Bachelorarbeit

Verteilte Suche für BitTorrent-Netzwerke

Tilo Eckert

Version 1.0 **30. September 2010**

Autor: Tilo Eckert

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

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet Datenbanken und Verteile 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 30. September 2010

_____

Tilo Eckert

Inhaltsverzeichnis

1 Einleitung

Das Austauschen von Dateien in Netzwerken, insbesondere dem Internet, ist ein essenzieller Bestandteil der heutigen Kommunikation. Es existieren unzählige Lösungen für dieses Problem. Die Mehrheit dieser Lösungen basiert auf dem Client-Server-Ansatz: Ein Rechner (Client) fordert eine oder mehrere Dateien bei einem Zentralrechner (Server) an, dieser liefert die Dateien an den Client aus. Die verwendeten Protokolle haben neben dem reinen Übertragen von Dateien oft noch weitere Aufgaben. Dies können unter Anderem die Absicherung der Daten vor Übertragungsfehlern, das Wiederaufnehmen abgebrochener Transfers, das Sichern der Daten vor unautorisiertem Einblick auf dem Übertragungsweg (Verschlüsselung), Flußkontrolle, Überlastkontrolle oder die Übertragung von Metadaten sein.

Für die Auslieferung kleiner Dateien, wie sie meistens im World Wide Web zu finden sind, skaliert die Client-Server-Architektur in den meisten Fällen ausreichend gut. Sollen aber große Datenmengen in einem kurzen Zeitraum an eine große Anzahl von Nutzern übertragen werden, dann kann es mit diesem Ansatz zu starken Problemen kommen, denn das Übertragen großer Datenmengen von einem zentralen Server erfordert hohe Bandbreiten auf dem Übertragungsweg und oft auch teure Hardware. Sind diese beiden Faktoren nicht ausreichend dimensioniert, ist die Datenübertragung nur sehr langsam oder schlimmstenfalls gar nicht mehr möglich. Neben den hohen Anschaffungs- und Instandhaltungskosten ist der Betrieb einer entsprechenden Infrastruktur auch mit hohem Arbeitsaufwand verbunden. Dies gilt insbesondere dann, wenn die Empfänger der Daten auf der ganzen Welt verteilt sind, denn dies erfordert meist entsprechend verteilte Rechenzentren, damit die Übertragungsstrecke ausreichend kurz ist, um hohe Übertragungsgeschwindigkeiten zu erreichen und eine Überlastung von Netzknoten zu vermeiden. Aus diesem Bedürfnis heraus sind Content Delivery Netzwerke (CDN) entstanden. [CDN02] Zu den bekanntesten davon zählt Akamai mit mehreren tausend, global verteilten Servern zur Auslieferung von Inhalten.

1.1 Peer-To-Peer Netzwerke

Eine kostengünstige und effiziente Alternative zur Verteilung von Daten bieten Peer-To-Peer (P2P) Netzwerke. Es sind keine zentralen Server nötig. Ein einzelner Rechner, der die kompletten zu verteilenden Daten besitzt, kann bereits ausreichend sein. Im Gegensatz zur Client-Server-Architektur ist der Empfänger (Client) nicht nur passiv am Datentransfer beteiligt, sondern überträgt während des Herunterladens bereits fertiggestellte Teile der Daten wiederum an andere Nutzer. Er übernimmt damit gleichzeitig die Rolle von Client und Server, weshalb er auch Peer genannt wird. Gegenüber dem Client-Server-Ansatz hat dies einige wichtige Vorteile:

1. Der Ausfall eines einzelnen Peers hat nur geringe Folgen für die Verteilung der Daten. Fällt der zentrale Server aus, ist kein Herunterladen mehr möglich (Single Point of Failure).

2. Der Datenverkehr - und damit die Kosten - werden unter allen Peers aufgeteilt.

3. Da alle Peers mit hoher Wahrscheinlichkeit auf mehrere Netze verteilt sind, werden die Übertragungsnetze entlastet, da die Daten über viele verschiedene Netzsegmente übertragen werden, anstatt dass alle eines oder einige wenige Segmente passieren müssen.

1.2 BitTorrent

BitTorrent [BT03] ist ein solches Peer-To-Peer Netzwerk und laut ipoque Internetstudie 2008/2009 [IPO08] das am weitesten verbreitete Protokoll zum dezentralen Austausch von Dateien. Sein relativ einfaches, aber dennoch flexibles Protokoll hat es durch zahlreiche Anwendungsszenarien beliebt gemacht. Es wird etwa zum Verteilen von Software, Updates, kommerzieller und freier Musik, Videos, Filmen und anderen Inhalte sowohl kommerziell als auch nicht-kommerziell genutzt. Auch zum Tauschen von privaten Daten unter Freunden wird es eingesetzt. Es spielt keine große Rolle, an wie viele Peers Daten verteilt werden sollen, da jeder neue Peer sich automatisch an der Weiterverbreitung beteiligt. Durch die gegenseitige Unterstützung beim Herunterladen der Daten kommt ein schwarmartiges Verhalten zustande, weshalb die Summe aller an einem Torrent beteiligten Peers auch als Schwarm bezeichnet werden.

Die zwei wichtigsten Komponenten für den Datentausch mit BitTorrent sind die Torrent-Datei und das Wissen um andere Peers, die dasselbe Torrent verteilen. Die wenige Kilobytes große Torrent-Datei wird in einem BitTorrent-Programm geöffnet und enthält Meta-Informationen

über die zu verteilenden Dateien, Prüfsummen über Datenblöcke fester Größe und einen oder mehrere Tracker zum Finden weiterer Peers. Die gesuchte Torrent-Datei kann allerdings oft schwer zu finden sein, da Torrents zumeist über Webseiten angeboten werden, deren Zahl sehr groß ist. Zudem sind solche Seiten auch nicht immer verfügbar oder werden abgeschaltet, wodurch sich die betreffenden Torrents nicht mehr auffinden lassen. Die Suche nach Peers kann über die in der Torrent-Datei eingetragenen Tracker (meist ein HTTP-Server) erfolgen. Dies hat jedoch den großen Nachteil, dass es nicht mehr möglich ist, weitere Peers anzufordern, wenn keiner der eingetragenen Tracker verfügbar ist. Um dieses Problem zu lösen, gibt es mehrere existierende Tracking-Ansätze, die es ermöglichen, Informationen über andere Peers ohne eine zentrale Komponente zu erhalten. Diese werden im Abschnitt *Ähnliche Systeme* näher behandelt.

Den Datenaustausch nehmen die beteiligten Peers untereinander vor. Beim Verbindungsaufbau mit einem anderen Peer wird als erstes ein Handshake durchgeführt, bei dem sich die beiden Peers identifizieren und sich über das relevante Torrent, unterstützte Protokollerweiterungen, sowie die bereits fertiggestellten Teile des Torrents austauschen. Nach dem erfolgreichen Abschluß des Handshakes beginnt der eigentliche Tausch der Daten. Jeder Peer entscheidet anhand eines Algorithmus, in welcher Reihenfolge angefragte Datenblöcke gesendet und welche Datenblöcke angefordert werden. Diese Komponente ist jedoch nicht Bestandteil dieser Arbeit.

Nähere Details zum Protokoll lassen sich in der Spezifikation nachlesen. [Spec10]

1.3 Ziel der Arbeit

In dieser Bachelorarbeit geht es um einen weiteren P2P-Ansatz. Auf Basis der probabilistischen, erschöpfenden Suche des Peer-to-Peer Systems BubbleStorm wurde eine Möglichkeit entwickelt, um Informationen über Peers in einem dezentralen Netzwerk abzulegen und wiederzufinden. Die in dieser Arbeit vorgestellte Lösung wurde in eine java-basierte BitTorrent-Applikation für den praktischen Einsatz integriert.

Aus Benutzersicht hat BitTorrent ein großes Usability-Problem: das Finden von Torrents. Diese sind über hunderte von Internetseiten verstreut, was es für den Benutzer schwierig machen kann, für ihn interessante Torrents zu finden. Als eine mögliche Lösung wurde eine Volltextsuche zum Auffinden von Torrents über das BubbleStorm-Netzwerk implementiert. Durch die Eingabe von Suchbegriffen erhält der Benutzer, wie bereits aus anderen Systemen zum Tauschen von Dateien bekannt, eine Liste der verfügbaren Torrents. Wenn sich der Benutzer für ein Torrent entscheidet, wird dieses ebenfalls aus dem dezentralen Netzwerk angefordert und anschließend über das BitTorrent-Protokoll heruntergeladen.

Das Ziel war es, BitTorrent unabhängig von jeder zentralen Infrastruktur zu machen und alle dafür nötigen Dienste in ein einziges dezentrales Netzwerk zu integrieren. Dadurch soll die Robustheit des Protokolls gestärkt und die Benutzung durch leichter zu findende Torrents vereinfacht werden.

2 Vorarbeiten

Es liegen zwei weitere Arbeiten zu Grunde, auf die das in dieser Arbeit entwickelte System aufbaut.

2.1 BubbleStorm

2.1.1 Kurzbeschreibung

BubbleStorm [BBS07] ist ein unstrukturiertes Peer-To-Peer Netzwerk, das das Suchen von Daten auf probabilistische und erschöpfende Weise durchführt. Im Gegensatz zu anderen P2P Systemen, wie z. B. verteilten Hash-Tabellen (DHT), sind sowohl Daten als auch die Netzwerktopologie strukturlos. Dies ermöglicht es BubbleStorm auch bei größeren Ausfällen von Netzwerkknoten die Topologie in kurzer Zeit zu heilen und Suchanfragen erfolgreich zu beantworten. BubbleStorm wird im Fachgebiet Datenbanken und Verteilte Systeme des Fachbereichs Informatik der Technischen Universität Darmstadt unter der Leitung von Prof. Alejandro P. Buchmann entwickelt.

2.1.2 Idee

Die Topologie von BubbleStorm basiert auf einem zufälligen Multigraphen. Dieser Graph ist nicht regelmäßig. Jedem Knoten in diesem Graph kann ein Grad zugewiesen werden, der proportional zur Bandbreite ist, da der durchschnittliche Traffic auf allen Kanten identisch ist. Dadurch kann die Last in einem Netzwerk mit unterschiedlichen Bandbreiten einzelner Knoten leichter verteilt werden.

Dem Grad eines Knotens ist stets ein fester Wert zugeordnet. Tritt ein Knoten dem Netzwerk bei, dann wird jeweils eine neue Kante zwischen zwei Knoten, zwischen denen bereits eine Kante besteht, und dem beitretenden Knoten aufgebaut. Wurden die beiden neuen Kanten erfolgreich hergestellt, dann wird die Kante zwischen den beiden zuvor bereits integrierten Knoten aufgelöst. Verlässt ein Knoten das Netzwerk, wird dies rückgängig gemacht. Die zurückbleiben-

den Nachbarn bauen also jeweils eine Kante untereinander auf. Dadurch bleibt der Grad jedes Knotens (bis auf den beitretenden bzw. verlassenden Knoten) zu jedem Zeitpunkt konstant.

Daten und Suchanfragen werden in so genannten Bubbles durch das Netz verschickt, die aus einer Teilmenge aller Knoten im Netzwerk gebildet werden. Dies erfolgt in einem so genannten Bubblecast. Ein Bubblecast repliziert Bubbles auf Peers im Netzwerk. Dazu wird eine Mischung aus Random Walk und Flooding angewendet. Einem Bubblecast wird, neben den zu replizierenden Daten, ein Gewicht w (Anzahl der Repliken) und ein Teilungsfaktor s zugeordnet. Beim Empfang eines Bubblecasts wird w um eins dekrementiert und die Replik lokal verarbeitet, das Datum also gespeichert bzw. die Suchanfrage bearbeitet. Wenn w nicht auf null gesunken ist, dann wird der Bubblecast an s zufällig gewählte Nachbarn weitergeleitet, wobei die Weiterleitung niemals an den Sender des Bubblecasts erfolgt. Das hat zur Folge, dass Nachbarn von Knoten mit hohem Grad nicht mehr durchschnittlichen Traffic erhalten als Knoten von Nachbarn mit geringerem Grad, was in einem reinen Flooding System nicht gilt. Beim Weiterleiten eines Bubblecasts wird außerdem darauf geachtet, dass dieser bei Existenz von Doppelkanten nicht zweimal an denselben Nachbarn oder bei einer Schleife mit dem eigenen Knoten nicht an sich selbst gesendet wird, da dies die Anzahl der Repliken reduzieren würde. Größere Schleifen werden allerdings nicht erkannt und führen entsprechend zu einer niedrigeren Replikenzahl. Zufallsgraphen neigen jedoch dazu, eher große Schleifen zu bilden, sodass dies selten auftritt.

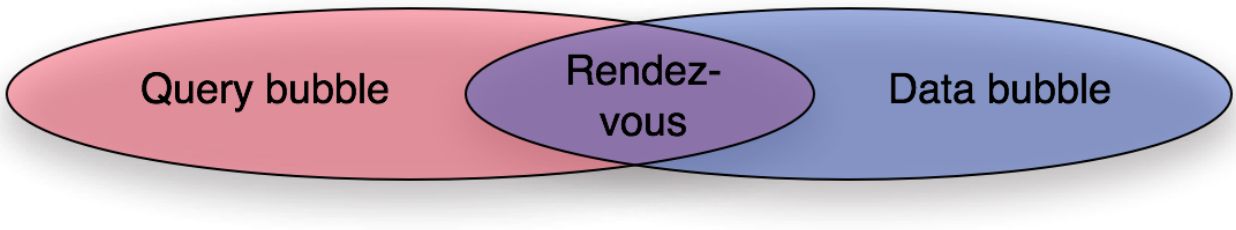

Abbildung 2.1: Aufeinandertreffen von Query Bubble und Data Bubble [BBS07, Seite 2]

Eine Suchanfrage ist immer genau dann erfolgreich, wenn sie durch einen Bubblecast einen Knoten erreicht, der zuvor Daten eines Data Bubbles erhalten und gespeichert hat, die bei der Suche im lokalen Datenbestand einen Treffer ergeben. Ein Knoten bei dem Query Bubble und Data Bubble aufeinandertreffen, wird als Rendezvous-Knoten bezeichnet. Dies wird in Abbildung 2.1 illustriert. Der Suchalgorithmus ist von BubbleStorm nicht definiert. Er wird von der jeweiligen Applikation implementiert, wodurch beliebig komplexe Suchanfragen möglich sind. Bei einem Treffer wird dieser direkt an den Absender der Suchanfrage übermittelt.

Wie bereits erwähnt, verfolgt BubbleStorm bei der Suche nach Daten einen probabilistischen Ansatz. Dies bedeutet, dass es keine Garantie gibt, eine Antwort zu erhalten. Jedoch lässt sich die Wahrscheinlichkeit einer ergebnislosen Suche minimieren: Angenommen über n Knoten werden

ein Data Bubble der Größe d und ein Query Bubble der Größe s gleichmäßig per Zufall verteilt, dann ist die Wahrscheinlichkeit, dass kein Ergebnis zustande kommt kleiner als $e^{-ds/n}$. Das Produkt $d \cdot s$ ist dabei das wichtigste Merkmal, denn durch Erhöhung von d und Absenkung von s (oder umgekehrt) kann die Wahrscheinlichkeit erhalten werden. Wenn beispielsweise gilt, dass $ds = 4n$, dann beträgt die Wahrscheinlichkeit kein Ergebnis zu erhalten $e^{-4} \approx 0,018$. Anhand dieser Beobachtung lässt sich ein Kompromiss finden, der möglichst zuverlässige Suchergebnisse ermöglicht. Weitere Details hierzu sind in [BBS07] Abschnitt 2.4 zu finden.

2.1.3 Implementierung und Verwendung

BubbleStorm ist als Bibliothek in der funktionalen Programmiersprache StandardML geschrieben. Um die Bibliothek in einer Applikation zu benutzen, steht eine API zur Verfügung. Diese API bietet einen sehr hohen Grad der Abstraktion. Die Applikation kommt mit keinerlei Funktionalität in Berührung, die die Funktion des Netzwerkes betrifft. Die Anwendung hat daher keine Kontrolle darüber, wie Kanten auf- und abgebaut werden, sowie ob und wie Daten mit Nachbarn ausgetauscht werden. Das Protokoll, der Aufbau und Erhalt des Graphen und die Replikation werden transparent durchgeführt. Zur Nutzung der Bibliothek in anderen Programmiersprachen als StandardML stehen Bindings für verschiedene Programmiersprachen zur Verfügung. Zum Zeitpunkt dieser Bachelorarbeit waren Bindings für C++ und Java verfügbar.

In der vorliegenden Bachelorarbeit kommt die BubbleStorm API mit Binding für Java zum Einsatz. Dieses bindet die von BubbleStorm benötigten Bibliotheken über das Java Native Interface (JNI) [JNI05] ein. Da auf plattformspezifischen Programmcode zurückgegriffen wird, müssen die Bibliotheken für die jeweilige Plattform separat kompiliert werden.

Die für diese Arbeit verwendete BubbleStorm Java API beschränkt sich im Wesentlichen auf folgende Funktionen:

- Übergabe von Bootstrap-Adressen
- Beitritt zum Netzwerk (Join)
- Verlassen des Netzwerks (Leave)
- Senden von Suchanfragen und Bubblecasts

Es ist jedoch zu beachten, dass sich die API noch im Entwicklungsstadium befindet. Die zur Verfügung stehenden Funktionen können sich daher dem Bedarf entsprechend ändern.

2.2 TUD Torrent

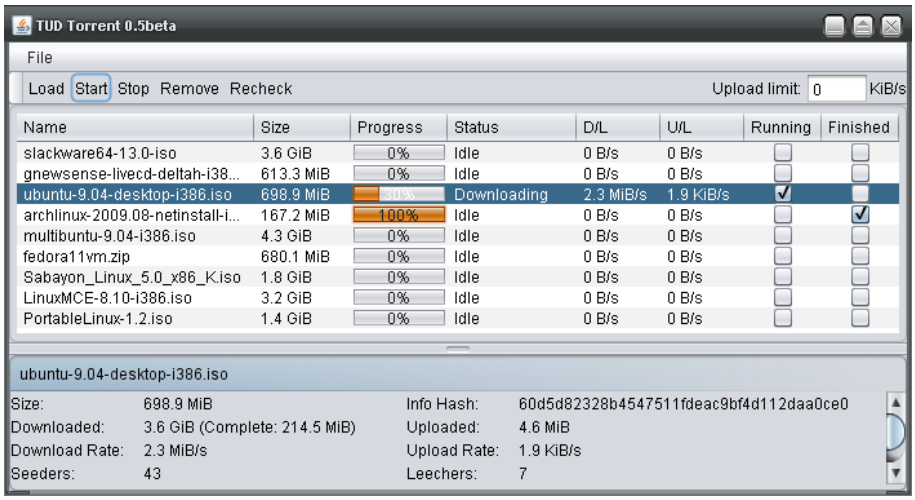

Abbildung 2.2: Screenshot der grafischen Oberfläche von TUD Torrent

2.2.1 Kurzbeschreibung

TUD Torrent [TUT09] ist ein plattformunabhängiger BitTorrent-Client. Er wurde von Andreas Teuber und Tilo Eckert im Jahr 2009 als Praktikum an der Technischen Universität Darmstadt entwickelt. TUD Torrent wurde in Java entwickelt, ist event-basiert und in der Lage mehrere Prozessorkerne zu verwenden.

2.2.2 Funktionen

Implementiert wurde das Basisprotokoll, wie es in der Spezifikation [Spec10] zu finden ist. Es ist also alle notwendige Funktionalität vorhanden, um mit anderen Benutzern Torrents auszutauschen. Dazu gehören das Erhalten von Peer-Informationen von einem HTTP-Tracker und die Kommunikation der Peers eines Torrents untereinander.

Darüber hinaus ist TUD Torrent modular aufgebaut und eignet sich daher für diverse Erweiterungen, wovon in dieser Arbeit Gebrauch gemacht wird. Es existieren zwei Benutzeroberflächen: eine grafische Benutzeroberfläche und eine Kommandozeilenversion. Die grafische Benutzeroberfläche unterstützt das Tauschen mehrerer Torrents gleichzeitig (Screenshot in Abb. 2.2). Die Kommandozeilenversion erlaubt jedoch nur ein Torrent.

2.2.3 Design

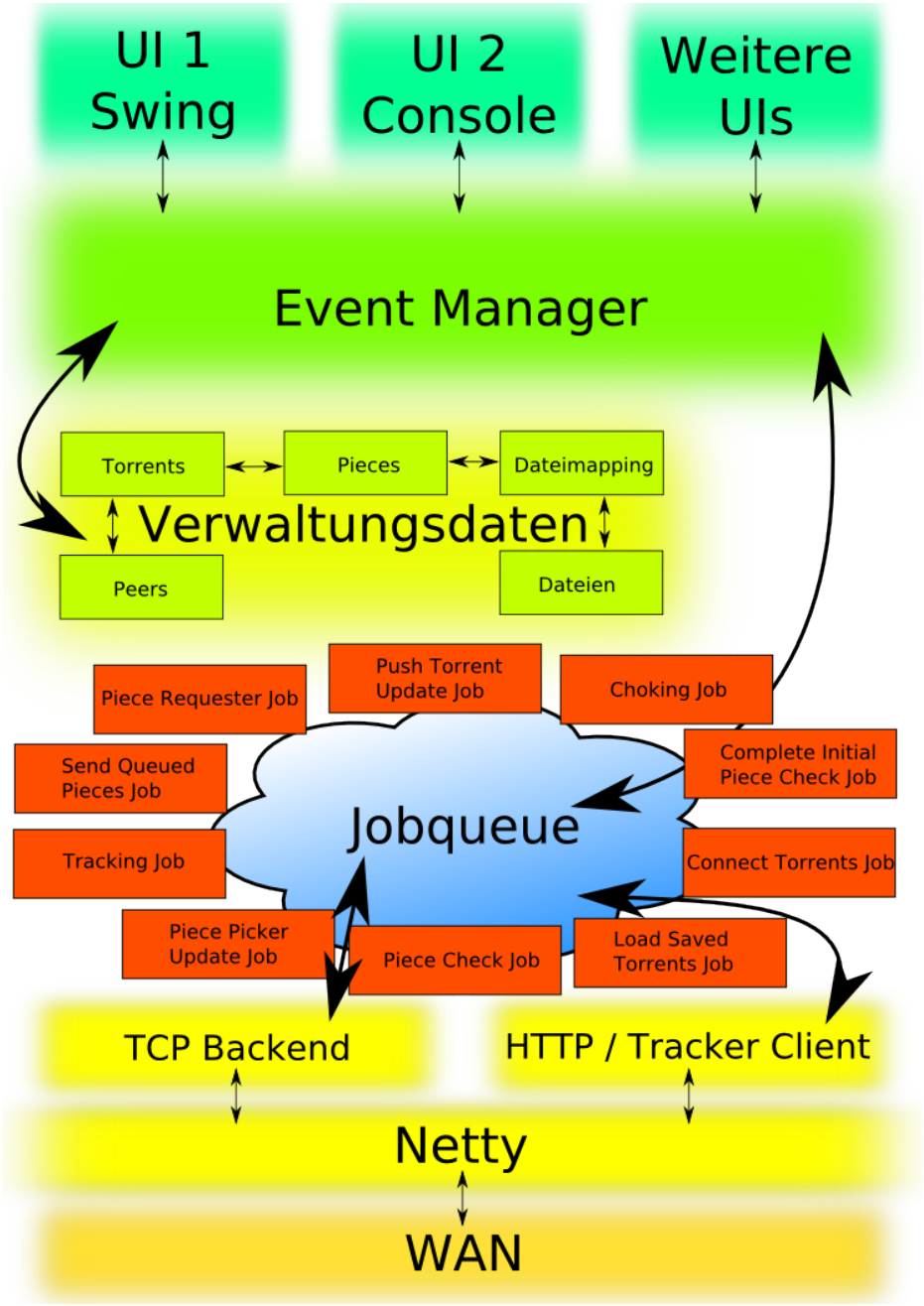

Abbildung 2.3: Grafik der Ebenen des Designs von TUD Torrent [TUT09, Seite 6]

TUD Torrent ist in mehrere logische Ebenen unterteilt, wodurch einige Komponenten bei Bedarf ausgetauscht werden können. Ebenso sind so Erweiterungen leichter umzusetzen. In Abbildung 2.3 ist das Design der Anwendung schematisch dargestellt. Die Ebenen haben (von unten nach oben) folgende Funktionen:

WAN

Stellt das Netzwerk dar, über das kommuniziert wird (üblicherweise das Internet).

Netty

Netty ist eine Java Bibliothek, die die Nutzung von Java NIO [NIO02] zur asynchronen, nichtblockierenden Kommunikation erleichtert. Die von Trustin Lee entwickelte Bibliothek ist für Einsatzzwecke entwickelt worden, die hohe Skalierbarkeit erfordern. Netty kann für Server- und Client-Anwendungen eingesetzt werden. Die Bibliothek stellt eine Pipeline bereit, die mit Upstream- und Downstream-Handlern gefüllt wird. Diese Handler eignen sich zur Implementierung des Protokoll-Stacks der Anwendung. Es können Prüfungen und Transformationen der zu sendenden bzw. der empfangenen Daten vorgenommen werden. Jeder Handler kann dabei von mehreren Verbindungen gleichzeitig genutzt werden oder dediziert für jede Verbindung existieren. Beide Arten können auch in einer Pipeline gemischt vorkommen.

TCP Backend

Das TCP Backend ist für die P2P-Komunikation mit anderen BitTorrent-Clients zuständig. Es implementiert das Kommunikationsprotokoll von BitTorrent, baut also Verbindungen auf, sendet Daten an andere Peers und erzeugt Jobs für empfangene Daten, die in die Job Queue eingefügt werden. Es verwendet dazu die Netty-Bibliothek.

HTTP / Tracker Client

Das Tracking, also das Finden anderer Peers, geschah bisher ausschließlich über die am weitesten verbreitete Methode, nämlich per HTTP-Anfrage. In regelmäßigen Zeitabständen wird eine Anfrage an den Tracker geschickt, der eine Liste von IP/Port-Kombinationen (und implementationsabhängig weitere Details) zurückliefert.

Job Queue

Der größte Teil der Programmlogik ist in Jobs unterteilt. Dies sind kleine Arbeitseinheiten, die in sehr kurzer Zeit auszuführen sind. Beispiele dafür sind das Senden einer Anforderung

nach Torrent-Teilen oder das Speichern empfangener Daten. Diese Warteschlange lässt sich von mehreren Threads parallel abarbeiten, wodurch mehrere CPU-Kerne genutzt werden können.

Verwaltungsdaten

In dieser Ebene wird der Zustand der Torrents gespeichert. Dazu zählen auch die bekannten Peers, fertige Blöcke (Pieces) und das Mapping von Torrent-Pieces auf Dateien. Letzteres ist nötig, da ein Torrent in gleich große Pieces aufgeteilt wird. Ein Piece kann sich über einen Teil einer Datei oder mehrere Dateien eines Torrents erstrecken.

Event Manager

Der Event Manager ist die Schnittstelle zwischen der Benutzeroberfläche und dem Kern des Programms. Es kommt der Publish-Subscribe-Ansatz zum Einsatz. Der Event Manager stellt mehrere Channels bereit, über die Events gesendet werden. So gibt es z. B. einen Channel für Push-Updates, über den eine GUI automatisch über Zustandsänderungen benachrichtigt wird und Channels für Polling-Updates. Programmkern, respektive grafische Oberfläche können sich beim Event Manager für einen oder mehrere Channels registrieren und werden über zukünftige Event-Nachrichten informiert, die an den jeweiligen Channel gesendet werden. Damit der Empfänger eines Events die Antwort auf eine Anfrage identifizieren kann, wird bei Antworten immer eine Referenz auf das ursprüngliche Event mitgesendet, das die Anfrage beinhaltet.

User Interfaces

Da der Event Manager die einzige Schnittstelle vom User Interface zum eigentlichen BitTorrent-Client ist, lassen sich mehrere unterschiedliche UI-Typen (z. B. GUI, Konsole, Web Interface) gleichzeitig nutzen. Es ist auch möglich zur Laufzeit das User Interface auszutauschen oder ganz abzukoppeln.

2.2.4 Tracking

Der für das dezentralisierte Finden von Peers über BubbleStorm relevante Teil von TUD Torrent ist der Tracking-Mechanismus. Es werden beliebig viele Tracking-Methoden unterstützt. Jede Tracking-Methode muss ein Interface (ITracking) implementieren. Neben Funktionen für das

Abfragen von Statusinformationen wie den Aktivierungszustand, Bezeichner, Anzahl fertiger und nicht fertiger Peers, sowie Statustexten, müssen vier Funktionen implementiert werden, die vom BitTorrent-Client in entsprechenden Situationen aufgerufen werden:

- **update()**: Diese Methode wird alle 30 Sekunden aufgerufen, wenn die Tracking-Methode und das zugehörige Torrent aktiv sind. Sie dient zum regelmäßigen Aktualisieren der Peer-Informationen. Falls Updates seltener als alle 30 Sekunden ausgeführt werden sollen, muss dieser Aufruf entsprechend oft ignoriert werden. Ein Update darf sowohl blockierend, als auch nicht-blockierend ausgeführt werden. Nicht-blockierende Implementierungen sind allerdings zu bevorzugen, da andernfalls nachfolgende Aufgaben verzögert werden können.

- **start()**: Wenn ein Torrent vom inaktiven in den aktiven Zustand wechselt, wird diese Methode aufgerufen. Dies erlaubt es, dass sich der eigene Peer beim Start des Torrents gegebenenfalls registrieren kann (z. B. bei einer HTTP-Tracker).

- **stop()**: Entsprechend wird diese Methode beim Wechsel in den inaktiven Zustand ausgeführt, um eine eventuelle Abmeldung zu erlauben.

- **complete()**: Mit der Fertigstellung des Torrents wird diese Methode aufgerufen. So kann eine entsprechende Signalisierung gesendet werden.

Die Tracking-Objekte, die jedem Torrent als Instanz der jeweiligen Tracking-Methode zugewiesen werden, werden von einer Factory-Klasse erzeugt. Beim Erzeugen wird der Factory-Methode der Name der Tracking-Methode, das Torrent, der Port für eingehende Verbindungen und zwei weitere Strings übergeben, die zusätzliche Parameter wie beispielsweise URLs enthalten können.

Zur Verdeutlichung der Funktionsweise sind im Klassendiagramm in Abb. 2.4 alle Klassen und ihre Abhängigkeiten dargestellt, die für das Tracking über das HTTP-Protokoll wichtig sind.

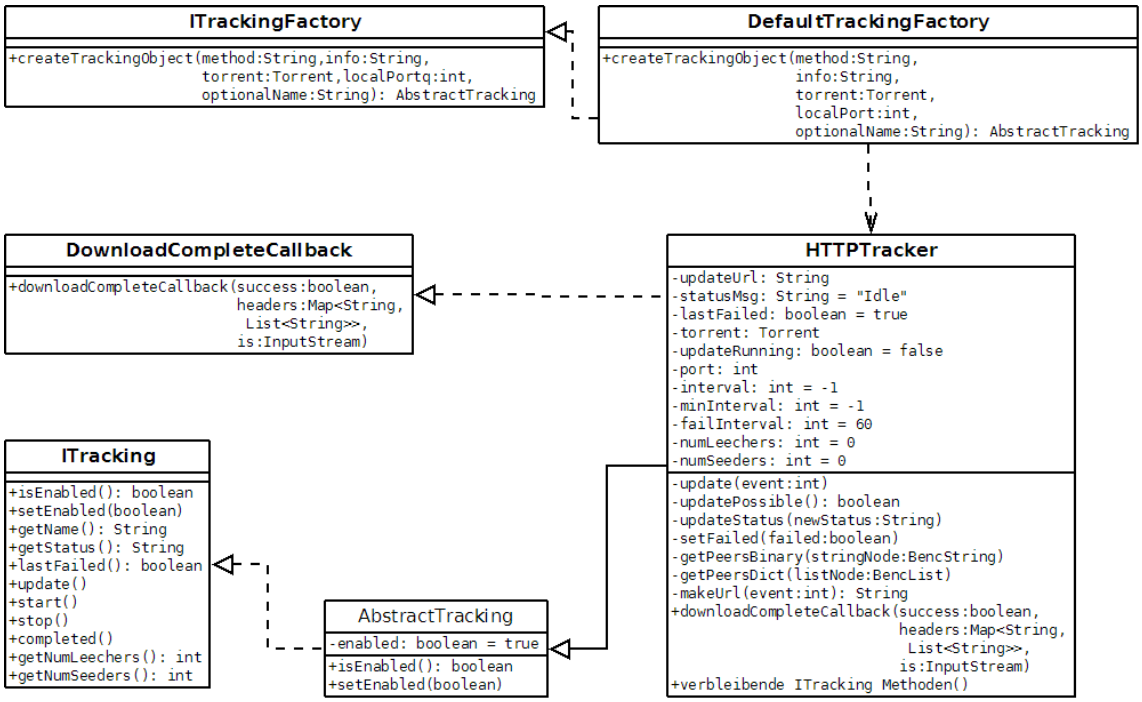

Abbildung 2.4: Klassendiagramm des HTTP-Trackings in TUD Torrent [TUT09, Seite 27]

3 Verwandte Systeme

Es gibt bereits einige Ansätze für das dezentrale Finden von Peers und Torrents. Sie alle liegen bisher nicht in einer finalen Version vor bzw. befinden sich noch im Entwicklungsstadium. Einige werden aber bereits im praktischen Einsatz und mit teils weiter Verbreitung verwendet.

3.1 Distributed Hash Table / DHT

Seit Januar 2008 existiert ein Draft, das den trackerlosen Austausch von Peer-Informationen spezifiziert. [DHT08] Dieser Draft beschreibt, wie mit Hilfe von verteilten Hash Tabellen (DHT) Informationen über Peers gespeichert werden sollen. Jeder Peer fungiert dabei als ein Tracker. Der Austausch zwischen den Knoten erfolgt über das UDP-Protokoll. Das beschriebene DHT-Protokoll basiert auf Kademlia [KAD02].

Jeder Knoten im DHT-Netzwerk besitzt eine eindeutige zufallsgenerierte ID von 160 Bit Länge und entspricht damit der Länge der Infohash-Werte über die Torrents eindeutig identifiziert werden. Es kommt eine Distanzmetrik zum Einsatz anhand der bestimmt wird, wie weit die eigene ID von der eines anderen Knotens entfernt ist. Als Metrik wird XOR verwendet. Das Ergebnis der XOR-Operation auf zwei IDs wird als vorzeichenloser Integer interpretiert. Je kleiner der Wert ist, desto geringer ist die Distanz zwischen den Knoten. Wird nach Peers für ein Torrent gesucht, dann wird eine Anfrage an die bekannten Knoten gesendet, die zum Infohash die geringste Distanz besitzen. Kennt ein kontaktierter Knoten Peers des Torrents, sendet er die Informationen über diese an den Absender. Andernfalls wird mit einer Liste von DHT Knoten geantwortet, deren Distanz am nächsten zum Infohash ist. So wird iterativ weiter nach Knoten gesucht, die möglichst nahe am Infohash liegen. Kann kein näherer Knoten mehr gefunden werden, dann werden Kontaktinformationen über den eigenen Peer an die Knoten gesendet, die dem Infohash nun am nächsten sind.

3.2 Magnet Links

Mit dem Magnet URI-Schema [MAG02] können Ressourcen unabhängig von ihrem Ablageort referenziert werden. Typischerweise werden damit Dateien referenziert, die verschiedene Peer-to-Peer Programme herunterladen können. Sie werden in der Regel anhand eines Hashes identifiziert. Ein Magnetlink kann dabei mehrere Hashes und auch explizite Downloadlokationen angeben (z. B. URLs). Für BitTorrent werden Magnet Links genutzt, um Torrent-Dateien zu verlinken, ohne dass diese auf einer Internetseite heruntergeladen werden müssen. Dazu wird im Magnet Link der Infohash angegeben. Ein BitTorrent Magnet Link kann beispielsweise wie folgt lauten:
`magnet:?xt=urn:btih:1234567890abcdef1234567890abcdef12345678&dn=Torrent+Name`

Beim Öffnen des Magnet Links in einem unterstützenden BitTorrent-Client, wird das Torrent zunächst gesucht. Dazu wird im DHT-Netzwerk nach Peers gesucht, wie im vorherigen Abschnitt beschrieben. Wird mindestens ein Peer gefunden, wird die Torrent-Datei von diesem Peer heruntergeladen, sofern er die „Extension for Peers to Send Metadata Files“ [SMF08] unterstützt. Sobald die Metadaten vollständig empfangen und mit dem Infohash verifiziert wurden, beginnt das Herunterladen der vom Torrent beschriebenen Daten.

3.3 Tribler

Tribler [TRB07] ist ein BitTorrent-Client der derzeit gemeinsam von der Delft University of Technology und der Vrije Universiteit Amsterdam entwickelt wird. Er funktioniert vollkommen dezentral und beinhaltet soziale Komponenten. Es können Torrents nach Kategorien und Stichworten gesucht werden. Freundschaften können geschlossen werden, um kooperative Downloads durchzuführen, wobei ein Helfer einen Teil seiner Upload-Bandbreite bereitstellt, um den Freund beim Download eines Torrents zu unterstützen. Außerdem schlägt das Programm weitere Torrents vor, die anhand von Vergleichen der eigenen Downloadhistorie mit den Historien anderer Benutzer ermittelt werden und unterstützt Live-Streaming von Videoinhalten über das BitTorrent-Protokoll.

3.4 Cubit

Mit Cubit [CUB08] lassen sich Daten anhand von Stichwörtern finden. Es arbeitet ähnlich dem DHT-Netzwerk, das bei der Suche von Peers benutzt wird. Anstelle von Hashes werden allerdings Stichwörter verwendet. Bei der Integrierung ins Netzwerk wählt der Knoten ein zufälliges Stichwort aus, das für die Dauer seines Aufenthalts im Netzwerk seine Knoten ID ist. Bei der Suche nach einem Schlüsselwort werden Daten gesucht, deren Schlüsselworte möglichst nahe am Suchwort liegen. Als Distanzmetrik wird hier die Anzahl der Veränderungen genutzt, die nötig ist, um ein Schlüsselwort in ein anderes zu transformieren. Eine Suchanfrage wird an eine definierte Anzahl bekannter Knoten gesendet, die die niedrigste Distanz zum Schlüsselwort besitzen, um weitere Knoten zu finden, die eine noch niedrigere Distanz haben. Iterativ werden die zurückgesendeten Knoten nach noch näheren Knoten befragt. Können keine näheren Knoten mehr ermittelt werden, werden die n nächsten Knoten ausgewählt und nach Objekten gefragt, die sich möglichst nahe am Schlüsselwort befinden.

4 Konzept

4.1 Dezentrale Peer-Suche mit BubbleStorm

Um Peers dezentral über das BubbleStorm-Netzwerk finden zu können, muss zu diesem natürlich zunächst eine Verbindung hergestellt werden. Um dies tun zu können, muss mindestens ein aktiver Teilnehmer des Netzwerkes bekannt sein, ein so genannter Bootstrap-Knoten. Da sich die Bootstrapping API von BubbleStorm in Zukunft komplett ändern wird, werden einige Adressen für den Beitritt ins Netzwerk vorläufig aus einer Datei geladen.

Für den Datenaustausch erfordert BubbleStorm, dass so genannte Bubble-Typen angelegt werden. Es wird zwischen Data Bubbles unterschieden, die Nutzdaten enthalten, welche im Netzwerk gespeichert werden sollen, und Query Bubbles, die Suchanfragen darstellen. Üblicherweise existiert mindestens ein Data Bubble und ein Query Bubble. Beim Tracking von BitTorrent ist es jedoch nicht sinnvoll, dass zwischen der Datenspeicherung und Suche nach anderen Peers unterschieden wird, denn wenn ein Peer weitere Peers sucht, möchte er selbst auch gefunden werden können. Jede neue Suchanfrage signalisiert außerdem, dass der eigene Peer noch aktiv ist. Daher ist es naheliegend, Data Bubble und Query Bubble zusammenzufassen. Für die Suche nach neuen Peers ist es ausreichend, als Suchparameter den Infohash des Torrents im Bubble anzugeben, denn der Infohash ist eine SHA-1 Prüfsumme, die über einen bestimmten Teil der Torrent-Datei - dem Info-Schlüssel - gebildet wird und das Torrent eindeutig identifiziert. Zur Bekanntgabe der eigenen Verbindungsinformationen werden zusätzlich folgende Informationen in das Bubble aufgenommen:

- Die eigene IP und der vom BitTorrent-Client abgehörte Port für eingehende Verbindungen
- Der BubbleStorm-Port, damit der BubbleStorm Cache mit Kontaktdaten zu weiteren Knoten gefüllt werden kann
- Ein aktueller Zeitstempel, um veraltete Peer-Informationen löschen bzw. zwischen veralteten und neueren Daten unterscheiden zu können

In regelmäßigen Intervallen werden diese Daten zum Finden neuer Peers als Query in das BubbleStorm-Netzwerk gesendet. Erhält ein anderer Knoten diese Query, dann speichert er die Informationen in seiner lokalen Datenbank unter dem Schlüssel des Infohashes des Torrents, der

im Bubble enthalten ist. Außerdem prüft er, ob er zu dem Infohash weitere Peers gespeichert hat. Ist dies der Fall, dann wird eine Liste der lokal gespeicherten Peers zu diesem Infohash an den Absender der Query gesendet.

Da es derzeit noch nicht möglich ist, alle Replikate der Peer-Informationen zu löschen, wenn ein Peer ein Torrent stoppt oder aus sonstigen Gründen nicht mehr erreichbar ist, haben alle Datensätze der Bubbles nur eine Lebenszeit von 30 Minuten nach ihrem Versand. Danach werden sie automatisch gelöscht.

4.2 Dezentrale Torrent-Suche mit BubbleStorm

4.2.1 Vorüberlegungen

Für die Entwicklung eines Konzepts mussten zunächst einige Entscheidungen getroffen werden, die das weitere Vorgehen beeinflussen. Da BubbleStorm nicht definiert, wie komplex Queries zu sein haben, stellte sich zunächst die Frage, welche Daten überhaupt durchsuchbar und welche Arten von Suchanfragen möglich sein sollen. Da Benutzer von einem neuen Suchkonzept erwarten, dass es mindestens die gleichen Möglichkeiten bietet wie bereits etablierte Systeme, ist es sinnvoll, deren Suchoptionen zu analysieren.

Abbildung 4.1: Suchfeld von The Pirate Bay

Auf vielen Internetseiten, die einen durchsuchbaren Index von Torrent-Dateien bereitstellen, kann mit Stichworten gesucht werden. Auf größeren Indexseiten wie The Pirate Bay wird in den Namen der Torrents und optional auch in benutzerdefinierten Tags nach den eingegebenen Stichwörtern gesucht. Dabei werden oft Suchmodifikatoren wie + und - zum Inkludieren bzw. Exkludieren von Suchwörtern in der Suche unterstützt. Typischerweise lässt sich auch die Kategorie einschränken, in der gesucht werden soll. Kleinere Seiten, die in der Regel weniger Last bewältigen müssen, durchsuchen auch Beschreibungen, die von einem Benutzer zusammen mit dem jeweiligen Torrent eingestellt wurden.

Zusätzlich zu den vier genannten Kriterien (Name des Torrents, Tags, Beschreibung, Kategorie) kann es in manchen Fällen auch wünschenswert sein, nach Torrents zu suchen, die

bestimmte Dateinamen enthalten oder innerhalb eines bestimmten Zeitraums eingestellt wurden. Manchmal kennt ein Benutzer die exakte Schreibweise eines Suchworts nicht oder begeht Rechtschreibfehler bei der Eingabe. Eine in der Praxis hilfreiche, aber von kaum einer Torrent-Suchmaschine unterstützte Funktion ist daher Fuzzy Search, also die ungenaue Suche, die auch ähnliche Wörter als Treffer liefert.

An dieser Stelle war es nötig zu entscheiden, welche Bibliothek für Indizierung und Suche genutzt werden soll, da sie im Wesentlichen bestimmt, welche Suchmöglichkeiten effizient umgesetzt werden können. In Frage kamen dabei das relationale Datenbanksystem SQLite [SQL10] und die Volltextsuchmaschine Apache Lucene [LUC10]. Beide Bibliotheken sind nur wenige hundert Kilobytes groß, unterstützen Volltextsuche und komplexe Suchanfragen und speichern ihre Daten in wenigen Dateien auf dem Dateisystem des Benutzers. SQLite kennt - RDBMS-typisch - verschiedene Datentypen, was insbesondere Operationen auf Zahlenwerte sehr effizient macht. Für die Indizierung mit Lucene müssen alle Zahlen in eine String-Repräsentation umgewandelt werden, dennoch werden Range-Queries auf Zahlen unterstützt, was für die Suche innerhalb eines Zeitintervalls notwendig ist. Jedoch hat SQLite drei entscheidende Nachteile: Es wird keine Fuzzy Search unterstützt, ein Ranking der Suchergebnisse zum Platzieren besserer Ergebnisse am Anfang der Ergebnisliste fehlt und für die Tokenisierung, also das Generieren von effizient zu indizierenden Wörtern aus einem Text heraus, stehen nur 3 vordefinierte Tokenizer zur Verfügung, von denen keiner für die Tokenisierung von Torrent-Namen geeignet ist, da sie nicht konfigurierbar und nur für den Einsatz auf Prosatext sinnvoll sind. Es können zwar eigene Tokenizer implementiert werden, diese müssen aber in C geschrieben und entsprechend kompiliert werden. Lucene ist ausschließlich in Java geschrieben, auf Volltextsuche spezialisiert, unterstützt Fuzzy Search, eigene Tokenizer in Java und Result Scoring. Aus diesen Gründen ist die Wahl auf Lucene als verwendete Suchbibliothek gefallen.

4.2.2 Veröffentlichung von Torrents

Um ein Torrent im BubbleStorm-Netzwerk zu veröffentlichen, muss der Benutzer über eine grafische Oberfläche einige Informationen eingeben:

- Der Pfad zum zu veröffentlichenden Torrent
- Optional ein Name für das Torrent. Wird keiner eingegeben, wird der im Torrent gespeicherte Name verwendet.
- Eine optionale Beschreibung über den Inhalt des Torrents als Plaintext.
- Optionale Tags zum leichteren Auffinden des Torrents

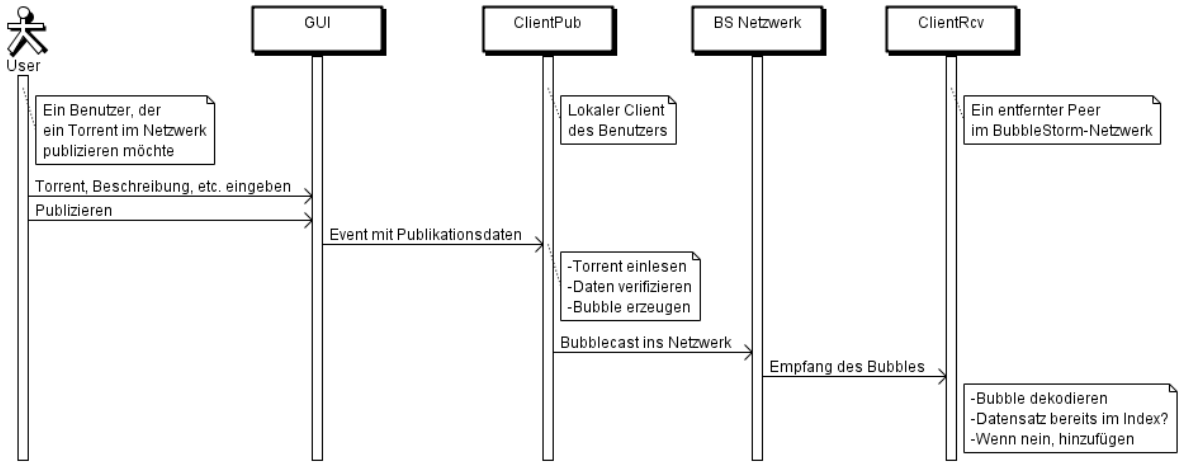

Abbildung 4.2: Veröffentlichung von Torrents im BubbleStorm-Netzwerk

- Eine passende Kategorie (kann Oberkategorie wie „Musik” oder Unterkategorie wie „Jazz” sein)
- Auswahl, ob die in der Torrent-Datei eingetragenen Tracker gespeichert werden oder nur dezentrales Tracking möglich sein soll

Wenn der Benutzer versucht ein als privat markiertes Torrent einzustellen, muss der Versuch mit einer Fehlermeldung scheitern, da dann laut Spezifikation ausschließlich die in der Torrent-Datei eingetragenen Tracker für die Peer-Suche verwendet werden dürfen.

Aus der Torrent-Datei werden der Infohash, die Dateinamen (inkl. Pfade), Dateigrößen und (sofern nicht vom Benutzer angegeben) der Name ausgelesen. Zusammen mit aktuellem Datum und Uhrzeit, einer eindeutigen ID und den übrigen Angaben des Benutzers wird aus diesen Informationen ein Bubble gebildet und per Bubblecast über BubbleStorm verbreitet. Empfangende Knoten prüfen zunächst, ob im Lucene-Index ein Dokument mit der gegebenen ID bereits vorhanden ist. Falls nicht, werden die Daten für die Indizierung durch Lucene aufbereitet und als neues Dokument dem lokalen Index hinzugefügt.

Zusammenfassend können also folgende Daten indiziert werden:

- Eindeutige ID
- Infohash des Torrents
- Name des Torrent
- Eine benutzerdefinierte Beschreibung
- Vom Benutzer festgelegte Tags
- Kategorie
- Tracker URLs

- Größe des gesamten Torrents
- Dateien und -größen
- Zeitpunkt der Veröffentlichung

Die Größe einer Torrent-Datei kann mehrere hundert Kilobytes oder gar einige Megabytes betragen. Würde der Inhalt der kompletten Torrent-Datei im Netzwerk abgelegt werden, dann würde in einem größeren Netzwerk mit einigen 10.000 Clients pro eingestelltem Torrent eine erhebliche Menge an Traffic entstehen, da die Verteilung eines Bubbles in BubbleStorm mit $O(\sqrt{n})$ skaliert, um die nötige Redundanz herzustellen. Darin inbegriffen sind noch keine zusätzlichen Informationen über das Torrent (Beschreibung, Tags, etc.), mit denen ein Torrent genauer beschrieben werden kann. Die für diese Art der Ablage notwendige Übertragungskapazität würde nicht mehr für den Austausch der Dateidaten von Torrents zur Verfügung stehen und damit die Effizienz des BitTorrent-Protokolls beeinträchtigen. Daher werden nur diejenigen Informationen im Netzwerk abgelegt, die für die Suche und das eindeutige Identifizieren von Torrents relevant sind (siehe obige Liste). Dadurch schrumpft die abzulegende Datenmenge auf ein- bis zweistellige Kilobyte-Werte.

4.2.3 Suche nach Torrents

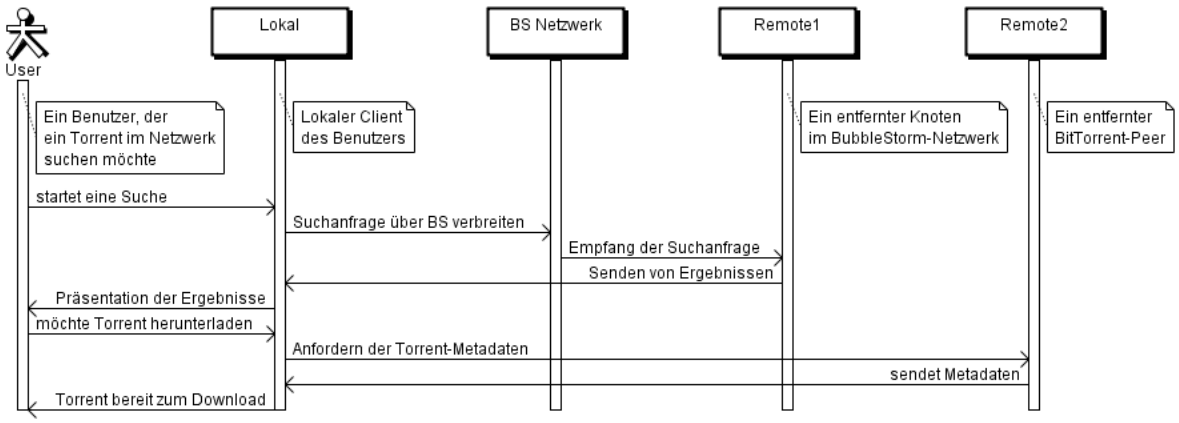

Abbildung 4.3: Suche nach Torrents im BubbleStorm-Netzwerk

Für das Durchsuchen des BubbleStorm-Netzwerkes muss natürlich zunächst eine Suchanfrage durch den Benutzer formuliert werden. Apache Lucene verwendet eine eigene Query-Sprache, die denen von Suchmaschinen wie Google nicht unähnlich ist. Jedoch ist die Sprache deutlich komplexer, da das zu durchsuchende Feld für jeden Suchterm definiert werden muss, jeder Suchterm eine Reihe von Modifikatoren besitzen kann, zwei Arten von booleschen Operatoren existieren und Suchausdrücke beliebig geklammert werden können.

Ausgehend von einem fiktiven Webindex, der Internetseiten mit Titel, Inhalt und Abrufdatum indiziert, könnte eine Query beispielsweise folgendermaßen aussehen:
`(+titel:“Apache Lucene” -text:.NET text:Java) AND date:[20090101 TO 20100101[`

Diese Query würde alle Seiten finden, die den Term *Apache Lucene* im Titel enthalten und in dessen Text nicht das Wort *.NET* vorkommt. Wenn das Wort *Java* im Text vorkommt, dann wird die entsprechende Seite in der Ergebnisliste weiter vorne platziert, als Seiten in denen das Wort nicht vorkommt. Außerdem müssen alle Treffer im Jahr 2009 indiziert worden sein. Die eckigen Klammern zeigen an, dass die Bereichssuche inklusive des 01.01.2009, aber exklusive des 01.01.2010 erfolgen soll.

Diese Syntax ist offensichtlich zu komplex, um das Ziel möglichst umfangreicher Suchmöglichkeiten mit einer vielen Benutzern bereits bekannten Syntax zu erreichen. Es ist sinnvoller, für die einzelnen Suchfelder ein möglichst einfaches Format zu verwenden. Dieses sieht wie folgt aus:

- **Torrent-Name**: Er ist in den meisten Fällen das wahrscheinlich wichtigste Suchkriterium. Es gibt zwei Typen von Suchtermen:
  - Wörter in Form von alphanumerischen Zeichenfolgen
  - Phrasen: eine zusammenhängende Folge von Wörtern, die von Anführungszeichen umschlossen werden (z. B. *“Apache Lucene*")

  Wörter werden per Fuzzy Search gesucht. Zu Suchwörtern ähnliche Wörter liefern also ebenfalls Treffer. Je kleiner die Distanz zwischen Suchwort und Wort im Index ist, desto besser ist die Platzierung im Ergebnis. Phrasen müssen immer exakte Treffer ergeben. Jedem Suchterm kann einer von 2 Modifikatoren vorangestellt werden:
  - Ein + vor Termen forciert, dass der Term in allen Ergebnissen vorhanden sein muss (z. B. +*Lucene* oder +*“Apache Lucene*")
  - Ein - vor Termen schließt alle Suchergebnisse aus, die den Term enthalten (z. B. -*.NET* oder -*“Apache Webserver*")

  Bei jeder Suche muss mindestens ein Term auf jeden Torrent-Namen im Ergebnis zutreffen, sofern das Suchfeld Name mindestens einen Term enthält. Je mehr Terme jedoch zutreffen, desto besser ist die Platzierung in der Liste der Suchergebnisse.

- **Beschreibung**: Für das Durchsuchen der Beschreibung gelten dieselben Regeln wie für Torrent-Namen.

- **Tags**: Tags sind durch Leerzeichen getrennte Wörter. Werden mehrere Tags angegeben, dann werden Torrents im Ergebnis besser platziert, je mehr Tags zwischen Suche und Indexeintrag übereinstimmen.

- **Dateinamen**: Es gelten die selben Regeln wie bei Tags. Eine Übereinstimmung bedeutet hier jedoch, dass ein Pfadsegment (also ein Dateiname oder Ordner auf dem Pfad dorthin) zwischen Suche und Index übereinstimmt. (z. B. Suchwort *Apache* ergibt eine Übereinstimmung mit *src/apache/lucene/Index.java*)

- **Zeitpunkt der Veröffentlichung**: Zum Einschränken des Veröffentlichungszeitraums werden zwei Zeitpunkte benötigt von denen mindestens einer in der Vergangenheit liegen muss. Beide Zeitpunkte werden bei der Suche inklusiv berücksichtigt.

Hat der Benutzer seine Suchanfrage formuliert, wird sie vom lokalen Knoten per Bubblecast im BubbleStorm-Netzwerk verbreitet. Erreicht die Suchanfrage einen Knoten im Netzwerk, führt er die Suche über den mit Lucene gespeicherten Index aus. Ist die Suche erfolgreich, d.h. wurde mindestens ein Ergebnis erzielt, dann werden alle gefundenen Ergebnisse an den Absender der Suchanfrage gesendet.

Die Ergebnisse werden dem Benutzer in Form einer Liste angezeigt. Zu jedem Eintrag lassen sich die Details anzeigen, die mit den Suchergebnissen übermittelt wurden und die Auswahl eines (oder mehreren) herunterzuladenden Torrents erleichtern soll.

Wenn sich der Benutzer dafür entscheidet, ein Torrent herunterzuladen, dann müssen nun die in der Torrent-Datei enthaltenen Metadaten bezogen werden. Da der Infohash durch die Suche bereits bekannt ist, kann eine Suche nach Peers für das Torrent gestartet werden, obwohl der Inhalt der Torrent-Datei noch fehlt. Die Suche kann sowohl über HTTP-Tracker als auch über die BubbleStorm Peer-Suche geschehen. Jeder Peer im Schwarm eines Torrents kennt mindestens den Info-Key eines Torrents, der den wichtigsten Schlüssel in einer Torrent-Datei bildet. Dieser enthält Informationen über Dateinamen/-pfade und Prüfsummen zur Verifikation der Daten. Diese Metadaten können über eine Erweiterung des BitTorrent-Protokolls von einem oder mehreren Peers heruntergeladen werden. Sobald die Metadaten komplett sind, kann der lokale BitTorrent-Client dem Schwarm beitreten und die Daten des Torrents herunterladen.

5 Implementierung

5.1 BubbleStorm

5.1.1 Integration ins Netzwerk

Für die Kommunikation über BubbleStorm wird eine Singleton-Klasse `tud.torrent.bubblestorm.BSInstance` verwendet. Beim ersten Aufruf wird die plattformabhängige BubbleStorm-Bibliothek geladen und über JNI eingebunden. Um eine Verbindung zum Netzwerk herzustellen, müssen folgende Aktionen ausgeführt werden:

- Alle für diese BubbleStorm-Sitzung benötigten Bubble-Paare, bestehend aus Data Bubble und Query Bubble, sowie Handler für eintreffende Bubbles werden festgelegt.
- Als Listen-Port für eingehende Pakete wird der Port des BitTorrent-Clients verwendet, da BubbleStorm ausschließlich UDP und BitTorrent ausschließlich TCP als Übertragungsprotokoll verwendet.
- Der Host Cache wird initial mit Peers aus einer Datei gefüllt (siehe auch Abschnitt 5.1.2). Ist der Host Cache leer, bildet der eigene Knoten einen Ring mit sich selbst und fungiert damit als Bootstrap.
- Die festgelegten Bubble Typen und ein Query-Objekt werden erzeugt und zusammen mit den Handlern bei der BubbleStorm Bibliothek registriert.
- Um festzustellen, wann die Integration ins Netzwerk abgeschlossen ist, wird ein Objekt mit einer Callback-Methode erzeugt. Nach dem erfolgreichen Beitritt wird die Singleton-Instanz für das Senden von Bubbles/Queries aktiviert. Vorher ist kein Senden möglich.
- Der Beitritt ins Netzwerk wird gestartet.

Außerdem registriert sich das Singleton beim EventManager für den Empfang von Events im Kanal `CHANNEL_UI_POLLING_REQUEST`, um das Event `EVENT_POLLING_BUBBLESTORM_STATUS` beantworten zu können. Ist die Instanz mit dem BubbleStorm-Netzwerk verbunden, dann wird ein Event `EVENT_POLLING_BUBBLESTORM_CONNECTED` über den Kanal `CHANNEL_UI_POLLING_UPDATE` gesendet, andernfalls ein Event `EVENT_POLLING_BUBBLESTORM_NOT_CONNECTED`.

5.1.2 Host Cache

Um sich mit anderen Knoten zu verbinden, müssen IP und Port mindestens eines weiteren aktiven Knotens bekannt sein. Bekannte Knoten werden in einem Host Cache gespeichert. Beim Initialisieren des BubbleStorm-Clients werden diese aus der Textdatei *hostcache.dat* im aktuellen Verzeichnis geladen, damit der Kontakt zum Netzwerk hergestellt werden kann. Die Datei enthält pro Zeile einen Eintrag im Format *IP:Port*. Nach dem Integrieren ins Netzwerk, wird der Host Cache mit weiteren Knoten gefüllt. Dies geschieht, indem die Informationen über IP und Port eingetragen werden, die mit Peer-Queries und Antworten auf eigene Peer-Queries eintreffen. Wenn ein Knoten aus dem Host Cache angefordert wird, wird ein zufälliger Knoten aus der Liste ausgewählt und übergeben.

5.1.3 Verlassen des Netzwerks

Sobald die Verbindung mit dem BubbleStorm-Netzwerk hergestellt wurde, registriert sich die BSInstance zusätzlich auf dem Kanal `CHANNEL_UI_COMMAND_REQUEST` und wartet auf das Event `EVENT_COMMAND_SHUTDOWN`. Dieses Event wird gesendet, wenn der BitTorrent-Client durch den Benutzer beendet wird. Das Singleton reagiert auf das Event, indem der lokale BubbleStorm-Knoten durch das Abmelden von allen benachbarten Knoten das Netzwerk verlässt. Würde diese Abmeldung nicht passieren, dann wären alle benachbarten Knoten temporär degradiert, da einer ihrer Nachbarn nicht mehr reagieren würde. Nach Verlassen des Netzwerks wird die Library von BubbleStorm inklusive zugehöriger Ressourcen freigegeben.

5.2 Peer-Suche mit BubbleStorm

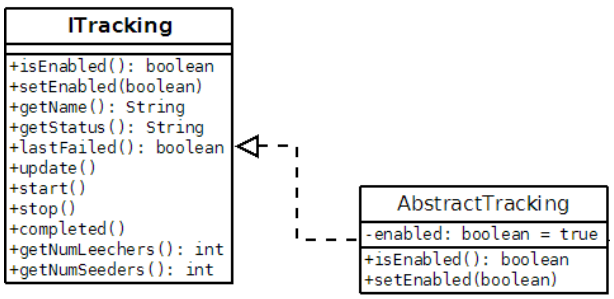

Abbildung 5.1: Tracking Interface

Für die Implementierung der Peer-Suche wird das Interface ITracking implementiert (siehe Abb. 5.1), das die API zwischen BitTorrent-Client und der implementierten Tracking-Methode darstellt (in diesem Fall BubbleStorm). Aufgrund der dezentralen Eigenschaften von BubbleStorm können nicht alle Methoden sinnvolle Funktionen erhalten. Die Methoden sind daher wie folgt implementiert:

- **getName()**: Gibt immer den String "BubbleStorm P2P Tracking" zurück.

- **getStatus()**: Liefert eine Meldung zurück, die angibt, ob der eigene Peers ins BubbleStorm-Netzwerk integriert ist oder nicht.

- **lastFailed()**: Wenn das letzte Update über BubbleStorm fehlgeschlagen ist, ist dieser Wert auf *true* gesetzt, sonst *false*. Ein Update gilt immer dann als fehlgeschlagen, wenn der eigene Knoten nicht ins BubbleStorm-Netz integriert ist, die eigene IP-Adresse nicht ermittelt werden kann oder ein Ein-/Ausgabefehler auftritt.

- **update()**: Wenn das Private-Feld im Info-Key des Torrents auf 1 gesetzt ist, dann dürfen nur die im Torrent eingetragenen Tracker verwendet werden. In dem Fall kehrt diese Methode sofort zurück. Andernfalls wird zunächst geprüft, wie viel Zeit seit dem letzten Aufruf vergangen ist. Diese Methode wird vom BitTorrent-Client alle 30 Sekunden aufgerufen. Es wäre aber nicht sinnvoll in diesem Intervall Anfragen im BubbleStorm-Netz zu verschicken, da dies zu sehr hoher Last führen würde und vergleichsweise selten Peers ein Torrent verlassen oder zu ihm hinzustoßen. Daher müssen seit dem letzten erfolgreichen Update mindestens 10 Minuten vergangen sein. Ist dies der Fall und der eigene Knoten ist im BubbleStorm-Netzwerk integriert, dann wird eine Query mit Timestamp, Kontaktinformationen zum eigenen Peer und dem Infohash des Torrents gesendet. Antworten werden asynchron über ein Callback erhalten.

- **start()**: Beim Start eines Torrents wird der Zeitpunkt des letzten Updates zurückgesetzt und danach update() aufgerufen.

- **stop()**: Es ist nicht möglich, Informationen über den eigenen Peer zu löschen, da nicht bekannt ist, bei welchen BubbleStorm-Knoten die Daten gespeichert sind. Diese Methode hat daher keine Funktion. Wenn ein Peer ein Torrent stoppt, wird er durch den BubbleTimeout nach spätestens 30 Minuten automatisch gelöscht.

- **completed()**: Die Information, ob ein Peer ein Torrent fertiggestellt hat, wird nicht im BubbleStorm-Netzwerk abgelegt. Die Methode hat keine Funktion.

- **getNumLeechers()**: Aufgrund der Dezentralität des Trackings ist es nicht möglich, zuverlässig zu ermitteln, wie viele Peers ein Torrent gerade herunterladen. Die Anzahl der erhaltenen Peers zurückzugeben ist unangemessen, da diese Zahl bereits von der PeerManager-Klasse ermittelt wird. Daher wird immer -1 zurückgegeben.

- **getNumSeeders()**: Äquivalent zu getNumLeechers() wird auch hier -1 zurückgegeben.

5.2.1 Queries

Da Query und Data Bubble identisch sind, ist nur ein Bubble-Typ nötig. Dieser wird als Query immer dann gesendet, wenn neue Peers angefordert werden. Da die Query auch Informationen zum eigenen BitTorrent-Peer und dem BubbleStorm-Knoten enthält, können diese vom Data Store der empfangenen Knoten gespeichert werden. Der Payload des Bubbles wird mittels Bencoding [Spec10] kodiert, das auch in Torrent-Dateien und der Kommunikation mit HTTP-Trackern verwendet wird, da Parser und Encoder dafür bereits vorhanden sind. Bencoding kennt die Datentypen Byte String, Integer, List und Dictionary. Als Wurzelknoten wird ein Dictionary verwendet, in dem Schlüssel/Wert-Paare gespeichert werden können. In diesem Dictionary sind die Schlüssel *timestamp*, *ip*, *port*, *bsport* und *hash* enthalten. Dem Schlüssel *hash* wird der Infohash des Torrents als Byte String zugewiesen. Alle anderen Schlüssel verweisen auf Integer. Im Bencoding gibt es für Integer keine Größenbeschränkung. Daher muss beim Dekodieren darauf geachtet werden, dass der verwendete Datentyp ausreichend große Werte erlaubt. *timestamp* enthält die vergangenen Millisekunden seit dem 1. Januar 1970 (UTC), *ip* die eigene externe IP-Adresse in Network Byte Order, *port* den Listening-Port für Verbindungen zum eigenen BitTorrent-Peer und *bsport* den Listening-Port des BubbleStorm-Knotens.

5.2.2 Data Store

Der Data Store wird als Handler eingehender Queries registriert und speichert eingehende Peer-Daten lokal, um Suchanfragen beantworten zu können. Dazu wird eine HashMap verwendet, deren Schlüssel der Infohash des jeweiligen Torrents ist. Zu jedem Infohash enthält die HashMap eine PriorityQueue mit Peer-Daten, da eine Sortierung nach Alter der Peer-Informationen nötig ist, um tote Peers zu entfernen. Die Operationen add() und remove() der PriorityQueue werden in O(log(n)) und peek() in O(1) ausgeführt, was sie ausreichend schnell für diese Aufgabe macht. Wird eine Suchanfrage empfangen, wird zunächst die PriorityQueue in der HashMap gesucht, die zum Infohash gehört, welcher in der Suchanfrage enthalten ist. Existiert eine solche

PriorityQueue, werden alle Peer-Daten des angefragten Torrents gelöscht, die älter als 30 Minuten sind, da davon ausgegangen werden muss, dass ein Peer nicht mehr verfügbar ist, wenn er innerhalb dieser Zeit keine Suchanfrage gesendet hat. Dazu wird solange das vorderste Element der PriorityQueue entfernt, wie die peek() Operation veraltete Datensätze vom Kopf der Queue zurückgibt. Ist danach noch mindestens ein Peer vorhanden, dann werden die Daten aller verbliebenen Peers an den Absender der Anfrage gesendet. Das Datenformat ist dabei mit dem der Query Bubbles identisch.

5.3 Dezentrale Torrent-Suche mit BubbleStorm

5.3.1 Lucene

Zum Verständnis der Implementierung müssen zunächst einige Eigenschaften von Lucene geklärt werden.

Lucene speichert Daten in Form von Dokumenten. Jedes Dokument besteht aus einem oder mehreren Feldern, wobei jedes Feld einen Namen trägt. Insoweit sind Dokumente also mit Tupeln aus relationalen Datenbanken vergleichbar. Im Gegensatz zu diesen besitzt ein Dokument allerdings keine per Schema festgelegte Anzahl von Attributen. Zwei Dokumente im selben Index können unterschiedlich viele Felder mit unterschiedlichen Namen enthalten. Die in den Feldern enthaltenen Daten müssen Texte (also Strings in Java) oder Binärdaten sein. Lucene kann allerdings nur Felder mit Textinhalt indizieren. Felder mit Binärdaten können also nur für die Speicherung zusätzlicher Informationen zu einem Dokument genutzt werden. Ein weiterer Unterschied zu relationalen Datenbanken besteht darin, dass in jedem Feld eines Dokuments eine beliebige Anzahl von Werten gespeichert werden kann. So lassen sich in einem einzigen Feld ganze Listen ablegen. Für Felder mit Textdaten gibt es drei Arten, auf die diese im Index gespeichert werden können:

- Felder können gespeichert und nicht indiziert werden. In diesem Fall kann das jeweilige Feld nach dem Speichern des Dokuments im Index zwar ausgelesen, aber nicht durchsucht werden.
- Felder können indiziert und nicht gespeichert werden. Nach dem Speichern des Dokuments im Index kann das Feld durchsucht werden. Der ursprüngliche Text wird aber nicht gespeichert und kann daher nicht ausgelesen werden.
- Felder können sowohl indiziert als auch gespeichert werden. Das Feld kann also durchsucht werden und der ursprüngliche Inhalt bleibt verfügbar.

Lucene ist zwar eine Bibliothek für die Volltextsuche, trotzdem ist es möglich Zahlenwerte zu speichern und nach ihnen zu suchen. Auch die Suche innerhalb eines Wertebereichs ist möglich. Da Lucene nur Texte durchsuchen kann, gibt es einen speziellen Feldtyp namens NumericField, der die Indizierung von `int`, `long`, `float` und `double` Werten optimiert. Diese werden intern als 7 Bit ASCII-Zeichen in einem Trie gespeichert. NumericFields werden immer indiziert. Soll der Wert auch gespeichert werden, geschieht dies entsprechend der toString() Methode des jeweiligen Zahlentyps.

Indizierte Felder können sowohl analysiert als auch nicht analysiert im Index abgelegt werden. Analysierte Felder werden durch einen Analyzer geleitet, der den jeweiligen Text in Terme (z. B. Einzelwörter von Sätzen) zerlegt und ggf. weitere Filter anwendet. Das Ergebnis des Analyzers wird dann im Index abgelegt.

Für die Suche innerhalb des Indexes gibt es eine abstrakte Schnittstelle für Queries. Queries können entweder über einen Parser in der in Abschnitt 4.2.3 gezeigten Query-Sprache formuliert oder mittels einer Reihe von Query-Klassen über die API von Lucene konstruiert werden. Für die Torrent-Suche wurde letztere Möglichkeit gewählt, da sie der vom Lucene Projekt vorgeschlagene Weg ist und der Parser für Eingaben in der Query-Sprache ebenfalls auf die Query API zurückgreift. Die in dieser Arbeit verwendeten Query-Klassen werden im Folgenden beschrieben:

- BooleanQuery: Mehrere Queries können hiermit zusammengefasst werden. Jede Unterquery erhält ein Attribut, das definiert, ob sie auf jedes Ergebnis zutreffen muss, zutreffen kann oder nicht zutreffen darf.
- TermQuery: Sucht nach exakten Entsprechungen eines einzelnen Terms in einem Dokumentenfeld. Terme müssen vor dem Übergeben an eine TermQuery nach den selben Regeln wie beim Hinzufügen der Dokumente tokenisiert worden sein.
- PhraseQuery: Sucht nach einer Folge von Termen in einem Dokumentenfeld. Die zu suchende Phrase muss ebenfalls zuvor in Terme aufgespalten worden sein.
- FuzzyQuery: Entspricht weitgehend der TermQuery, findet aber - basierend auf der Levenshtein-Distanz - auch Terme, die zu einem gewissen Grad vom Suchterm abweichen. (Standarddistanz: 0.75 $\times$ Termlänge)
- NumericRangeQuery: Sucht in einem Dokumentenfeld nach Zahlen innerhalb eines angegebenen Wertebereichs.

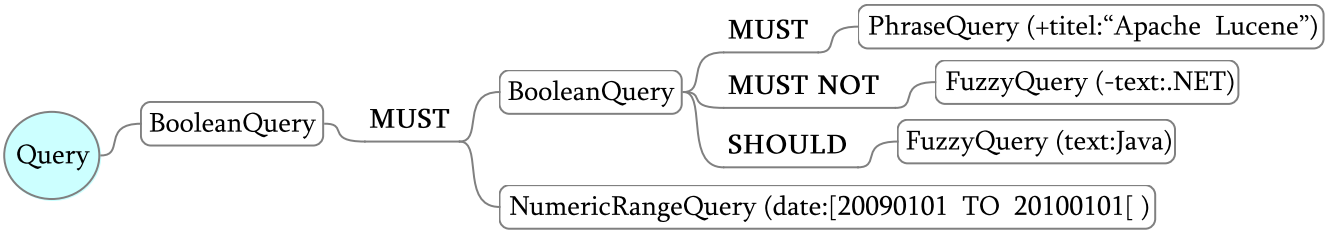

Abbildung 5.2: Beispiel für die Struktur einer Query

Es wurde bereits beispielhaft folgende Query erwähnt, die der Syntax des Parsers für Queries folgt:
`(+titel:“Apache Lucene” -text:.NET text:Java) AND date:[20090101 TO 20100101[`

Die Abbildung 5.2 stellt die Struktur dieser Query dar, wie sie für die Torrent-Suche in Form einer baumartig angeordneten Menge von Objekten an Lucene übergeben wird. Jeder in einer Blase enthaltene Knoten repräsentiert ein Objekt, das einen Teil der Query definiert. In Klammern hinter dem Namen des Typs der einzelnen Query-Objekte befindet sich jeweils der Teil der Query in Lucene-Syntax, der von dem Objekt repräsentiert wird. Die Schlüsselwörter MUST, MUST NOT und SHOULD geben an, in welcher Weise nachfolgende Objekte als Suchkriterien angewendet werden, ob das Kriterium also inkludierend, exkludierend oder als optional behandelt wird.

Zu jedem Suchergebnis wird ein Score in einem Vector Space Model berechnet. Im Wesentlich hängt die Höhe des Scores davon ab, ob und wie häufig Terme aus der Query im Dokument enthalten sind. Lucene erlaubt es auch, das Scoring durch sogenannte Boosts von Termen, Queries und Dokumenten zu beeinflussen. Diese Funktion wird für den vorliegenden Zweck jedoch nicht verwendet.

Lucene arbeitet transaktionsbasiert. Damit Änderungen des Indexes bei der Suche sichtbar werden, muss also ein Commit stattfinden. Wenn der Index vor dem Zeitpunkt des Commits bereits zum Lesen geöffnet war, dann bleiben die Änderungen bei der Suche ebenfalls unsichtbar bis der Index erneut geöffnet wird.

5.3.2 Bubbles

Für die Übermittlung neuer Torrents und Suchanfragen wird jeweils ein Bubble-Typ verwendet.

Das Data Bubble dient zum Publizieren eines Torrents per BubbleStorm. Jedes dieser Bubbles enthält ein TorrentItem, das alle in Abschnitt 4.2.2 beschriebenen Daten enthält. Für die Übertragung werden sie per Bencoding kodiert. Das TorrentItem wird gemäß der Spezifikation für das Bencoding in ein Directory gespeichert, das aus einer Liste mit den folgenden Schlüssel/Wert-Paaren besteht (die Bezeichnungen String, Integer und Liste sind als Datenstrukturen des Bencoding zu verstehen):

- `name`: Name des Torrents als UTF-8 String
- `added`: Zeit der Veröffentlichung in Sekunden seit der UNIX-Epoche als Integer
- `size`: Größe des gesamten Torrents als Integer
- `desc`: Beschreibung des Torrents als UTF-8 String
- `cat`: Nummer der Kategorie als Integer
- `infohash`: 20 Byte SHA-1 Hash des Info-Keys des Torrents in Binärform als String
- `tags`: Liste mit Tags als String; Schlüssel ist nur vorhanden, wenn Tags definiert sind
- `trackers`: Liste mit Trackern als String; Schlüssel ist nur vorhanden, wenn Tracker definiert sind
- `files`: Liste mit Dateinamen des Torrents inklusive relativem Pfad als String; Unix oder Windows Pfadtrennzeichen
- `filesizes`: Liste mit Dateigrößen als Integer; Liste hat dieselbe Länge wie `files` und enthält an jeder Position die Größe der Datei an derselben Position wie in `files`.

Das bencodete Directory wird mit dem gzip-Algorithmus komprimiert, um die zu sendende Datenmenge zu reduzieren.

Das Query Bubble wird für das Senden von Suchanfragen verwendet. Es enthält eine Search-Query, die ebenfalls bencodet wird und folgende Schlüssel/Wert-Paare beinhaltet:

- `uuid`: Eine UUID gemäß RFC 4122 als String zur Vermeidung doppelter Suchen auf einem Knoten
- `name`, `desc`, `tags` und `files`: Such-String für die Felder Torrent-Name, Beschreibung, Tags und Dateinamen als String
- `cat`: Nummer einer Kategorie
- `fromTime` und `toTime`: untere bzw. obere Zeitgrenze für Veröffentlichungszeitpunkt; wenn gesetzt muss `toTime` $\geq$ `fromTime` gelten

Das Feld `uuid` ist das einzige Pflichtfeld, alle anderen sind optional.

Für jedes durch eine Query gefundene Torrent wird eine Datenstruktur namens `SearchResult` an den Absender des Query Bubbles gesendet. Dieses bencodete Directory hat folgende drei Einträge:

- `uid`: Eindeutiger, 20 Bytes langer Identifier als binärer String, der verwendet wird, um doppelte Suchergebnisse erkennen zu können. Dieser Wert entspricht der ID des Data Bubbles, mit dem das Torrent veröffentlicht wurde. Sie wurde durch BubbleStorm basierend auf dem Hash der Nutzdaten des Data Bubbles generiert.
- `item`: Dieser Schlüssel enthält ein Directory mit einem TorrentItem, dessen Kodierung mit der des Data Bubbles identisch ist.
- `score`: Scorewert der Relevanz des Suchergebnisses in Bezug auf die Suchanfrage. Der Score entspricht einem `float`-Wert. Da Bencoding nur Integer unterstützt, wird der Wert mithilfe der Funktion Float.floatToRawIntBits() in einen `int`-Wert konvertiert.

5.3.3 Initialisierung

Die Hauptklasse befindet sich in `tud.torrent.search.SearchMain`. Um die verteilte Suche von Torrents über BubbleStorm verwenden zu können, muss der Konstruktor dieser Klasse aufgerufen werden, bevor die Verbindung zum Netzwerk hergestellt wird. Zur Initialisierung werden die Handler für eintreffende Data und Query Bubbles erzeugt, und der BubbleStorm Instanz als Konfiguration für die beiden benötigten Bubble-Typen übergeben.

Die Klasse initiiert neue Suchanfragen und Veröffentlichungen von Torrents, allerdings besitzt sie für diesen Zweck keine öffentlichen Methoden. Stattdessen wartet sie auf dem Kanal `CHANNEL_UI_COMMAND_REQUEST` auf die beiden Events `EVENT_COMMAND_NEW_BUBBLESTORM_SEARCH` und `EVENT_COMMAND_ANNOUNCE_BUBBLESTORM_TORRENTITEM`.

5.3.4 Veröffentlichung und Indizierung von Torrents

Wenn der Benutzer die nötigen Daten für die Veröffentlichung eines Torrents über die grafische Oberfläche eingegeben hat (siehe Abschnitt 4.2.2) und sie absendet, wird zunächst ein `TorrentItemInfo` Objekt erzeugt, das die eingegebenen Informationen zum Torrent und den Pfad zur Torrent-Datei aufnimmt. Dieses Objekt wird durch das Event `EVENT_COMMAND_ANNOUNCE_BUBBLESTORM_TORRENTITEM` an die Hauptklasse `SearchMain` dele-

giert. Es wird ein Objekt der Klasse `tud.torrent.search.Publisher` erzeugt, das die Veröffentlichung des Torrents in zwei Schritten durchführt, für die jeweils eine Methode existiert:

prepare(): Der Prepare-Schritt dient dazu, die Eingaben zu validieren und alle noch fehlenden Informationen aus der Torrent-Datei auszulesen. Die Methode gibt einen Fehlercode ungleich 0 zurück, wenn ungültige Eingaben vorliegen oder ein Fehler auftritt. Ungültige Eingaben liegen vor, wenn:

- der Torrent-Name nur aus Zeichen besteht, die Trennzeichen für die spätere Tokenisierung sind,
- zum spezifizierten Pfad keine Datei existiert,
- die Datei keine Torrent-Datei ist,
- das Bencoding in der Datei fehlerhaft ist oder
- das Torrent als privat markiert ist.

Das Auslesen der Torrent-Datei erfolgt über bereits vorhandene Funktionen des BitTorrent Clients. Hat der Benutzer keinen Namen für das Torrent eingegeben, dann wird der Name aus dem Info-Key der Torrent-Datei verwendet. Die Dateiliste, Dateigrößen, der Infohash und, falls vom Benutzer gewünscht, die eingetragenen Tracker-URLs werden ausgelesen und zusammen mit den übrigen Benutzereingaben (Beschreibung, Kategorie, Tags) in einem TorrentItem-Objekt angelegt, das serialisiert und versendet werden kann.

announce(): Liefert die Funktion prepare() keinen Fehler zurück, wird das erzeugte TorrentItem wie bereits beschrieben beencodet und als Data Bubble per Bubblecast verbreitet. Die ID des Bubbles wird von einem Hash der kodierten Daten abgeleitet.

War die Veröffentlichung im BubbleStorm-Netzwerk erfolgreich, dann wird ein Event `EVENT_COMMAND_REPLY_ANNOUNCE_BUBBLESTORM_TORRENTITEM_SENT` über den Kanal `CHANNEL_UI_COMMAND_REPLY` gesendet. Im Fehlerfall wird `EVENT_COMMAND_REPLY_ANNOUNCE_BUBBLESTORM_TORRENTITEM_FAILED` mit einem Fehlercode gesendet.

Wird ein Data Bubble empfangen und das enthaltene TorrentItem erfolgreich dekodiert, wird es an ein Objekt der Klasse `SearchEngine` weitergereicht, in der Indizierung und Suche implementiert sind. Zunächst wird geprüft, ob das TorrentItem bereits indiziert ist. Dazu wird die ID des Data Bubbles verwendet, mit dem das TorrentItem eingetroffen ist. Zunächst wird die ID in einer Liste der erst kürzlich hinzugefügten und noch nicht committeten Dokumente gesucht, da sich diese noch nicht über den Index finden lassen. Führt dies zu keinem Ergebnis wird der Index von Lucene nach der ID durchsucht. Ergibt eine der beiden Suchen einen Treffer, wird nicht

mit der Indizierung fortgefahren, da das Torrent bereits indiziert ist. Andernfalls wird ein neues Lucene Dokument erzeugt, in dem für jedes Datum im TorrentItem ein Feld angelegt wird. Alle Felder werden indiziert und gespeichert, ausgenommen das Feld für Tracker URLs, das ohne Indizierung gespeichert wird. Außerdem werden die Felder für Torrent-Name, Beschreibung und Dateinamen analysiert. Ein Tokenizer zerteilt dafür den Namen und die Beschreibung in einzelne Wörter, die durch nicht-alphanumerische Zeichen getrennt werden. Einige gängige englische Wörter wie „a" und „the" werden herausgefiltert, da sie nur einen geringen Wert bei der Suche haben und den Index aufblähen. Dateinamen werden lediglich nach den Pfadtrennern / bzw. \ in Wörter zerteilt.

Nach der Erstellung des Dokuments wird es einem IndexWriter zum Schreiben in den Index übergeben und die ID zur Liste uncommitteter IDs hinzugefügt. Falls seit dem letzten Commit mindestens 60 Sekunden vergangen sind oder seitdem 1.000 Dokumente zum Index hinzugefügt wurden, wird ein neuer Commit ausgeführt, wodurch die neuen Dokumente durchsuchbar werden. Alle 30 Minuten oder 10.000 hinzugefügte Dokumente wird der Index außerdem optimiert, um die Suchperformanz zu steigern. Diese Operation ist mit relativ hohen Rechenkosten verbunden und wird daher nicht öfter ausgeführt.

5.3.5 Suchen von Torrents

Hat der Benutzer die Felder für Torrent-Name, Beschreibung, Tags, Kategorie und Suchzeitraum entsprechend seiner gewünschten Suche ausgefüllt und die Suche gestartet, wird aus den Angaben ein `SearchQuery` Objekt erzeugt und als Event `EVENT_COMMAND_NEW_BUBBLESTORM_SEARCH` versendet, das wiederum durch das Objekt der Hauptklasse `SearchMain` empfangen wird. Ein Objekt der Klasse `Search` wird erzeugt, das die Suche repräsentiert. Es erhält eine ID, über die sich die Ergebnisse einer Suche zuordnen lassen, wenn mehrere Suchanfragen zur gleichen Zeit ablaufen. Das Search-Objekt erhält die SearchQuery, die auf Validität geprüft (mindestens ein Suchkriterium ausgefüllt?), serialisiert und als Query Bubble verschickt wird. Bei Erhalt eines Suchergebnisses wird die onResponse() Methode aufgerufen, die das Suchergebnis über das Event `EVENT_PUSH_SEARCH_RESULT` an die GUI weiterleitet. Soll die Suche gestoppt und kein weiteres Ergebnis mehr empfangen werden, muss dies über das Event `EVENT_COMMAND_STOP_BUBBLESTORM_SEARCH` geschehen, dem die ID übergeben wird, die die Suche zuvor erhalten hat.

Auf der Empfängerseite des Query Bubbles wird die SearchQuery nach der Dekodierung an die Klasse SearchEngine übergeben. Da die BubbleStorm-Bibliothek derzeit doppelt empfangene Bubbles weiterreicht, wird geprüft, ob eine SearchQuery mit derselben UUID innerhalb der

letzten 3 Minuten empfangen wurde. Ist das der Fall, wird die SearchQuery ignoriert. Andernfalls wird aus der SearchQuery eine Query-Struktur für die Suche mit Lucene erzeugt. Für die Suche wird eine BooleanQuery verwendet, bei der jede Unterquery zutreffen muss. Sie enthält folgende Unterqueries:

- BooleanQuery enthält weitere Unterqueries:
  - Eine PhraseQuery für jede in Anführungszeichen (") eingeschlossene Phrase. Ist ein + oder - einer Phrase vorangestellt muss die Phrase auf jedes Suchergebnis zutreffen oder darf es nicht.
  - Eine FuzzyQuery für jeden Term im Suchfeld Name und Beschreibung. + und - vor Termen bewirken dieselbe Semantik wie vor Phrasen.
  - Eine TermQuery für jeden Term im Suchfeld für Dateien. Jeder Term muss auf jedes Ergebnis zutreffen.
  - Eine TermQuery für jedes Tag, wobei auch jedes einen Treffer auf jedes Ergebnis ergeben muss.
- NumericRangeQuery für die Suche innerhalb eines Veröffentlichungsintervalls
- *Entweder* NumericRangeQuery, um auch alle Unterkategorien zu durchsuchen, wenn eine Oberkategorie angegeben ist (z. B. Musik)
- *oder* TermQuery, wenn eine Unterkategorie zum Durchsuchen angegeben wurde (z. B. Jazz)

Alle Unterqueries sind nur dann vorhanden, wenn das jeweilige Suchfeld gesetzt ist bzw. gültige Suchbegriffe enthält.

Nachdem die Query für Lucene erstellt wurde, wird der Index neu geöffnet, falls nötig. Dies geschieht dann, wenn der Index bisher nicht geöffnet war oder seit der letzten Nutzung ein Commit stattgefunden hat. Die Query wird an einen IndexSearcher von Lucene übergeben, der die Suche ausführt und eine nach einem Score sortierte Liste mit IDs von Dokumenten zurückgibt. Die Suche wird wegen der zu sendenden Datenmenge auf maximal 50 Ergebnisse beschränkt. Zu jedem Ergebnis wird das Dokument geladen und die Informationen wieder in ein TorrentItem transformiert. Dieses TorrentItem wird zusammen mit der gespeicherten ID des Torrents und dem Score des Suchergebnisses über BubbleStorm an den Absender der Suche gesendet.

5.3.6 Erlangung der Torrent Metadaten

Mit den im BubbleStorm-Netzwerk gespeicherten Daten ist es nicht direkt möglich ein Torrent herunterzuladen, da die Informationen, die den Großteil des Datenvolumens einer Torrent-Datei ausmachen, nicht im Index abgespeichert werden. Da aber jeder aktive Peer des Schwarms eines Torrents mindestens den kompletten Inhalt des Info-Keys der Torrent-Datei kennt, kann dieser von den aktiven Peers heruntergeladen werden. Es existieren bereits zwei Spezifikationen für Erweiterungen des BitTorrent-Protokolls als Entwürfe, die eben dies umsetzen:

Das BitTorrent Enhancement Proposal 10 (BEP 10) [EP08] beschreibt das „Extension Protocol“, das dem aktuellen BitTorrent-Protokoll einen neuen Nachrichtentyp hinzufügt, über den beliebige Erweiterungen zwischen Clients kommunizieren können - sofern beide Kommunikationspartner mindestens eine gleiche Erweiterung unterstützen. Ein Client, der diese Erweiterung unterstützt, setzt im regulären Handshake das Bit 20 (von rechts gezählt). Wenn die Clients an beiden Enden einer Verbindung das Extension Protocol unterstützen, tauschen sie sich nach dem regulären Handshake mit einer weiteren Handshake-Nachricht darüber aus, welche Erweiterungen sie unterstützen und welche ID der jeweils andere Client in der weiteren Kommunikation verwenden muss, um mit der entsprechenden Erweiterung zu kommunizieren.

BEP 9 [SMF08] spezifiziert die „Extension for Peers to Send Metadata Files“. Sie nutzt das in BEP 10 beschriebene Protokoll, um den Info-Key von Torrent-Dateien von anderen Peers herunterzuladen. Dies funktioniert so ähnlich wie das Herunterladen der Inhalte des Torrents. Die Metadaten werden in Blocke von je 16 KiB unterteilt, die heruntergeladen werden können. Spezifiziert sind lediglich drei Nachrichten:

- Request: Anfordern eines Blockes von Metadaten, wobei der 0-basierte Blockindex angegeben wird
- Data: Übermittlung eines maximal 16KiB großen Blocks von Metadaten. Jede Data-Nachricht enthält außerdem die Gesamtgröße der Metadaten des kompletten Info-Keys.
- Reject: Abweisung eines Requests

Da der Infohash bereits bekannt ist, kann die Integrität der Metadaten verifiziert werden, sobald alle Blöcke empfangen wurden.

Der bestehende BitTorrent-Client wurde um die beiden BEPs erweitert, wozu die Pipeline für die Protokollerweiterung leicht angepasst werden musste. Dies war nötig, um den neuen Handshake gemäß BEP 10 senden und empfangen zu können. Das Herunterladen der Metadaten wird

genauso behandelt, wie das Herunterladen der Inhalte. Es gibt lediglich drei Unterschiede: Während des Herunterladens der Metadaten werden alle Nachrichten außer den Nachrichten von BEP 9 und 10 ignoriert, es wird immer für alle noch ausstehenden Datenblöcke je ein Request in zufälliger Reihenfolge gesendet und bei Fertigstellung des Herunterladens wird der Download entfernt und die erhaltenen Metadaten als ein neues Torrent geladen.

5.3.7 GUI

Für die Benutzung der Veröffentlichungs- und Suchfunktionen durch Endanwender fehlt noch eine geeignete Benutzeroberfläche. Da der bestehende BitTorrent-Client eine mit Swing implementierte Oberfläche besitzt, soll damit fortgefahren werden. Genauso wie diese kommuniziert die Benutzeroberfläche ausschließlich über den EventManager mit der Implementierung im Backend, wodurch die Oberfläche leicht austauschbar ist. Die möglichen Events entsprechen dabei den zuvor beschriebenen.

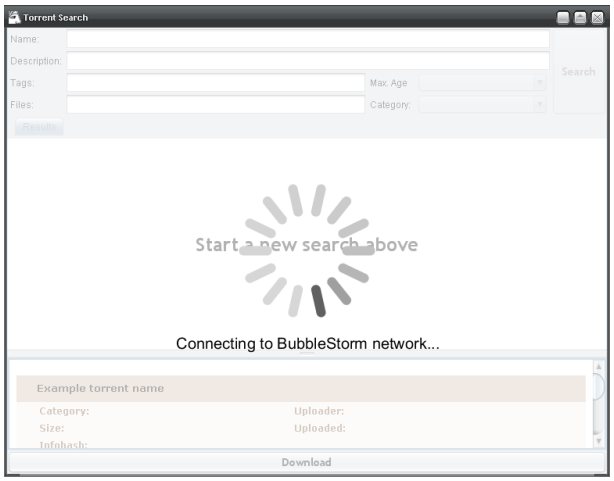

Abbildung 5.3: GUI wartet auf Netzwerkverbindung

Für Veröffentlichung und Suche ist jeweils ein eigenes Fenster vorhanden. Beim Öffnen eines dieser Fenster wird zunächst geprüft, ob eine Verbindung mit dem BubbleStorm-Netzwerk besteht. Solange dies nicht der Fall ist, wird der Fensterinhalt blockiert und von einer halbtransparenten, runden, unendlichen Fortschrittsanzeige überlagert (siehe Abb. 5.3), die ursprünglich von Roman Guy [WAIT05] entwickelt und in einer verbesserten Version in die Effektebibliothek SwingFX [SFX08] übernommen wurde.

Veröffentlichung

Abbildung 5.4: GUI für das Veröffentlichen von Torrents

Die Veröffentlichung von Torrents erfolgt über ein Fenster mit einem Formular (siehe Abb. 5.4). In diesem wählt der Benutzer die zu veröffentlichende .torrent Datei aus, gibt Name, Tags und eine Beschreibung ein, wählt eine passende Kategorie aus und bestimmt, ob die in der .torrent Datei hinterlegten Tracker ebenfalls veröffentlicht werden sollen.

Nach einem Klick auf den Button „Publish“ werden die Angaben über den EventManager an das Backend delegiert. Wenn das Backend einen Fehler zurückmeldet, dann wird eine entsprechende Fehlermeldung ausgegeben. Im Erfolgsfall wird der Benutzer darüber informiert, dass die Veröffentlichung im Gange ist, aber keine weitere Meldung erfolgt, sobald das Torrent tatsächlich im Netzwerk auffindbar ist, da sich nicht bestimmen lässt, ob der Bubblecast wirklich gesendet wurde und mindestens einen anderen Knoten erreicht hat.

Suche

Wie in der Abbildung 5.5 zu sehen ist, ist die Suchoberfläche in drei Bereiche unterteilt: Je einem für die Eingabe der Suche, die Ergebnisliste und für die Anzeige von Details zu den Suchergebnissen.

In der Eingabemaske gibt es Eingabefelder für die Suche in den vier Feldern Name, Beschreibung, Tags und Dateinamen, sowie zwei Auswahllisten (Dropdown) zum Einschränken der Suche nach Alter und Kategorie. Durch Drücken der Eingabetaste oder Betätigen der Search-

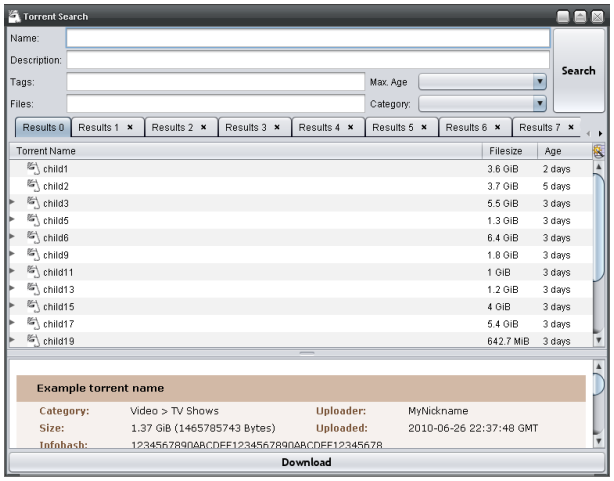

Abbildung 5.5: GUI der Torrent-Suche

Buttons wird die Suche gestartet. Kann die Suche nicht gestartet werden (z. B. wegen fehlenden Eingaben), wird ein Fehlerdialog angezeigt, andernfalls öffnet sich im mittleren Bereich des Fensters ein neues Tab.

Pro Suche wird ein eigenes Tab geöffnet. Jedes Tab erhält als Namen die Sucheingabe des ersten nicht leeren Eingabefeldes, damit der Benutzer zwischen verschiedenen Suchanfragen unterscheiden kann. Tabs können über die von Browsern bekannten Tastenkombinationen Strg+Tab, Strg+Umsch+Tab und Strg+W vorwärts und rückwärts traversiert bzw. geschlossen werden. Solange ein Tab geöffnet ist, werden kontinuierlich die eintreffenden Suchergebnisse angezeigt. Mit Schließen eines Tabs wird die entsprechende Suche gestoppt. Die Ergebnisse werden in einer Outline-Komponente angezeigt, die den Quellen von NetBeans entnommen ist. [OUT08] Outline ist eine Kombination aus einem JTree und einer JTable. Es können ausklappbare Baumstrukturen mit mehreren Spalten angezeigt werden. Dies wird dafür benutzt, mehrfach veröffentlichte Torrents anzuzeigen, die zwar den selben Infohash besitzen, zu denen es aber beispielsweise mehrere Beschreibungen gibt. Ergebnisse, die sich auf dasselbe Torrent beziehen, aber nicht einem bereits empfangenen identischen Suchergebnis entsprechen, werden in einem Ast des Baums gruppiert und zunächst nur das zuerst erhaltene Ergebnis angezeigt. Die alternativen Ergebnisse können durch Expandierung über das vorangestellte Symbol angezeigt werden. Swing (und auch Outline) arbeitet nach dem Model-View-Controller Prinzip. Da Outline aus zwei Komponenten besteht, werden zwei Models erwartet: Das TreeModel bringt die erhaltenen Daten durch die Gruppierung der Torrents mit identischem Infohash in Baumform, liefert die Knoten des Baumes auf Anfrage an die View und hält die Daten für die erste (expandierbare) Spalte der Tabelle bereit. Bei Änderungen des Baumes wird die View entsprechend informiert. Ein RowModel wird für die Anzeige aller weiteren Spalten benötigt. Es bekommt von der View den anzuzeigenden Baumknoten übergeben und gibt den anzuzeigenden Wert der

gewünschten Spalte formatiert zurück. Dadurch können Größe und Alter des Torrents, sowie die Ergebnisrelevanz in einer leichter lesbaren Form präsentiert werden. Des Weiteren kann nach mehreren Spalten gleichzeitig sortiert werden (Umschalt-Taste), Spalten lassen sich ausblenden und können beliebig vertauscht werden.

Durch Selektieren eines Suchergebnisses werden im unteren Fensterbereich weitere Details angezeigt. Dafür wird ein JEditorPane im HTML-Modus verwendet, das HTML und CSS teilweise implementiert. Zur Generierung der Seite wird HTML.Template.java von Philip S. Tellis [HTJ02] verwendet. Die Bibliothek ist an das Perl-Modul HTML::Template.pm angelehnt. Sie generiert aus einem Template - durch die Zuweisung von Variablen - eine HTML-Seite und übernimmt das Escaping von Sonderzeichen wie den Begrenzungszeichen von HTML-Tags, sodass beispielsweise <script></script> Blöcke nicht als ausführbare Skripte interpretiert werden. In der Detailansicht werden alle verfügbaren Angaben zum Torrent angezeigt, darunter Größe, Veröffentlichungsdatum, Liste der Tracker, Tags und die Beschreibung.

6 Evaluierung

Um die neuen Funktionen des BitTorrent-Clients zu testen, wurde in drei Stufen vorgegangen:

Lokaler Test, eine Instanz

Um zu überprüfen, dass die Suche mit Lucene über BubbleStorm grundsätzlich funktioniert, wurde zunächst nur eine Instanz des BitTorrent-Clients gestartet. Da diese Instanz der einzige Knoten im Netzwerk ist, fungiert sie als Bootstrap Knoten. Als einzige Instanz im Netzwerk werden alle Bubbles immer an die eigene Instanz gesendet, sodass sichergestellt ist, eine Antwort zu erhalten.

Als Datenbasis für das Evaluieren der Suchfunktion diente eine 424.000 Torrents umfassende Liste von Namen, die von einer bekannten Suchseite für Torrents im Internet stammen. Sie wurden mit zufällig generierten Angaben für Größe, Dateien, Beschreibung und Kategorie dem Index hinzugefügt. Tags wurden aus den Namen erzeugt. Dies sind bedeutend mehr Torrents, als in der Praxis auf einem Netzwerkknoten zu erwarten sind (Erinnerung: Verteilung der Torrents auf Knoten mit $O(\sqrt{n})$). Dadurch ließ sich aber die Performanz von Lucene gut abschätzen. Eine komplexere Suche mit mehreren Suchbegriffen (inklusive Phrasen), Tags und eingeschränkter Kategorie dauerte selten länger als 200ms auf einem Zweikernsystem. Damit arbeitet die Suche schnell genug, um Suchanfragen auch bei höherem Suchaufkommen zeitnah beantworten zu können.

In den nachfolgenden zwei Tabellen sind einige Stichproben von Antwortzeiten auf zwei Suchanfragen festgehalten. Die Tests wurden in einem LAN auf zwei Knoten durchgeführt, wobei ein Knoten einen Index mit 20.000 Dokumenten im lokalen Lucene-Index besaß und ein zweiter Knoten mit leerem Index Suchanfragen startete.

Lokaler Test, mehrere Instanzen

Zum Testen der dezentralen Peer-Suche und des Herunterladens der Metadaten von anderen Peers wurden zwei bis neun Instanzen des Clients verwendet. Auf einer davon wurde ein

bereits fertiggestelltes Torrent im Seeding-Modus gestartet. Dieses Torrent wurde auch per BubbleStorm veröffentlicht, sodass die anderen Clients danach suchen konnten. Um sicherzustellen, dass nur über BubbleStorm und nicht über HTTP Tracking nach neuen Peers gesucht wird, wurde Letzteres deaktiviert. Die anderen Clients haben nach dem Torrent gesucht und es zu ihren Downloads hinzugefügt. Nachdem der Peer im Seeding-Modus gefunden wurde, wurden die Metadaten des Torrents von diesem heruntergeladen, das Torrent in den Client geladen und mit dem Herunterladen der Daten begonnen.

Die folgende Tabelle zeigt die Messreihe einer mehrfachen Suche nach Torrent-Namen mit nur einem einzigen *Stichwort*. Die Suche ergab 50 Ergebnisse. Es wurde jeweils die Zeit gemessen, die zwischen Absenden der Suche durch den Benutzer bis zum Eintreffen der Ergebnisse vergangen ist:

Erstes Ergebnis nach	219ms	91ms	82ms	89ms	67ms	74ms	68ms
Letztes Ergebnis nach	379ms	140ms	132ms	145ms	169ms	141ms	127ms

Die zweite Tabelle enthält Messwerte einer komplexeren Anfrage. Diese ist eine Suche nach Torrent-Namen in der Form *Wort +"Wortgruppe als Phrase" -Ausschlusswort*. Diese Suche lieferte nur vier Suchergebnisse:

Erstes Ergebnis nach	217ms	73ms	43ms	35ms	40ms	40ms	38ms
Letztes Ergebnis nach	237ms	91ms	65ms	48ms	43ms	56ms	39ms

Man erkennt hier zwei Dinge:

1. Der jeweils erste Meßwert ist deutlich höher als die restlichen Werte. Der Grund dafür ist, dass der Index durch Lucene erst mit dem Eintreffen der ersten Suchanfrage geöffnet wird. Dies ist relativ rechenintensiv, weshalb versucht wird, den Index bei der Torrent-Suche so selten wie möglich neu zu öffnen.

2. Die komplexere Suchanfrage benötigt weniger Zeit. Offensichtlich hängt die Zeit, die für die Suche vergeht, von der Anzahl der Ergebnisse ab. Mögliche Gründe dafür sind die Berechnung des Scores aller Ergebnisse, die Verwendung der Fuzzy-Suche oder Lucenes Suchmechanismus.

Test per Internet, mehrere Instanzen

In der letzten Teststufe wurde mit 50 Instanzen über das Internet getestet. Die meisten davon liefen auf drei Linux-Servern, die keine grafische Oberfläche besitzen. Um diese Instanzen steuern zu können, war ein Interface ohne GUI erforderlich. Da es wünschenswert war, dass die Instanzen ferngesteuert werden können, erhielten sie ein Socket Interface, das TCP-Verbindungen annimmt. Per Telnet können darüber textbasierte Kommandos an die Instanz gesendet werden und der aktuelle Status abgefragt werden. Konkret werden folgende Kommandos unterstützt:

- status: Liefert Liste der geladen Torrents und Statistiken
- bsstatus: Liefert Verbindungsstatus zum BubbleStorm Netzwerk
- enablepush: Aktiviert sekündliches Push-Update des Torrent-Status
- disablepush: Deaktiviert sekündliches Push-Update des Torrent-Status
- shutdown: Beendet den Client
- start <id>: Startet das Torrent mit der angegebenen ID
- stop <id>: Stoppt das Torrent mit der angegebenen ID
- remove <id>: Startet das Torrent mit der angegebenen ID
- retrievebyhash <infohash> [<tracker1>, <tracker2>, ...]: Fügt ein Torrent anhand des angebenen Infohashes hinzu und startet das dezentrale Herunterladen der Metadaten

TUDTorrent Peers:

IP:Port	Status	BubbleStorm	Links
.96.78:1111	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63002	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63004	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63006	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63008	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63010	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63012	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63014	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63016	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63018	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63020	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63022	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63024	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63026	ONLINE	CONNECTED	CONNECT STDOUT LOGFILE DEBUGLOGFILE SHUTDOWN KILL INSTANCE
.96.78:63028	OFFLINE	N/A	LAUNCH KILL INSTANCE
.96.78:63030	OFFLINE	N/A	LAUNCH KILL INSTANCE
.96.78:63032	OFFLINE	N/A	LAUNCH KILL INSTANCE

Abbildung 6.1: Webinterface zur Steuerung von Instanzen

Um nicht zu jeder Instanz einzeln eine Verbindung herstellen und Kommandos eingeben zu müssen, gibt es ein Webinterface, über das alle Instanzen gesteuert werden können. Auf der Hauptseite (Abb. 6.1) wird angezeigt, welche Instanzen aktiv sind und ob BubbleStorm eine Verbindung zum Netzwerk hergestellt hat. Es lassen sich neue Instanzen starten, Einblick in die Logs nehmen, Instanzen beenden oder forciert terminieren. Für jede Instanz gibt es eine Detailansicht, in der die geladen Torrents angezeigt werden, die sich starten, stoppen, entfernen und als Infohash hinzufügen lassen.

Dies ist über eine Reihe von PHP- und Shell-Skripten realisiert. Die PHP-Skripte nutzen das zuvor beschriebene Socket-Interface sowie eine SSH-Verbindung, um die Shell-Skripte auszuführen, die dem Starten und Terminieren von Instanzen, sowie dem Abrufen der Logs dienen.

Mithilfe des Webinterfaces wurde das dezentrale Laden der Metadaten von Torrents und die Peer-Suche nochmals getestet, dieses mal über das Internet. Über eine weitere Instanz mit GUI auf einem dritten Rechner wurden Tests der dezentralen Suche und Veröffentlichung durchgeführt, um zu prüfen, dass die Antwortzeiten über das Internet zufriedenstellend sind.

7 Zusammenfassung

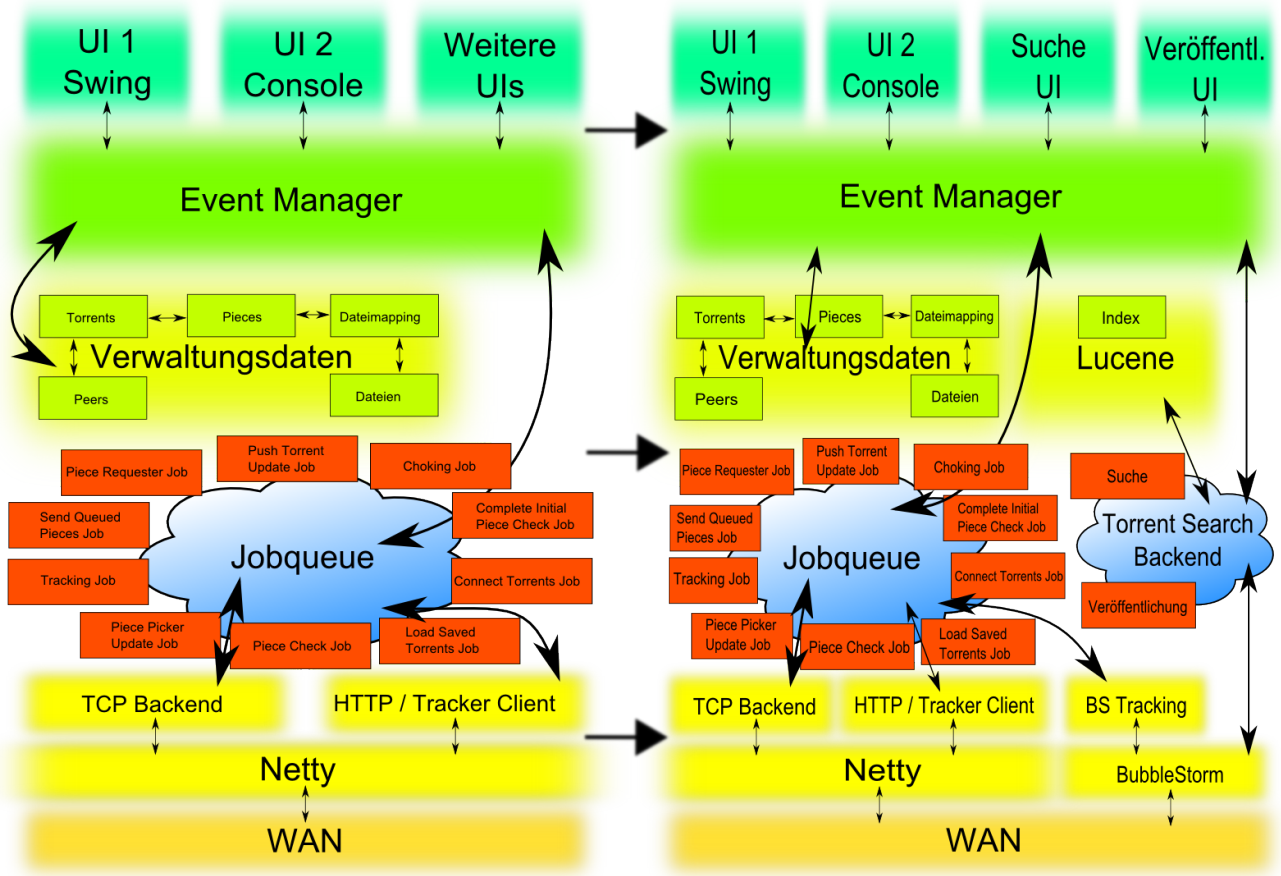

Abbildung 7.1: Schematische Darstellung des Entwicklungsstandes vor und nach dieser Arbeit

Das Ziel dieser Bachelorarbeit war es, einen bestehenden, Java-basierten BitTorrent-Client so zu erweitern, dass sich Torrents und Peers über eine dezentrale Netzwerkstruktur auffinden lassen, ohne von zentralen Komponenten wie einem Tracker oder einer Website für die Suche nach Torrents abhängig zu sein.

Mithilfe des unstrukturierten Peer-to-Peer Netzwerks BubbleStorm wurde eine Erweiterung entwickelt, die die Suche nach neuen Peers für ein Torrent im Netzwerk propagiert. Mit jeder Suche werden automatisch die Verbindungsdaten zum eigenen Peer übermittelt, sodass jede Suche gleichzeitig einer Aufforderung zur Kontaktaufnahme durch andere Peers entspricht.

Zum Finden und Publizieren von Torrents über BubbleStorm war es wichtig, die auf den einzelnen Knoten gespeicherten Suchinformationen möglichst effizient zu indizieren und durchsuchbar zu machen. Dafür wird die Open Source Textsuchmaschine Lucene verwendet, die bei der Umwandlung der vom Benutzer eingegeben Texte in die benötigten Tokens hilft, und komplexe Suchanfragen ausführen kann. Der Benutzer kann über eine grafische Oberfläche neue Torrents einstellen, sowie nach vorhandenen Torrents suchen. Die Suchergebnisse werden in einer Liste präsentiert, aus der die Torrents zum Herunterladen ausgewählt werden können.

Da die Suchfunktion aus Bandbreitengründen nicht alle Meta-Informationen über ein Torrent speichert bzw. übermittelt, wurde eine dritte Erweiterung entwickelt, die die beiden zuvor genannten Erweiterungen zusammenführt. Die Metadaten zu einem Torrent, die üblicherweise in der Form von .torrent Dateien in den BitTorrent geladen werden, lassen sich nur mit Kenntnis des Infohashes eines Torrents von anderen Peers herunterladen, die dasselbe Torrent verteilen. Dies geschieht über die beiden Erweiterungen des BitTorrent-Protokolls BEP 9 und BEP 10, die auch für das Funktionieren von Magnet-Links (siehe 3.2) in einigen aktuellen BitTorrent-Clients verantwortlich sind. Peers werden hier aber nicht über DHTs gefunden, sondern über BubbleStorm.

Die Abbildung 7.1 gibt - von links nach rechts - einen zusammenfassenden Überblick über die Evolution des BitTorrent-Clients seit Beginn dieser Arbeit. Im rechten Teil der Grafik sind die Erweiterungen zu finden. Wie sich leicht erkennen lässt, ist die Suchfunktion weitgehend unabhängig vom Rest des BitTorrent-Clients, was sie gut geeignet für weitere ähnliche Projekte macht.

8 Ausblick

Dieses Kapitel widmet sich möglichen Weiterentwicklungen, die bestehende Probleme und neue zusätzliche Funktionen betreffen, die die Benutzung verbessern können.

8.1 Funktionalität

Diese Arbeit umfasst lediglich die grundlegende Funktionalität, um Torrents ohne jegliche zentrale Entität zu veröffentlichen, zu finden und herunterzuladen. Für einen dauerhaften Erfolg in großen, öffentlichen Netzen ist dieser Funktionsumfang allerdings nicht ausreichend. In so gut wie allen File Sharing Systemen kam es mit wachsender Anzahl der Nutzer auch vermehrt zu unsachgemäßer Verwendung der vorhandenen Funktionen. Beispielsweise werden häufig Dateien verteilt, deren Name und/oder Beschreibung keinen Bezug zum Dateiinhalt hat (sog. Fakes), deren Inhalt von sehr schlechter Qualität ist oder die Schadsoftware enthalten.

Aus diesen Gründen wurden viele File Sharing Systeme um soziale Funktionen erweitert, die es Benutzern erlauben, andere Benutzer über die Qualität der Inhalte zu informieren. Dies geschieht beispielsweise über Bewertungssysteme, individuelle Kommentare oder direkte Empfehlungen an andere Nutzer, bzw. eine Kombination dieser Möglichkeiten.

Solche sozialen Komponenten fehlen in der vorliegenden Version und bieten daher einen sinnvollen Ansatz für weitere Projekte.

8.2 Datenverfügbarkeit

Des Weiteren wird die Verfügbarkeit der im Netzwerk hinterlegten Daten in der vorliegenden Version der BubbleStorm-Bibliothek nicht garantiert. Wenn ein oder mehrere Knoten das Netzwerk verlassen, findet keine Umverteilung der Replikate statt, die vom verlassenden Knoten vorgehalten wurden. Daraus folgt, dass Informationen verloren gehen können, wenn alle Knoten, die ein bestimmtes Data Bubble lokal gespeichert haben, nicht mehr verfügbar sind. Dieses Problem wird mit einer Nachfolgeversion der BubbleStorm-Bibliothek behoben.

8.3 Sicherheit

Die aktuelle Implementierung bietet Angriffsvektoren sowohl für Angriffe auf das BubbleStorm-Netz, einzelne Nutzer, als auch Rechner im Internet, die keine Beziehung zum Netzwerk haben.

Derzeit gibt es keine Schutzmaßnahmen gegen Flooding- und Spoofing-Attacken. Ein Angreifer, der mehrere 1000 Bubbles pro Sekunde in das BubbleStorm-Netzwerk senden kann, ist in der Lage, mehrere Knoten (insbesondere Adjazenten) zur Überlastung zu bringen und so für Teilausfälle zu sorgen.

Das dezentrale Tracking lässt sich in der aktuellen Form dazu missbrauchen, DDoS-Attacken gegen beliebige Rechner zu starten. Da jeder Knoten seine IP-Adresse und den verwendeten Port selbst bekannt gibt, können beide prinzipiell beliebig gewählt werden. Ein Angreifer könnte so beliebige Kombinationen aus IP und Port verwenden und sie für beliebige Torrents anmelden. Verwendet der Angreifer die Adresse seines Angriffsziels für die Anmeldung im Schwarm sehr populärer Torrents, dann wird das Opfer mit unzähligen Verbindungsversuchen von BitTorrent-Clients penetriert, was zu Beeinträchtigungen der Funktionalität und Erreichbarkeit des Opfers führen kann. Dies ist insbesondere gegen Ports effektiv, auf denen bereits bei der Herstellung einer Verbindung Rechenzeit investiert wird, z. B. für die Zufallszahlengenerierung von SSL. Solche Angriffe werden in der Praxis genutzt, um gezielt Server zu überlasten. [1]

Dieselbe Art von Angriff lässt sich auch dafür verwenden, es Peers aus dem BitTorrent-Schwarm nahezu unmöglich zu machen, legitime andere Peers zu finden. Dazu müssen ausreichend viele gefälschte Adressen dem Schwarm beitreten, sodass die Wahrscheinlichkeit sehr gering wird, dass ein Peer beim Herstellen neuer Verbindungen einen legitimen Peer wählt.

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

9 Glossar

Bencoding

Eine Kodierung für einfach strukturierte Daten, die ASCII-Zeichen zur Trennung von Datenfeldern verwendet. Sie wird in verschiedenen Teilen von BitTorrent eingesetzt, um Metadaten zu übermitteln und besteht aus den vier Datentypen Byte String, Integer, List und Dictionary.

BitTorrent

BitTorrent ist ein Peer-to-Peer Filesharing-Protokoll. Für jedes Torrent wird ein eigenes Verteilnetz aufgebaut. Es hat sich besonders für die schnelle Verteilung größerer Datenmengen etabliert.

Bootstrap

Als Bootstrap wird in einem Peer-to-Peer Netzwerk ein Knoten bezeichnet, der für weitere Knoten als Eintrittspunkt in das Netzwerk dient. Oft ist dieser Knoten derjenige, der das Netzwerk gegründet hat.

Bubble

Ein Bubble ist ein zyklenfreier Teilgraph über Knoten eines Bubblestorm-Netzwerks. Die Nutzdaten eines Bubbles werden auf allen Knoten eines Bubbles repliziert. Es gibt zwei Arten von Bubbles: Data Bubbles und Query Bubbles.

Bubblecast

Den Vorgang der Replikation der Daten in einem Bubble wird als Bubblecast bezeichnet.

BubbleStorm

BubbleStorm ist ein Peer-to-Peer System das Daten und Suchanfragen über ein strukturloses, dezentrales Netzwerk verbreitet. Die Replikation von Daten erfolgt in sogenannten Bubbles.

Infohash

Der Infohash ist eine SHA-1 Prüfsumme, die jedes Torrent eindeutig identifiziert. Sie wird über den Info-Schlüssel gebildet, der Teil jeder .torrent-Datei ist und alle wichtigen Informationen über die zu verteilenden Daten enthält.

Lucene

Apache Lucene ist eine dokumentenbasierte Open Source Bibliothek für Aufgaben im Bereich Information Retrieval. Sie indiziert Texte und ermöglicht Volltextsuche, die für beliebige Projektgrößen skaliert und komplexe Suchanfragen auf dem Index erlaubt.

Netty

Als NIO Framework ermöglicht Netty die einfache Implementierung von Client- und Server-Software mit asynchroner Kommunikation über TCP und UDP. Daten werden in einer Filterkette zwischen Applikation und Netzwerk bidirektional transformiert, sodass die vom jeweiligen Protokoll definierten Datenstrukturen von der Applikation leicht verarbeiten werden können und umgekehrt.

Peer

Als Peer wird bei BitTorrent jeder Teilnehmer bezeichnet, der an der Verteilung eines Torrents beteiligt ist.

Schwarm

Ein Schwarm bezeichnet die Menge aller Peers eines Torrents.

Tokenizer

Für die Indexierung müssen Texte in kleinere Einheiten eingeteilt werden, den Tokens. Ein Algorithmus, der einen Text in eine Menge von Tokens überführt, wird als Tokenizer bezeichnet.

Torrent

Ein Torrent ist eine logische Einheit für die Verteilung von Daten über BitTorrent. Ein oder mehrere Dateien und weitere Informationen wie z. B. Dateigrößen, Prüfsummen und Tracker URLs werden als ein sogenanntes Torrent zusammengefasst und in einer Torrent-Datei gespeichert. Die in dieser Datei enthaltenen Informationen ermöglichen es, die von ihr beschriebenen Daten mithilfe eines BitTorrent Clients herunterzuladen und zu verteilen. Oft wird diese Datei abkürzend ebenfalls als Torrent bezeichnet.

Tracker

Tracker dienen bei BitTorrent dazu, die Peers eines Torrents einander zu vermitteln. Der Tracker ist also die zentrale Komponente, die das Verteilen eines Torrents ermöglicht. Peers fragen beim Tracker regelmäßig eine Liste bekannter Peers für das Torrent an. Gleichzeitig registriert sich der lokale Peer beim Tracker, um selbst von anderen Peers kontaktiert werden zu können.

Vector Space Model

Das Vector Space Model ist ein algebraisches Modell aus dem Bereich Information Retrieval. Dokumente werden als Vektoren von Bezeichnern (z. B. Suchtermen) in einem mehrdimensionalen Vektorraum repräsentiert. Die Distanz zwischen Vektoren kann dazu benutzt werden, eine Rangliste für die Ergebnisse einer Suche über eine Menge von Dokumenten zu bilden.

10 Abbildungsverzeichnis

11 Literaturverzeichnis

[BBS07] W. W. Terpstra, J. Kangasharju, C. Leng, A. P. Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. *SIGCOMM'07*, 2007.

[BT03] B. Cohen. Incentives Build Robustness in BitTorrent. 2003.
http://www.bittorrent.org/bittorrentecon.pdf

[CDN02] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An Analysis of Internet Content Delivery Systems. *OSDI '02*, 2002.
http://www.usenix.org/events/osdi02/tech/export/saroiu/saroiu_html/

[CUB08] B. Wong, A. Slivkins, E. Gün Sirer. Approximate Matching for Peer-to-Peer Overlays with Cubit. Cornell University, 2008.
http://hdl.handle.net/1813/11651

[DHT08] A. Loewenstern. DHT Protocol. 2008.
http://www.bittorrent.org/beps/bep_0005.html

[EP08] A. Norberg, L. Strigeus, G. Hazel. Extension Protocol. 2008.
http://www.bittorrent.org/beps/bep_0010.html

[HTJ02] P. S. Tellis. HTML.Template.java Projektseite. 2002.
http://html-tmpl-java.sourceforge.net/

[IPO08] ipoque. Internetstudie 2008/2009. 2009.
http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009

[JNI05] A. Ofterdinger. Java Native Interface ab J2SE 1.4. *Javaspektrum*. Nr. 5, 2005, S.32-35.
http://www.sigs.de/publications/js/2005/05/ofterdinger_JS_05_05.pdf

[KAD02] P. Maymounkov, D. Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. New York University, 2002.

http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf

[LUC10] Apache Lucene Projektseite. 2010
http://lucene.apache.org/

[MAG02] G. Mohr. MAGNET v0.1 Created 2002-06-12; Revised 2002-06-17. 2002.
http://magnet-uri.sourceforge.net/magnet-draft-overview.txt

[NIO02] R.Hitchens. Java Nio. O'Reilly & Associates, Inc., 2002.

[OUT08] T. Boudreau. Egads! An actual Swing Tree-Table! 2008.
http://weblogs.java.net/blog/timboudreau/archive/2008/06/egads_an_actual.html

[SFX08] SwingFX Projektseite. 2008.
https://swingfx.dev.java.net/

[SMF08] G. Hazel, A. Norberg. Extension for Peers to Send Metadata Files. 2008.
http://bittorrent.org/beps/bep_0009.html

[Spec10] Bittorrent Protocol Specification v1.0. Stand Juni 2010.
http://wiki.theory.org/index.php?title=BitTorrentSpecification&oldid=3418

[SQL10] SQLite Home Page. 2010.
http://www.sqlite.org/

[TRB07] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, H. J. Sips1. TRIBLER: a social-based peer-to-peer system. Delft University of Technology, Vrije Universiteit, 2007
https://www.tribler.org/attachment/wiki/File/CPE-Tribler-final.pdf?format=raw

[TUT09] T. Eckert, A. Teuber. Praktikum - Entwicklung eines BitTorrent-Clients. Technische Universität Darmstadt, 2009.

[WAIT05] R. Guy. Wait with style in Swing. 2005.
http://www.curious-creature.org/2005/02/15/wait-with-style-in-swing/