Bachelorarbeit

Soziale Komponenten für BitTorrent via BubbleStorm

Autor: Andreas Teuber

Prüfer: Prof. Dr. Alejandro P. Buchmann
Betreuer: Christof Leng

Version: 1.0
Darmstadt, 3. Juli 2012

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet Datenbanken und Verteilte Systeme (DVS)

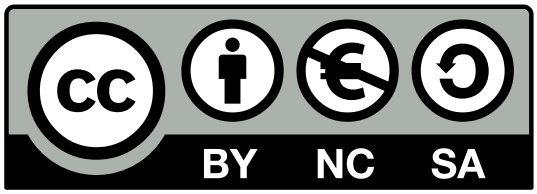
Erklärung zur Bachelor-Thesis

Hiermit versichere ich die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 3. Juli 2012

_____

(Andreas Teuber)



Inhaltsverzeichnis

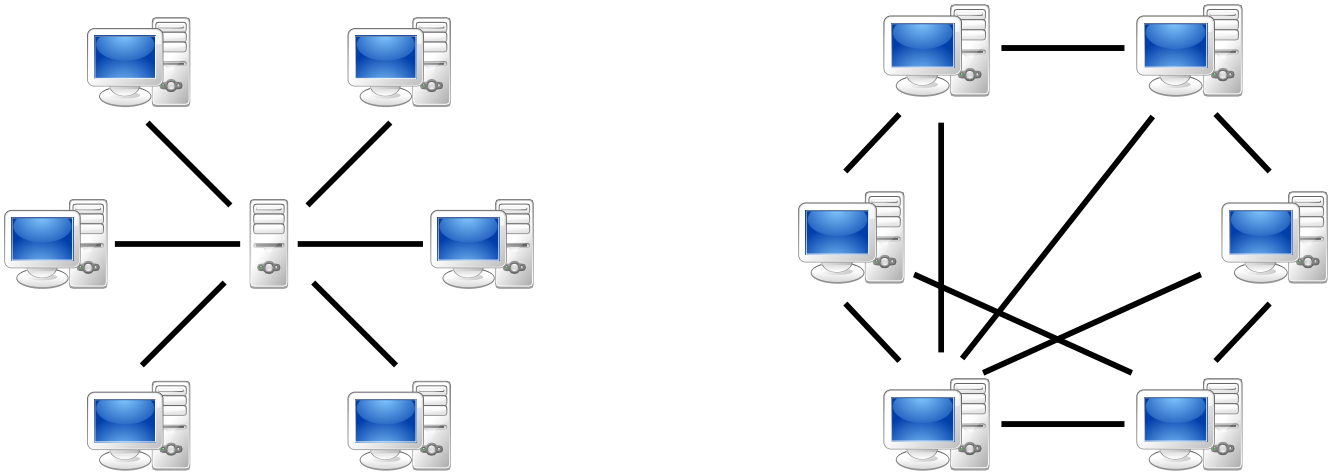
1 Einleitung

In heutigen Kommunikationsnetzen, insbesondere im Internet, spielt das Austauschen von Dateien eine bedeutende Rolle. Es existieren viele Lösungen dies zu ermöglichen, die sich aber in zwei Ansätze kategorisieren lassen.

Der erste Ansatz ist das Client-Server-Modell (siehe Abbildung 1.1 [a]), bei dem ein Dienstnutzer (Client) eine oder mehrere Dateien von einem Dienstleister (Server) anfordert, welcher diese Dateien daraufhin ausliefert. Dafür kommen meistens Protokolle zum Einsatz, die neben dem reinen Übertragen der Dateien auch noch andere Aufgaben übernehmen. Dies umfasst unter anderem Verfahren zur Fehlerkorrektur, Datenflusskontrolle, Verhalten bei Verbindungsabbrüchen, Fortsetzen von Übertragungen, Verschlüsselung und Überlastkontrolle. [12]

Der zweite Ansatz ist das Peer-to-Peer-Modell (P2P, siehe Abbildung 1.1 [b]), bei dem jeder Teilnehmer (Peer) gleichberechtigt ist, also gleichzeitig Dienstnutzer und Dienstleister ist. Existierende Peer-to-Peer-Systeme besitzen jedoch teilweise zentrale Elemente, wie beispielsweise Knoten, die eine wichtigere Rolle übernehmen als andere Knoten (sogenannte Supernodes) oder Server, welche die Suchfunktionalität im Netzwerk übernehmen. Man unterscheidet zwischen strukturierten und unstrukturierten Peer-to-Peer-Systemen. Erstere zeichnen sich dadurch aus, dass Objekte einem oder mehreren Peers deterministisch zugeordnet werden. Der Zugriff erfolgt über eine Lookup-Operation. Zusätzlich kommen häufig verteilte Hashtabellen [32] zum Einsatz, welche das Suchen über einen verteilten Index ermöglichen. Beispiele für strukturierte Peer-to-Peer-Systeme sind Chord [5], Kademlia [17] und Pastry [22]. [12] Bei unstrukturierten Peer-to-Peer-Systemen findet keine deterministische Zuordnung von Objekten zu Peers statt. Der Speicherort der Objekte ist vollständig von der Netztopologie entkoppelt, sie können auf einem beliebigen Peer liegen. Peers können im Netzwerk nach Objekten suchen, die gewisse Kriterien erfüllen. [12] Beispiele für unstrukturierte Peer-to-Peer-Systeme sind Gnutella [28], Freenet [6] und *BubbleStorm* (2.1) [26]. Manche unstrukturierten, rein dezentralen Peer-to-Peer-Systeme sind als Friend-to-Friend Netzwerk (auch *Web of Trust*-Netzwerk [1.2]) konzipiert. Bei dieser Art von Netzwerk werden nur Verbindungen zu vertrauenswürdigen anderen Peers aufgebaut. Ein Beispiel hierfür ist RetroShare [23].

Im Internet wird momentan häufig der Client-Server-Ansatz für die Auslieferung von Dateien verwendet. Dieser hat jedoch den Nachteil, dass bei hoher Nachfrage eine entsprechend hohe



(a) Client-Server-Modell [3] **(b)** Peer-to-Peer-Modell [2]

Abbildung 1.1.: Vergleich zwischen Client-Server-Modell und Peer-to-Peer-Modell

Bandbreite und leistungsfähige Hardware beim Server erforderlich ist. Die hierfür benötigte Infrastruktur bedeutet hohe Kosten für den Serverbetreiber. Bei Peer-to-Peer-Systemen werden die Kosten dagegen gleichmäßig auf alle Teilnehmer verteilt. Der Client-Server-Ansatz hat außerdem den Nachteil, dass der Inhalt des Servers bei einem Ausfall nicht mehr abrufbar ist. Ein rein dezentrales Peer-to-Peer-System besitzt keinen solchen Single Point of Failure[1], daher hat der Ausfall eines einzelnen Peers nur geringe Folgen. Eine immer größere Verbreitung finden Dienste zum Speichern von Dateien im Internet. Diese haben den Vorteil, dass derjenige, der eine Datei bereitstellt, nicht mit dem Internet verbunden sein muss, wenn ein anderer Benutzer die Datei abruft. Außerdem wird seine Verbindung nur einmal belastet, nämlich beim Hochladen an den Dienstanbieter. Jeder Abruf der Datei belastet ausschließlich die Verbindung des Dienstanbieters. Häufig werden zudem Funktionen angeboten, die über das reine Speichern von Dateien hinausgehen, beispielsweise Dateisynchronisierung und Benutzerverwaltung. Ein sehr beliebtes Programm mit diesen Funktionen ist Dropbox [8]. Demnächst beabsichtigt auch Facebook einen solchen Dienst anzubieten[2]. Bei Peer-to-Peer-Systemen gelangen Datenpakete zwischen verschiedenen Peers über unterschiedliche Routen zum Ziel. Dies kann die Netze entlasten, da weniger Daten über einzelne Netzsegmente übertragen werden. Ein Problem von Peer-to-Peer-Systeme ist jedoch, dass ein Peer sich mit anderen, oft unbekannten, Peers verbindet. Dies ermöglicht verschiedene Formen missbräuchlicher Nutzung, z. B. Angriffe auf einen Peer oder das Bereitstellen falscher Daten. Mittlerweile existiert sogar ein Startup, dessen Geschäftsmodell es ist, Dateiübertragungen über *BitTorrent* (1.1) zu unterbinden[3]. Aus diesem Grund werden in vielen Peer-to-Peer-Systemen Verfahren zur Einstufung der Vertrauenswürdigkeit eingesetzt.

1 SPOF: Teil eines Systems, dessen Defekt den Ausfall des gesamten Systems verursacht
2 http://www.gulli.com/news/18800-facebook-startet-filesharing-fuer-gruppen-2012-05-14
3 https://torrentfreak.com/microsoft-funded-startup-aims-to-kill-bittorrent-traffic-120513/

1.1 BitTorrent

BitTorrent [7] ist ein solches Peer-to-Peer-Netzwerk und hat laut ipoque-Internetstudie einen Anteil von 70% am Peer-to-Peer-Traffic und damit einen Anteil von 37% am gesamten Internet-Traffic in Deutschland. [25] Das BitTorrent-Protokoll ermöglicht den dezentralen Austausch von Dateien auf technisch einfache Weise und wird in vielen unterschiedlichen Anwendungsgebieten eingesetzt. Diese sind unter anderem das Verteilen von Software, Softwareupdates, kommerziellen und freien Medien jeglicher Art sowie der Austausch von privaten Daten zwischen Freunden.

BitTorrent baut für jeden Torrent ein separates Verteilnetz – Schwarm genannt – auf. Jeder Peer, der sich an diesem Schwarm beteiligt, lädt nicht nur Daten herunter, sondern verteilt die heruntergeladenen Daten schon während des Downloads automatisch auch an andere Peers im Schwarm. Dadurch kann die Geschwindigkeit des Downloads bei einer zunehmenden Anzahl Peers zunehmen, während sie mit dem Client-Server-Ansatz dann oft abnimmt.

Zum Herunterladen oder Verteilen von Dateien muss eine Torrent-Datei in den BitTorrent-Client geladen werden. Diese Datei enthält Meta-Informationen über die zu verteilenden Dateien, Prüfsummen über Datenblöcke fester Größe und eine Liste von Trackern zum Finden anderer Peers. Torrent-Dateien werden meistens auf Webseiten angeboten und sind über spezielle Suchmaschinen auffindbar.

Zum Auffinden anderer Peers kontaktiert ein Peer die in der Torrent-Datei eingetragenen Tracker. Ein Tracker ist meistens ein HTTP-Server, der eine Liste von Peers im Schwarm bereithält.

Sowohl für das Finden von Torrent-Dateien als auch zum Finden von anderen Peers wird also standardmäßig auf den Client-Server-Ansatz zurückgegriffen. Dies bringt die in der *Einleitung* (1) dieser Arbeit beschriebenen Probleme mit sich. Es existieren allerdings Ansätze, beide Funktionen dezentral zu realisieren. Ein neuer Ansatz wird im Abschnitt *Verteilte Suche für BitTorrent-Netzwerke* (2.3) beschrieben. Da es mit diesem Ansatz jedoch möglich ist, falsche, schadhafte oder beschädigte Torrent-Dateien und auch falsche Informationen über Peers zu verbreiten, wird er in dieser Bachelorarbeit erweitert (siehe *Ziel der Arbeit* [1.3]).

Der Datenaustausch beginnt, sobald einem Peer andere Peers im Schwarm bekannt sind. Zum Aufbau der Verbindung führen zwei Peers einen Handshake durch, bei dem sich die beiden Peers identifizieren. Dabei wird außerdem übertragen, um welchen Torrent es sich handelt, welche Protokollerweiterungen die Peers unterstützen und welche Teile des Torrents bereits

fertiggestellt wurden. Jeder Peer entscheidet anhand eines Algorithmus, in welcher Reihenfolge er angefragte Datenblöcke sendet und welche Datenblöcke er anfragt.

Die Details des BitTorrent-Protokolls werden in der Spezifikation [4] beschrieben.

1.2 Web of Trust

Mit Web of Trust (WoT) wird ein Konzept bezeichnet, welches in Pretty Good Privacy (PGP) [30], GnuPG [29] und anderen OpenPGP-kompatiblen [21] Systemen benutzt wird, um die Authentizität eines öffentlichen Schlüssels sicherzustellen, also dass der Schlüssel dem angegebenen Benutzer gehört. Dies geschieht über das gegenseitige Signieren von öffentlichen Schlüsseln und das Setzen von Vertrauenswerten (Owner Trust) für Schlüssel. Als öffentlichen Schlüssel bezeichnet man in einem asymmetrischen Kryptosystem einen Schlüssel, mit dem jeder Klartext verschlüsseln oder Signaturen verifizieren kann. Zu jedem öffentlichen Schlüssel existiert ein privater Schlüssel, mit welchem der Besitzer Chiffretext entschlüsseln oder Klartext signieren kann (also genau die Umkehrung der möglichen Operationen mit dem öffentlichen Schlüssel). Der öffentliche Schlüssel kann beliebig verbreitet werden, auf den privaten Schlüssel darf jedoch nur der Besitzer Zugriff haben. Zur Verbreitung von öffentlichen Schlüsseln existieren Schlüsselserver, auf die jeder seinen öffentlichen Schlüssel hochladen und öffentliche Schlüssel von anderen herunterladen kann. Neben den öffentlichen Schlüsseln können so auch Signaturen verbreitet werden. In der vorliegenden Bachelorarbeit wird eine dezentrale Alternative zu den Schlüsselservern entwickelt, welche auf BubbleStorm aufsetzt (siehe *Ziel der Arbeit* [1.3]).

Folgende Werte können als Owner Trust gesetzt werden:

- unknown: für Benutzer, über die man keine weiteren Informationen besitzt
- none: für Benutzer, denen nicht zugetraut wird, die Authentizität von anderen richtig zu prüfen
- marginal: für Benutzer, denen nicht voll zugetraut wird, die Authentizität von anderen richtig zu prüfen
- complete/full: für Benutzer, deren Signatur als genauso gut angesehen wird, wie eine eigene
- absolute: wird in der Regel nur für den eigenen Schlüssel genutzt

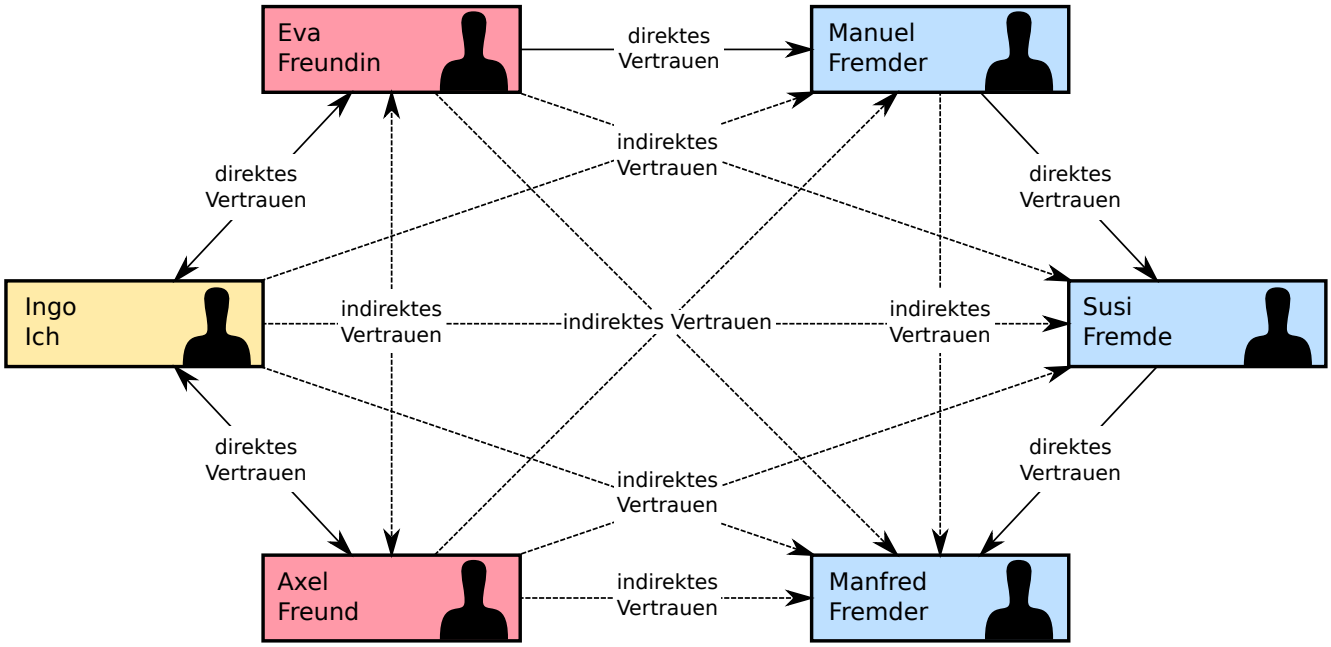


Abbildung 1.2.: Schematische Darstellung eines Web of Trust [20]

Die Basis für ein Web of Trust bilden transitive Vertrauensbeziehungen. Dies ist in Abbildung 1.2 veranschaulicht. Dort vertraut jeder Teilnehmer nur seinem unmittelbaren Nachbarn direkt (außer Axel Freund und Manfred Fremder), allen anderen aber indirekt über andere Teilnehmer am Web of Trust.

Neben dem Owner Trust, welcher Benutzern zugeordnet wird, gibt es den sogenannten Signatory Trust. Dieser bezieht sich auf Signaturen und wird nicht durch den Benutzern festgelegt, sondern ergibt sich aus den Owner Trusts. Dies ist in folgendem Beispiel erläutert: Signiert Alice den öffentlichen Schlüssel von Bob, so kann Carl diese Signatur benutzen, um die Authentizität des öffentlichen Schlüssels von Bob zu beurteilen. Hat Carl den öffentlichen Schlüssel von Alice selbst mit dem Owner Trust „marginal“ oder „full“ signiert, so erhält die Signatur von Alice genau diesen Wert. Hat Carl den öffentlichen Schlüssel von Bob selbst signiert, so erhält die Signatur von Alice ebenfalls den Signatory Trust „complete“. In allen anderen Fällen wird der Signatur der Signatory-Trust „none“ zugeordnet.

Ein öffentlicher Schlüssel K wird als gültig betrachtet, wenn er von genügend gültigen Schlüsseln signiert wurde. Dazu muss er entweder:

- selbst signiert worden sein,
- von mindestens einem Schlüssel signiert worden sein, dem voll vertraut wird
- oder von mindestens drei Schlüsseln mit „marginal“-Trust signiert worden sein.

Dabei darf der Pfad von signierten Schlüsseln vom Schlüssel K zum eigenen Schlüssel nicht länger als fünf Kanten sein.

Ein Web of Trust stellt eine dezentrale Alternative (von den Schlüsselservern abgesehen) zu den weit verbreiteten Public Key Infrastrukturen (PKI) dar, bei denen die Authentizität eines Schlüssels über eine Zertifizierungsstelle oder eine Hierarchie von Zertifizierungsstellen sichergestellt wird. Das Konzept wurde erstmals 1992 im PGP-Handbuch [35] von Phil Zimmermann beschrieben. Details sind im RFC 4880 zu OpenPGP [21] und im Manual von GnuPG [29] enthalten.

1.3 Ziel der Arbeit

In dieser Bachelorarbeit wird die Lösung zum Suchen und Verteilen von Daten, die Tilo Eckert in seiner Bachelorarbeit *Verteilte Suche für BitTorrent-Netzwerke* (2.3) [9] entwickelt hat, um soziale Komponenten erweitert. Für diese Komponenten wird das dezentrale Peer-to-Peer-Netzwerk BubbleStorm verwendet, sodass das gesamte System komplett dezentral bleibt. Die Arbeit ist ein Schritt hin zum Einsatz dieses Systems in der Praxis.

Eine Komponente soll es Benutzern erlauben, Torrents zu beglaubigen, zu bewerten und zu kommentieren. Mit der Beglaubigungsfunktion soll vermieden werden, dass Torrents verteilt werden, deren Name oder Beschreibung nichts mit den Dateien zu tun hat, die sie enthalten. Mit der Bewertungs- und Kommentarfunktion soll die Verbreitung von Torrents vermieden werden, die Dateien von schlechter Qualität oder Schadsoftware enthalten.

Um sicherzustellen, dass bei dieser Komponente keine Manipulationen vorgenommen werden, wird ein Web of Trust eingeführt. Dieses wird benutzt, um zu entscheiden, ob die Beglaubigung oder die Bewertung eines Benutzers in die Gesamtbewertung (fortlaufend bezeichnet als Rating) eines Torrents einfließen soll. Außerdem wird es dazu benutzt, um zu entscheiden, ob Kommentare anderer Benutzer zu einem Torrent angezeigt werden sollen. In dem Web of Trust dieser Bachelorarbeit werden die aus PGP bekannten Regeln benutzt, um zu bestimmen, ob Benutzer vertrauenswürdig sind (siehe *Web of Trust* [1.2]). Prinzipiell eignet es sich aber auch für die Implementierung aufwändigerer Reputationsverfahren. Zwei solcher Verfahren sind im Abschnitt *Reputationsmanagement* (3.5) beschrieben.

Mit dieser Arbeit wird das System außerdem an die Änderungen angepasst, die an BubbleStorm selbst geschehen sind. Torrents werden nun als Durable Bubbles (siehe *Durable Replication* [2.1.2]) veröffentlicht, sodass sie nicht mehr verloren gehen, wenn ein Peer das Netzwerk verlässt.

2 Vorarbeiten

2.1 BubbleStorm

Bei BubbleStorm handelt es sich um ein unstrukturiertes Peer-to-Peer-Netzwerk. Das Suchen erfolgt auf probabilistische und erschöpfende Weise. Da die Topologie von BubbleStorm strukturlos ist, richten Ausfälle von Netzwerkknoten keinen größeren Schaden an. In diesem Fall werden einfach neue Verbindungen zu anderen Peers aufgebaut und Daten auf weitere Peers repliziert. Suchanfragen sind somit weiterhin erfolgreich. BubbleStorm wird am Fachgebiet Datenbanken und Verteilte Systeme (DVS) des Fachbereichs Informatik der Technischen Universität Darmstadt unter der Leitung von Prof. Dr. Alejandro P. Buchmann entwickelt.

2.1.1 Funktionsweise

Die Topologie von BubbleStorm entspricht einem zufälligen Multigraphen. Dabei ist die Anzahl der Verbindungen eines Peers zu anderen Peers abhängig von der Bandbreite, mit der er mit dem Netzwerk verbunden ist. Da der zu erwartende Traffic auf allen Verbindungen identisch ist, lässt sich die Last bei abweichenden Bandbreiten mit der Schaffung einer unterschiedlichen Anzahl von Verbindungen verteilen.

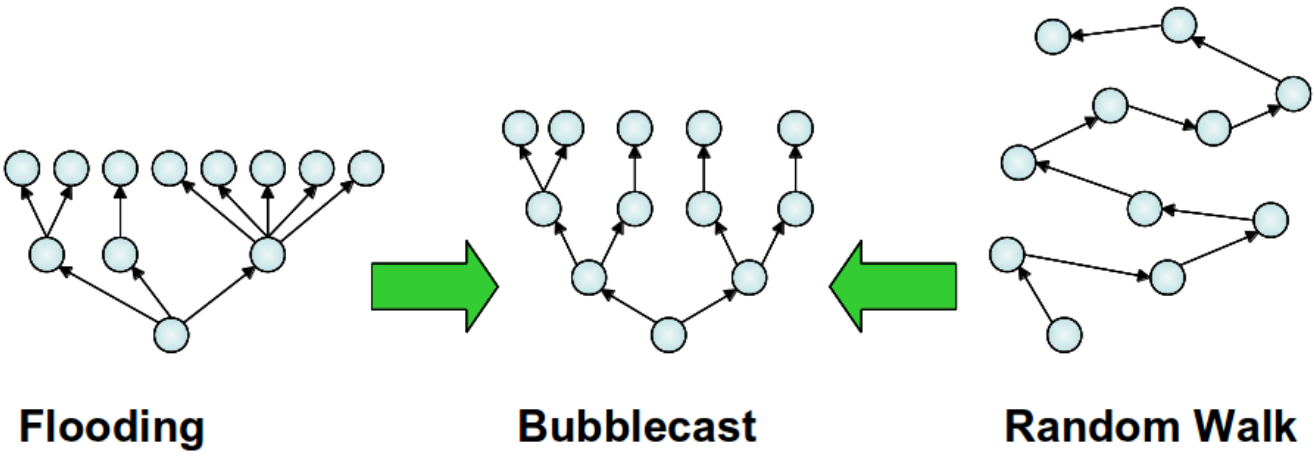


Abbildung 2.1.: Bubblecast im Vergleich zu Flooding und Random Walk [27]

Um Daten im Netzwerk verfügbar zu machen oder Suchanfragen durchzuführen, werden diese als sogenannte Bubbles auf einen Teil der Peers im Netzwerk repliziert. Bei diesem Vorgang, auch als Bubblecast bezeichnet, wird eine Mischung aus Random Walk und Flooding verwendet (siehe Abbildung 2.1). So wird die präzise kontrollierte Replikation des Random Walk mit der geringen Latenz des Floodings kombiniert.

Ein Bubblecast besteht aus den zu replizierenden Daten, einem Gewicht w für die Anzahl Replikate und einem Teilungsfaktor s. Empfängt ein Peer einen Bubblecast, so wird das Replikat lokal verarbeitet und w um eins dekrementiert. Ist w dann noch größer als Null, so wird ausgehend von diesem Peer ein weiterer Bubblecast an s zufällig ausgewählte Nachbarpeers gesendet, jedoch nicht an den Peer, von dem der nun sendende Peer den Bubblecast ursprünglich empfangen hatte. Außerdem wird bei vorhandenen Doppelkanten nur ein einziges Mal an den entsprechenden Nachbarn gesendet und im Falle einer Schleife mit dem eigenen Peer wird überhaupt nichts gesendet. Größere Schleifen werden nicht erkannt, führen also zu einer niedrigeren Replikatanzahl. Dies ist jedoch selten ein Problem, da sich bei Zufallsgraphen kaum lokale Schleifen bilden. Reines Flooding hätte an dieser Stelle den Nachteil, dass Peers, die Nachbarn von Peers mit vielen Kanten sind, mehr durchschnittlichen Traffic hätten, als Peers die keine oder weniger solcher Nachbarn haben.

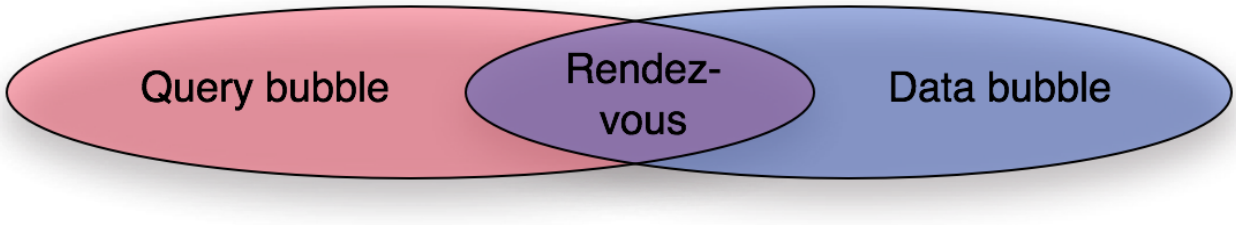


Abbildung 2.2.: Zusammenhang von Query Bubble, Data Bubble und Rendezvous-Peers [27]

Um Daten im Netzwerk zu veröffentlichen, startet ein Peer einen Bubblecast. Andere Peers speichern die so empfangenen Daten und replizieren sie weiter auf andere Peers. Alle Peers, auf denen schließlich die Daten gespeichert sind, werden zusammen als Data Bubble bezeichnet. Zur Durchführung einer Suche im Netzwerk, startet ein Peer ebenfalls einen Bubblecast. Peers, die diesen Bubblecast empfangen, gleichen die darin enthaltene Suchanfrage mit ihren lokal gespeicherten Daten ab. Wenn sich bei einem Peer ein Treffer ergibt, schickt dieser die Daten direkt an den Peer, der die Suche gestartet hat. Alle Peers, welche die Suchanfrage empfangen, werden zusammen als Query Bubble bezeichnet. Peers, die sowohl ein Teil der Query Bubble als auch der Data Bubble sind, nennt man auch Rendezvous-Peer (siehe Abbildung 2.2). Der Suchalgorithmus ist bei diesem Verfahren nicht definiert, sodass je nach Applikation unterschiedliche Algorithmen implementiert werden können.

In BubbleStorm gibt es keine Garantie dafür, dass eine Suche erfolgreich ist. Die Wahrscheinlichkeit P für eine erfolglose Suche lässt sich aber durch eine einfache mathematische Überlegung minimieren. Verteilt man eine Query Bubble der Größe q und eine Data Bubble der Größe d gleichmäßig durch Zufall auf n Peers, so gilt:

$$P < e^{\frac{-qd}{n}} \tag{2.1}$$

Man kann nun q und d so wählen, dass die Wahrscheinlichkeit klein genug ist, beispielsweise in der Art, dass $qd = 4n$ erfüllt ist. In diesem Fall beträgt die Wahrscheinlichkeit $e^{-4} \approx 0,018$. Diese Überlegung ist natürlich eine Vereinfachung, da die Bubbles in BubbleStorm nicht gleichmäßig durch Zufall verteilt werden. Die exakte Formel und deren Herleitung lassen sich im Abschnitt 2.4 des beschreibenden Papers nachlesen. [26]

2.1.2 Replikationsmodi

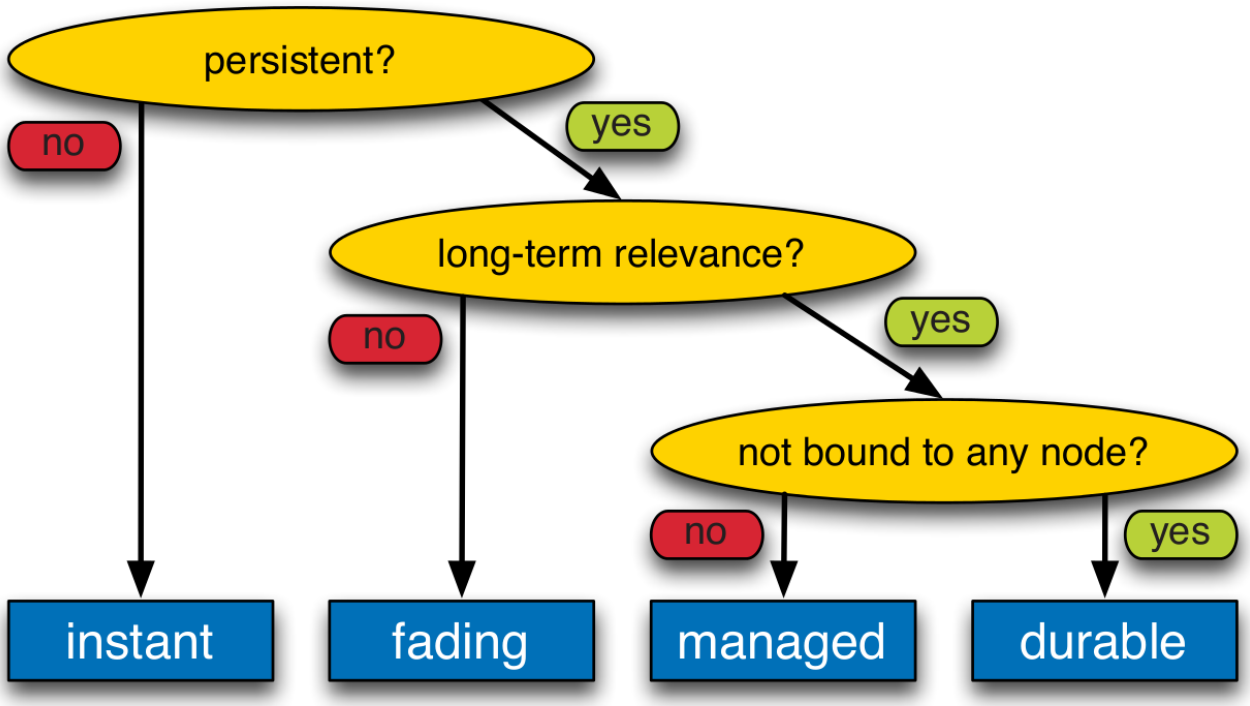


Abbildung 2.3.: Replikationsmodi in Peer-to-Peer-Systemen [16]

Mit Replikation bezeichnet man in einem verteilten System die Speicherung von Informationen auf mehreren unterschiedlichen Knoten. Dies wird in jedem Peer-to-Peer Suchoverlay benötigt, da sonst durch den Wegfall von Peers mit der Zeit die Indexstrukturen der Suche nicht mehr korrekt funktionieren würden. Durch die Replikation wird die Skalierbarkeit und die Trefferquote der Suche erhöht.

Im Kontext von Peer-to-Peer Suchoverlays zählen folgende Funktionen zu den Aufgaben der Replikation:

- Verbreiten: Speichern von Replikaten von neu erzeugten Objekten im Netzwerk.
- Verfügbarhalten: Sicherstellen, dass eine gewisse Anzahl Replikate existiert, auch wenn Peers das Netzwerk verlassen.
- Aktualisieren: Replikate aktuell und konsistent halten bei Aktualisierungen, Konflikte bei gleichzeitigen Aktualisierungen auflösen.

In BubbleStorm werden sowohl Daten repliziert als auch Suchanfragen. Dabei kommen verschiedene Replikationsmodi zum Einsatz, welche im Folgenden beschrieben werden. Der Entscheidungsbaum in Abbildung 2.3 veranschaulicht, worauf es bei der Auswahl des Replikationsmodus' ankommt. In der Tabelle 2.1 sind die Eigenschaften der Replikationsmodi zum besseren Vergleich übersichtlich dargestellt. Für jeden Replikationsmodus existiert in BubbleStorm eine Bubble Klasse. [16]

Modus	Verbreiten	Dauerhaft	Verfügbarhalten	Aktualisieren	Konfliktlösung	Anwendungsbeispiel
Instant	Ja	Nein	Nein	Nein	Nein	Query
Fading	Ja	Begrenzt	Nein	Nein	Nein	Caching
Managed	Ja	Begrenzt	Ja	Ja	Nein	Dateiliste
Durable	Ja	Ja	Ja	Ja	Ja	Wikiartikel

Tabelle 2.1.: Vergleich der Replikationsmodi in Peer-to-Peer-Systemen [16]

Instant Replication

Viele Objekte, z. B. Queries oder Publikationen in Publish/Subscribe-Systemen, müssen nicht gespeichert werden, sondern lösen nur etwas beim Empfänger aus oder werden direkt beantwortet. Für diese Objekte eignet sich der „Instant-Replikationsmodus". [16]

Fading Replication

Der „Fading-Replikationsmodus" eignet sich für Objekte, die gespeichert werden sollen, aber nicht verfügbar gehalten werden müssen. Ein Beispiel hierfür ist die Aktualisierung der Position

in einem Spiel. Außerdem bietet sich dieser Modus für Objekte an, die sowieso nach einer angegebenen Zeit gelöscht werden. [16]

Managed Replication

Der „Managed-Replikationsmodus“ kann benutzt werden, um Objekte dauerhaft zu speichern, dabei aber die Kontrolle über die Objekte zu behalten. Der Modus eignet sich beispielsweise zum Anbieten von Diensten (etwa eine Liste von freigegebenen Dateien) oder für Subscriptions in einem Publish/Subscribe-System. Das Verfügbarhalten und Aktualisieren des Objektes wird von einem Maintainer übernommen. Verlässt der Maintainer das Netzwerk, so ist das Objekt nicht mehr verfügbar. Es wird zwischen natürlichen Besitzern und dynamisch zugewiesenen Maintainern unterschieden. Erstere haben etwas mit dem Inhalt des gespeicherten Objektes zu tun, letztere übernehmen bloß die Verwaltung des Objektes. Maintainer sollten nur in zuverlässigen Umgebungen dynamisch zugewiesen werden und nicht in großen, für alle zugänglichen Overlays. In solchen kann es zu inkonsistenten Zuständen kommen, wenn der Maintainer wegen Churn[1] häufig wechselt. [16]

Durable Replication

Der „Durable-Replikationsmodus“ eignet sich für Objekte, die dauerhaft gespeichert und von beliebigen Peers verwaltet werden sollen. Ein Einsatzszenario hierfür sind Artikel in einem Peer-to-Peer Wiki oder öffentliche Schlüssel in einem Web of Trust. Diese sollen verfügbar bleiben, auch wenn alle Peers, die an ihnen etwas verändert haben, das Netzwerk verlassen haben. In Peer-to-Peer-Systemen, die eine verteilte Hashtabelle benutzen, lassen sich die Peers, die für ein Objekt zuständig sind, leicht finden, indem man einen Hash über das Objekt berechnet und einen Lookup durchführt. In unstrukturierten Peer-to-Peer-Systemen kann man alle Peers, die das Objekt replizieren, miteinander verbinden und Aktualisierungen an alle Peers in diesem Overlay durchreichen. Bei gleichzeitigen Änderungen muss der entstehende Konflikt gelöst werden. Um zu entscheiden, welche Änderung übernommen und welche zurückgewiesen wird, benutzen viele verteilte Systeme einen Zeitstempel, der von einer zuverlässigen Quelle ausgestellt wird. Eine andere verbreitete Strategie ist es, beide Änderungen zurückzuweisen. Beim Löschen eines Objektes werden die Replikate mit sogenannten „tombstones“ ersetzt, damit das Objekt nicht neu repliziert wird. [16]

[1] Churn: Fortlaufendes Hinzukommen und Wegfallen von Peers in einem Peer-to-Peer-Netzwerk

2.1.3 Implementierung

BubbleStorm ist in der funktionalen Programmiersprache Standard ML (SML) geschrieben und lässt sich als Bibliothek in die eigene Anwendung einbinden. Die API[2] bietet einen hohen Abstraktionsgrad, sodass die Anwendung überhaupt nichts mit der zugrunde liegenden Funktionalität des Netzwerks zu tun hat. Insbesondere hat die Anwendung keinerlei Kontrolle über den Aufbau und den Abbau von Verbindungen, sowie über den Datenaustausch mit Nachbarn im Netzwerk. Eine Anwendung, die BubbleStorm benutzt, besteht aus einem Frontend, welches die Anwendungslogik enthält, und einem Backend, welches als Datenspeicher fungiert. Das Frontend kommuniziert nur über das BubbleStorm-Netzwerk mit Backends auf anderen Peers und niemals direkt mit dem eigenen Backend. Dies ist in Abbildung 2.4 veranschaulicht.

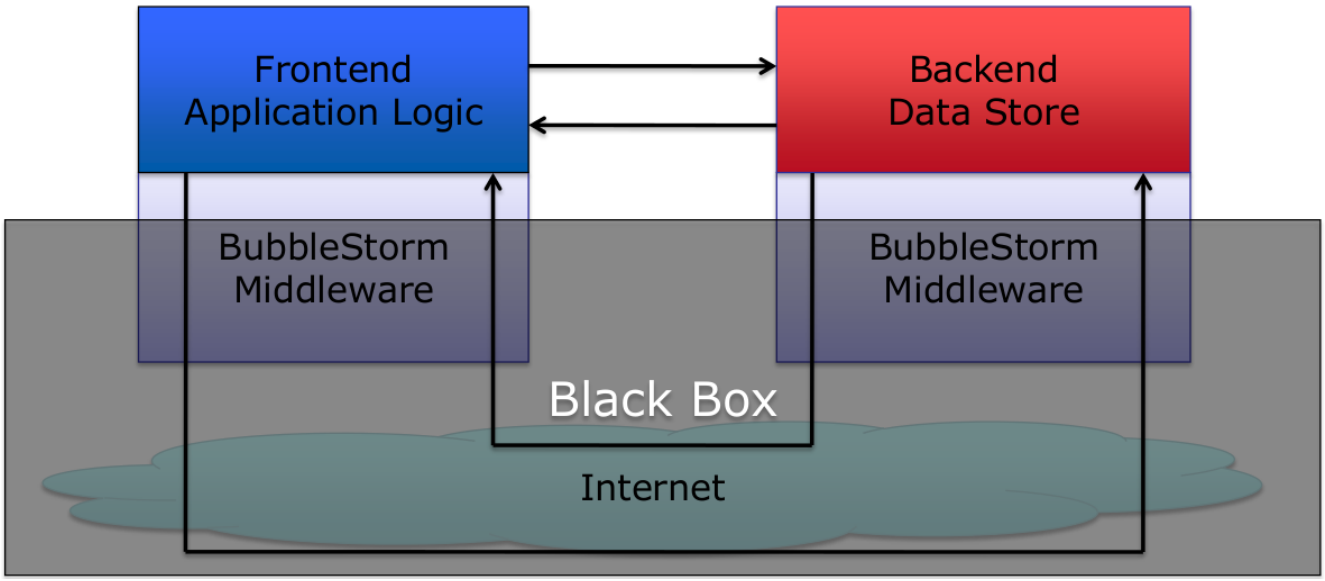


Abbildung 2.4.: Frontend und Backend einer BubbleStorm-Anwendung [26]

Es existieren Bindings[3] für C++ und Java. Da die Bibliothek plattformspezifischen Programmcode benutzt, muss sie für jede Plattform separat kompiliert werden. Das Java Binding bindet die von BubbleStorm benötigten Bibliotheken über das Java Native Interface (JNI) [19] ein. Es wird in dieser Bachelorarbeit verwendet und bietet folgende Funktionen:

- Übergeben der Bootstrap[4]-Adressen
- Betreten des Netzwerks (Join)
- Verlassen des Netzwerks (Leave)
- Senden von Bubblecasts (Query-Bubble und Data-Bubble)

[2] Application Programming Interface, also Programmierschnittstelle
[3] Binding: ermöglicht die Nutzung einer Bibliothek durch eine andere Programmiersprache
[4] Bootstrap: Knoten, über welchen der Zugang zu einem Peer-to-Peer-Netzwerk erfolgen kann

2.2 TUD Torrent

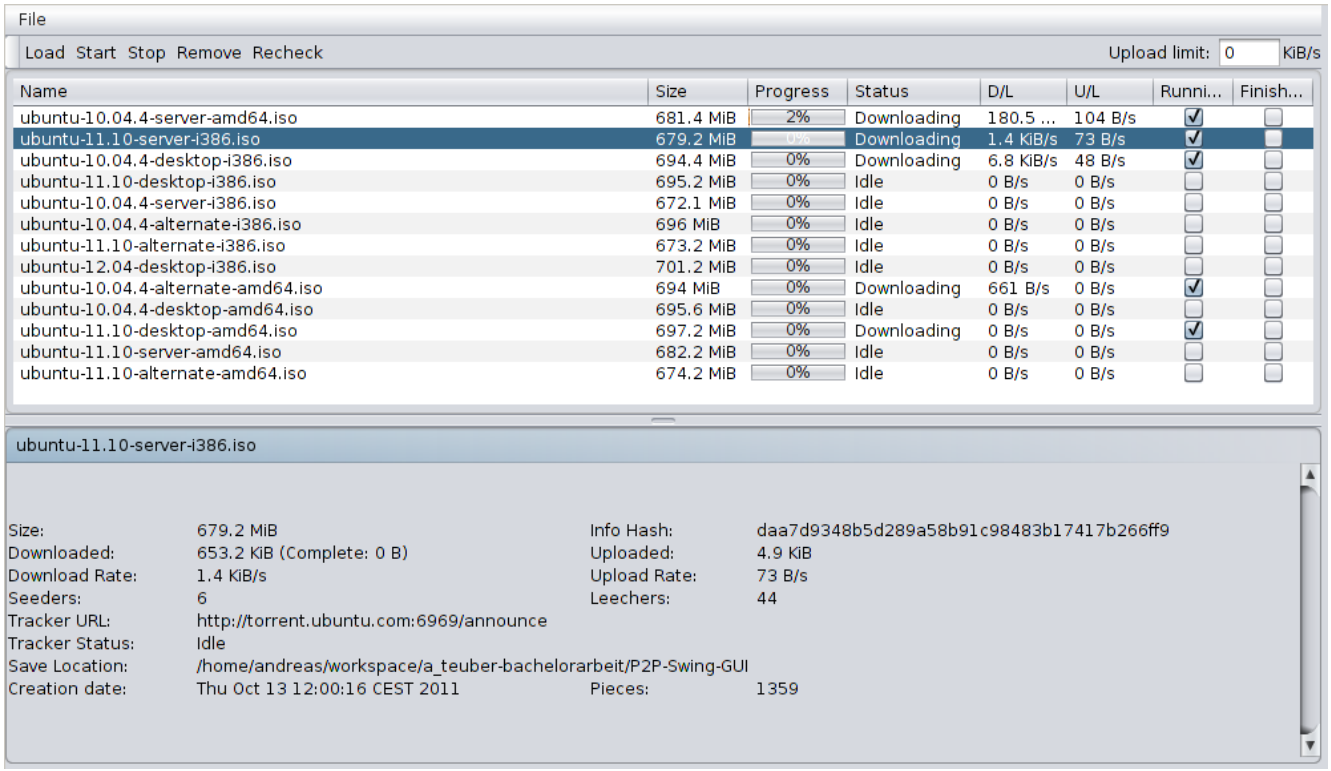


Abbildung 2.5.: Screenshot der grafischen Oberfläche des TUD Torrent Clients

TUD Torrent ist ein plattformunabhängiger BitTorrent-Client, welcher 2009 im Rahmen eines Praktikums an der Technischen Universität Darmstadt von Tilo Eckert und Andreas Teuber entwickelt wurde. Er ist in Java programmiert und modular aufgebaut, sodass er sich gut für Erweiterungen eignet. Außerdem basiert er auf einem Eventsystem und ist in der Lage, mehrere Prozessorkerne gleichzeitig zu benutzen (Multithreading). Der BitTorrent-Client erfüllt alle wesentlichen Funktionen der BitTorrent-Spezifikation [4]. Er kann also von einem HTTP-Tracker Informationen über andere Peers erhalten, um mit diesen Torrents, beziehungsweise Blöcke eines Torrents, auszutauschen. Es existiert eine grafische Benutzerschnittstelle (siehe Abbildung 2.5) und eine Benutzerschnittstelle für die Kommadozeile.

2.2.1 Design

TUD Torrent besteht aus mehreren logischen Schichten, sodass bei Bedarf Komponenten ausgetauscht und Erweiterungen hinzugefügt werden können. Deren Zusammenspiel ist in Abbildung 2.6 visualisiert, ihre Aufgaben und Funktionen werden im Folgenden beschrieben.

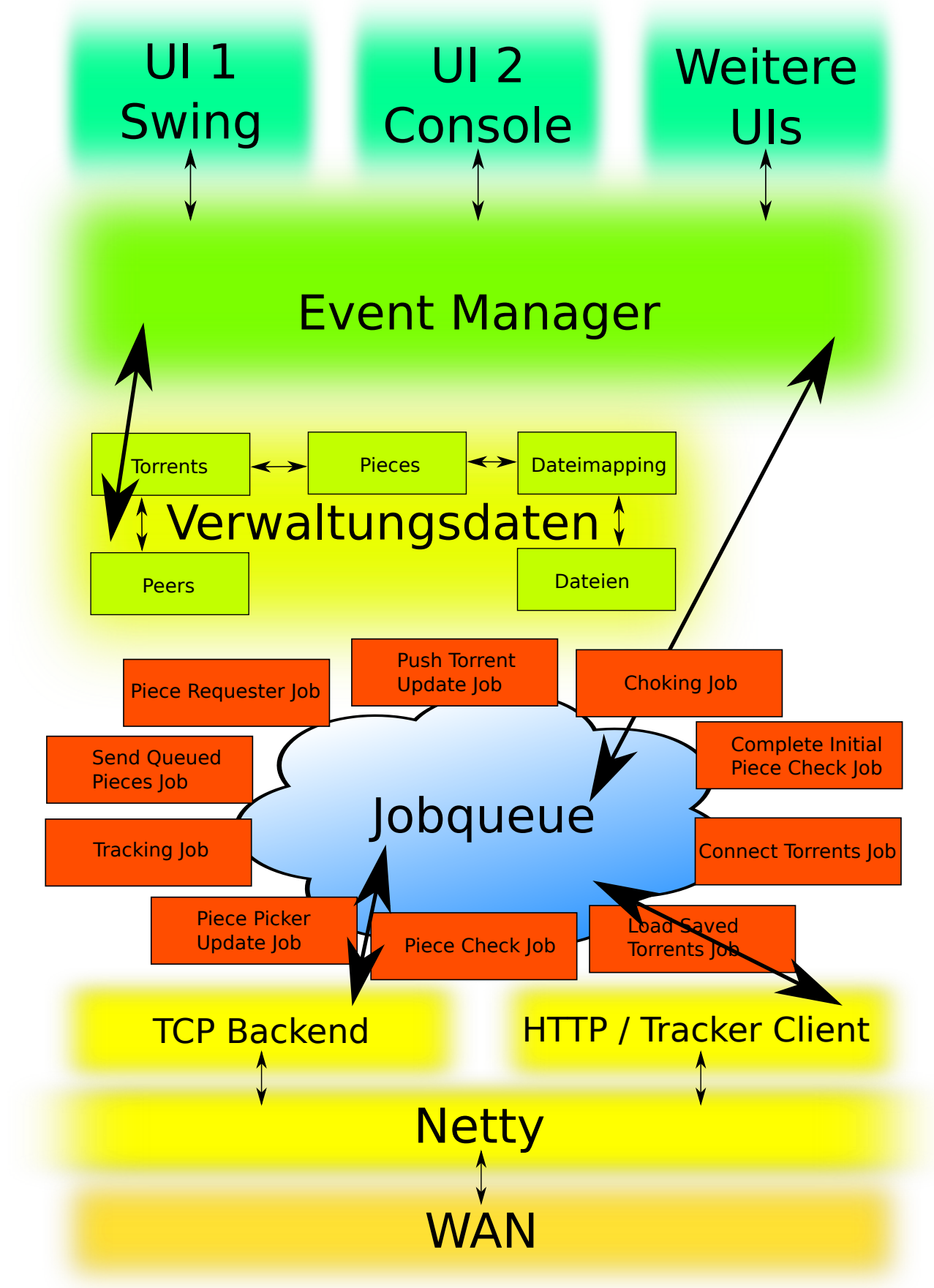


Abbildung 2.6.: Das Design von TUD Torrent mit seinen verschiedenen Schichten [10]

User Interfaces (UIs)

Die Benutzerschnittstellen sind für die Interaktion mit dem Benutzer zuständig. Durch das Event-System lassen sich mehrere UIs gleichzeitig nutzen und zur Laufzeit des Programms austauschen oder abkoppeln. Momentan existiert eine grafische Benutzerschnittstelle und eine, welche die Kommandozeile benutzt. Das Programm kann allerdings problemlos um weitere Benutzerschnittstellen (z. B. ein Webinterface) erweitert werden.

Event Manager

Über den Event Manager kommunizieren die UIs mit dem Programmkern (Datenhaltung, Logik). Er funktioniert nach dem Publish-Subscribe-Ansatz. Die Module des Programms können Nachrichten in einen Channel schreiben (publish) oder die Nachrichten aus einem Channel abonnieren (subscribe). Zur Identifikation der passenden Antwort wird bei diesen immer auch eine Referenz auf die ursprüngliche Anfrage übertragen.

Verwaltungsdaten

Diese Schicht ist für das Laden und Speichern von Daten zuständig, die von TUD Torrent benötigt werden. Dazu zählt der Zustand des Torrents, eine Liste bekannter Peers und eine Liste fertiger Torrent-Blöcke. Da ein Torrent in gleich große Blöcke aufgeteilt wird und sich diese über einen Teil einer Datei oder auch mehrere Dateien erstrecken können, muss zudem ein Mapping von Torrent-Blöcken auf Dateien gespeichert werden.

Job Queue

Fast alle Aufgaben, die in TUD Torrent anfallen, werden in kleine Teilaufgaben (Jobs) unterteilt und in eine Warteschlange (Job Queue) eingereiht. Das hat die Vorteile, dass die Jobs erst abgearbeitet werden, wenn genügend Rechenzeit verfügbar ist und dass sie automatisch auf mehrere Threads aufgeteilt werden können, sodass auch Multicore-Prozessoren ausgenutzt werden können. Die Jobs können interne Schritte ausführen, Nachrichten über das TCP-Backend senden oder empfangen oder ein Tracking-Update anstoßen.

HTTP Tracker Client

Der HTTP Tracker Client fragt von dem in der Torrent-Datei angegebenen Tracker regelmäßig Informationen über andere Peers ab und aktualisiert die Informationen über den eigenen Peer auf dem Tracker. Der Tracker liefert daraufhin eine Liste mit den IP-Adressen und den benutzten Ports der anderen Peers zurück.

TCP Backend

Das TCP Backend ist für die Kommunikation mit den anderen Peers zuständig. Es implementiert das Kommunikationsprotokoll von BitTorrent. Es übernimmt also den Verbindungsaufbau und das Senden von Daten an andere Peers. Außerdem trägt es von anderen Peers empfangene Daten in die Job Queue ein. Dabei kommt das Netty- Framework zum Einsatz.

Jboss Netty

Jboss Netty ist ein Java-Framework, das dem Entwickler die Nutzung von Java NIO [14] zur asynchronen, nicht-blockierenden Kommunikation vereinfacht. Es wurde für Einsatzzwecke entwickelt, die eine hohe Performance und Skalierbarkeit erfordern und eignet sich sowohl für Serveranwendungen als auch für Client-Anwendungen. Netty stellt eine Pipeline bereit, die mit eigenen Upstream- und Downstream-Handlern gefüllt wird, sodass sich sehr einfach ein eigener Protokollstack entwickeln lässt. Es können für jede Verbindung eigene Handler genutzt werden, verschiedene Verbindungen können sich jedoch auch Handler teilen. Beide Typen von Handlern können in einer gemeinsamen Pipeline genutzt werden.

2.3 Verteilte Suche für BitTorrent-Netzwerke

TUD Torrent wurde 2010 von Tilo Eckert in seiner Bachelorarbeit „Verteilte Suche für BitTorrent-Netzwerke“ [9] erweitert. Mit den in den folgenden beiden Abschnitten beschriebenen Änderungen lässt sich TUD Torrent völlig unabhängig von jeder zentralen Instanz benutzen und es wird kein weiteres Programm (z. B. ein Webbrowser) benötigt.

2.3.1 Dezentrale Peer-Suche mit BubbleStorm

Es ist nun möglich, die Informationen über Peers eines Schwarms im BubbleStorm-Netzwerk abzulegen und aus diesem zu empfangen. Diese Funktion ersetzt also das bisher über einen HTTP-Server stattfindende Tracking durch eine dezentrale Variante. Peers, die nach anderen Peers in einem Schwarm suchen, wollen ebenso von diesen gefunden werden. Deshalb wird beim Tracking über BubbleStorm nicht zwischen der Speicherung von Informationen über Peers und der Suche nach Peers unterschieden, also Data Bubble und Query Bubble zusammengefasst. Jeder Torrent besitzt einen Infohash, der den Torrent eindeutig identifiziert und daher für die Suche nach Peers genutzt wird. Der Infohash entspricht der Anwendung des SHA-1 Algorithmus auf den Info-Schlüssel der Torrent-Datei. Neben dem Infohash enthält eine Bubble bei der dezentralen Peer-Suche einen aktuellen Zeitstempel und folgende Informationen über die eigene Verbindung:

- die eigene IP-Adresse
- den Port, den BitTorrent für eingehende Verbindungen nutzt
- den Port, den BubbleStorm für eingehende Verbindungen nutzt (zum Empfangen von Informationen über weitere Peers im Schwarm)

Diese Bubble wird regelmäßig als Query in das BubbleStorm-Netzwerk geschickt, sodass auch neue Peers entdeckt werden. Ein Peer, der die Query empfängt, speichert die darin enthaltenen Informationen lokal ab und schickt dem Absender eine Liste mit weiteren Peers im Schwarm zurück. Die abgespeicherten Informationen über Peers behalten 30 Minuten lang ihre Gültigkeit und werden anschließend gelöscht.

2.3.2 Dezentrale Torrent-Suche mit BubbleStorm

Torrents lassen sich nun in das BubbleStorm-Netzwerk einstellen und auch darüber finden. Hierfür wurde eine Volltextsuche implementiert, welche auf Apache Lucene [1] basiert. Lucene ist eine Java-Bibliothek für Volltextsuchen und zeichnet sich unter anderem durch die Unterstützung von Fuzzy-Suche, eigenen Tokenizern und Result Scoring aus.

Veröffentlichung von Torrents

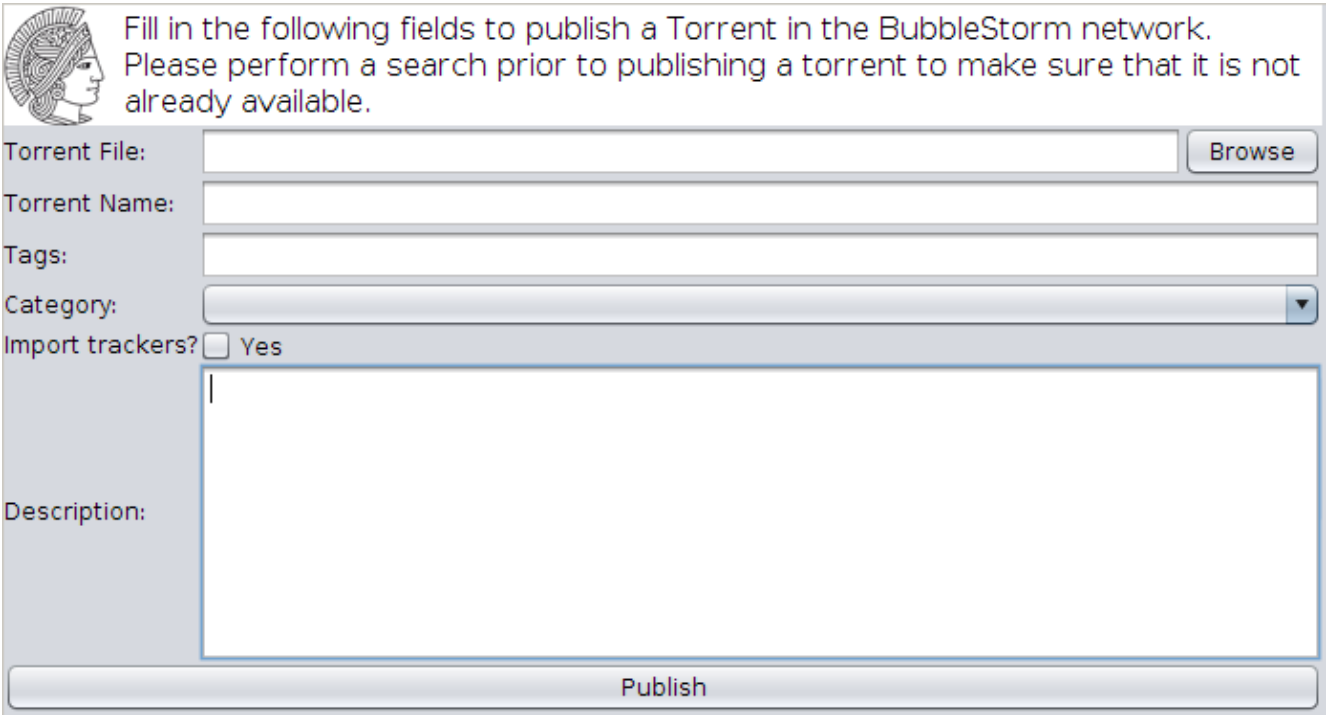


Abbildung 2.7.: Benutzeroberfläche für das Veröffentlichen von Torrents im BubbleStorm-Netzwerk

Um einen Torrent im BubbleStorm-Netzwerk zu veröffentlichen, werden eine Torrent-Datei und einige Angaben des Benutzers benötigt. Diese kann der Benutzer über eine grafische Oberfläche eingeben (siehe Abbildung 2.7). Danach wird eine Data Bubble mit folgenden Informationen per BubbleCast verschickt:

- eindeutige ID der Veröffentlichung
- Zeitpunkt der Veröffentlichung (Datum und Uhrzeit)
- Informationen aus der Torrent-Datei:
  - Infohash des Torrents
  - Name des Torrents (kann auch durch den Benutzer gesetzt werden)
  - Pfade und Dateinamen
  - Größe des Torrents und der einzelnen Dateien
- Angaben des Benutzers:
  - Beschreibung des Torrents
  - Schlagwörter (Tags)
  - Kategorie
  - Tracker URLs (als Alternative zum Tracking via BubbleStorm)

Es wird nicht einfach die Torrent-Datei selbst verschickt, da diese sehr groß sein kann. Ein Bubblecast skaliert mit $O(\sqrt{n})$, sodass bei sehr vielen Peers ein hoher Traffic entstehen könnte. Stattdessen werden nur die für die Suche relevanten Informationen verteilt. Empfängt ein anderer Peer im BubbleStorm-Netzwerk die Bubble, dekodiert er sie und prüft anhand der enthaltenen eindeutigen ID, ob der Torrent bereits im Lucene-Index enthalten ist. Ist dies nicht der Fall, so werden die Information als neues Dokument zum lokalen Lucene-Index hinzugefügt. Der Vorgang des Veröffentlichens ist in Abbildung 2.8 veranschaulicht.

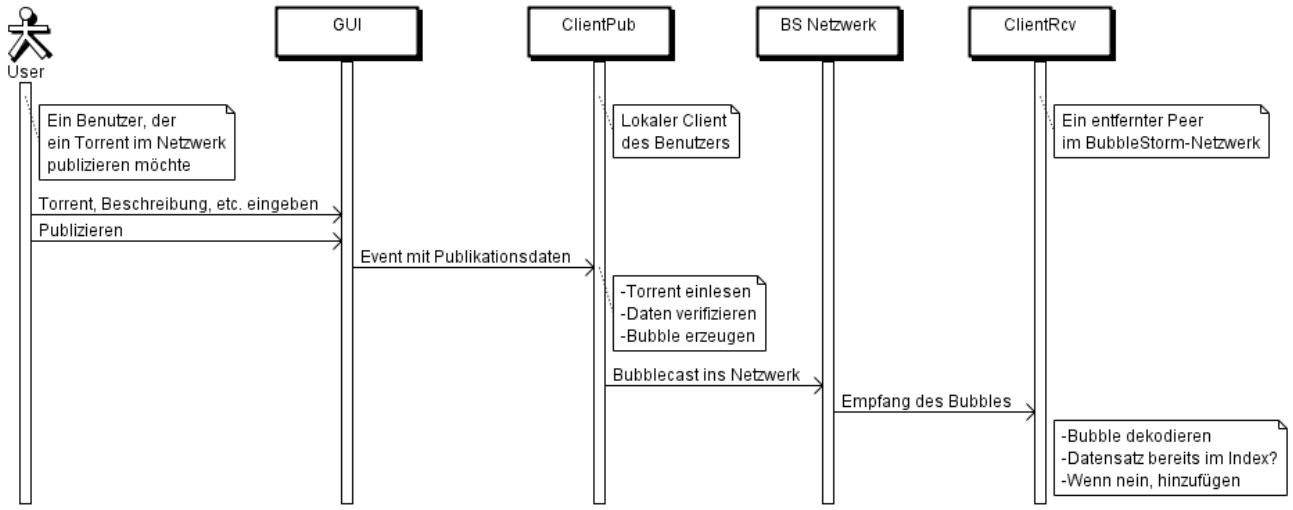


Abbildung 2.8.: Veröffentlichung von Torrents im BubbleStorm-Netzwerk [9]

Suche nach Torrents

Die Torrent-Suche im BubbleStorm-Netzwerk ist unscharf, sodass auch Treffer gefunden werden, die der Suchanfrage ähneln, jedoch nicht exakt mit ihr übereinstimmen (dies nennt man auch Fuzzy-Suche). Ein Benutzer kann im BubbleStorm-Netzwerk durch Angabe eines oder einer Kombination der folgenden Kriterien in einer grafischen Oberfläche (siehe Abbildung 6.1 im Kapitel *Benutzeroberfläche* [6]) nach Torrents suchen:

- Zeitpunkt der Veröffentlichung (Datum und Uhrzeit)
- Name des Torrents
- Beschreibung des Torrents
- Schlagwörter (Tags)
- Kategorie

Ein Peer, der eine Query Bubble mit einer Suchanfrage empfängt, führt lokal eine Suche über den mit Lucene erzeugten Index aus. Bei einer erfolgreichen Suche werden die Treffer an den Absender der Suchanfrage geschickt. Daraufhin kann der suchende Benutzer aus einer Liste aller Treffer die Torrents auswählen, die er herunterladen möchte. Dazu kann er sich auch Details zu jedem Torrent anzeigen lassen. Hat der Benutzer sich für einen Torrent entschieden, wird mit dem eindeutigen Infohash nach anderen Peers im Schwarm gesucht. Dies kann sowohl über einen HTTP-Tracker als auch über BubbleStorm geschehen (siehe *Dezentrale Peer-Suche mit BubbleStorm* [2.3.1]). Außerdem werden die in der Torrent-Datei enthaltenen Metadaten über eine Erweiterung des BitTorrent-Protokolls von anderen Peers im Schwarm heruntergeladen. Darin enthalten sind Informationen über Pfade und Dateinamen und SHA-1 Prüfsummen, mit denen die heruntergeladenen Daten verifiziert werden können.

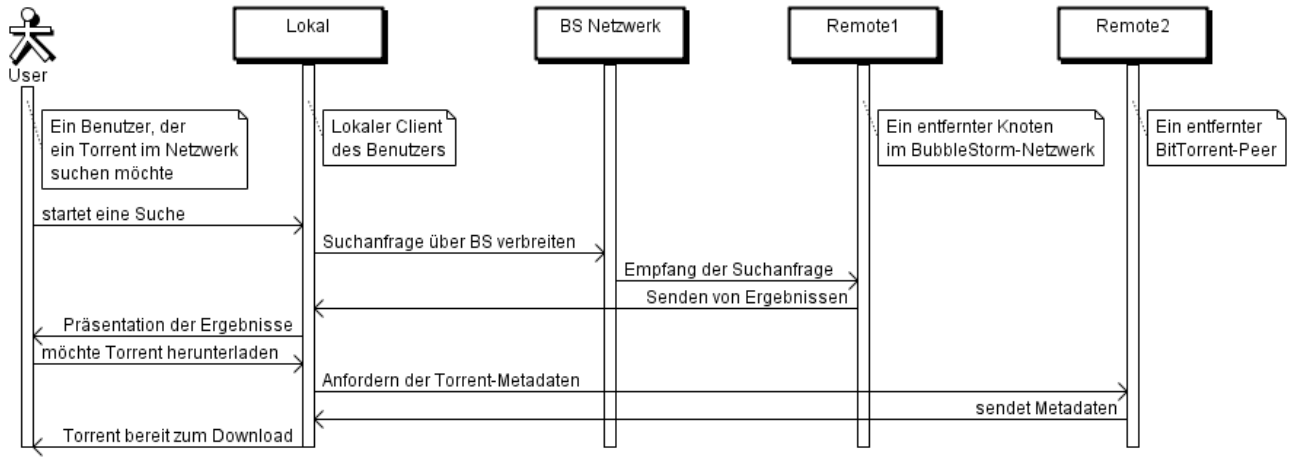


Abbildung 2.9.: Suche nach Torrents im BubbleStorm-Netzwerk [9]

Mit dem BitTorrent Enhancement Proposal 10 (BEP 10) [18] wird das BitTorrent-Protokoll um einen neuen Nachrichtentyp erweitert, mit dem Erweiterungen auf unterschiedlichen Clients miteinander kommunizieren können. Das BitTorrent Enhancement Proposal 9 (BEP 9) [13] spezifiziert die „Extension for Peers to Send Metadata Files“ und nutzt das BEP 10, um die Metadaten eines Torrents von anderen Peers im Schwarm herunterzuladen. Damit TUD Torrent mit diesen Protokollerweiterungen umgehen kann, wurde die Pipeline des TCP-Backends um einen neuen Handshake gemäß BEP 10 ergänzt. Das Herunterladen der Metadaten von Torrents funktioniert ähnlich wie das anschließende Herunterladen der Torrents selbst über das BitTorrent-Protokoll. Der gesamte Vorgang der Suche nach Torrents im BubbleStorm-Netzwerk ist im Sequenzdiagramm in Abbildung 2.9 dargestellt.

2.3.3 Zusammenfassung

Durch die Veränderungen, die Tilo Eckert an TUD Torrent vorgenommen hat, lässt es sich völlig unabhängig von jeglicher zentralen Instanz benutzen. Das Publizieren und Finden von Torrents funktioniert nun über das dezentrale BubbleStorm-Netzwerk, anstatt über einen HTTP-Server. Dasselbe gilt für das Finden von Peers in einem Schwarm. Die konkret hinzugekommenen Komponenten lassen sich in Abbildung 2.10 nachvollziehen. Da beim Entwickeln auf Modularität geachtet wurde, lassen sich TUD Torrent und die Suche nach Torrents auch getrennt voneinander verwenden.

Die Kombination von TUD Torrent mit der dezentralen Peer-Suche und der dezentralen Torrent-Suche besitzt noch einige Probleme. So existiert beispielsweise keine Möglichkeit, die Qualität eines Torrents zu beurteilen, ohne dieses herunterzuladen. Dies wird vor allem zum Problem, wenn Benutzer Torrents mit einer Beschreibung veröffentlichen, die nichts mit dem Inhalt des Torrents zu tun hat (sogenannte Fakes), oder wenn Benutzer auf diese Weise Schadsoftware verbreiten. Abhilfe kann die Erweiterung des Systems um soziale Komponenten schaffen. In dieser Bachelorarbeit wird deshalb das System um eine Bewertungsfunktion und eine Kommentarfunktion erweitert. Dabei wird von einem Web of Trust Gebrauch gemacht, sodass nur Kommentare von Benutzern angezeigt werden, denen man vertraut. Ebenso fließen nur Bewertungen von vertrauenswürdigen Peers in das Rating eines Torrents ein.

Ein weiteres Problem ist, dass das System Angriffe auf das BubbleStorm-Netzwerk, auf einzelne Benutzer im Netzwerk und sogar auf beliebige andere Rechner im Internet erlaubt. Die ersten beiden Angriffe sind möglich, weil keine Schutzmaßnahmen gegen Flooding existieren. Ein Angreifer ist in der Lage durch eine hohe Anzahl Bubblecasts in kurzer Zeit mehrere Peers zu überlasten. Um beliebige Rechner im Internet mit einer DDoS-Attacke zu überlasten, muss ein Angreifer nur die IP-Adresse des anzugreifenden Rechners für die dezentrale Peer-Suche

verwenden (Spoofing). Der Angriff ist besonders effektiv, wenn dabei ein Port gewählt wird, bei dem bereits der Verbindungsaufbau Rechenzeit beansprucht. Dies ist z. B. bei schlechten Implementierungen von SSL der Fall, in denen nicht überprüft wird, ob es sich bei ankommenden Daten um ein SSL Client-Hello handelt, sondern direkt eine Zufallszahl berechnet wird. Diese Art von Angriff ist in der Praxis bereits mit PEX (Peer Exchange), einer Erweiterung des BitTorrent-Protokolls, möglich und wird auch genutzt[5]. Ein Angreifer kann außerdem die dezentrale Peer-Suche über BubbleStorm unmöglich machen, indem er mit vielen gefälschte IP-Adressen dem Schwarm beitritt. Damit reduziert er die Wahrscheinlichkeit, dass andere Peers im Schwarm sich mit einem „echten“ Peer verbinden.

[5] http://blog.fefe.de/?ts=b2b82dd0

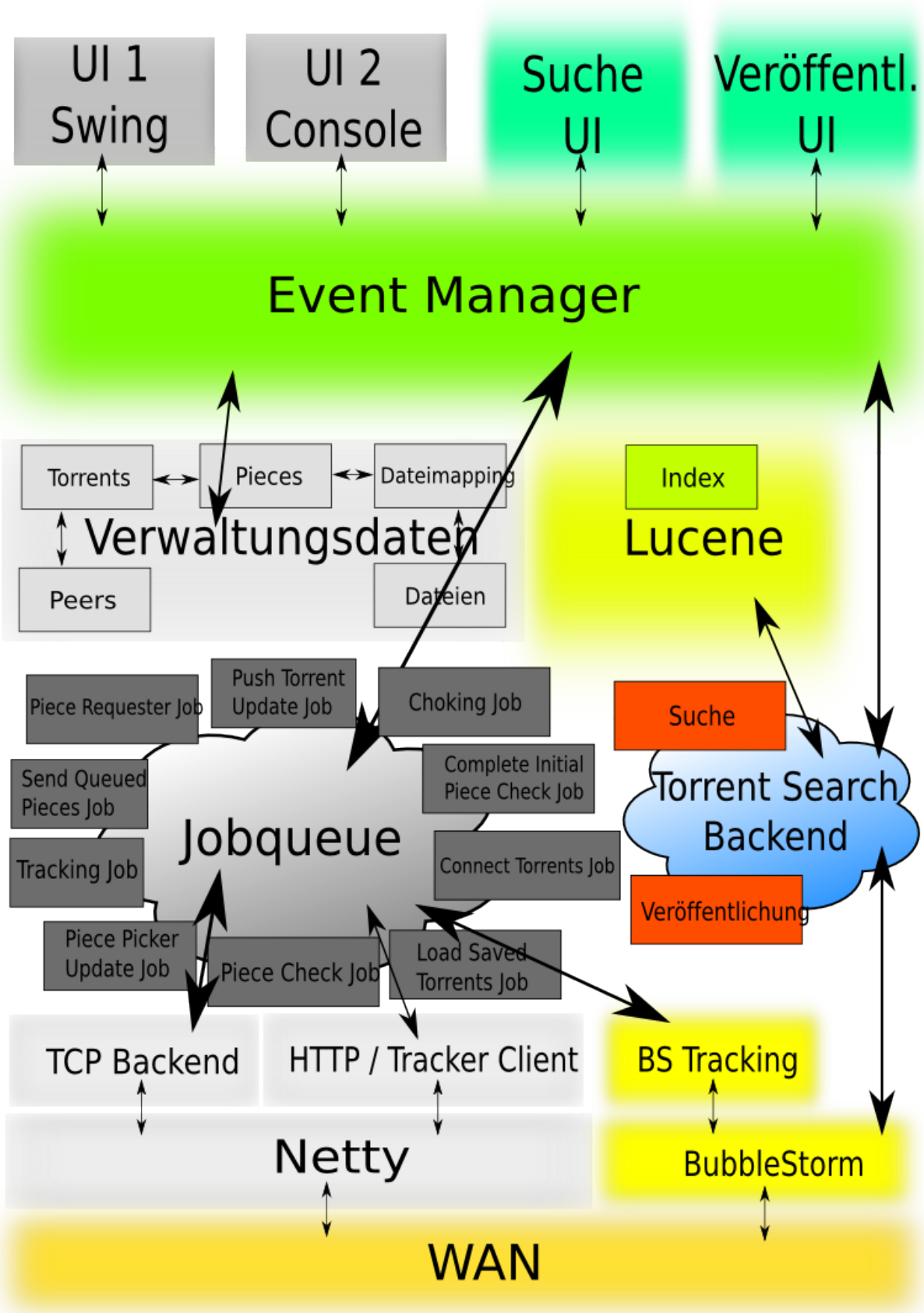


Abbildung 2.10.: Das Design von TUD Torrent mit seinen verschiedenen Schichten. Farbig hervorgehoben sind die mit der Bachelorarbeit von Tilo Eckert neu hinzugekommenen Komponenten und für diese relevante Schichten. [9]

3 Verwandte Systeme

Es existieren bereits einige Projekte, die ein Friend-to-Friend Netzwerk realisieren. Bei dieser Art von Netzwerk werden nur Verbindungen zu vertrauenswürdigen anderen Peers aufgebaut. Diesen muss der eigene Peer nicht unbedingt direkt vertrauen. Es genügt, wenn er einem anderem Peer vertraut, welcher wiederum einem dritten Peer vertraut (transitives Vertrauensverhältnis). Dabei existieren diverse Algorithmen zur Berechnung des Vertrauens. Friend-to-Friend Netzwerke unterscheiden sich von privaten Peer-to-Peer-Netzwerken, in denen jeder teilnehmende Peer jeden anderen teilnehmenden Peer kennt. Im Folgenden werden drei Friend-to-Friend Netzwerke vorgestellt und anschließend mit dem in dieser Bachelorarbeit implementierten Konzept verglichen. Dabei liegt der Fokus auf Freenet, weil dieses bereits am längsten entwickelt wird und deutlich besser erforscht ist als RetroShare oder Turtle F2F.

3.1 Freenet

Das Ziel von Freenet ist es, Daten verteilt zu speichern, um damit Zensur zu verhindern und einen anonymen Austausch von Informationen zu ermöglichen. Die Idee wurde 1999 in einem Paper von Ian Clarke mit dem Titel „Freenet: A Distributed Anonymous Information Storage and Retrieval System“ [6] beschrieben. Seit Version 0.7 (Veröffentlichung der Alpha-Version: 2006) lässt es sich so betreiben, dass der Zugang nur nach Einladung eines bereits teilnehmenden Benutzers möglich ist. Jeder Benutzer stellt einen Teil seiner Festplatte zur Verfügung, auf dem Freenet verschlüsselte Daten anderer Benutzer ablegen kann. Freenet lässt sich ähnlich wie das World Wide Web über einen Browser bedienen. Ein Dokument, das über Freenet erreichbar ist, nennt man Freesite. Dieses kann Links zu anderen Freesites oder sonstigen Daten in Freenet enthalten. Die Geschwindigkeit beim Surfen in Freenet wird über ActiveLinks erhöht. Diese kleinen PNG-Dateien führen dazu, dass entfernte Freesites bereits im Voraus geladen werden (Prefetching) und dass Freesites besser im Netzwerk verteilt werden. Freenet verfügt außerdem über ein E-Mail System, Diskussionsforen und Blogs (diese heißen im Freenet-Jargon Flogs).

Für das Routing (Key-based routing, KBR) in Freenet wird jedem Peer eine reelle Zahl zwischen Null und Eins zugeordnet. Wird eine Suche nach einem Schlüssel durchgeführt, schaut der suchende Peer zunächst in seinen lokalen Speicher. Wird er dort nicht fündig, so wird aus dem Hashwert des Schlüssels eine andere Zahl zwischen Null und Eins berechnet. Die Suchanfrage wird daraufhin an den Peer gesendet, dessen zugewiesene Zahl dieser berechneten Zahl

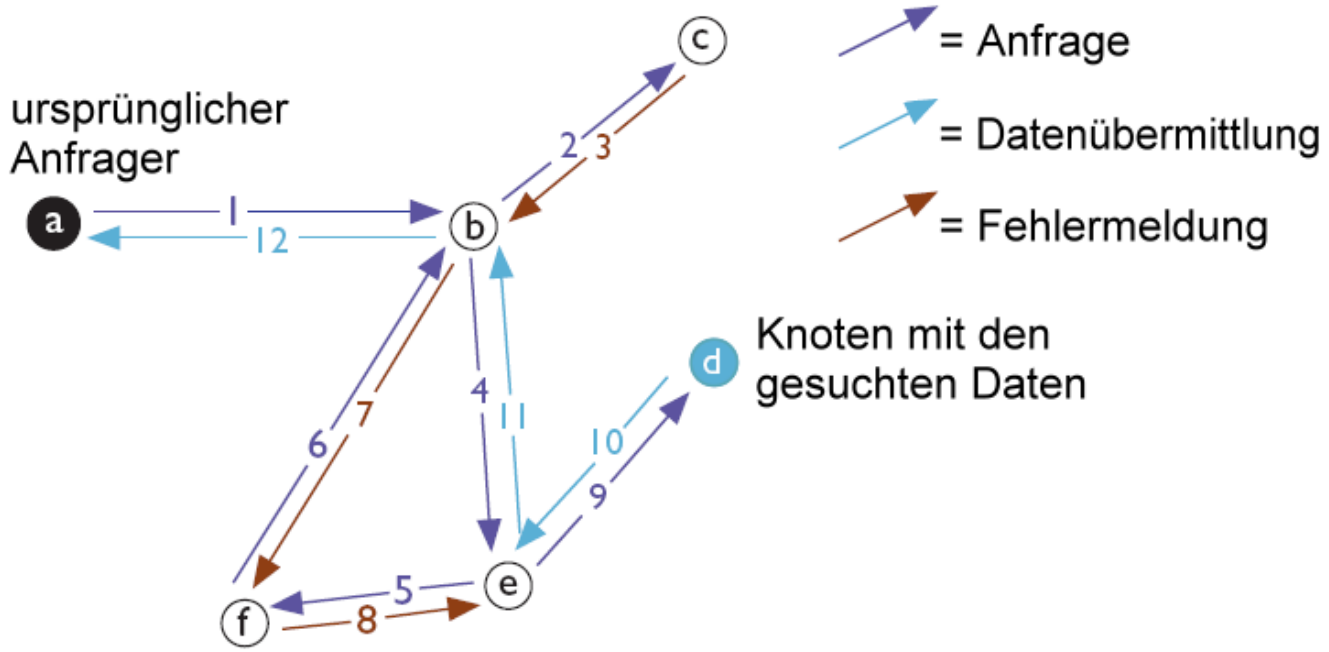


Abbildung 3.1.: Typischer Ablauf einer Anfrage in Freenet [11]

am nächsten ist. Dies geht so weiter, bis die maximale Anzahl an Hops durchgeführt wurde (gewöhnlich 20), bei allen Peers gesucht wurde oder die Suche erfolgreich war. Wenn die Suche erfolgreich war, werden der Schlüssel und die unter diesem Schlüssel gespeicherten Daten bei jedem Peer auf dem Pfad zurück zum suchenden Peer zwischengespeichert, sodass künftige Suchanfragen nach diesem Schlüssel schneller beantwortet werden können. Die Antwortzeit einer Suche in Freenet ist also davon abhängig, wie häufig bereits nach dem Schlüssel gesucht wurde. Das gleiche Verfahren wird auch zum Speichern von Daten genutzt. Hier werden die Daten solange geroutet, bis die maximale Anzahl an Hops erreicht wurde. Wenn nun keine Daten mit dem gleichen Schlüssel gefunden wurden, werden sie auf jedem Peer im Pfad gespeichert. Wenn dagegen Daten mit dem gleichen Schlüssel gefunden wurden (also eine Kollision stattfindet), werden diese an den Peer, von dem die Suchanfrage stammt, übertragen.

In Abbildung 3.1 ist dargestellt, wie eine Suchanfrage in Freenet ablaufen könnte. Zunächst endet sie bei Peer c, dann stellt Peer b eine Schleife fest und am Ende wird ein Peer gefunden, der die gesuchten Daten bereithält (Peer d). Dadurch, dass die gefundenen Daten über den Pfad der Suchanfrage zurück übertragen werden, müssen sich der suchende Peer und der Peer, welcher die Daten vorhält, nicht direkt verbinden. Dies ermöglicht es Peers in Freenet, anonym zu bleiben. [6]

Für Freenet existiert ein Web of Trust Plugin, das der Bekämpfung von Spam dient. Hierbei bekommt jeder Benutzer eine Identität und kann jeder anderen Identität einen positiven

oder negativen Vertrauenswert (trust value) zuteilen (von -100 bis +100). Zur Beurteilung von noch unbekannten Identitäten werden die Vertrauenswerte über die im Abschnitt *Freenet Web-of-Trust Algorithmus* (3.1.1) beschriebene Vorgehensweise zu einem Scorewert (score value) zusammengeführt. Nur von Identitäten mit einem Scorewert größer oder gleich Null werden Daten heruntergeladen. Neue Identitäten müssen CAPTCHAs[1] anderer Identitäten lösen, um von diesen einen Vertrauenswert von Null ausgesprochen zu bekommen und somit überhaupt sichtbar zu werden. Um glaubhafte Abstreitbarkeit zu gewährleisten, werden die Daten in Freenet verschlüsselt abgespeichert.

3.1.1 Freenet Web-of-Trust Algorithmus

Das Web of Trust Plugin von Freenet berechnet automatisch Scorewerte von jeder eigenen Identität (ein Peer in Freenet kann mehrere Identitäten im Web of Trust Plugin besitzen) zu jeder fremden Identität. Existiert ein eigener Vertrauenswert für eine Identität, wird einfach dieser als Scorewert genommen. Andernfalls wird nach dem im Folgendem beschriebenen Algorithmus vorgegangen. Zunächst wird für jede Identität ein Rang berechnet. Dies ist die kürzeste Distanz zur eigenen Identität, also die minimale Anzahl Hops im Web of Trust, die zum Erreichen der Identität benötigt werden. Zusätzlich kommen die folgenden Regeln zur Anwendung:

1. Identitäten, denen von der eigenen Identität der Vertrauenswert 0 zugeordnet wurde, bekommen einen Rang von unendlich.
2. Identitäten, die nur über Identitäten erreichbar sind, denen ein Vertrauenswert von 0 zugeordnet wurde, bekommen ebenfalls einen Rang von unendlich.
3. Identitäten, die einen Rang von unendlich haben, können diesen nicht vererben. Ansonsten könnte sich der Besitzer durch das Lösen von CAPTCHAs neue Identitäten mit dem Vertrauenswert 0 erzeugen.
4. Identitäten, die nicht über das Web of Trust erreichbar sind, bekommen den Rang „null“.

Nachdem der Rang berechnet wurde, wird der Scorewert berechnet, indem empfangene Vertrauenswerte nach ihrem Rang gewichtet summiert werden (siehe Tabelle 3.1). [34]

Rang	1	2	3	4 oder höher	unendlich oder „null“
Einfluss	40%	16%	6%	16%	0%

Tabelle 3.1.: Einfluss des Vertrauenswertes auf den Scorewert abhängig vom Rang der Identität

[1] CAPTCHA: „Completely Automated Public Turing test to tell Computers and Humans Apart“ – ein Test den nur Menschen lösen können sollen, keine Computer

3.2 RetroShare

RetroShare ist ein Programm, das es ermöglicht, verschlüsselt zu kommunizieren und Dateien zu tauschen. Es basiert auf einer verteilten Hashtabelle, stellt aber die Authentizität von Peers über GnuPG sicher. Es handelt sich um ein Friend-to-Friend-Netzwerk, d. h. Peers verbinden sich nur mit ihnen bekannten Peers, beziehungsweise Peers, deren GnuPG-Schlüssel von ihnen beglaubigt wurde. Um auf Freigaben von Peers zuzugreifen, die nicht zu den eigenen Freunden zählen, können Anfragen (Suche, Zugriff, Upload, Download) über die eigenen Freunde geroutet werden. Das Routing erfolgt über maximal sechs Peers. RetroShare bietet noch weitere Funktionen, darunter verschiedene Chatmöglichkeiten, interne Foren und demnächst VoIP. [23]

3.3 Turtle F2F

Turtle F2F ist ein Programm, um Dateien zu teilen und anonyme Kommunikation zu ermöglichen. Es wird u. a. von Andrew Tanenbaum an der Vrije Univeriteit in Amsterdam entwickelt und verfolgt den Friend-to-Friend-Ansatz. Jeder Peer verbindet sich nur über sichere Kanäle mit authentifizierten anderen Peers. Die Authentifikationsschlüssel müssen über einen Kanal außerhalb von Turtle F2F ausgetauscht werden („out-of-band“). Anfragen werden geroutet, sodass nur Peers miteinander kommunizieren, die sich vertrauen. Dadurch hält sich der durch kompromittierende Peers entstehende Schaden in Grenzen. Turtle F2F ist außerdem immun gegen Sybil-Attacken und resistent gegen Denial of Service Angriffe. Implementiert ist Turtle F2F als Plugin für gIFT. gIFT (Internet File Transfer) ist ein Daemon zum Abstrahieren von GUI und Filesharing-Protokoll. [31]

3.4 Vergleich der Konzepte

Die unterschiedlichen Eigenschaften der Systeme sind in Tabelle 3.2 dargestellt. Freenet, RetroShare und auch die in dieser Bachelorarbeit vorgestellte Lösung (BubbleStorm Web of Trust) benutzen die Eigenschaft von asymmetrischen Kryptosystemen, dass Signieren und Verifizieren mit zwei unterschiedlichen Schlüsseln funktioniert, um andere Peers zu authentifizieren. Dabei sind RetroShare und Turtle F2F echte Friend-to-Friend-Netzwerke, erlauben aber über Routing auch Zugriff auf Dateien von Peers, die nicht zu den eigenen Freunden zählen. Dadurch bieten sie Anonymität. Um sich mit dem Netzwerk zu verbinden, muss der Benutzer jedoch erst andere Peers kennen. RetroShare und auch das BubbleStorm Web of Trust erlauben immerhin den Import bestehender GnuPG-Schlüssel. BubbleStorm Web of Trust ist eigentlich eine Anwendung, die ein bestehendes Peer-to-Peer-System nutzt, um die Vertrauenswürdigkeit von Peers zu bestimmen und darüber zu entscheiden, ob ein bestimmter Inhalt (Kommentare,

Programm	Kryptosystem	echtes F2F	Anonymität	„Ebene“ des WoT	DHT	GnuPG
Freenet	Asymmetrisch	Nein	Ja	P2P Overlay	Ja (KBR)	Nein
RetroShare	Asymmetrisch	Ja	Ja	P2P Overlay	Ja	Ja
Turtle F2F	Symmetrisch	Ja	Ja	P2P Overlay	Nein	Nein
BubbleStorm WoT	Asymmetrisch	Nein	Nein	Anwendung	Nein	Ja

Tabelle 3.2.: Vergleich von BubbleStorm WoT mit verwandten Systemen

Bewertungen) angezeigt werden soll oder nicht. Es setzt also eine Ebene höher an als RetroShare und verhindert nicht, dass zu nicht vertrauenswürdigen Peers eine Verbindung aufgebaut wird. Das hat den Nachteil, dass es keine Sicherheit auf Protokollebene gegen bösartige Peers und auch keine Anonymität bietet. Dafür lässt es sich prinzipiell mit unterschiedlichen Peer-to-Peer-Netzwerken benutzen, sofern diese eine ähnliche Schnittstelle wie BubbleStorm besitzen. Das Ziel von Freenet ist hauptsächlich, anonymen Informationsaustausch zu ermöglichen. Dies funktioniert ähnlich wie bei RetroShare und Turtle F2F über Routing. Dabei ist Freenet aber kein reines Friend-to-Friend-Netzwerk.

3.5 Reputationsmanagement

Beim Reputationsmanagement werden Aktionen von Entitäten und Meinungen anderer Entitäten über diese Aktionen benutzt, um bei der Auswahl vertrauenswürdiger Interaktionspartner zu helfen. Entitäten sind dabei meistens Menschen oder deren Endgeräte. Es existieren Verfahren, um die Reputation einer Entität mathematisch zu bestimmen. Diese Verfahren eignen sich sehr gut, um die Vertrauenswürdigkeit von Peers in Peer-to-Peer-Netzwerken einzustufen. Im Folgenden werden der sehr bekannte EigenTrust-Algorithmus und das etwas neuere CertainTrust vorgestellt.

3.5.1 EigenTrust

EigenTrust [15] ist ein Algorithmus zum Reputationsmanagement in Peer-to-Peer-Netzwerken. Jedem Peer in einem Netzwerk wird ein eigener globaler Vertrauenswert zugewiesen, der sich aus den bisherigen Uploads dieses Peers berechnet. Auf diese Weise soll die Anzahl von falschen Dateien in Peer-to-Peer-Netzwerken verringert werden. Der EigenTrust-Algorithmus basiert auf der Idee des transitiven Vertrauens. Jeder Peer i berechnet einen lokalen Vertrauenswert für je-

den Peer j, indem er die Anzahl der nicht zufriedenstellenden Interaktionen von der Anzahl der zufriedenstellenden Interaktionen subtrahiert:

$$s_{ij} = sat(i,j) - unsat(i,j) \tag{3.1}$$

Dieser lokale Vertrauenswert wird auf einen Wert zwischen Null und Eins normalisiert, damit böswillige Peers nicht für andere böswillige Peers beliebig hohe, oder für normale Peers niedrige Werte vergeben können:

$$c_{ij} = \frac{max(s_{ij}, 0)}{\sum_j max(s_{ij}, 0)} \tag{3.2}$$

Die normalisierten lokalen Vertrauenswerte werden aggregiert: Der Peer i multipliziert seinen lokalen Vertrauenswert in den Peer j mit dem lokalen Vertrauenswert dieses Peers j in den Peer, für den das Vertrauen berechnet werden soll (k). Dies tut er für jeden ihm bekannten Peer j und summiert anschließend die Ergebnisse:

$$t_{ik} = \sum_j c_{ij} \cdot c_{jk} \tag{3.3}$$

Kennt der Peer die c_{ij} für das gesamte Netzwerk, dann lässt sich wie folgt ein Vertrauensvektor berechnen, der die Vertrauenswerte des Peers i in alle Peers im Netzwerk enthält:

$$\bar{t}_i = C^T \bar{c}_i \tag{3.4}$$

Dieser Vertrauensvektor ist ein Eigenvektor der Matrix C. Dies gibt dem Algorithmus seinen Namen. Häufig konvergieren Potenzen von C, woraus sich die folgende, einfachste (nicht-verteilte) Version des EigenTrust-Algorithmus ableiten lässt:

$$\begin{aligned} &\bar{t}_0 = \bar{e}; \\ &\textbf{repeat} \\ &\quad \bar{t}^{(k+1)} = C^T \bar{t}^{(k)} \\ &\quad \delta = \|t^{(k+1)} - t^{(k)}\| \\ &\textbf{until} \quad \delta < error; \end{aligned} \tag{3.5}$$

Die verteilte Version von EigenTrust ist deutlich komplizierter; deshalb wurde EigenTrust in dieser Bachelorarbeit nicht implementiert. [15]

3.5.2 CertainTrust

CertainTrust ist 2009 im Rahmen der Dissertation von Sebastian Ries [24] an der TU Darmstadt entstanden. CertainTrust basiert auf der Idee von Bayes'schen Vertrauensmodellen [33]. Bei diesen wird das Vertrauen in einen Knoten über ein Bayes'sches Netz berechnet. Es unterscheidet sich zu bisherigen Verfahren dieser Art hauptsächlich durch folgende Eigenschaften:

- Charakteristika des Anwendungskontextes können in Form von Parametern einfließen. Beispiele für solche Parameter sind: Grundvertrauen, Alterung von Hinweisen, maximale Anzahl Hinweise
- Es wird eine für Menschen verständliche Repräsentation eingeführt (Human Trust Interface, HTI) und definiert, wie sich diese auf die Bayes'sche Repräsentation abbilden lässt. Somit kann der Vertrauenswert sowohl von Menschen als auch von Computern interpretiert und aktualisiert werden.
- Wissen von Dritten über Interaktionspartner (Empfehlungen) kann einbezogen werden.

In CertainTrust ist es (in der HTI-Repräsentation) möglich, zusätzlich zu einem Vertrauenswert („trust") anzugeben, wie sicher man sich mit dieser Einschätzung ist („certainty"). Die Eingabe kann dabei über ein Diagramm erfolgen, bei der die x-Achse für den Vertrauenswert steht und die y-Achse für die Sicherheit der Einschätzung (siehe Abbildung 3.2). Dabei liefert ein Farbverlauf von rot zu grün einen Hinweis darauf, wie positiv oder negativ die Bewertung ist. In der Bayes'schen Repräsentation wird die Sicherheit der Einschätzung von der Anzahl positiver und negativer Evidenzen, also beispielsweise erfolgter positiver und negativer Interaktionen, abhängig gemacht. Außerdem wird sie kontextabhängig erfasst, sodass für Anwendungen, die ein unterschiedliches Risiko bergen, verschiedene Einschätzungen möglich sind. CertainTrust eignet sich zur Einstufung der Vertrauenswürdigkeit von Peers in Peer-to-Peer-Netzwerken, wurde aber wegen seiner hohen Komplexität nicht im Rahmen dieser Bachelorarbeit implementiert. [24]

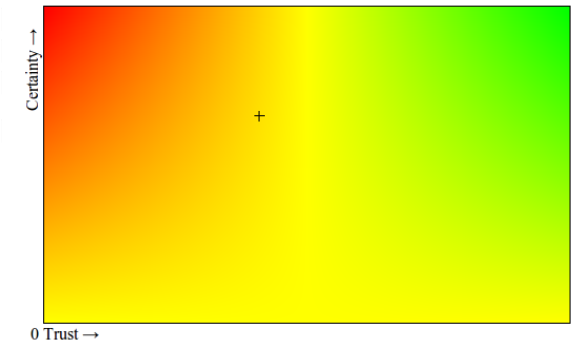


Abbildung 3.2.: Eingabeelement für die HTI-Repräsentation von CertainTrust

4 Konzept

4.1 BubbleStorm Web of Trust

In dieser Bachelorarbeit wird ein Web of Trust über das BubbleStorm-Netzwerk realisiert. Dabei generiert jeder Peer ein Schlüsselpaar, bestehend aus einem öffentlichen und einem privaten Schlüssel. Der private Schlüssel wird lokal abgespeichert und nur für das Signieren von anderen öffentlichen Schlüsseln verwendet. Der öffentliche Schlüssel wird als Durable Bubble in das BubbleStorm-Netzwerk gestellt. Durable Bubbles können prinzipiell durch jeden Peer geändert oder gelöscht werden. Der Datenspeicher für öffentliche Schlüssel im Backend der BubbleStorm-Anwendung ist jedoch so implementiert, dass nach dem Veröffentlichen eines Schlüssels nur noch Signaturen hinzugefügt werden können. Es ist also nicht möglich, Signaturen oder den öffentlichen Schlüssel zu ändern oder zu löschen.

Um den öffentlichen Schlüssel eines anderen Peers und die zugehörigen Signaturen zu empfangen, wird ein Lookup nach dem Fingerprint dieses Schlüssels im BubbleStorm-Netzwerk durchgeführt. Wenn nur die User-ID, nicht aber der Fingerprint bekannt ist, kann auch eine Suche ausgeführt und danach der passende Schlüssel ausgewählt werden. Für die Suche wird ein Bubblecast mit einer Instant Bubble durchgeführt. Die Peers, welche über den gesuchten öffentlichen Schlüssel verfügen, schicken daraufhin ihre Antwort direkt an den suchenden Peer. Der Vorgang ist in Abbildung 4.1 als „Matching Graph“ dargestellt.

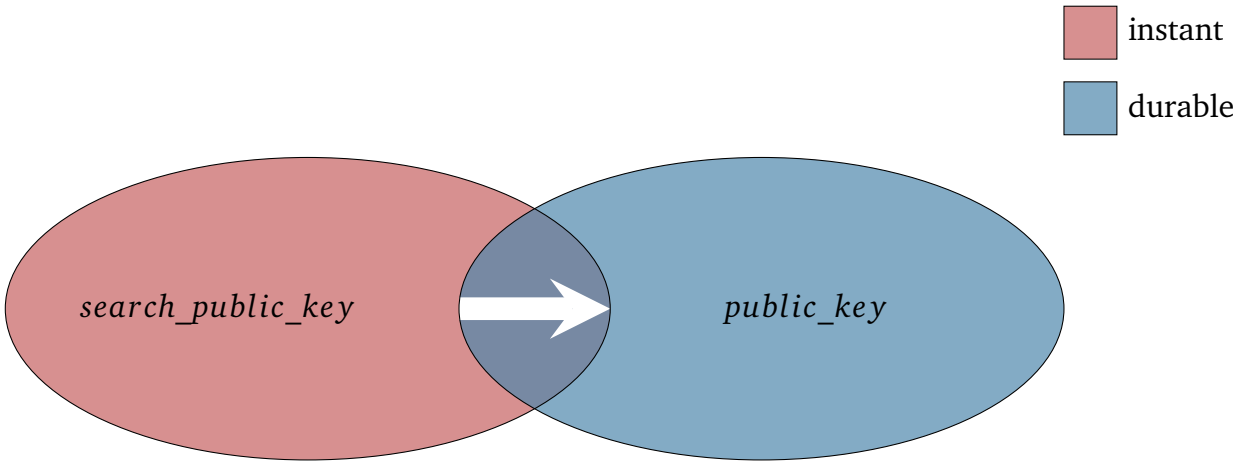


Abbildung 4.1.: Matching Graph für das Web of Trust

Anschließend werden die zu den Signaturen passenden öffentlichen Schlüssel heruntergeladen. Mit diesen öffentlichen Schlüsseln werden die Signaturen verifiziert. Aus den gültigen Signaturen wird berechnet, ob dem ursprünglich angefragten öffentlichen Schlüssel vertraut wird. Dazu muss der Peer jedoch wissen, wie sehr er den Schlüsseln der gültigen Signaturen vertrauen kann. Der Vorgang wiederholt sich also. In OpenPGP ist die Regel, dass der Pfad von signierten Schlüsseln vom Schlüssel K zum eigenen Schlüssel nicht länger als fünf Kanten sein darf. In dem in dieser Bachelorarbeit implementierten Web of Trust wurde der Owner Trust an die Signaturen gebunden (sogenannte Trust Signatures). Deshalb reicht es aus, einen Pfad von nur zwei Kanten zu betrachten. Dies liefert eine bessere Performance in einem Peer-to-Peer-Netzwerk, reduziert jedoch auch die Anzahl Peers, denen vertraut wird. Ansonsten werden die Regeln von OpenPGP übernommen.

Im Folgenden sind die Regeln des implementierten Web of Trust noch einmal aufgeführt. Beim Signieren eines Schlüssels können folgende Werte als Owner Trust gesetzt werden:

- unknown: für Benutzer, über die man keine weiteren Informationen besitzt
- none: für Benutzer, denen nicht zugetraut wird, die Authentizität von anderen richtig zu prüfen
- marginal: für Benutzer, denen nicht voll zugetraut wird, die Authentizität von anderen richtig zu prüfen
- complete: für Benutzer, deren Signatur als genauso gut angesehen wird, wie eine eigene

Ein öffentlicher Schlüssel K wird als gültig betrachtet, wenn er von genügend gültigen Schlüsseln signiert wurde. Dazu muss er

- selbst signiert worden sein,
- von mindestens einem Schlüssel signiert worden sein, dem voll vertraut wird (anpassbar)
- oder von mindestens drei Schlüsseln mit „marginal“-Trust signiert worden sein (anpassbar).

Dabei darf der Pfad von signierten Schlüsseln vom Schlüssel K zum eigenen Schlüssel nicht länger als zwei Kanten sein.

4.2 Bewertungen und Kommentare für Torrents

Die grafische Oberfläche zur Ansicht von Informationen eines Torrents im BubbleStorm-Netzwerk wird um drei Eingabeelemente erweitert:

- ein Button zum Bestätigen, dass es sich bei dem Torrent nicht um einen Fake handelt (also der Inhalt des Torrents zur Beschreibung passt)
- eine Dropdown-Liste zum Bewerten der Qualität (1 = sehr schlecht, 10 = sehr gut)
- ein Textfeld zum Verfassen von Kommentaren

Neben diesen neuen Eingabeelementen wird die Ansicht eines Torrents um die folgenden Ausgabeelemente erweitert:

- ein Feld „Verifications“, dessen Wert der Anzahl vertrauenswürdiger anderer Benutzer entspricht, welche beglaubigt haben, dass es sich bei diesem Torrent nicht um einen Fake handelt
- ein Feld „Rating“, dessen Wert der durchschnittlichen Bewertung vertrauenswürdiger anderer Benutzer entspricht
- eine Liste von Kommentaren zu diesem Torrent, die von vertrauenswürdigen Benutzern hinterlassen wurden

Zur Bestimmung, ob ein Benutzer vertrauenswürdig ist – also ob seine Eingaben in die Torrent-Ansicht einfließen – wird das im vorherigen Kapitel beschriebene Web of Trust verwendet.

4.3 Umverteilung von Replikaten beim Verlassen des Netzwerkes

In der bisherigen Version blieben in das BubbleStorm-Netzwerk gestellte Torrents nicht unbegrenzt verfügbar. Mit der neuen Klasse der Durable Bubbles lassen sich nun Daten unbegrenzt im BubbleStorm-Netzwerk speichern. Verlässt ein Peer das Netzwerk, werden bei ihm gespeicherte Replikate von Durable Bubbles auf andere Peers verteilt. Deshalb wurde die bisherige Version der verteilten Torrent-Suche an Durable Bubbles angepasst.

5 Implementierung

5.1 BubbleStorm Web of Trust

Die Implementierung des Web of Trust macht hauptsächlich Gebrauch von Funktionen von BubbleStorm und BouncyCastle. Im Folgenden wird erklärt, wie BubbleStorm und BouncyCastle eingebunden und genutzt werden. Außerdem wird auf die Implementierung der wichtigsten Funktionen des Web of Trust eingegangen.

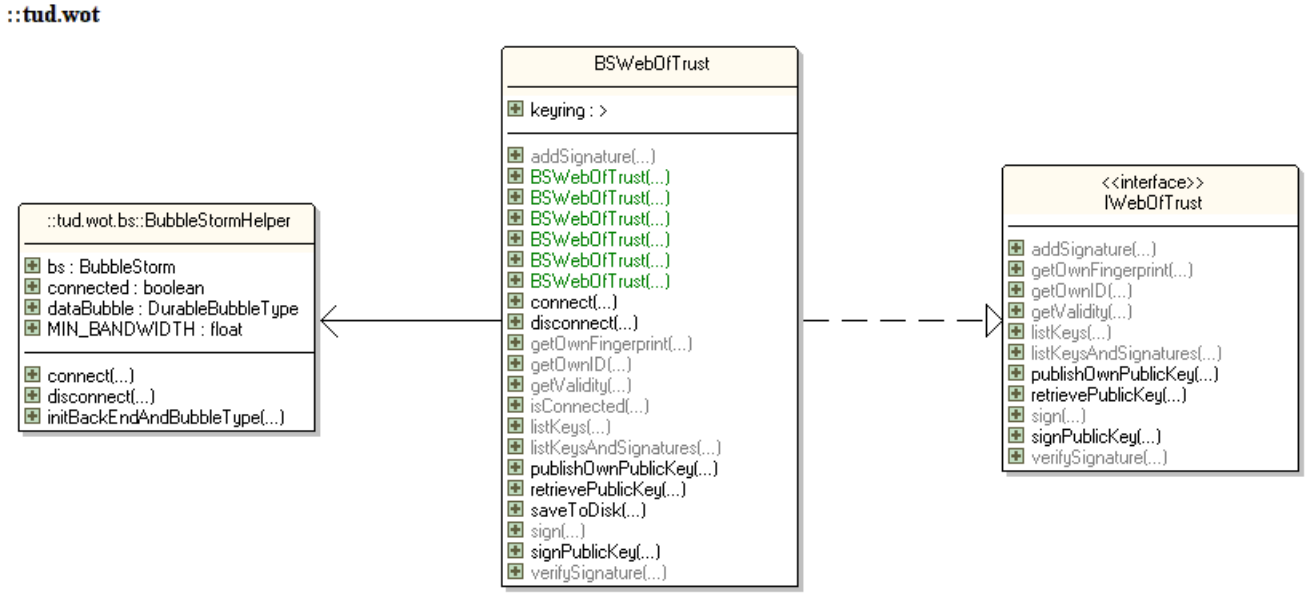


Abbildung 5.1.: Klassendiagramm des BubbleStorm Web of Trust

5.1.1 BubbleStorm

Die wichtigste Klasse des Web of Trust ist die BSWebOfTrust-Klasse (siehe Abbildung 5.1). Diese implementiert das IWebOfTrust-Interface, welches die minimalen Anforderungen an ein Web of Trust definiert. Die BSWebOfTrust-Klasse besitzt sowohl Konstruktoren zur Übergabe einer bestehenden BubbleStorm-Instanz als auch zur Übergabe der notwendigen Parameter zum Erzeugen einer neuen BubbleStorm-Instanz. Die anderen Konstruktoren kommen dadurch zu-

stande, dass die Wahl gelassen wird, ob Schlüssel aus einer Datei geladen werden oder neu generiert werden sollen.

Die BubbleStorm-Funktionalität wurde in eine eigene BubbleStormHelper-Klasse ausgelagert. Zum Verbinden mit dem BubbleStorm-Netzwerk wird die connect-Methode der BubbleStorm-Helper-Klasse aufgerufen. Diese benötigt eine Bootstrap-Adresse und einen Listen-Port[1]. In der Methode werden folgende Schritte zur Herstellung der Verbindung abgearbeitet:

1. Ein BubbleStorm-Endpunkt wird erzeugt.
2. Das Backend (die PGPPublicKeyStoreDurable-Klasse) wird initialisiert.
3. Ein Durable Data Bubble Typ mit diesem Backend wird erzeugt.
4. Eine Query, welche mit diesem Data Bubble Typ „matcht“, wird erzeugt.
5. Wurde als Bootstrap-Adresse „null“ übergeben, so wird ein neues BubbleStorm-Netzwerk aufgebaut.
6. Andernfalls wird dem BubbleStorm-Netzwerk über den Bootstrap-Peer beigetreten.

Existiert bereits eine BubbleStorm-Instanz, so wird die initBackEndAndBubbleType-Methode anstelle der connect-Methode aufgerufen. Bei dieser entfallen die Schritte 1, 5 und 6. Nachdem der Verbindungsvorgang abgeschlossen ist, lassen sich in der BSWebOfTrust-Klasse über das dataBubble-Feld, beziehungsweise das query-Feld, in der Helperklasse einfach Bubblecasts durchführen. Über das dataBubble-Feld sind außerdem Lookups möglich.

5.1.2 BouncyCastle

BouncyCastle ist eine Sammlung von Kryptographie-Bibliotheken. BouncyCastle enthält Schnittstellen für Java und C# und besteht aus zwei Hauptkomponenten: einer API für die zugrunde liegenden kryptographischen Algorithmen und dem JCE (Java Cryptography Extension) Provider. Letzterer wird durch Erweiterungen, wie PGP oder S-MIME, genutzt.

Im Web of Trust wird die Java Version von BouncyCastle mit der OpenPGP-Erweiterung (BCPG) benutzt. Ausschlaggebend für diese Entscheidung war, dass BCPG die einzige aktuelle und frei verfügbare OpenPGP-Implementierung ist, welche in Java programmiert wurde. Sie ist kompatibel zu anderen OpenPGP-Implementierungen, sodass sich Schlüsselringe aus dem weit verbreiteten GnuPG importieren lassen. Damit lässt sich der Schlüsselring auch mit einem anderen Programm bearbeiten (z. B. mit dem Thunderbird-Addon Enigmail).

[1] Port auf welchem auf eingehende Verbindungen gehorcht wird

Leider ist die Dokumentation lückenhaft und Bezeichnungen in BCPG unterscheiden sich teilweise zu denen in GnuPG. Beispielsweise enthält ein Keyring in BCPG nur Schlüssel eines einzigen Benutzers; für Schlüssel unterschiedlicher Benutzer muss eine Keyring Collection verwendet werden.

BouncyCastle wird von der BSWebOfTrust-Klasse für folgende Funktionen genutzt:

- Generieren von Schlüsselpaaren
- Verwalten von Schlüsseln in einem Schlüsselring
- Signieren eines öffentlichen Schlüssels
- Verifizieren einer Signatur eines Schlüssels
- Signieren von Daten
- Verifizieren einer Signatur über Daten
- Lesen des eigenen Schlüsselpaares aus einer secring.gpg-Datei
- Speichern des eigenen Schlüsselpaares in eine secring.gpg-Datei
- Lesen eines öffentlichen Schlüsselrings aus einer pubring.gpg-Datei
- Speichern eines öffentlichen Schlüsselrings in eine pubring.gpg-Datei

5.1.3 Schlüssel veröffentlichen

Damit der eigene öffentliche Schlüssel von anderen Peers gefunden werden kann, muss er im BubbleStorm-Netzwerk veröffentlicht werden. Dazu wird eine Durable Bubble erzeugt und ein Bubblecast durchgeführt. Als ID wird der Fingerprint des Schlüssels benutzt. Da dieser durch die kryptographische Hashfunktion SHA-1 berechnet wird, ist es unwahrscheinlich, dass zwei Schlüssel in einer Bubble mit der gleichen ID gespeichert werden sollen. Das Backend ist so implementiert, dass Schlüssel nur gespeichert werden, wenn noch kein Schlüssel mit der gleichen ID existiert und die ID mit dem tatsächlichen Fingerprint des Schlüssels übereinstimmt.

5.1.4 Schlüssel signieren

Um einen öffentlichen Schlüssel zu signieren und ihm einen Vertrauenswert zuzuweisen, wird dieser zunächst aus dem BubbleStorm-Netzwerk heruntergeladen (siehe *Schlüssel abrufen / suchen* [5.1.5]). Anschließend wird mit dem eigenen privaten Schlüssel eine Signatur für diesen öffentlichen Schlüssel erzeugt. Bei dieser sogenannten *Trust Signature* fließt der eingegebene Vertrauenswert mit ein. Es wird, wie in RFC 4880 (OpenPGP) empfohlen, für „marginal“-Trust der Wert 60 und für „complete“-Trust der Wert 120 benutzt. Die Signatur wird per Durable Bubble Update an alle die Bubble replizierenden Peers übertragen. Danach wird erneut der öffentliche Schlüssel heruntergeladen und überprüft, ob er die eigene Signatur enthält. Ist dies

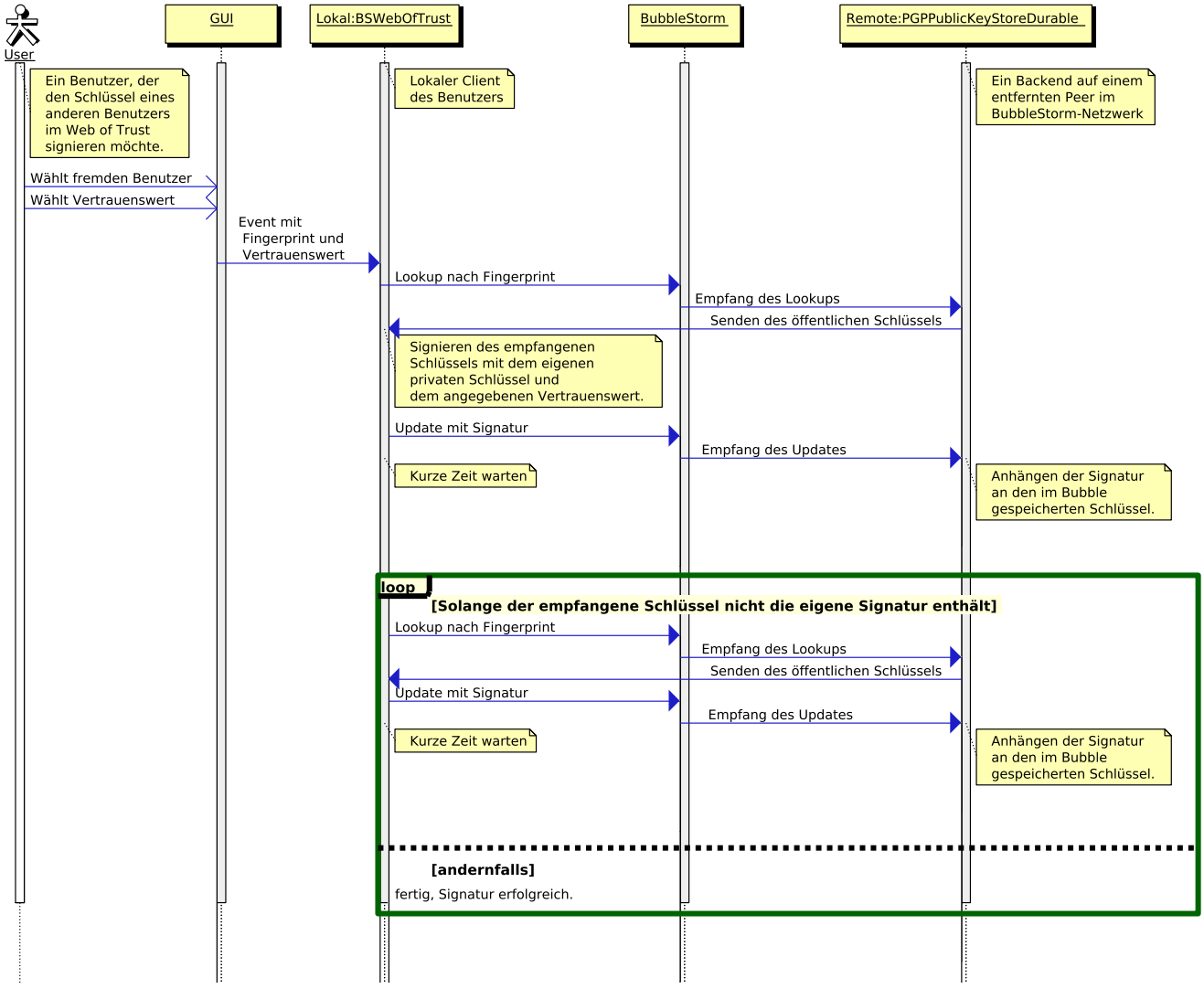


Abbildung 5.2.: Signieren eines öffentlichen Schlüssels im BubbleStorm-Netzwerk

nicht der Fall, so wird das Update mit der neu heruntergeladenen Durable Bubble erneut versucht. Der Vorgang wiederholt sich so lange, bis die eigene Signatur im heruntergeladenen öffentlichen Schlüssel enthalten ist. Das ist notwendig, weil in der Zwischenzeit ein anderer Peer die Durable Bubble aktualisiert haben könnte und Updates mit einem älteren Zeitstempel ignoriert werden. Für diesen Anwendungsfall wäre es eventuell sinnvoll, die Durable Bubbles von BubbleStorm um eine Methode zum Hinzufügen von Objekten zu erweitern. Der gesamte Vorgang ist in Abbildung 5.2 visualisiert.

5.1.5 Schlüssel abrufen / suchen

Zum Abrufen eines öffentlichen Schlüssels aus dem BubbleStorm-Netzwerk wird ein Lookup nach dem Fingerprint durchgeführt. Der heruntergeladene Schlüssel wird dem lokalen Schlüsselring hinzugefügt. Außerdem wird über das Eventsystem mitgeteilt, dass der Schlüssel abgerufen wurde.

Für die Suche nach einem öffentlichen Schlüssel wird eine Query (also eine Instant-Bubble) erzeugt, in welcher die User-ID des Schlüssels enthalten ist. Diese wird per Bubblecast in das BubbleStorm-Netzwerk repliziert. Empfängt ein Peer die Bubble, so prüft er in seinem Backend, ob bei ihm ein Schlüssel mit der User-ID der Query gespeichert ist. Ist dies der Fall, so überträgt er den Schlüssel direkt an den anfragenden Peer. Die Suche kann im Gegensatz zum Lookup mehrere Objekte zurückliefern und lässt sich prinzipiell um weitere Schlüsseleigenschaften erweitern.

5.1.6 Validität eines Schlüssels berechnen

Das Berechnen der Validität eines Schlüssels erfolgt über eine abgeänderte Version des entsprechenden OpenPGP-Algorithmus. Im Gegensatz zu diesem werden nur Pfade mit zwei Kanten betrachtet. Es existieren drei Validitätsstufen: none, marginal und complete. In Abbildung 5.3 ist der Ablauf der Berechnung dargestellt.

Zunächst wird der Schlüsselring aktualisiert (bei einem großen Schlüsselring sollten in Zukunft nur einzelne Schlüssel aktualisiert werden, sobald diese benötigt werden). Dann wird überprüft, ob der Fingerprint, für welchen die Gültigkeit berechnet werden soll, dem Fingerprint des eigenen Schlüssels entspricht. In diesem Fall soll die Gültigkeit des eigenen Schlüssels berechnet werden. Da diese immer „complete“ ist, wird „complete“ zurückgegeben.

Handelt es sich nicht um den eigenen Schlüssel, so wird überprüft, ob der Schlüsselring den betroffenen Schlüssel bereits enthält. Ist dies nicht der Fall, so wird er aus dem BubbleStorm-Netzwerk heruntergeladen. Ist dies nicht erfolgreich, so wird „none“ zurückgeliefert.

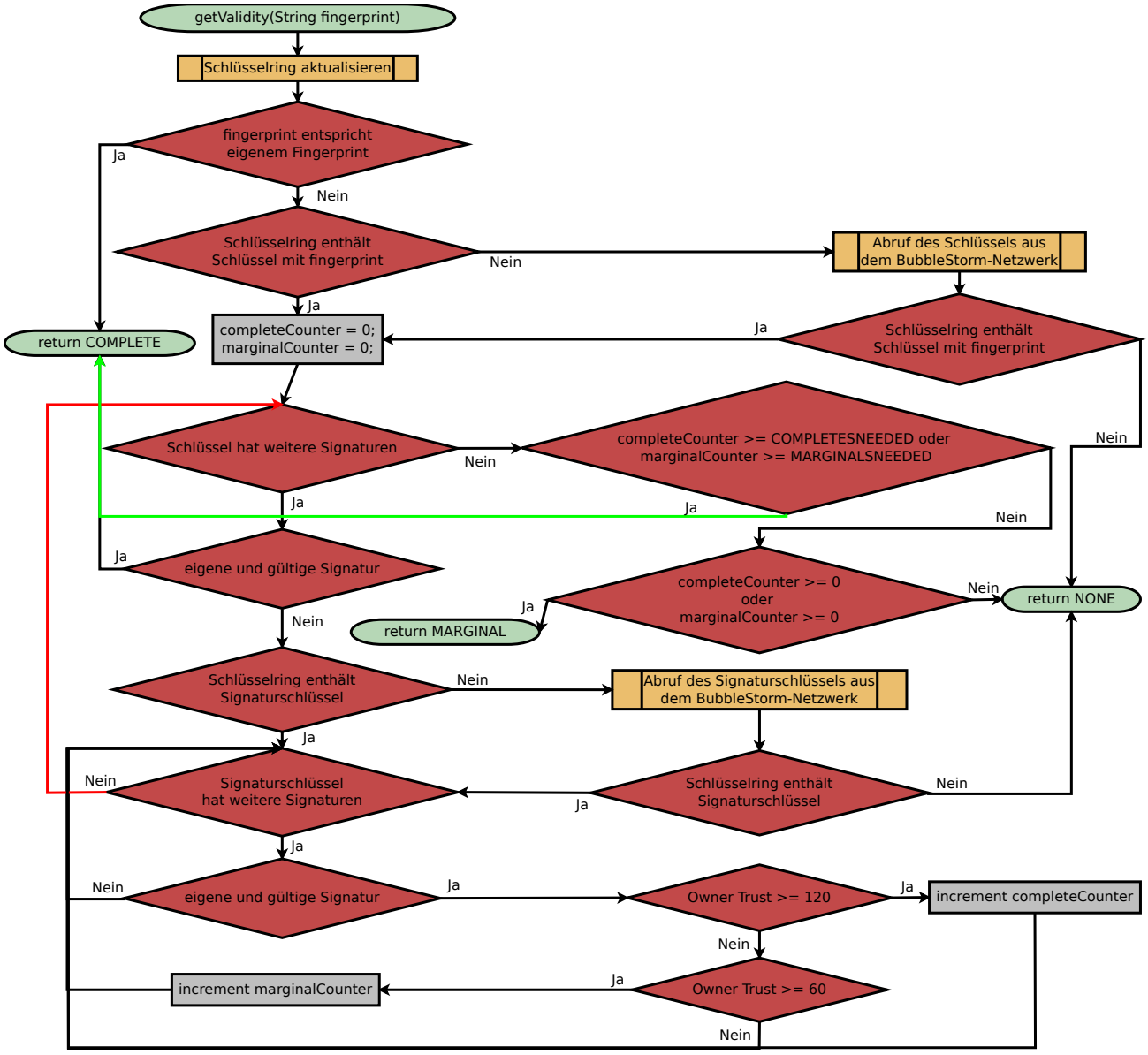


Abbildung 5.3.: Flussdiagramm für die Berechnung der Validität eines Schlüssels. Die Pfeilfarben dienen nur zum Unterscheiden der Pfeile.

Anschließend werden die Zähler für die Anzahl an „complete“- und „marginal“-OwnerTrusts initialisiert (completeCounter und marginalCounter). Danach wird durch die Signaturen des Schlüssels iteriert. Wird dabei eine eigene und gültige Signatur gefunden, so wird direkt „complete“ zurückgeliefert. Andernfalls wird überprüft, ob der Schlüsselring den Schlüssel, mit dem die Signatur erzeugt wurde, bereits enthält. Ist dies nicht der Fall, so wird er aus dem

BubbleStorm-Netzwerk heruntergeladen. Ist dies nicht erfolgreich, so wird „none“ zurückgeliefert.

Danach wird durch die Signaturen des Signaturschlüssels iteriert. Wird dabei eine eigene und gültige Signatur gefunden, so wird abhängig vom enthaltenen Owner Trust entweder der completeCounter (Owner Trust ≥ 120) oder der marginalCounter (Owner Trust ≥ 60) inkrementiert.

Sobald über alle Signaturen iteriert wurde, wird überprüft, ob genügend „complete“- oder „marginal“-OwnerTrusts gezählt wurden, um „complete“ zurückzuliefern. Die nötige Anzahl lässt sich in den Einstellungen definieren; standardmäßig werden mindestens ein „complete“-OwnerTrust oder drei „marginal“-OwnerTrusts benötigt. Wenn nicht genügend gezählt wurden, wird überprüft, ob überhaupt ein „complete“- oder ein „marginal“-OwnerTrust gezählt wurde. In diesem Fall wird „marginal“ zurückgeliefert, andernfalls „none“.

5.2 Bewertungen und Kommentare für Torrents

In dieser Komponente wird das Web of Trust benutzt, um die Vertrauenswürdigkeit von Peers festzustellen und so zu entscheiden,

- ob ein Torrent die Daten enthält, die er vorgibt, zu enthalten (also kein Fake ist),
- ob ein Kommentar zu einem Torrent angezeigt werden soll
- oder ob eine Bewertung eines Torrents in das Rating einfließt.

Dazu wurde das Backend der Torrent-Suche über BubbleStorm erweitert. Hinzugekommen ist eine Klasse SocialTorrent, in welcher ein Torrent und Signaturen dieses Torrents verwaltet werden. Durch das Signieren eines Torrents beglaubigen Peers, dass ein Torrent kein Fake ist. Zusätzlich enthält die Klasse eine Liste von Bewertungen und eine Liste von Kommentaren zu diesem Torrent. Zu jeder Bewertung und zu jedem Kommentar wird eine Signatur des Peers gespeichert, von welchem die Bewertung oder der Kommentar stammt.

Hinzugekommen ist außerdem die Klasse SocialSearchDataStore, welche die Klasse SearchDataStore ersetzt. Diese implementiert einen Datenspeicher für Durable Bubbles, sodass gespeicherte Daten durch unterschiedliche Peers geändert werden können. Es ist nicht wünschenswert, dass gespeicherte Torrents durch andere Peers geändert oder gelöscht werden können. Deshalb können nur weitere Signaturen eines Torrents, neue Kommentare oder Bewertungen hinzugefügt, aber nichts geändert oder gelöscht werden.

Sucht ein Peer nun im BubbleStorm-Netzwerk nach einem Torrent, so bekommt er nicht mehr ein TorrentItem als Antwort, sondern ein SocialTorrent. Dieses Objekt enthält alle Signaturen für den Torrent, alle Kommentare (jeweils mit Signatur) und alle Bewertungen (ebenfalls jeweils mit Signatur). Zunächst überprüft der Peer die Signaturen und sortiert ungültige Signaturen, Kommentare und Bewertungen aus. Anschließend benutzt er das Web of Trust, um die Vertrauenswürdigkeit der Peers zu bestimmen, von denen die Signaturen stammen. Zuletzt werden dem Benutzer der Torrent und die zugehörigen Kommentare angezeigt, welche von vertrauenswürdigen Peers stammen. Zusätzlich wird ihm ein Rating angezeigt, welches sich aus den Bewertungen von vertrauenswürdigen Peers ergibt, und wie viele vertrauenswürdige Peers den Torrent beglaubigt haben.

5.3 Umverteilung von Replikaten beim Verlassen des Netzwerkes

Um zu verhindern, dass Torrents, Bewertungen und Kommentare verloren gehen, wenn Peers das BubbleStorm-Netzwerk verlassen, implementiert die SearchDataStore-Klasse im Package tud.torrent.search nun das Interface DurableBubbleType.DataStore anstelle des Interfaces BubbleHandlerWithId. Dadurch wird die Bubble beim Verlassen auf einen anderen Peer repliziert und es gehen keine Daten verloren.

6 Benutzeroberfläche

6.1 Suche nach Torrents

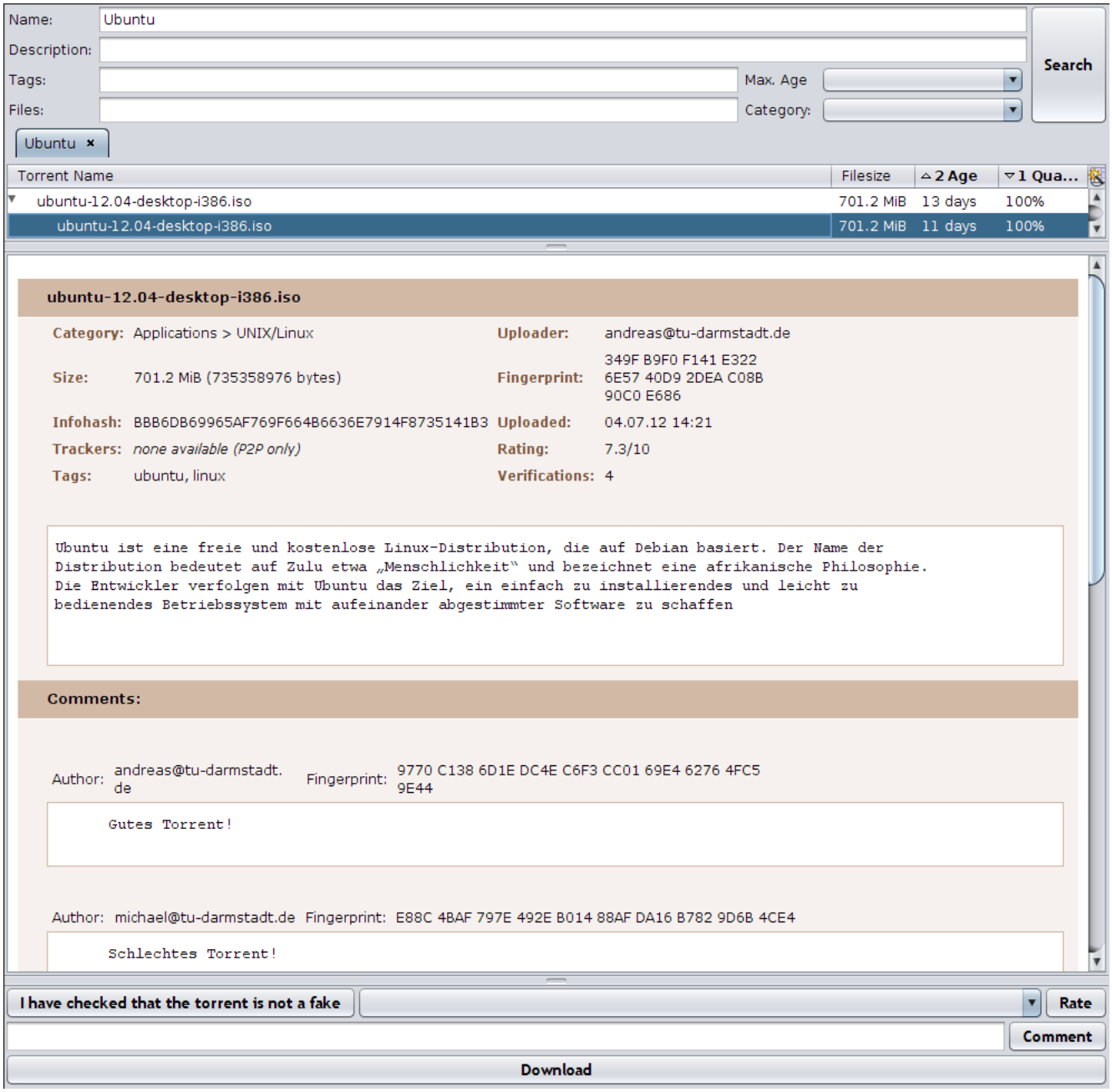


Abbildung 6.1.: Grafische Benutzeroberfläche für die um Kommentare und Rating erweiterte Suche nach Torrents im BubbleStorm-Netzwerk

Im Rahmen der Erweiterung der Benutzeroberfläche wurde darauf geachtet, dass der Stil der alten Oberfläche beibehalten wird, damit das Erscheinungsbild konsistent bleibt. Bei der Suche nach Torrents im BubbleStorm-Netzwerk wurden folgende Elemente hinzugefügt (siehe Abbildung 6.1):

- User-ID (Uploader) und Fingerprint des öffentlichen Schlüssels desjenigen, der den Torrent in das BubbleStorm-Netzwerk gestellt hat
- Ein Feld „Verifications“, dessen Wert der Anzahl vertrauenswürdiger anderer Peers entspricht, welche den Torrent beglaubigt haben
- Ein Feld „Rating“, welches aus den abgegebenen Bewertungen aller vertrauenswürdigen Peers berechnet wird
- Kommentare, die vertrauenswürdige Peers zu diesem Torrent abgegeben haben
- Eingabezeile für eigene Kommentare
- Dropdown-Liste für eine eigene Bewertung des Torrents
- Button zum beglaubigen, dass es sich bei dem Torrent um kein Fake handelt

Eigene Kommentare werden erst nach einer erneuten Suche nach dem Torrent angezeigt. Ebenso fließt die eigene Bewertung erst dann in das angezeigte Rating ein. Beim Kommentieren oder Bewerten erscheint ein Hinweis, dass der Kommentar oder die Bewertung in das BubbleStorm-Netzwerk transferiert wird und kein weiterer Hinweis über den Erfolg oder Misserfolg dieses Vorgangs.

6.2 Suche nach Schlüsseln

Neu hinzugekommen ist eine Benutzeroberfläche zum Suchen und Abrufen von öffentlichen Schlüsseln aus dem Web of Trust (siehe Abbildung 6.2). Gesucht werden kann nach dem Fingerprint, der User-ID und der Schlüssel-ID eines öffentlichen Schlüssels. Außerdem ist es möglich, die Anzahl der Suchergebnisse durch die Angabe des maximalen Alters des Schlüssels einzuschränken. Unter der Ergebnisliste werden die Signaturen des ausgewählten Schlüssels und der Schlüssel selbst angezeigt. Ganz unten lässt sich der ausgewählte Schlüssel signieren. Dazu wählt der Benutzer in der Dropdown-Liste einen Vertrauenswert aus (“marginal“ oder „complete“) und klickt anschließend auf Sign. Auch hier erscheint ein Hinweis, dass der Schlüssel signiert wird und kein weiterer Hinweis über den Erfolg oder Misserfolg des Signierens.

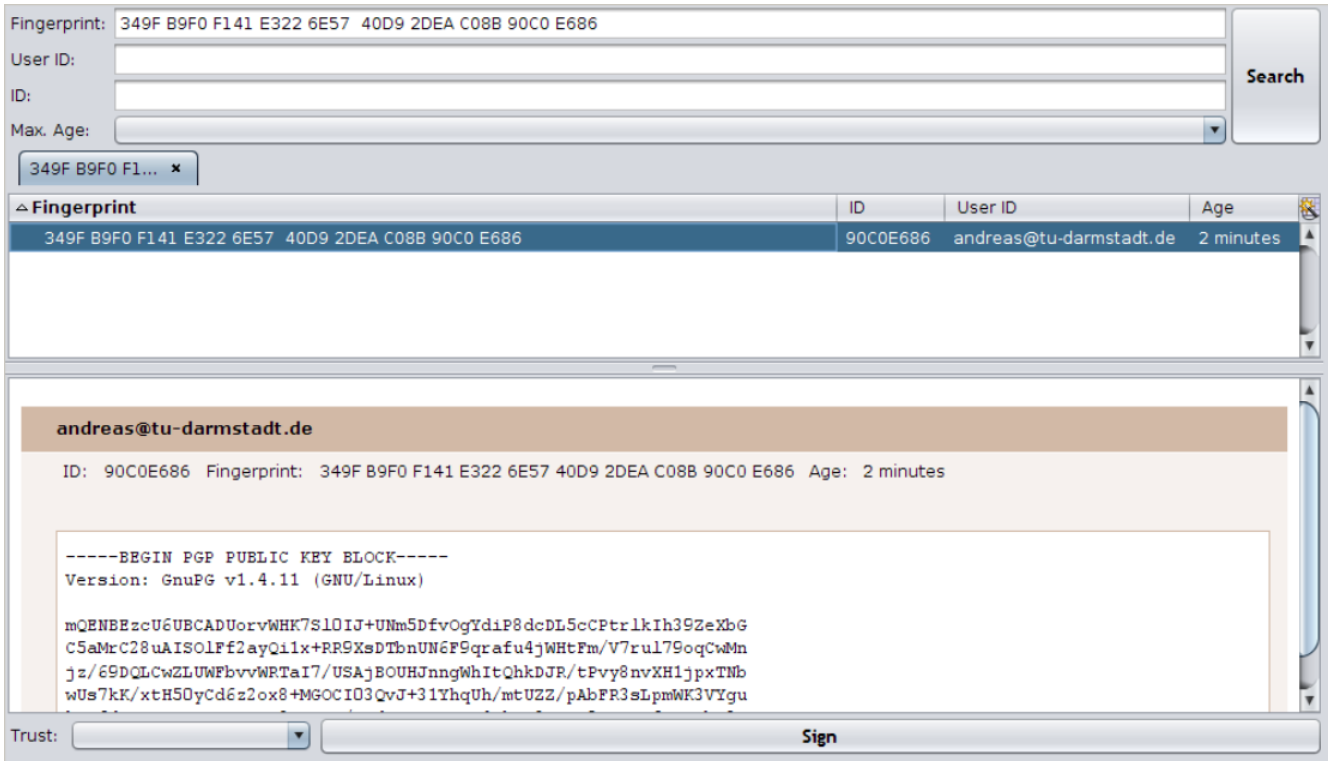


Abbildung 6.2.: Benutzeroberfläche für die Suche nach Schlüsseln im BubbleStorm-Netzwerk und das Signieren des ausgewählten Schlüssels mit einer Trust Signature

6.3 Anzeigen des Schlüsselrings

Neu hinzugekommen ist außerdem eine Benutzeroberfläche zur Anzeige des Schlüsselrings. Diese ähnelt der Oberfläche für die Suche nach Schlüsseln. Es entfallen jedoch die Felder für die Sucheingabe. Stattdessen gibt es Buttons zum manuellen Aktualisieren und zum Importieren oder Exportieren des Schlüsselrings. Dabei wird das GPG-Format benutzt, sodass sich der Schlüsselring prinzipiell auch in anderen Programmen bearbeiten lässt (beispielsweise im Thunderbird-Addon Enigmail). Außerdem können einzelne Schlüssel importiert oder exportiert werden.

7 Zusammenfassung

Das Ziel dieser Bachelorarbeit war es, die von Tilo Eckert in seiner Bachelorarbeit „Verteilte Suche für BitTorrent-Netzwerke“ [9] entwickelte Lösung zum Suchen und Verteilen von Daten um soziale Komponenten zu erweitern. Dazu wurde ein Bewertungssystem eingeführt, mit dem Benutzer die Qualität von Torrents bewerten können und angeben können, ob ein Torrent die Daten enthält, die es laut Titel und Beschreibung enthalten soll. Außerdem wurde die Möglichkeit eingebaut, Torrents zu kommentieren. Diese Funktion lässt sich sowohl zum Äußern von Meinungen als auch zum Stellen von Fragen benutzen. Die Bewertungen und Kommentare anderer Peers helfen dem Benutzer in der Entscheidung, ob er einen Torrent herunterladen möchte oder nicht.

Damit im Bewertungs- und Kommentarsystem keine Manipulationen vorgenommen werden können, wurde ein Web of Trust implementiert, welches ebenfalls das BubbleStorm-Netzwerk benutzt. Dieses Web of Trust ermöglicht es, dass dem Benutzer nur Kommentare von vertrauenswürdigen anderen Peers angezeigt werden und nur Bewertungen von vertrauenswürdigen anderen Peers in das Rating eines Torrents einfließen.

Das Web of Trust wurde so modular programmiert, dass es auch für andere Applikationen verwendet werden kann. Dies wird vor allem in Abbildung 7.1 deutlich, in der folgende Komponenten farbig hervorgehoben sind:

- neu hinzugekommene Komponenten: Web of Trust Backend, UI zur Schlüsselverwaltung und Schlüsselsuche
- erweiterte Komponenten: Suche UI (Bewerten und Kommentieren von Torrents)
- von neuen Komponenten genutzte Schichten: Event Manager, BubbleStorm und WAN

Ein weiteres Ziel dieser Bachelorarbeit war es, die „verteilte Suche für BitTorrent-Netzwerke“ den an BubbleStorm erfolgten Änderungen anzupassen. Dank der Durable Bubbles bleiben Torrents nun zeitlich unbegrenzt im Netzwerk. Das gleiche gilt für Bewertungen und Kommentare von Torrents.

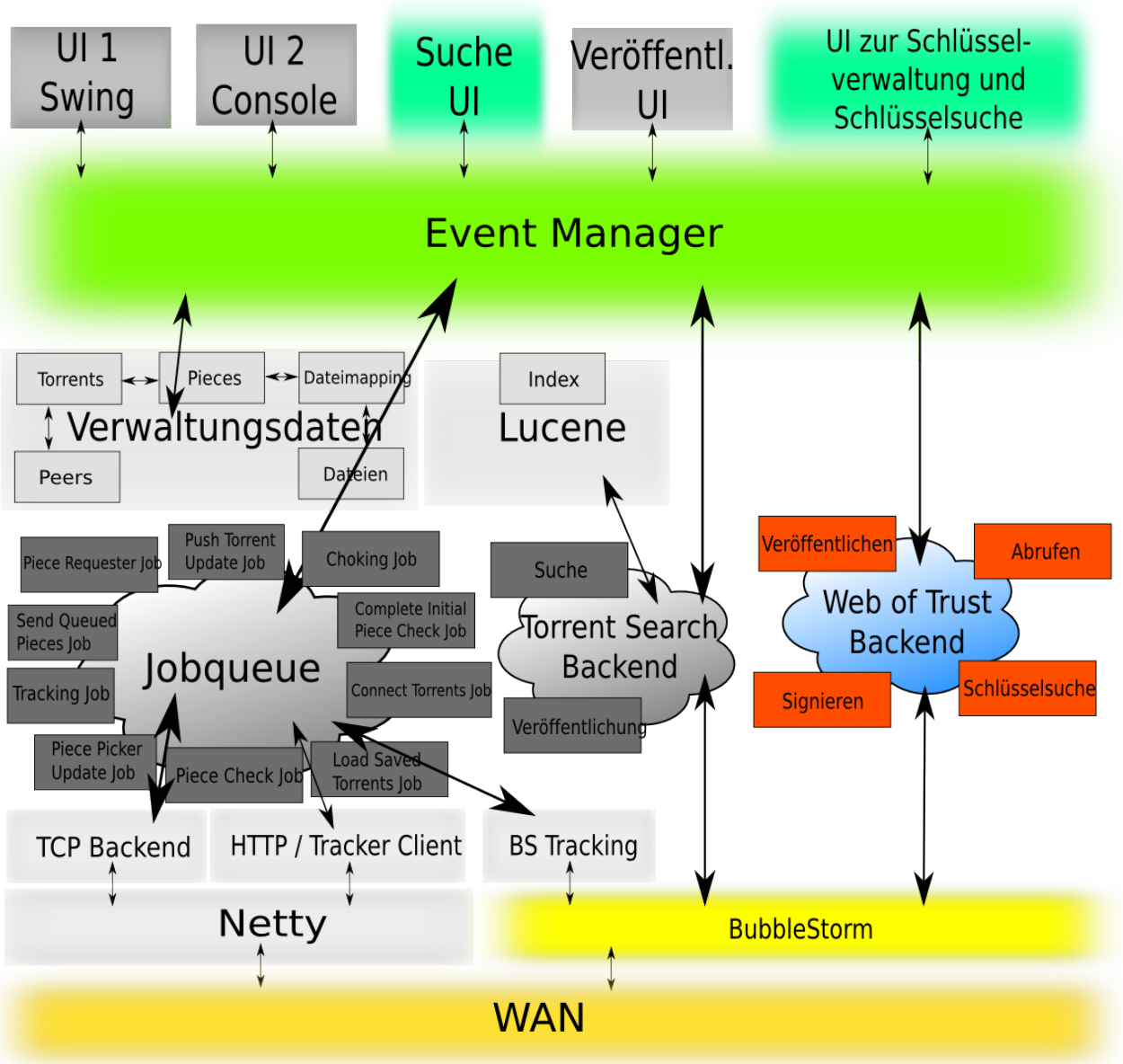


Abbildung 7.1.: Das Design von TUD Torrent mit seinen verschiedenen Schichten. Farbig hervorgehoben sind die in dieser Bachelorarbeit neu hinzugekommenen oder erweiterten Komponenten und für diese relevante Schichten.

8 Ausblick

In diesem Kapitel werden mögliche Weiterentwicklungen der in dieser Bachelorarbeit implementierten Software beschrieben.

8.1 Reputationsverfahren

In der derzeitigen Version des Web of Trust kommt nur ein einfacher Algorithmus zum Berechnen der Vertrauenswürdigkeit eines Peers zum Einsatz. Bei diesem an OpenPGP angelehnten Verfahren müssen Vertrauenswerte manuell durch den Benutzer festgelegt werden. Wenn ein Benutzer einen anderen Peer nur durch die von diesem in das BubbleStorm-Netzwerk gestellten Torrents oder Kommentare kennt, fällt es ihm schwer, diesen Vertrauenswert selbst sinnvoll zu bestimmen.

Deshalb wäre es wichtig, das System um ein Reputationsverfahren zu erweitern. Bei diesen Verfahren wird aus einzelnen Interaktionen mit anderen Peers eine Reputation für diese berechnet. Im Kontext von TUD Torrent bieten sich vor allem Torrents und Kommentare als Interaktion an. Der Benutzer bewertet dabei einen Torrent oder einen Kommentar als positive oder negative Interaktion. Aus allen so getätigten Bewertungen wird dann die Reputation des bewerteten Peers berechnet. Zwei solcher Reputationsverfahren sind im Abschnitt *Reputationsmanagement* (3.5) beschrieben.

8.2 Verändern und Löschen

Bisher ist es einzelnen Peers nur möglich, neue Schlüssel oder Signaturen in das BubbleStorm-Netzwerk einzustellen. Wünschenswert wäre es, wenn der Besitzer eines Schlüssels diesen auch wieder zurückziehen könnte, sodass ein kompromittierter Schlüssel keinen größeren Schaden anrichten kann. Außerdem wäre es sinnvoll, dass auch Signaturen zurückgezogen und Vertrauenswerte in Signaturen geändert werden können, da sich die Einschätzung eines Peers mit der Zeit ändern kann.

Auch das Ändern und Löschen von Torrents, Kommentaren und Bewertungen durch deren Urheber wäre eine wünschenswerte Funktion. Andernfalls bleiben Inhalte im Netzwerk, selbst wenn der Urheber sich von ihnen distanzieren möchte.

Damit sich eine solche Funktion nicht missbrauchen lässt, müssten Peers im BubbleStorm-Netzwerk im Backend überprüfen, ob der Peer, welcher einen Auftrag zum Ändern oder Löschen gegeben hat, dazu auch berechtigt ist. Der anfragende Peer signiert dazu seinen Auftrag kryptographisch. Danach verifizieren die Peers, zu denen der Auftrag propagiert wurde, die Signatur und überprüfen, ob sie zum Auftrag passt. Da ein einzelner Peer so einen Verifikationsvorgang, also eine rechenaufwändige Operation, auf mehreren anderen Peers im BubbleStorm-Netzwerk anstoßen kann, entsteht hier theoretisch die Möglichkeit, mit einem Angriff das Netzwerk zu überlasten. Eventuell könnte zum Verhindern eines solchen Angriffs wieder das in dieser Bachelorarbeit entwickelte Web of Trust eingesetzt werden.

8.3 Vertrauenswürdige BitTorrent-Peers

Das implementierte Web of Trust lässt sich nutzen, um sicherzustellen, dass sich der BitTorrent-Client nur mit vertrauenswürdigen Peers verbindet.

Ohne diese Absicherung kann ein Angreifer versuchen, das System für seine DDoS-Attacken zu missbrauchen; vorausgesetzt es hat eine ausreichend große Menge an Benutzern. Dazu müsste er lediglich die IP-Adresse des anzugreifenden Rechners in die dezentrale Peer-Suche eintragen. Der Angriff wäre besonders effektiv, wenn er dabei einen Port wählt, der bereits beim Verbinden auf der Gegenseite Rechenzeit beansprucht. Dies ist beispielsweise in schlechten Implementierungen von SSL der Fall.

Ohne diese Absicherung kann ein Angreifer außerdem die dezentrale Peer-Suche über BubbleStorm unmöglich machen, indem er mit vielen gefälschten IP-Adressen dem Schwarm beitritt. Damit reduziert er die Wahrscheinlichkeit, dass andere Peers im Schwarm sich mit einem „echten“ Peer verbinden.

Diese Absicherung verhindert keine Flooding- oder Spoofing-Attacken gegen Benutzer im BubbleStorm-Netzwerk. Es ist sogar das Gegenteil der Fall: Durch die kryptographischen Operationen besteht ein höherer Rechenaufwand und solche Angriffe werden einfacher. Schutzmaßnahmen gegen diesen Angriffsvektor müssten im Peer-to-Peer-Layer, also in BubbleStorm selbst, getroffen werden.

A Glossar

BitTorrent

BitTorrent ist ein Peer-to-Peer Filesharing-Protokoll. Für jeden Torrent wird ein separates Verteilnetz, Schwarm genannt, aufgebaut. Jeder Peer, der sich an diesem Schwarm beteiligt, lädt nicht nur Daten herunter, sondern verteilt die heruntergeladenen Daten schon während des Downloads automatisch auch an andere Peers im Schwarm. Dies ermöglicht eine schnelle Verteilung großer Datenmengen.

Bootstrap

Als Bootstrap bezeichnet man in einem Peer-to-Peer-Netzwerk einen Knoten, der neu hinzukommenden Knoten die notwendigen Informationen liefert, um erfolgreich dem Peer-to-Peer-Netzwerk beizutreten.

Bubble

Eine Bubble ist ein zyklenfreier Teilgraph über Knoten im BubbleStorm-Netzwerk. Die Daten einer Bubble werden auf alle Knoten in dieser Bubble repliziert. Es gibt verschiedene Bubble Klassen, die sich in ihrem Replikationsmodus unterscheiden. Diese sind Instant Bubbles, Fading Bubbles, Managed Bubbles und Durable Bubbles. Zudem gibt es zwei Arten von Bubbles: Query Bubbles und Data Bubbles.

Bubblecast

Als Bubblecast bezeichnet man die Replikation von Daten in einer Bubble von BubbleStorm.

BubbleStorm

BubbleStorm ist ein unstrukturiertes Peer-to-Peer-Netzwerk. Daten werden in sogenannten Bubbles auf einen Teil der Peers im Netzwerk repliziert. Bei diesem auch Bubblecast genannten Vorgang sind verschiedene Modi möglich (siehe Bubble).

BouncyCastle

BouncyCastle ist eine Sammlung von Kryptographie-Bibliotheken. Es enthält Schnittstellen für Java und C# und besteht aus zwei Hauptkomponenten: einer API für die zugrunde liegenden kryptographischen Algorithmen und dem JCE (Java Cryptography Extension) Provider. Letzterer wird durch Erweiterungen wie PGP oder S-MIME genutzt.

CertainTrust

CertainTrust ist ein Verfahren zum Reputationsmanagement. Es basiert auf der Idee von Bayes'schen Vertrauensmodellen und bietet darüber hinaus verschiedene Parameter und die Möglichkeit, Empfehlungen einzubeziehen. Außerdem besitzt es eine Schnittstelle für Menschen, sodass die Reputation einer Entität sowohl automatisch berechnet als auch interaktiv eingegeben werden kann.

DHT

In einer verteilten Hashtabelle (engl. distributed hash table, DHT) werden Daten auf mehreren Knoten gespeichert. Der Knoten, auf dem Daten gespeichert werden, bestimmt sich über eine konsistente (also kollissionsresistente, gleichverteilte und effiziente) Hashfunktion. Eine DHT besitzt mindestens zwei Funktionen: publish(Schlüssel, Inhalt) und lookup(Schlüssel).

EigenTrust

EigenTrust ist ein Algorithmus zum Reputationsmanagement in Peer-to-Peer-Netzwerken. Er basiert auf der Idee des transitiven Vertrauens.

Friend-to-Friend

Ein Friend-to-Friend-Netzwerk ist eine spezielle Form des Peer-to-Peer-Netzwerkes. Bei diesem verbindet sich jeder Peer nur mit als vertrauenswürdig eingestuften anderen Peers (Friends).

GnuPG

GnuPG (GNU Privacy Guard) ist ein Programm zum Verschlüsseln, Entschlüsseln und Signieren von Daten sowie zum Verifizieren von Signaturen. Es implementiert den OpenPGP-Standard nach RFC 4880 und wird hauptsächlich für E-Mails genutzt. Im Gegensatz zu PGP ist es Freie Software.

OpenPGP

OpenPGP ist ein Standard (RFC 4880) für Verschlüsselungssoftware. Es beschreibt das Format für verschlüsselte Daten, Signaturen und Zertifikate.

Owner Trust

Mit Owner Trust bezeichnet man das Vertrauen, das ein Benutzer dem Schlüssel eines anderen Benutzers zuordnet.

PGP

PGP (Pretty Good Privacy) ist ein Programm zum Verschlüsseln, Entschlüsseln und Signieren von Daten sowie zum Verifizieren von Signaturen. Es wurde 1991 von Phil Zimmermann entwickelt. Im Gegensatz zu GnuPG ist es kommerzielle Software.

Peer

Als Peer bezeichnet man einen Teilnehmer in einem Peer-to-Peer-Netzwerk.

Peer-to-Peer

In einem Peer-to-Peer-Netzwerk ist jeder Teilnehmer (Peer) gleichrangig. Es unterscheidet sich vom Client-Server-Modell. Dort verbinden sich meistens viele Clients mit nur einem Server.

Reputation

Die Reputation eines Peers ist ein Maß für seine Vertrauenswürdigkeit. Sie ergibt sich in Reputationsverfahren aus den erfolgten Interaktionen, welche positiv oder negativ bewertet wurden.

Schlüssel

Als Schlüssel bezeichnet man in der Kryptologie eine Information zum Verschlüsseln, Entschlüsseln und Signieren von Daten sowie zum Verifizieren von Signaturen. In asymmetrischen Verfahren existiert ein öffentlicher Schlüssel zum Verschlüsseln von Daten und Verifizieren von Signaturen und ein geheimer Schlüssel zum Entschlüsseln und Signieren von Daten.

Schwarm

Als Schwarm bezeichnet man alle Peers, die gerade einen Torrent herunterladen oder anbieten.

Signatory Trust

Mit Signatory Trust bezeichnet man das Vertrauen eines Benutzers in eine Signatur.

SPOF

Ein Single Point of Failure ist eine Systemkomponente, deren Defekt den Ausfall des gesamten Systems zur Folge hat.

Torrent

Die Torrent-Datei enthält Informationen wie Dateigrößen, Prüfsummen und Tracker URLs für zusammengehörende Inhalte. Sie wird benötigt, um an einem Schwarm teilzunehmen, also um Daten herunterzuladen und zu anderen Peers hochzuladen.

Tracker

Ein Tracker vermittelt Peers zueinander. Sucht ein Peer nach anderen Peers, so fragt er den Tracker nach einer Liste bekannter Peers für einen Torrent. Gleichzeitig registriert er sich beim Tracker, sodass er durch andere Peers gefunden werden kann.

Web of Trust

Mit Web of Trust (WoT) wird ein Konzept bezeichnet, um die Authentizität eines öffentlichen Schlüssels sicherzustellen, also dass der Schlüssel dem angegebenen Benutzer gehört. Dies geschieht über das gegenseitige Signieren von öffentlichen Schlüsseln und das Setzen von Vertrauenswerten (Owner Trust) für Schlüssel.

B Abbildungsverzeichnis

C Tabellenverzeichnis

D Literatur

[1] *Apache Lucene*, aufgerufen am 03.07.2012. Adresse: https://lucene.apache.org/.

[2] M. Bieg, *File:P2P-network.svg*, Wikimedia Commons, aufgerufen am 03.07.2012, 2007. Adresse: https://commons.wikimedia.org/w/index.php?title=File:P2P-network.svg&oldid=37889931.

[3] —, *File:Server-based-network.svg*, Wikimedia Commons, aufgerufen am 03.07.2012, 2007. Adresse: https://commons.wikimedia.org/w/index.php?title=File:Server-based-network.svg&oldid=37031544.

[4] *Bittorrent Protocol Specification v1.0. Stand Februar 2012*, aufgerufen am 03.07.2012. Adresse: http://wiki.theory.org/index.php?title=BitTorrentSpecification&oldid=4886.

[5] *Chord*, aufgerufen am 03.07.2012. Adresse: http://pdos.csail.mit.edu/chord/.

[6] I. Clarke, O. Sandberg, B. Wiley und T. Hong, „Freenet: A distributed anonymous information storage and retrieval system“, in *Designing Privacy Enhancing Technologies*, Springer, 2001, S. 46–66. DOI: 10.1007/3-540-44702-4_4.

[7] B. Cohen, „Incentives build robustness in BitTorrent“, in *Workshop on Economics of Peer-to-Peer systems*, aufgerufen am 03.07.2012, Bd. 6, 2003, S. 68–72. Adresse: http://www.bittorrent.org/bittorrentecon.pdf.

[8] *Dropbox*, aufgerufen am 03.07.2012. Adresse: https://www.dropbox.com/.

[9] T. Eckert, *Bachelorarbeit: Verteilte Suche für BitTorrent-Netzwerke*, Technische Universität Darmstadt, 2010.

[10] T. Eckert und A. Teuber, *Praktikum - Entwicklung eines BitTorrent-Clients*, Technische Universität Darmstadt, 2009.

[11] Eldred, *File:Freenet - Ablauf einer Anfrage.png*, aufgerufen am 03.07.2012, 2009. Adresse: https://commons.wikimedia.org/w/index.php?title=File:Freenet_-_Ablauf_einer_Anfrage.png&oldid=39636840.

[12] K. Graffi, A. Kovacevic, P. Mukherjee, M. Benz, C. Leng, D. Bradler, J. Schröder-Bernhardi und N. Liebau, „Peer-to-Peer-Forschung - Überblick und Herausforderungen (Peer-to-Peer Research - Overview and Challenges)“, *it - Information Technology*, Bd. 49, Nr. 5, S. 272–279, 2007. DOI: 10.1524/itit.2007.49.5.272.

[13] G. Hazel und A. Norberg, *Extension for Peers to Send Metadata Files*, aufgerufen am 03.07.2012, 2008. Adresse: http://www.bittorrent.org/beps/bep_0009.html.

[14] R. Hitchens, *Java Nio*. O'Reilly Media, 2002.

[15] S. D. Kamvar, M. T. Schlosser und H. Garcia-Molina, „The Eigentrust algorithm for reputation management in P2P networks", in *Proceedings of the 12th international conference on World Wide Web*, Ser. WWW '03, aufgerufen am 03.07.2012, Budapest, Hungary: ACM, 2003, S. 640–651, ISBN: 1-58113-680-3. DOI: 10.1145/775152.775242. Adresse: http://doi.acm.org/10.1145/775152.775242.

[16] C. Leng, „BubbleStorm: Replication, Updates, and Consistency in Rendezvous Information Systems", Diss., TU Darmstadt, 2012.

[17] P. Maymounkov und D. Mazieres, „Kademlia: A peer-to-peer information system based on the xor metric", *Peer-to-Peer Systems*, S. 53–65, 2002. DOI: 10.1007/3-540-45748-8_5.

[18] A. Norberg, L. Strigeus und G. Hazel, *Extension Protocol*, aufgerufen am 03.07.2012, 2008. Adresse: http://www.bittorrent.org/beps/bep_0010.html.

[19] A. Ofterdinger, „Java Native Interface ab J2SE 1.4", *JavaSpektrum*, Bd. 5, S. 32–35, 2005.

[20] Ogmios, *File:Web of Trust.svg*, aufgerufen am 03.07.2012, 2009. Adresse: https://commons.wikimedia.org/w/index.php?title=File:Web_of_Trust.svg&oldid=34125750.

[21] *OpenPGP Message Format*, aufgerufen am 03.07.2012. Adresse: https://tools.ietf.org/html/rfc4880.

[22] *Pastry*, aufgerufen am 03.07.2012. Adresse: http://www.freepastry.org/.

[23] *RetroShare*, aufgerufen am 03.07.2012. Adresse: http://retroshare.sourceforge.net/.

[24] S. Ries, „Certain trust: a trust model for users and agents", in *Proceedings of the 2007 ACM symposium on Applied computing*, Ser. SAC '07, aufgerufen am 03.07.2012, Seoul, Korea: ACM, 2007, S. 1599–1604, ISBN: 1-59593-480-4. DOI: 10.1145/1244002.1244342. Adresse: http://doi.acm.org/10.1145/1244002.1244342.

[25] H. Schulze und K. Mochalski. (2009). Internet Study 2008/2009. aufgerufen am 03.07.2012, ipoque, Adresse: http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf.

[26] W. W. Terpstra, J. Kangasharju, C. Leng und A. P. Buchmann, „BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search", in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, Ser. SIGCOMM '07, ACM, Bd. 37, Kyoto, Japan, 2007, S. 49–60, ISBN: 978-1-59593-713-1. DOI: 10.1145/1282427.1282387.

[27] —, „BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search (Slides)“, Ser. SIGCOMM ’07, aufgerufen am 03.07.2012, ACM, Kyoto, Japan, 2007. Adresse: http://www.dvs.tu-darmstadt.de/publications/ppt/bubblestorm-sigcomm.ppt.

[28] *The Annotated Gnutella Protocol Specification v0.4*, aufgerufen am 03.07.2012. Adresse: http://rfc-gnutella.sourceforge.net/developer/stable/index.html.

[29] *The GNU Privacy Handbook*, aufgerufen am 03.07.2012. Adresse: http://www.gnupg.org/gph/en/manual.html.

[30] *The International PGP Home Page*, aufgerufen am 03.07.2012. Adresse: http://www.pgpi.org/.

[31] *Turtle F2F*, aufgerufen am 03.07.2012. Adresse: http://www.turtle4privacy.org/new/.

[32] *Verteilte Hashtabelle*, aufgerufen am 03.07.2012. Adresse: https://de.wikipedia.org/w/index.php?title=Verteilte_Hashtabelle&oldid=99463937.

[33] Y. Wang und J. Vassileva, „Bayesian Network-Based Trust Model“, in *Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence*, Ser. WI ’03, aufgerufen am 03.07.2012, Washington, DC, USA: IEEE Computer Society, 2003, S. 372–, ISBN: 0-7695-1932-6. Adresse: http://dl.acm.org/citation.cfm?id=946251.946986.

[34] *Web of Trust*, aufgerufen am 03.07.2012. Adresse: https://wiki.freenetproject.org/Web_of_Trust.

[35] P. R. Zimmermann, *The official PGP user’s guide*. Cambridge, MA, USA: MIT Press, 1995, ISBN: 0-262-74017-6.