dienste über das Tor-Netz angesprochen. In den durchgeführten Experimenten konnte, obwohl es technische Probleme mit der genutzten Implementierung des Tor-Protokolls gab, gezeigt werden, dass die beschriebenen Konzepte valide sind. Besonders das Konzept des hybriden Filesharings ist zu beachten, da hier die Vorteile von anonymer Datenübertragung mit der überlegenen Geschwindigkeit von direkten nicht anonymen Verbindungen kombiniert werden, ohne dabei die Nachteile zu haben. Weiterhin wurden die Erwartungen an das Verhalten des Systems in verschiedenen Konfigurationen bei Übertragung über anonyme und nicht anonyme Verbindungen bestätigt. Besonders dahingehend, dass durch die Art der Verbindung, über 6 Zwischenstationen, im Tor-Netz, die Übertragungsgeschwindigkeit nicht von der geographischen Entfernung der beiden Endpunkte beeinflusst wird.

Da Onion^{Core} als Prototyp entwickelt wurde, wären zur Entwicklung eines im Produktivbetrieb einsetzbaren Systems, allerdings einige Detail-Veränderungen der eingesetzten Konzepte notwendig. Besonders ist hier zu sagen, dass zur Übertragung von großen Datenmengen das Tor-Netzwerk ungeeignet ist. Onion^{Core} ist dabei aber so angelegt dass das Unterliegende Anonymisierungsnetzwerk ausgetauscht werden kann, solange es die gleichen Konzepte, wie z.B. versteckte Dienste unterstützt. Weiterhin müssten weitere Untersuchungen durchgeführt werden, um sicherzustellen dass das System die Anonymität der Nutzer nicht ungewollt gefährdet. Zusätzlich sind einige Aspekte wie z.B das genutzte DHT nur simuliert, was das Verhalten des Systems im Betrieb beeinflussen könnte. Eine wichtige Änderung für ein Produktivsystem wäre das Gruppenmanagement und die Implementierung einer Erweiterung des Aufnahmeprotokolls, sodass z.B. mehrere Mitglieder über die Aufnahme eines neuen Mitgliedes abstimmen müssen. Überhaupt nicht beachtet ist hier ein Protokoll zum Ausschluss eines Mitgliedes oder das Verwalten von freiwilligen Austritten.

Zusammengefasst kann Onion^{Core}, mit seinen Konzepten, als Vorlage für ein weiterentwickeltes System genutzt werden.

Literatur

[1] Arun Chokkalingam and Firasath Riyaz. Bittorrent protocol specification v 1.0, 2004. 4

[2] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996. 5

[3] The Tor-Project. Tor: Overview. `https://www.torproject.org/about/overview.html.en`, Juni 2015. 7

[4] Tobias Boese. Silvertunnel-ng, java library for easy accessing tor network. `http://sourceforge.net/projects/silvertunnel-ng/`, März 2015. 8

[5] Alessandro Bahgat. Bitlet, a small java bittorrent library. `https://github.com/bitletorg/bitlet`, Juni 2015. 8

[6] Andrew Loewenstern and Arvid Norberg. Dht protocol. `http://www.bittorrent.org/beps/bep_0005.html`. 17

[7] Bram Cohen. The bittorrent protocol specification. `http://www.bittorrent.org/beps/bep_0003.html`, Juni 2015. 19

[8] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004. 24

[9] Bassam Zantout and Ramzi Haraty. I2p data communication system. In *ICN 2011, The Tenth International Conference on Networks*, pages 401–409, 2011. 24

[10] Ian Clarke, Oskar Sandberg, Matthew Toseland, and Vilhelm Verendel. Private communication through a network of trusted connections: The dark freenet. *Network*, 2010. 25

[11] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P 5: A protocol for scalable anonymous communication. *Journal of Computer Security*, 13(6):839–876, 2005. 25

[12] Kevin Bauer, Damon McCoy, Dirk Grunwald, and Douglas Sicker. Bitblender: Light-weight anonymity for bittorrent. 2008. 25

A Abkürzungsverzeichnis

P2P	Peer-to-Peer
Tor	The Onion Router
I2P	Invisible Internet Project
DHT	Distributed hash table
ISP	Internet Service Provider
P^5	P^5-Projekt
$Onion^{Core}$	$Onion^{Core}$
TLS	Transport Layer Security
AWS	Amazon Web Services