Peer-to-peer Netzwerke

Protokolle und der Einsatz in der verteilten Software-Entwicklung

Hauke Stieler

Fachbereich Informatik, TGI, Universität Hamburg

Inhaltsverzeichnis

Zusammenfassung Durch die zunehmende Vernetzung der gesamten Welt kann ein einzelner Haushalt immer mehr Daten pro Sekunde empfangen womit sich sowohl neue Möglichkeiten, als auch Probleme ergeben. Um einen möglichst großen Datendurchsatz zu gewährleisten gibt es Technologien wie Peer-to-Peer (P2P). Anhand von BitTorrent wird hier die Technologie dahinter, sowie das BitTorrent-Protokoll selbst, erläutert und Einsatzgebiete betrachtet.
Mit Hilfe von P2P wird die Umsetzung dezentraler Werkzeuge vereinfacht, welche eine ebenfalls dezentrale Software-Entwicklung fördern. Als Beispiele hierzu werden Softwarewerkzeuge zum Verbreiten und Versionieren von Software, sowie zum verfolgen von Tickets herangezogen.

1 Motivation

Die Menge an Daten, die über das Internet versendet werden, erfährt seit jeher ein exponentielles Wachstum mit einer jährlichen Steigerung um mehr als 20%. Bereits 2015 waren dies weltweit rund 72,4 Exabyte pro Monat, was sich bis 2018 verdoppeln wird [Cisco, 2015].
Dieser Zuwachs bringt jedoch einige Probleme mit Sich:

Ansprüche an Hardware werden höher

Durch diese enormen Zuwächse innerhalb kürzester Zeit ändern sich auch die Ansprüche an Teile der gesamten Infrastruktur wie z.B. Server und Router – Innerhalb von drei bis vier Jahren müssen sie die doppelte Datenmenge in der gleichen Zeit bearbeiten bzw. liefern können.

Möglich wird das mit dieser Architektur nur durch den Austausch von Hardware was ein enormen Kostenpunkt darstellt. Gemeinnützige Organisationen sind dabei auf Spenden angewiesen, wie zum Beispiel der Halifax Server von der RWTH Aachen, der Hardware von Fujitsu gespendet bekam als der damalige Server nicht mehr ausreichte [Fabelje et al., 2016].

Tragweite einer Ausfalls wird höher

Nimmt die Wichtigkeit eines Servers durch die zu liefernde Datenmenge zu, so steigt auch die Tragweite eines Ausfalls. Um einen solchen Ausfall jedoch zu vermeiden gibt es verschiedene Schutzmaßnahmen wie etwa eine unterbrechungsfreie Stromversorgung (USV), Redundanz der Daten auf mehreren Festplatten oder gar Redundanz auf mehreren Servern.

Eine USV schützt allerdings nicht vor allen Ausfallgründen (z.B. Hardwareschäden oder Wartungsarbeiten) und vollständige Redundanz verursacht höhere Kosten.

Administrator trägt volle Verantwortung

Die Administrierenden tragen bei Servern die volle Verantwortung, soll heißen, dass sie direkten Zugriff auf die Daten haben, die zum Beispiel bei Hosting-Anbietern nicht dem Unternehmen, sondern den Kunden gehören. Bei reinen Download Servern haben somit wenige Menschen die gesamte Kontrolle über die Daten und im Falle einer Löschung sind diese ohne weiteres nicht mehr verfügbar.

Auch bildet die Administratorebene eine Schwachstelle, die von beispielsweise Geheimdiensten leicht ausgenutzt werden kann. Eindrucksvoll demonstriert wurde das bei TrueCrypt, einer weit verbreiteten und bis zur Einstellung sicheren Verschlüsselungssoftware [Schmidt, 2014]:

Im Mai 2014 wurde das Projekt eingestellt und seit dem ist die aktuellste Version 7.2, welche nur noch *ent*schlüsseln, aber nicht mehr *ver*schlüsseln kann. Sehr wahrscheinlich ist ein *National Security Letter* der Grund für die Einstellung von TrueCrypt. Da sich TrueCrypt großer Beliebtheit erfreut ist die Version

7.1a (welche auch verschlüsseln kann) weiterhin verfügbar. Gäbe es keine weitere Verbreitung, könnte man TrueCrypt nicht mehr herunterladen und müsste veraltete oder alternative Software benutzen.

Viele Probleme mit P2P lösen

Die genannten Probleme lassen sich mit Hilfe von P2P-Netzwerken lösen, welche im optimalen Fall **volle Redundanz** der Daten und einen **hohen Datendurchsatz** verbunden mit **niedrigen Kosten** bieten. Anhand vom BitTorrent Protokoll wird gezeigt, wie File-sharing mittels P2P funktioniert.

Die Frage die nach den technischen Details beantwortet werden soll geht jedoch mehr in Richtung Softwareentwicklung:

Wie kann P2P in der verteilten Softwareentwicklung eingesetzt werden?

2 Einführung

Dieses Kapitel gibt einen Einblick in die Grundlegende Technik hinter P2P-Netzwerken und BitTorrent als Netzwerkprotokoll. Darüber hinaus befinden sich hier Beispielhafte Anwendungsfälle für P2P-Netzwerke aus verschiedenen Themengebieten.

2.1 P2P Netzwerke

Ein *reines* P2P-Netzwerk besteht aus mehreren gleichberechtigten Computern, die alle die selbe Aufgabe haben (s. Abbildung 1). Es gibt also keinen Computer (Server) der eine höhere Stellung hat als andere Mitglieder des Netzwerkes. Der Name *Peer-to-Peer* kommt von der Bezeichnung der einzelnen Rechner, den *Peers*.

2.2 BitTorrent

BitTorrent ist ein Protokoll zum aufbauen und betreiben eines P2P Netzwerkes. Der Aufbau des Netzes entspricht dabei weitestgehend dem eines normalen P2P Netzes, jedoch mit einer Einschränkung, dass es in einem klassischen BitTorrent Netzwerk einen zentralen Server (sog. *Tracker*) gibt, der Anfragen verwaltet und weiterleitet (s. Abbildung 2). Der Tracker ist ein zentraler Server, der Zugriffe verwaltet und Metainformationen speichert. Er hält selbst keine Daten, die versendet werden sollen. Bei einem Ausfall kann somit ein anderer Tracker-Server einspringen ohne, dass Daten verloren gehen.

BitTorrent wird vor allem zum Austausch von Dateien (*file-sharing*) benutzt, wobei immer nur kleinste Teilstücke angefordert und gesendet werden. Dadurch kann auch ein Downloader mit unvollständigen Daten bereits heruntergeladene

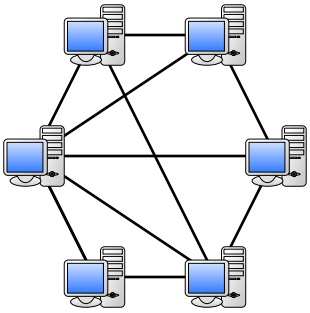

Abbildung 1. Reines P2P-Netzwerkes: Jeder ist gleichberechtigt.

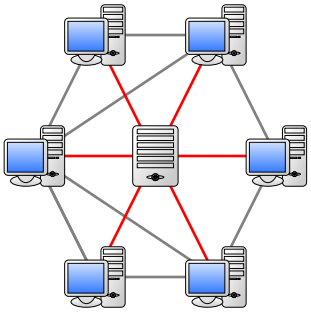

Abbildung 2. BitTorrent-Netzwerkes: Über den Tracker läuft die Vermittlung der Peers.

Teile senden, wodurch eine maximale Netzwerkauslastung (und somit eine minimale Download Zeit) angestrebt wird.

2.3 Beispielhafte Anwendungsfälle

Video-streaming: Dies ist zwar nicht der klassische Anwendungsfall von BitTorrent, bzw. P2P, er lässt sich damit jedoch sehr gut realisieren.

Da immer mehr Video-On-Demand- oder auch IPTV-Dienste (herkömmliches Fernsehen über das Internet) angeboten werden, wird die Belastung für die dahinter liegenden Server immer größer. Auch eine höhere Qualität des Materials, wie es beispielsweise bei Übertragungen mit einer 4K-Auflösung der Fall ist, belastet zunehmend die Server.

Um dennoch ein breites Angebot bieten zu können entwickelt die BBC beispielsweise an einem P2P Netzwerk samt eigenem Client für ihre Videothek – dem iPlayer [p2pfoundation, 2005]. Der Stream hat dadurch eine Latenz von teilweise mehreren Minuten und eignet sich somit nicht für Live-Übertragungen, Filme lassen sich dadurch aber sehr gut streamen.

Spielehersteller: Sie benutzen ebenfalls P2P Netzwerke für die Verbreitung von Inhalten. Blizzard Entertainment benutzt P2P (genauer gesagt BitTorrent) zwischen den Nutzern von Spielen wie World-Of-Warcraft, Starcraft 2 und Diablo 3 [various, 2008], [Blizzard, 2014]. Große Patches werden somit über P2P versendet, was sich jedoch optional abstellen lässt.

Auch kleinere Spiele wie zum Beispiel Rennspiele der TrackMania Serie setzen beim Laden von Rennstrecken und Spieler eigenen skins (Lackierung der Rennwagen) auf P2P Netzwerke [Nadeo, 2011].

3 Das BitTorrent Protokoll

In diesem Kapitel werden Begriffe geklärt und definiert, die später verwendet werden. Zudem gibt es den ersten Einblick in das BitTorrent Protokoll, sowie die `.torrent` Dateien und verteilten Hash-Tabellen.

3.1 Die Terminologie

Der Begriff *Peer* beschreibt einen downloadenden Teilnehmer und *peering* analog das herunterladen von Daten. Jemand der den kompletten Datensatz besitzt und somit ausschließlich Daten hoch lädt wird dabei als *Seeder* bezeichnet und *seeding* somit der Vorgang des hochladens. Besitzt ein Peer nur Teile der Daten, lädt diese aber bereits hoch, so sprich man von einem *Leecher*. Da Seeder und Leecher Daten hochladen und die dazugehörigen Konzepte für beide identisch sind, wird im Folgenden nur noch von Seeder gesprochen.

Als *Tracker*-Server wird ein Server bezeichnet, der zu allen Peers und Seedern eine Verbindung hält und Anfragen zu passenden Peers/Seedern umleitet [Shen et al., 2009]. Auf diesem Server sind insbesondere auch Informationen zu den Teilstücken eines Peers gespeichert, er weiß somit welcher Peer welches Teilstück gespeichert hat [Shen et al., 2009, S. 98].

Jeder Seeder und Leecher besitzt Teilstücke von Dateien, die sogenannten *Pieces*, die von anderen Peers angefordert werden können.

3.2 Die `.torrent`-Dateien

Die `.torrent`-Dateien enthalten Informationen zum Erreichen des Trackers, zum Dateinamen sowie Checksummen (SHA-1 Hashes) zum Verifizieren einer korrekten Übertragung.

Zunächst stehen allgemeine Informationen als *bencode*-String in der Datei, wobei bencode die bei BitTorrent verwendete Codierung ist, die im Grunde wie folgt aufgebaut ist:

`<Länge-der-Information>:<Information>`

So codiert zum Beispiel `5:hello4:world` die Worte `hello` und `world`. Bestimmte Informationen werden gesondert codiert und erhalten beispielsweise Präfixe, was jedoch hier nicht von tragender Bedeutung ist [Cohen, 2008]. Erwähnt werden sollte noch, dass bencode die Struktur *dictionary* benutzt um Schlüssel-Wert-Paare zu speichern.

Als Beispiel soll hier eine `.torrent` Datei zum herunterladen von `Fedora-WorkstationLivex86_64-24_Beta` dienen. Der Inhalt der Datei sieht dabei wie folgt aus:

```
d8:announce
46:http://torrent.fedoraproject.org:6969/announce
13:creation datei1462843336e
```

```
4:info
d5:files
ld6:lengthi1117e
4:path
l39:Fedora-Workstation-24_Beta-1.6-CHECKSUMee
d6:lengthi1460666368e
4:path
l46:Fedora-Workstation-Live-x86_64-24_Beta-1.6.isoeee
4:name
38:Fedora-Workstation-Live-x86_64-24_Beta
12:piece lengthi262144e
6:pieces
111460:<liste aller Hashes>
```

Hier sieht man zunächst die Länge der Information (als Zahl im bencode Format), einen Doppelpunkt und dann die Information. Im wesentlichen besteht jede Datei aus dem `announce` und dem `info` Key, wobei der `info` Key ein Dictionary enthält.

Man kann unter Linux die Informationen in einer `.torrent` Datei mittels `transmission-show` aus dem `transmission-cli` Paket auslesen und anzeigen. Hier am Beispiel der obigen torrent-Datei gezeigt (der Dateiname `FedoraWorkstation-Livex86_64-24_Beta` wurde aus Platzgründen zu `Fedora-23-x64` gekürzt):

```
~ $ transmission-show ./Fedora-23-x64.torrent
    Name: Fedora-23-x64
    File: Fedora-23-x64.torrent

    GENERAL

      Name: Fedora-23-x64
      Hash: 04cbb0172031a4956dbb71dcfb6c425cce7f9276
      Created by:
      Created on: Tue May 10 03:22:16 2016
      Piece Count: 5573
      Piece Size: 256.0 KiB
      Total Size: 1.46 GB
      Privacy: Public torrent

    TRACKERS

      Tier #1
      http://torrent.fedoraproject.org:6969/announce

    FILES

      Fedora-23-x64/Fedora-23-x64-1.6-CHECKSUM (1.12 kB)
      Fedora-23-x64/Fedora-23-x64-1.6.iso (1.46 GB)
```

In diesem Informationsblock findet man bereits sehr wichtige Informationen zum seeding und peering, unter anderem nämlich die `Piece Count` (im Protokoll selbst als `piece length` bezeichnet), also die Anzahl der Pieces, die angefordert werden können.

Nach der `piece length` werden SHA-1 Hashes für jedes einzelne Teilstück der Datei gespeichert. Da es SHA-1 Hashes sind hat jeder die Länge 160 Bit, bzw. 20 UTF-8 codierte Zeichen.

Der Nutzen: Bei der `.torrent` Datei liegt er darin, dass sie zum herstellen der Verbindung zum Tracker, zum prüfen der Integrität (also der Verifikation einer korrekten Übertragung) und zum Finden bisher fehlender Dateistücke benötigt wird.

3.3 Distributed-Hash-Table

Eine verteilte Hash-Tabelle (*distributed hash table* oder kurz *DHT*) ist eine Lösung zum Vermeiden eines Trackers [Loewenstern, 2008]. Die Information wo welches Teilstück einer Datei liegt wird dabei auf sogenannten *nodes* gespeichert, was Computer oder Server sind, die das DHT-Protokoll für BitTorrent unterstützen und auf einem UDP Socket arbeiten. Eine `.torrent` Datei wird trotzdem benötigt um den Einstieg zu ermöglichen oder die Integrität der bereits geladenen Daten mittels der Hashwerte zu prüfen. Es ist jedoch eine Liste von möglichen Einstiegsnodes gegeben und nicht mehr die URL zu einem zentralen Tracker.

Durch das verteilen der Information ist eine höhere Ausfallsicherheit gewährleistet, da die Metadaten nicht mehr zentralisiert auf dem Tracker-Server, sondern verteilt auf verschiedenen Nodes liegen.

Statt eines `announce` Keys mit der URL zum Tracker gibt es in der `.torrent` Datei einen `nodes` Key mit den K am nächsten gelegenen Seedern.

Die Nähe ist dabei nicht physisch sondern auf den Unterschied von den IDs zweier Nodes bezogen, der mittels einer simplen `xor` Operation ermittelt wird. Man kann also die Distanz wie folgt ganz einfach und performant bestimmen [Loewenstern, 2008]:

$$distance(A, B) = |A \oplus B|$$

4 BitTorrent im Detail: Abläufe beim File-sharing

In diesem Kapitel dreht sich alles um das senden und empfangen von Daten via BitTorrent.

Um bei BitTorrent eine Datei herunterzuladen oder selbst zu verbreiten muss man sich zunächst mit den Tracker-Server verbinden, damit dieser Kontakt zu anderen Peers herstellt. Danach kann man als Peer, Leecher oder Seeder agieren.

4.1 Verbindung zum Tracker-Server herstellen

Zunächst muss die Adresse des Tracker-Servers aus der `.torrent` Datei ausgelesen werden. Ist das geschehen, so kann man eine `HTTP GET` Anfrage an den Server stellen, wobei die Form der Anfrage in `BEP 003`[1] spezifiziert ist. Die Anfrage enthält bestimmte Keys mit Metainformationen zum Peer [Cohen, 2008]:

`info_hash`	Hashwert der `info` Sektion aus der `.torrent` Datei.
`peer_id`	Die ID des Peers. Jeder Peer erstellt einen eigenen, zufälligen SHA-1 Hashwert.
`ip`	Optional: Enthält die IP oder den DNS Namen des peers.
`port`	Der Port auf dem der peer hört. Standardmäßig 6881, falls er nicht genutzt wird, werden alle bis 6889 ausprobiert.
`uploaded`	Die Anzahl bereits hochgeladener Bytes.
`downloaded`	Die Anzahl bereits heruntergeladener Bytes.
`left`	Die Anzahl an noch herunterzuladenden Bytes. Falls ein Download fortgesetzt oder wiederholt werden muss kann diese Größe nicht aus den vorherigen Daten berechnet werden.
`event`	Optional: Gibt mit den Werten `started`, `completed` und `stopped` an ob der Download gestartet, beendet oder angehalten wurde.

Als Antwort erhält der Peer eine Liste aller Seeder, die er erreichen kann. Es ist an dieser Stelle noch nicht bekannt welcher Seeder welches Piece besitzt.

4.2 Anfordern einer Datei

Um eine Datei herunterzuladen muss sich zunächst der Peer mit dem Seeder verbinden. Dazu sendet er über das Peer-Protocol fest spezifizierte Nachrichten, sogenannte `messages`, welche alle spezifizierte Teile und Größen besitzen.

Zu Beginn wird eine `handshake` Anfrage vom Peer an den Seeder gesendet [Cohen, 2008]. Diese Anfrage stellt die Verbindung her und sichert durch das senden des SHA-1 Hashwertes vom `info` Key aus der `.torrent` Datei, dass beide Parteien auf der gleichen Datei arbeiten.

So wird danach die Nachricht `bitfield` an den Peer gesendet, welche angibt was für Teilstücke der Seeder besitzt. Danach kann mit `request` ein Teil einer Datei angefordert werden, wobei drei Werte mitgegeben werden müssen:

`index`	Der Index vom gewünschten Teilstück
`begin`	Der Startpunkt (in Bytes) innerhalb des Teilstücks. Es kann somit auch ein Teil eines Teilstücks (Block) angefordert werden.

[1] Die *BitTorrent Enhancement Proposals* (kurz *BEP*s) sind die offiziellen Dokumente der Spezifikation für Features im BitTorrent Protokoll. Jedes BEP hat eine Nummer und kann unter bittorrent.org eingesehen werden.

`length`	Die Länge (in Bytes) des Blocks

Als Antwort erhält der Peer vom Seeder eine Nachricht vom Typ `piece` (s.u.). Darüber hinaus wird mit einer `have` Nachricht ein fertig und korrekt heruntergeladenes Teilstück dem Seeder gegenüber bestätigt.

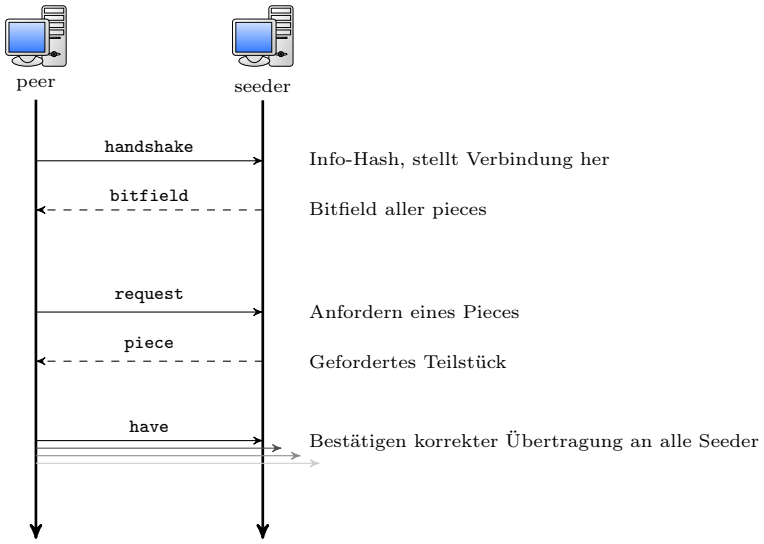

Abbildung 3. Nachrichtenaustausch beim Aufbauen der Verbindung Peer – Seeder.

4.3 Bereitstellen einer Datei

Als Seeder muss man Anfragen der Form `request` bereitstellen und korrekte Ergebnisse mittels `piece` zurückgeben können. Auch muss die Anfrage nach dem `bitfield` korrekt bearbeitet werden.

Optional kann man Peers blockieren, wieder freigeben und Übertragungen abbrechen, sobald eine `cancel` Nachricht empfangen wird.

Wird ein Teilstück mittels `request` angefordert, so muss eine Antwort als `piece` Nachricht zurückgegeben werden, welche die folgenden drei Teile beinhalten muss:

`index`	Der Index vom gewünschten Teilstück
`begin`	Der Startpunkt (in bytes) innerhalb des Teilstücks.
`block`	Die Daten aus dem geforderten Teilstück/Blocks.

Der `block` Teil der Antwort enthält die geforderten Daten und die beiden anderen Keys (`index` und `begin`) dienen dem Peer dabei dazu um die Antwort in die bisherigen Daten einordnen zu können.

5 Deploying mittels P2P: Beispiel `apt-p2p`

Besonders im Bereich Deploying (also Verbreitung von Software) lassen sich P2P Netzwerke sehr gut verwenden. Sie haben gegenüber zentralen Systemen drei sehr nützliche Eigenschaften: Hohe Ausfallsicherheit, hohe Performanz und geringe Kosten. Letzteres kommt vor allem Projekten für freie Software zugute.

Mit `apt-p2p` gibt es eine dezentralisierte Implementation des `apt` Paketmanagers [Dale and Liu, 2014]. Normalerweise wird über Mirror-Server die Software heruntergeladen, doch `apt-p2p` leitet normale `apt` Anfragen um und benutzt eine DHT mit eigenem Protokoll zum Download, Package-Lookup und Anfragen.

Tatsächlich bildet `apt-p2p` nur einen Proxy zwischen Clients und Server. Falls im Netzwerk eine angefragte Datei nicht, sehr selten oder nur über eine langsame Verbindung vorhanden sein sollte, wird diese vom Server heruntergeladen und dann über das P2P Netzwerk verbreitet. Im Folgenden wird der Fokus jedoch auf dem P2P Netzwerk, dessen DHT und das Protokoll liegen.

5.1 Warum ein eigenes Protokoll?

Man könnte zunächst auf die Idee kommen direkt BitTorrent für `apt-p2p` zu verwenden, was auch bei der Implementation berücksichtigt wurde, doch gibt es einige Nachteile im Bezug auf die Effizienz von BitTorrent für Paketmanager.

Das BitTorrent Protokoll wird nicht direkt verwendet, da zu viele nicht benötigte Informationen übertragen werden. Auch ist es nicht möglich ein Software-Paket zu aktualisieren ohne, dass jeder Peer erneut eine Menge Daten herunterladen muss.

Ein weiterer Grund ist die oftmals kleine Paketgröße im apt-Repository. Für jedes Paket ein Torrent anzulegen wäre zu ineffizient, da bei der Menge an Torrents zu viel unnötige Information übertragen würde. Die entgegengesetzte Idee wäre aus allen Paketen ein großes Torrent bilden, was jedoch zu ineffizient im verfolgen (tracken) wäre.

Die Lösung ist ein eigenes Torrent-Protokoll bei dem man kleine Dateien effizient und leicht verbreiten kann und alle Pakete ohne großen Aufwand aktualisieren kann.

5.2 Das `apt-p2p` Protokoll

Entgegen des BitTorrent Protokolls kennt das von `apt-p2p` nur Paket-Hashes, die zusammen mit den Peers, welche das Paketstück besitzen, in der DHT gespeichert werden. In der DHT wird deswegen jeweils ein *torrent-string*, der die einzelnen Piece-Hashes enthält.

Die DHT ist nach dem selben Schema wie bei BitTorrent aufgebaut und auch hier wird jeder node eine zufällige ID zugewiesen. Über eine Suche nach einem Wert (Value) zu einem Schlüssel (Key) im Netzwerk, die wie oben schon mal erwähnt über die Distanz zweier SHA-1 Hashwerte funktioniert, werden dann Peers zu gewünschten Teilstücken einer Datei ermittelt. Der Key ist dabei die ID des Peers und der Value ist der Hashwert des Paketes, welches der Peer besitzt.

Das Protokoll von `apt-p2p` erlaubt zwei wichtige Anfragen an das Netzwerk: `find_node`, was eine liste von Nodes zurück gibt die dem Key am nächsten sind, und `find_value`, was direkt den Wert zu einem Key zurück gibt.

5.3 Spezialisierung auf kleine Pakete

Entgegen der Software, die klassischer Weise über BitTorrent (oder anderen Systemen) verbreitet wird ist die Größe von Softwarepaketen im apt-Repository in den meisten Fällen sehr klein. Im Paper von Dale und Liu ([Dale and Liu, 2014]) wurde das Debian Repository von Januar 2008 untersucht und dabei festgestellt, dass die meisten Pakete (ca. 92,5%) nur maximal vier Piece-Hashes im Torrent-String benötigen. Die restlichen 1667 Pakete (der insgesamt 22298) benötigen mindestens fünf Hashwerte.

Ein Piece-Hash hat dabei den selben Aufbau und die selbe Funktion wie bei BitTorrent: Es ist ein SHA1 Hashwert auf einem Piece der Datei und dient zum Überprüfen einer korrekten Übertragung und zum ermitteln fehlender Stücke.

Um eine Fragmentierung der Pakete zu vermeiden, wird ein UDP Paket auf 1472 Bytes Payload[2] begrenzt. Diese Größe ergibt sich aus der eines UDP Paketes, welche bei 1500 Bytes liegt, minus der des UDP Headers von 20 Bytes und dem IP Header von 8 Bytes. Ist der Torrent-String kleiner als diese 1472 Bytes, kann er direkt in der DHT abgespeichert werden, da er als ganzes an die Peers gesendet werden kann. Sollte er mehr als 1472 Bytes haben (was bei Metadaten zuzüglich fünf Paket-Hashes der Fall ist), so enthält der Torrent-String einen Verweis auf einen extra Eintrag in der DHT in denen die Hashwerte stehen. Für den zweiten Fall ist dann eine weitere Anfrage an die DHT erforderlich um alle Piece-Hashes zu bekommen.

5.4 Rück- und Ausblick

BitTorrent stoßt bei sehr kleinen Dateien an seine Grenze. Eine Abwandlung und an die Vorteile von BitTorrent anknüpfendes Protokoll kann jedoch viele Vorteile schafften. Genau das haben die Entwickler von `apt-p2p` untersucht und implementiert, wodurch ein für Paketmanager geeignetes Protokoll entstanden ist, welches kleine Dateien über P2P Netzwerke effizient übertragen kann.

Statistiken mit erhobenen Daten aus dem Produktivbetrieb zeigten, dass mit `apt-p2p` Serverkosten tatsächlich gesenkt werden konnten und die Implementation gut funktioniert.

[2] Payload ist die Größe in einem Paket, die für Nutzdaten übrig bleibt. Es gilt also $Payload = Gesamtgrösse - Metadata$.

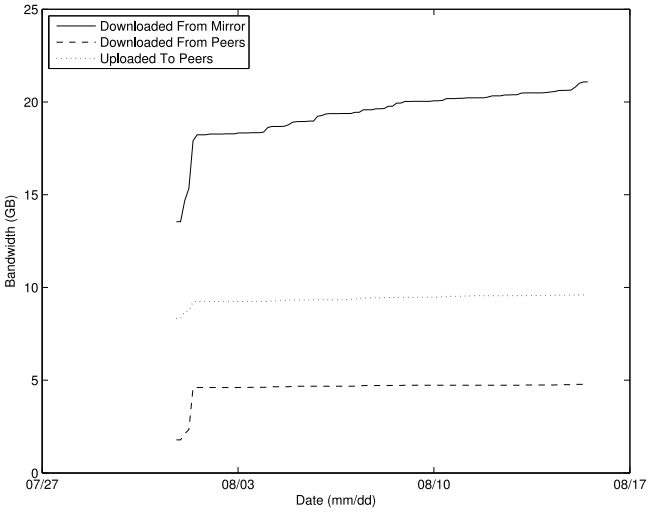

Abbildung 4. Die Verteilung von Daten, die vom Mirror und anderen Peers heruntergeladen wurde.

Um von Spenden jeglicher Art unabhängig zu sein bietet `apt-p2p` tatsächlich eine Alternative zum herkömmlichen, zentralen `apt`.

6 Issue-Tracking

Neben dem Deployment ist bereits in der Entwicklung von Software eine skalierbare und ausfallsichere Lösung für Werkzeuge wie zum Beispiel Issue-Tracking sehr von Vorteil [Agneeswaran et al., 2011]. Bisherige Lösungen wie Jira, Bugzilla oder auch Redmine sind alle zentralisiert was sowohl Kontrolle, als auch Server angeht.

Besonders in Open-Source Projekten ist der Wunsch oft groß die Art und Weise der Verbreitung von Daten selbst zu bestimmen, was mit einem verteilten Issue-Tracking System gegenüber einem zentralisierten möglich ist. Zentralisierte Lösungen sind zudem nicht in dem Maße skalierbar und fehlertolerant wie verteilte Lösungen mittels P2P.

6.1 Die Herausforderungen

Es gibt drei wesentliche Eigenschaften, die ein P2P Netzwerk für Issue-Tracking besitzen sollte.

Scalability: Wie schon der Grundgedanke eines P2P Netzwerkes ist, so sollte natürlich auch ein verteiltes Issue-Tracking System skalierbar sein. Das bedeutet für das Netzwerk, dass es in der Performanz nicht abbauen darf, wenn Speicherbedarf, Netzwerk- oder Prozessorlast ansteigt.

Semantische Heterogenität: Ein verteiltes Issue-Tracking System sollte in der Lage sein Daten verschiedener Kontexte zu speichern.

Sicherheit: Wenn Daten über das Netzwerk verteilt werden, muss sichergestellt sein, dass sie so verteilt wurden, wie es um Nutzer spezifiziert worden ist.

6.2 Framework für Overlay-Netzwerke

Mit GridVine gibt es ein Framework für Overlay-Netzwerke mit dem ein solches P2P basiertes Issue-Tracking System realisiert werden kann. Ein Oberlay-Netz bildet eine Abstraktion auf einem bestehenden Netzwerk[Ohio State University, 2006]. Das Prinzip findet gerade bei P2P Netzwerken seinen Einsatz, da hier eigene Protokolle auf bestehenden basieren. Beispielsweise benutzt BitTorrent als Protokoll standardmäßig TCP [Cohen, 2008] und mit der DHT Erweiterung UDP[Loewenstern, 2008].

Mit GridVine werden strukturierte P2P Netzwerke erstellt bei denen die Ressourcen (also in diesem Fall die Issue-Tickets) bei den Endknoten und Besitzer sowie Index des Eintrags in der DHT gespeichert werden.

Auch wenn es technisch möglich ist und viele Vorteile bietet, so hat sich ein P2P basiertes Issue-Tracking System noch nicht durchgesetzt.

7 P2P-Versionsverwaltung mit PlatinVC

PlatinVC ist ein *Versionsverwaltungssystem* (*Version Control System* oder kurz *VCS*), welches vollständig dezentral arbeitet [Mukherjee, 2011] und vereint dabei Vorteile von zentralisierten und verteilten Systemen.

Es basiert lokal auf dem Mercurial VCS und bietet darüber hinaus einen Mechanismus für ein globales Repository, welches mittels P2P verbreitet wird.

7.1 Arten von VCS

Es gibt verschiedene Arten von Versionsverwaltungen, nämlich verteilte (*dVCS* – distributed VCS), zentrale (*cVCS* – centralized VCS) und dezentrale (hier kurz *dcVCS* – decentralized VCS).

Verteilte VCS: Bei ihnen besitzt jeder, der am Repository arbeitet, eine komplette Kopie davon.

Zentrale VCS: Hier liegt alles auf einem zentralen Server, der alle nötigen Operationen unterstützt. Das hat jedoch zur Folge, dass nach den meisten Operationen der aktuelle Commit neu heruntergeladen werden muss (etwa beim wechseln in einen anderen branch).

Dezentrale VCS: Sie arbeiten komplett dezentral, was bedeutet, dass jeder Nutzer Teile des (oder gar das ganze) Repository besitzt (Eigenschaft von *dVCS*), es aber keinen zentralen Server gibt, sondern ein P2P Netzwerk.

7.2 Die Architektur

Um die Architektur von PlatinVC zu erklären wird hier dahingehend abstrahiert, dass das P2P Netzwerk das remote-Repository bildet und man dahin push und davon pull Operationen ausführt. Tatsächlich gibt es aber *keine* zentrale Instanz, es dient nur zur Veranschaulichung.

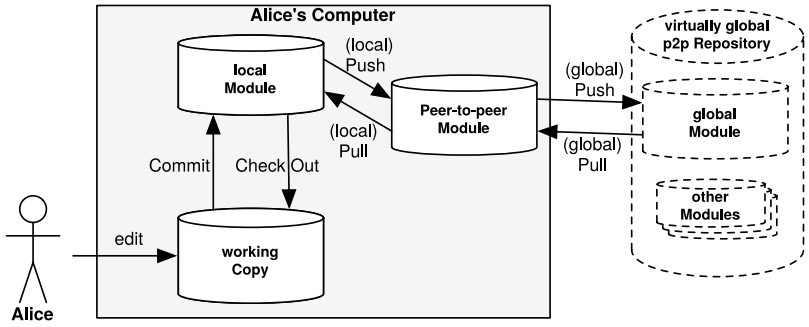

Abbildung 5. Die grundlegende Architektur von PlatinVC.

Wie bei git oder anderen dVCS besitzt Alice (s. Abbildung 5) eine komplette working-Copy auf dem lokalen Rechner. Änderungen werden von dort aus in das lokal-Module (s.u.) und danach in das – ebenfalls lokal befindliche – P2P-Module geschrieben. Das Repository mit dem sich Alice letztendlich synchronisiert ist dann das P2P Netzwerk welches hier aber *virtual global Repository* genannt wird.

Das virtual global repository: Das virtuelle, globale Repository (*virtual global Repository*) enthält alle Dateien des Projektes und ist nur ein konzeptuelles Konstrukt, da es keinen Server gibt auf dem alles liegt. Viel mehr ist das P2P Netzwerk das virtual global Repository, da der Zusammenschluss aller Rechner das gesamte Repository bildet. Gleiches gilt für das *virtual global Module*, doch dazu erst ein paar Worte zu Modulen.

7.3 Module

Um auf technischer Ebene zu verstehen wie PlatinVC funktioniert kommt man um Module nicht herum. Ein Modul ist nicht weiter als ein Projektordner, der ein bestimmtes Projekt oder bestimmte Ressourcen im Repository enthält [Mukherjee, 2011, S. 47]. Größere Projekte können aus mehreren Modulen bestehen, doch für die Versionskontrolle besteht keine Verbindung zwischen den Modulen.

Local module: Das local-Module ist der Ort zwischen der working-Copy und dem *peer-to-peer module* und ist – wie der Name schon sagt – lokal auf dem Rechner des Nutzers vorhanden. Auf dem lokalen Modul werden alle Mercurial Operationen ausgeführt, dadurch muss keine Internetverbindung bestehen um darauf arbeiten zu können.

Peer-to-peer module: Das P2P-Modul ist nichts anderes als eine Kopie vom local-Module mit der Einschränkung, dass hierauf alle PlatinVC Operationen ausgeführt werden und es sowohl vor dem Nutzer, als auch vor Mercurial versteckt ist.

Global module: Das globale Modul existiert nicht als ein reales Modul, welches auf einem der Peers gespeichert ist. Es ist viel mehr der Zusammenschluss aller existierender P2P-Module im Netzwerk. Auf Grund der Verteilung dieser existiert kein zentrales global-Module. Als Vorstellung der Struktur von PlatinVCs globalen Aktionen dient es aber sehr gut, genau wie das virtual global Repository.

7.4 Operationen

Unter dem Begriff Operation fallen Vorgänge wie etwa Push oder Pull, welche sowohl lokal, als auch global ausgeführt werden können. Lokal ausgeführte Operationen arbeiten nur auf dem lokalen- und P2P-Modul, wohingegen globale Operationen mit dem P2P- und globalen-Modul arbeiten.

Änderungen werden in PlatinVC in sogenannten *Snapshots* gespeichert, die den Zustand bestimmter Dateien zu einem bestimmten Zeitpunkt speichern [Mukherjee, 2011, S. 23].

Um die Verbreitung der Änderungen zu organisieren wird für jeden Ordner ein *Maintainer* bestimmt. Diese Idee stammt aus dem *maintainer based*-Ansatz, den PlatinVC verfolgt und sorgt dafür, dass eine zentrale Instanz für das verwalten (nicht speichern!) von Snapshots zuständig ist und sicher mit Konfliktsituationen (z.B. Merging) umgeht. Der Maintainer ist ein Peer, bei dem der Abstand seiner ID zu dem eines Ordners am geringsten ist. Wie schon bei BitTorrent bestehen auch hier IDs aus SHA-1 Hashwerten und so wird aus dem Ordnernamen und dem Pfad dahin die Ordner-ID bestimmt.

Wird eine Push-Operation mit mindestens einem neuen Snapshot ausgeführt, so ist der Maintainer der erste Peer, der den neuen Snapshot bekommt [Mukherjee, 2011, S. 116]. Alle anderen Peers wenden sich nun bei einer Pull-Operation an die DHT um die Peers zu ermitteln, die den gewünschten Snapshot gespeichert haben.

7.5 Evaluation

Nach ausgiebigen Tests wurde gezeigt, dass PlatinVC den Zielen einer höheren Skalierbarkeit, Performanz und Ausfallsicherheit gerecht wird. Schön sieht man

den Vorteil von P2P bei den globalen Operationen, denn hier sinkt die Dauer schon bei wenigen Teilnehmern [Mukherjee, 2011, S. 167]. So dauert ein Push bei 15 Teilnehmern rund neun Sekunden und bei 37 Teilnehmern nur noch drei. Mit Pull Operationen verhält es sich etwas anders, hier steigt die Dauer von zwei (15 Peers) auf drei Sekunden (37 Peers) an.

In einer zweiten Untersuchung mit anderen Testdaten zeigte sich, dass PlatinVC auch gegenüber etablierten VCS einen Geschwindigkeitsvorteil bietet (s. Abbildung 6). Was die Evaluation von Mukherjee nicht gezeigt hat ist ein sehr extremes Szenario mit mehreren Hundert oder gar tausend Nutzern. Interessant wird dies bei einem Wiki oder Open-Source Projekt (z.B. Linux mit rund 4000 Entwicklern [The Linux Foundation, 2015]).

	SVN	Pastwatch[3]	**PlatinVC**
commit / push	5,9 Sek.	3,74 Sek.	**4,8 Sek.**
update / pull	21,4 Sek.	2,7 Sek.	**1 Sek.**

Abbildung 6. PlatinVC ist teilweise schneller als etablierte VCS.

Tatsächlich aber wird PlatinVC im Produktivbetrieb nicht eingesetzt. Auf seiner Website schrieb Mukherjee, dass er PlatinVC bald veröffentlichen wollte, doch zuerst noch einmal über den Code schauen möchte [Mukherjee, 2012]. Das war jedoch 2013, PlatinVC ist bis heute (Oktober 2016) nicht veröffentlicht worden.

8 Abschluss

Zu Beginn dieser Arbeit stand die Frage im Raum wie P2P in der verteilten Softwareentwicklung eingesetzt werden kann. Um jedoch zu verstehen was P2P ist und wie es funktioniert wurde anhand von BitTorrent ein verbreitetes Protokoll detailliert betrachtet. Der Einsatz in der verteilten Softwareentwicklung wurde dann an drei Projekten untersucht, apt-p2p, GridVine und PlatinVC.

Apt-p2p bildet einen P2P basierten Paketmanager und verteilt somit Software über ein skalierbares und ausfallsicheres Netzwerk. Nach Untersuchungen ergab sich, dass viele Softwarepakete im konventionellen (und zentralisierten) apt Paketmanager sehr klein sind – zu klein um sie mit BitTorrent effizient zu verschicken. Um effizient mit diesen kleinen Dateien umgehen zu können wurde apt-p2p als eine Abwandlung des BitTorrent Protokolls entwickelt. Es erfüllt dabei die gegebenen Anforderungen und bewies sich im Produktivbetrieb.

Das GridVine Framework bietet eine Möglichkeit für ein Overlay-Netzwerk mit dem ein P2P basiertes Issue-Tracking System realisiert werden kann. Leider hat sich ein solches System bisher nicht durchsetzen können.

[3] Pastwatch ist ein anderen P2P-basiertes VCS [Yip et al., 2006].

Bei PlatinVC sieht es ähnlich aus: Hier wurde zwar ein funktionierender Prototyp vorgestellt, doch wurde er bis heute nicht veröffentlicht und somit auch nicht eingesetzt. Trotzdem zeigte PaltinVC sehr schön, dass eine komplett dezentrale Versionsverwaltung durchaus Sinn ergibt und teilweise schneller ist als herkömmliche Lösungen. Benutzt wurde dabei nicht BitTorrent, sondern ein teils eigenes auf einer DHT basiertes Protokoll.

Insgesamt bietet P2P viele Möglichkeiten für die verteilte Softwareentwicklung, doch auch wenn konventionelle Lösungen (z.B. Bugzilla, github oder apt) teurer und unzuverlässiger sind, hat sich P2P in der verteilten Softwareentwicklung noch nicht zur Gänze durchgesetzt.

Literatur

Agneeswaran et al., 2011. Agneeswaran, V. S., Narendula, R., and Aberer, K. (2011). Peer-to-peer issue tracking system: Challenges and solutions. https://infoscience.epfl.ch/record/116279/files/p2pIssueTracking.pdf.

Blizzard, 2014. Blizzard (2014). Blizzard downloader – häufige fehler und probleme. https://eu.battle.net/support/de/article/blizzard-downloader-hufige-fehler-und-probleme. last accessed: 12.06.2016.

Cisco, 2015. Cisco (2015). The zettabyte era—trends and analysis. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html. last accessed: 12.06.2016.

Cohen, 2008. Cohen, B. (2008). The bittorrent protocol specification. last accessed: 11.06.2016.

Dale and Liu, 2014. Dale, C. and Liu, J. (2014). apt-p2p: A peer-to-peer distribution system for software package releases and updates.

Fabelje et al., 2016. Fabelje, H., Heift, G., and Otto, D. C. (2016). http://ftp.halifax.rwth-aachen.de. http://ftp.halifax.rwth-aachen.de. last accessed: 12.06.2016.

Loewenstern, 2008. Loewenstern, A. (2008). Dht protocol. last accessed: 11.06.2016.

Mukherjee, 2011. Mukherjee, P. (2011). A fully decentralized, peer-to-peer based version control system. http://tuprints.ulb.tu-darmstadt.de/2488/. last accessed: 14.08.2016.

Mukherjee, 2012. Mukherjee, P. (2012). Platinvc. last accessed: 28.08.2016.

Nadeo, 2011. Nadeo (2011). Maniaplanet wiki - configure. http://wiki.maniaplanet.com/en/Configure. last accessed: 12.06.2016.

Ohio State University, 2006. Ohio State University (2006). System/application designs, optimizations and implementations on overlay networks. http://web.cse.ohio-state.edu/hpcs/WWW/HTML/internet-p2p.html. last accessed: 14.08.2016.

p2pfoundation, 2005. p2pfoundation (2005). P2p tv. http://p2pfoundation.net/P2P_TV#The_economics_of_netcasting. last accessed: 12.06.2016.

Schmidt, 2014. Schmidt, J. (2014). Kommentar: Truecrypt ist unsicher - und jetzt? last accessed: 29.08.2016.

Shen et al., 2009. Shen, X., Yu, H., Buford, J., and Akon, M. (2009). *Handbook of Peer-to-Peer Networking*. Springer Publishing Company, Incorporated, 1st edition.

The Linux Foundation, 2015. The Linux Foundation (2015). The linux foundation releases linux development report. last accessed: 28.08.2016.

various, 2008. various (2008). Blizzard downloader. http://wow.gamepedia.com/Blizzard_Downloader. last accessed: 12.06.2016.

Yip et al., 2006. Yip, A., Chen, B., and Morris, R. (2006). Pastwatch: a distributed version control system. last accessed: 28.08.2016.