Programmierung eines

Pseudo-Torrentnetzwerkes

Rechnerkommunikation und Middleware

Entwurfsarbeit der Studiengänge I-16/ I-17

Fachbereich Automatisierung und Informatik

Hochschule Harz

**Autoren:** Dirk Neumann (24160), Dominik Viererbe (25401)

**Dozent:** Prof. Dr. O. Drögehorn

**Veranstaltung/Semester:** Rechnerkommunikation und Middleware  
Wintersemester 2019/2020

I Inhaltsverzeichnis

[1 Einleitung 4](#_Toc33741045)

[2 Aufgabenstellung 4](#_Toc33741046)

[3 Architekturentwurf 6](#_Toc33741047)

[3.1 Logische Architektur 6](#_Toc33741048)

[3.2 Technische Architektur 8](#_Toc33741049)

[4 Implementierung 9](#_Toc33741050)

[4.1 Softwarekomponenten 9](#_Toc33741051)

[4.1.1 Client 9](#_Toc33741052)

[4.1.2 Trackerserver 11](#_Toc33741053)

[4.1.3 Torrentserver 12](#_Toc33741054)

[4.2 Schnittstellen 12](#_Toc33741055)

[4.2.1 gRPC 13](#_Toc33741056)

[4.2.2 MQTT 13](#_Toc33741057)

[5 Diskussion der Ergebnisse 13](#_Toc33741058)

[6 Fazit 13](#_Toc33741059)

[7 Ausblick 14](#_Toc33741060)

[8 Quellen 16](#_Toc33741061)

[8.1 Literaturquellen 16](#_Toc33741062)

[8.2 Bildquellen 16](#_Toc33741063)

[9 Anhang 16](#_Toc33741064)

[9.1 Glossar 16](#_Toc33741065)

[9.2 Verwendete Programme 16](#_Toc33741066)

[9.3 Projektdateien 16](#_Toc33741067)

II Abbildungsverzeichnis

[Figure 1: Logische Architektur des PTN 7](#_Toc33741041)

[Figure 2: Technische Architektur des PTN 8](#_Toc33741042)

[Figure 3: Hochladen einer Datei in das PTN 10](#_Toc33741043)

[Figure 4: Herunterladen einer Datei aus dem PTN 10](#_Toc33741044)

IV Abkürzungsverzeichnis

|  |  |
| --- | --- |
| Abkürzung | Bedeutung |
| PTN | Pseudo-Torrent-Netzwerk |
| CORBA | Common Object Request Broker Architecture |
| RMI | Remote Method Invokation |
| TCP | Transmission Control Protocol |
| IP | Internet Protocol |
| HTTP | Hyper Text Transfer Protocol |
| MQTT | Message Queuing Telemetry Transport |
|  |  |

V Versionsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Autor | Version | Datum | Änderungen |
| Dirk Neumann | 0.1 | 12.02.2020 | Dokument erstellt |
| Dirk Neumann | 0.2 | 14.02.2020 | Kapitel 1 + 2 |
| Dirk Neumann | 0.3 | 15.02.2020 | Kapitel 3 + 4 + 5  Kapitel 4 nur angelegt |
| Dirk Neumann | 0.4 | 16.02.2020 | Kapitel 4 Aufgliederung  Kapitel 5 + 6 angelegt |
| Dirk Neumann | 0.5 | 16.02.2020 | Kapitel 4.1 |
| Dirk Neumann | 0.6 | 22.02.2020 | Kapitel 5 + 6 geschrieben  Verzeichnisse angelegt |
| Dirk Neumann + Dominik Viererbe | 0.7 | 25.02.2020 | Kapitel 7 + 8 + 9 angelegt und teilweise befüllt  Kontrolllesung + Fehlerkorrektur + sprachliche Überarbeitungen |
| Dirk Neumann + Dominik Viererbe | 0.8 | 27.02.2020 | Kapitel 4.2 geschrieben |
| Dirk Neumann | 1.0 |  | Finale Abgabefassung |

# Einleitung

Diese Entwurfsarbeit wird im Zuge der Lehrveranstaltung „Rechnerkommunikation und Middleware“ durchgeführt. Die Studierenden sollen strukturierte Entwicklungsarbeit, mit dem Fokus auf Implementierung und Dokumentation, anhand einer selbst gewählten Aufgabenstellung erlernen und vertiefen.

Mit zunehmendem Einzug des Internets in alle Gesellschafsbereiche und einer steigenden Vernetzung von Computern und Netzwerken, hat sich der Umgang mit Daten stark verändert. Dies ist nicht zuletzt auch dem Fortschreiten von Datenanalysemethoden geschuldet welche den Nutzen und das Gefahrenpotential von Daten stark erhöht haben. Dementgegen standen und stehen immer neu Entwicklungen im Bereich von Datenbanken, Sicherheitsmechanismen und Softwareparadigmen wie z.B. Webservices oder Cloud Computing. Eine der inzwischen nicht mehr neusten aber noch immer weit verbreiteten Technologien ist das BitTorrent-Protokoll, welcher das dezentralisierte und redundante sowie, im Zusammenhang mit verschiedenen Sicherheitsmaßnahmen, sichere Abspeichern und -abrufen von Daten ermöglicht. Der Aspekt der Sicherheit ist dabei je nach Implementierung durch andere Sicherheitsziele wie z.B. Integrität, Verschlüsselung, Authentizität oder Anonymität geprägt.

# Aufgabenstellung

Die Datenhaltung wie sie in einem solchen Rechnernetzwerk mit BitTorrent-Protokoll realisiert wird, ist eine praktische Anwendung der in der Vorlesung „Rechnerkommunikation und Middleware“ vermittelten Inhalte, darunter Datenserialisierung und –deserialisierung, Rechnerkommunikation und Kommunikationsprotokolle. Diese Inhalte sollen an einem praktischen Beispiel erlernt werden, indem ein ähnliches Rechnernetz aufgebaut wird. Dazu sollen die entsprechend notwendigen Komponenten identifiziert, beschrieben und implementiert werden und als Gesamtsystem getestet werden. Im Kontext der Lehrveranstaltung wird der zu realisierende Umfang des Projekts wie folgt definiert:

* Erstellen Sie ein Torrentnetzwerk mit Java und C#, die via CORBA kommunizieren.
* Das Netzwerk besteht aus einem Client, einem Trackerserver und mehreren Torrentservern.
* Das Netzwerk soll es ermöglichen Dateien zentral hochzuladen und dezentral herunterzuladen.

Client:

* Der Nutzer besitzt einen Client. An diesem kann er eine Datei hochladen, die dann an den Trackerserver gesendet wird.
* Der Nutzer kann mit Hilfe des Clients den Datenbestand am Trackerserver anfragen und zu einer gewünschten Datei die Torrentserver ausfindig machen, welche die Fragmente der gewünschten Datei gespeichert haben.
* Der Client lädt diese Fragmente von den Torrentserver herunter und setzt diese dann selbstständig zur Zieldatei zusammen, welche der Nutzer unter einem gewünschten Pfad abgelegen kann.

Trackerserver:

* Der Trackerserver nimmt Dateien vom Client an und teilt diese in mehrere Fragmente fest konfigurierter Größe auf.
* Diese werden vom Trackerserver publiziert woraufhin sich die Torrentserver diese Dateifragmente selbstständig herunterladen.
* Der Trackerserver überwacht zentral über MQTT welcher Torrentserver welche Fragmente besitzt.
* Fragt ein Nutzer eine Datei an, so stellt der Trackerserver die nötigen Informationen zur Verfügung, um die ursprüngliche Datei wieder zusammenzusetzen.

Torrentserver

* Der Torrentserver abonniert die Informationskanäle, welche der Trackerserver für diese zur Verfügung stellt.
* Werden neue Datenfragment vom Trackerserver ausgeschrieben, so entscheiden die Torrentserver nach einem zu definierenden Protokoll, ob sie dieses Fragment herunterladen und abspeichern.
* Der Trackerserver vermerkt diese Entscheidung.
* Auf Anfrage stellt der Torrentserver die geforderten Datenfragmente zur Verfügung.

Die Abgabe enthält:

* Quellcode mit Quellcodedokumentation
* Gesamtdokumentation
* UML und andere Diagramme
* Entwurfsentscheidungen
* Arbeitsaufteilung aus welcher Bearbeitungsverantwortlichkeiten hervorgehen

Wie aus der Aufgabenstellung entnommen werden kann, handelt es sich bei dem Projekt nur um ein BitTorrent ähnliches Netzwerk. Dieses Pseudotorrentnetzwerk (PTN) besitzt einige, aber nicht alle Merkmale des BitTorrents, was das Projekt zum einen etwas leichter und damit innerhalb des Zeitrahmens realisierbar macht und zum anderen die Möglichkeit mit sich bringt Technologie wie MQTT zu verwenden, welche ebenfalls Vorlesungsinhalt war.

Modifikation der Aufgabenstellung:

Während der Konzeptions- und Technologierecherchephase des Projekts wurden einige Entscheidungen die zu Beginn bei der Formulierung des Projektziels getroffen wurden revidiert. Daraus ergaben sie sehr früh schon Änderungen an der Aufgabenstellung, welche im Folgenden kurz erläutert werden sollen. Die Aufgabenstellung wurde in Abstimmung mit dem betreuenden Professor im Nachhinein angepasst. Sowohl die Ausgangsaufgabenstellung als auch die überarbeitete Fassung sind unter den Dokumentationsartefakten im git repository zu diesem Projekt zu finden.

Die Anforderung, dass innerhalb des PTN mit CORBA kommuniziert werden soll wurde verworfen und durch die Anforderung ersetzt die Kommunikation stattdessen mit gRPC zu realisieren. Diese Entscheidung wurde erst später getroffen, bei Beginn der Implementierungsphase. Dies geschah, da die Schnittstellen als erstes implementiert werden sollten und sich bereits hier Probleme im Zusammenhang mit C# zeigten. Die aktuellsten Blogeinträge zu „CORBA in C#“ waren auf die Jahre 2006-2008 datiert, was ein Argument dafür ist eine andere Technologie zu wählen, denn Aktualität von Dokumentationen und Aktivität in der Community zu einem Thema können leicht zu einem Problem werden, wenn mitten in der Entwicklungsphase ein Problem auftritt, dass nicht gelöst werden kann. Aus diesem Grund, und dem Wissen über die Technologie gRPC wurde stattdessen gRPC als Technologie für die Kommunikation gewählt.

Eine weitere Änderung betrifft das Einbinden des Trackerservers beim Herunterladen von Daten. Hier war zu Beginn gefordert, dass notwendige Informationen für das Herunterladen vom Nutzer beim Trackerserver angefordert werden. Dies würde den Trackerserver, der nur einmal je PTN existiert nicht nur beim Hochladen, sondern auch beim Herunterladen von Daten zum „single point of failure“ machen. Um dies zu vermeiden. Kann der Nutzer nun jeden beliebigen Torrentserver beim Download ansprechen.

Eine weitere Änderung ist das Zusammensetzen der Datei erst beim Nutzer und nicht schon vorher beim Torrentserver. So kann zusätzliche Kommunikation innerhalb des PTN verhindert werden und der Nutzer weiß, dass niemand der Torrentserver bei dem er das Herunterladen der Datei initiiert hat niemals die komplette Datei bei sich hatte, was gut für den Aspekt Vertraulichkeit im PTN ist.

In diesem Zusammenhang wurde ebenfalls geändert, dass der Nutzer, bzw. der Client die Datei nicht nur beim Herunterladen selbst zusammensetzt, sondern sie auch vor dem Hochladen in das PTN selbst zerteilt. Das hat erneut einen positiven Effekt auf die Vertraulichkeit und zu anderen wird damit der Trackerserver stark entlastet. Müsste dieser für hunderte oder tausende Nutzer die Dateifragmente erzeugen und wieder zusammensetzen wäre dies ein sehr hoher Aufwand und in einer Rechnerarchitektur schwerer umzusetzen als hunderte/ tausende Nutzer-PCs diesen Vorgang jeden für sich bearbeiten zu lassen.

Nach erfolgreichem Anlegen einer finalen Aufgabenstellung konnte daraus ein lauffähiger Prototyp implementiert werden. Hier ist jedoch zu beachten, dass es sich dabei wie beschrieben nur um einen Prototyp handelt und dieser nicht eine komplette BitTorrent-Nachimplementierung darstellt.

# Architekturentwurf

Im Folgenden wird die Architektur, des zu implementierenden Gesamtsystems beschrieben. Dazu wird erst eine Unterteilung in verschiedene Aufgaben vorgenommen, welche dann im nächsten Schritt umgesetzt werden sollen.

## Logische Architektur

Der Nutzer kann mittels eines grafischen Interfaces mit dem PTN interagierten. Dieses wird vom PTN zur Verfügung gestellt und ist Teil des Projekts, gehört logisch aber nicht zum PTN. Der Nutzer kann über dieses Interface, wie in Abbildung 4 zu sehen, Daten in das PTN hochladen und Daten die er zuvor hochgeladen hat, oder auf die er Zugriff hat herunterlade. Das PTN verwaltet sich intern selbst. Es regelt welcher Teil der Daten wo abgespeichert wird und wie und ob ein Nutzer auf Daten zugreifen darf.

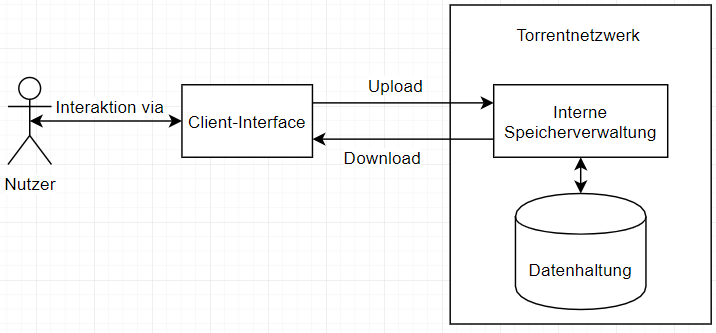


Figure 1: Logische Architektur des PTN

## Technische Architektur

Lädt ein Nutzer eine Datei in das PTN hoch, wird dieser vom Nutzerinterface zunächst in einzelne Datenpakete zerlegt, welche dann nach und nach hochgeladen werden. Der in Abbildung 4 als „Interne Speicherverwaltung“ bezeichnete Vorgang wird in Abbildung 5 weiter verfeinert. Beim Daten-in-das- PTN -Laden wird über einen Trackerserver organisiert, der die einzelnen Datenpakete ausschreibt. Die im Netzwerk befindlichen Torrentserver entscheiden ob sie das ausgeschriebene Datenpaket annehmen wollen oder nicht. Dies entscheidet der Torrentserver anhand der aktuellen Verteilung von Daten im Netzwerk. Möchte ein Nutzer eine Datei aus dem PTN herunterladen, so kann er sich an jeden beliebigen Torrentserver wenden, welcher dann die für die angeforderte Datei relevanten Datenpakete im PTN lokalisiert und diese Informationen zur Verfügung stellt. Im Anschluss lädt das Nutzer-Interface die restlichen benötigten Datenpakete von den entsprechenden Torrentservern herunter und fügt die Datei wieder zusammen. Der Nutzer kann dann die wieder zusammengefügte Datei abspeichern.

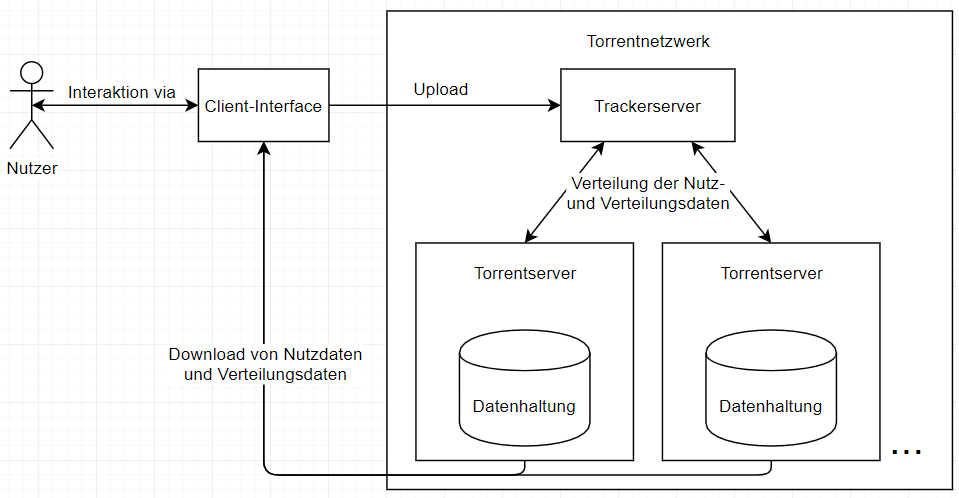


Figure 2: Technische Architektur des PTN

# Implementierung

Die Implementierung des Systems kann in zwei Teilbereiche untergliedert werden. Zum einen werden die Softwarekomponenten untersucht, welche implementiert wurden um die in Abbildung 2 gezeigte Architektur umzusetzen. Zum anderen sollen die Schnittstellen zwischen diesen Komponenten genauer betrachtet werden. Diese sind insofern interessant, da sie eigens für das PTN entworfen wurden und die dabei verwendeten Technologien erläutert werden sollen. Dieses Kapitel soll nur einen groben Überblick über die Implementation des PTN geben und die Ideen und Konzepte die dabei verwendet wurden. Zu jedem der Unterkapitel gibt es weitere Dokumentationsartefakte, wie zum Beispiel Codedokumentationen für die Softwarekomponenten und Sequenzdiagramme für die Schnittstellenbeschreibungen. Diese sind, wie auch der Quellcode online in einem github repository[3] zu finden.

## Softwarekomponenten

Die Softwarekomponenten kapseln ihre jeweilige Funktionalität und stellen diese über öffentlich bekannte Schnittstellen für andere Teilnehmer im PTN bereit. Ein Teilnehmer am PTN kann sowohl die Rolle eines Clients als auch die Rolle eines Torrentservers einnehmen, was den Grad an potentieller Dezentralisierung erhöht. Alternativ kann die Dezentralisierung auch gewährleistet werden, wenn ein Betreiber eines PTN mehrere Rechner seines Rechenclusters zur Verfügung stellt. Hier entspricht der Grad der Dezentralisierung allerdings eher einer Sicherheitsmaßnahme um zum Beispiel Redundanz und Integrität zu gewährleisten und die Verfügbarkeit des Systems über mehrere Endpunkte zu erhöhen. Durch den Dualismus zwischen Client und Torrentserver ist zusätzlich eine Teilung des Datenbesitzes möglich. Datenbesitz ist hier nicht im Sinne des rechtlichen Eigentümers der Daten zu interpretieren, sondern im Sinne der Verfügungsgewalt über die Daten. Im Folgenden wird für jede Softwarekomponente des PTN ein kurzer Überblick über Aufbau, Funktionalität und Ziele gegeben.

### Client

Der Client ist eine grafische Schnittstelle zwischen dem Nutzer des PTN und dem PTN selbst. Es ermöglicht ihm verschieden Aktionen auszuführen und dabei grafisch Feedback zu erhalten. Dies senkt die Barriere zwischen Nutzer und Software, welche zum Beispiel eine reine Konsolenanwendung mit sich bringen würde und erhöht zudem den Abstraktionsgrad der eigentlichen Funktionalität. Ein Nutzer muss sich so keine Gedanken darum machen, wie Algorithmen zur Dateidistribution parametrisiert sein müssen bzw. welche Parameter es dabei überhaupt gibt. Im Rahmen des Projekts ist der Client als WPF-Anwendung implementiert worden und damit an ein Windowssystem gebunden. Als weitere Einschränkung ist der Client im Rahmen der Entwurfsarbeit nur dafür ausgelegt Bilddateien im PTN zu speichern. Diese werden dem Nutzer in der grafischen Oberfläche Angezeigt um das Arbeiten mit dem PTN nutzerfreundlicher zu gestalten und. Zudem erleichterte dies die Kontrolle der korrekten Übertragung während der Test, da hier Fehler leichter zu erkennen sind als bei großen Textdateien.

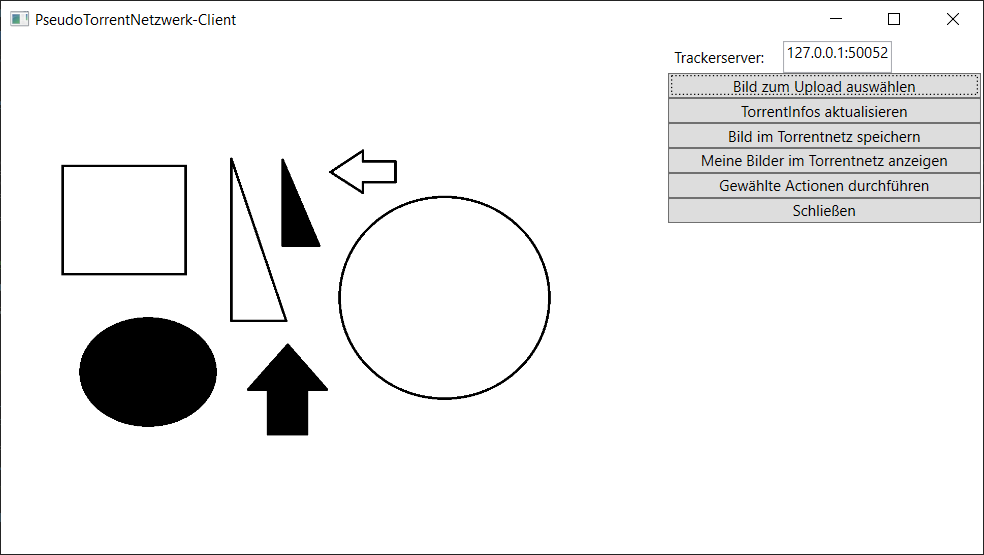


Figure 3: Hochladen einer Datei in das PTN

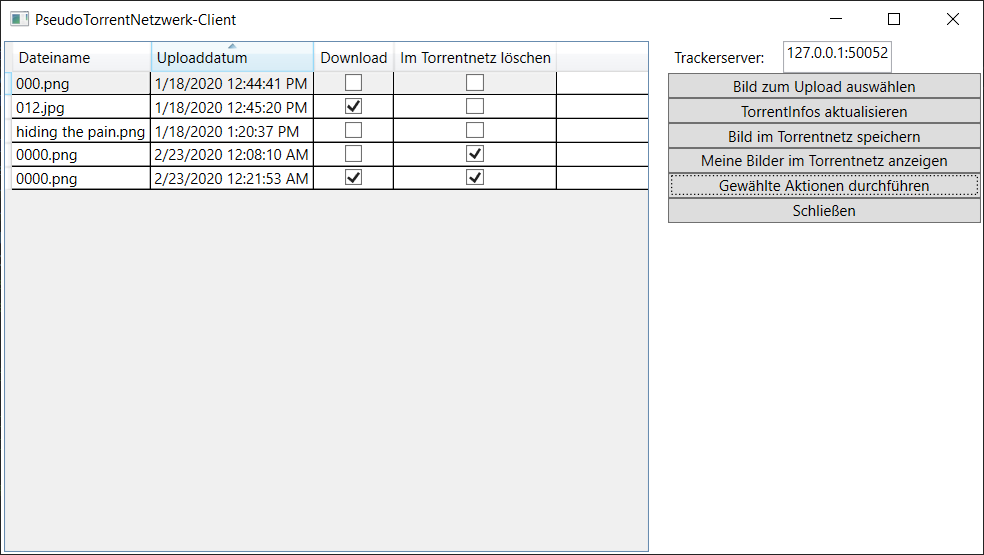


Figure 4: Herunterladen einer Datei aus dem PTN

Startet der Nutzer den Client kann er aus verschiedenen Aktionen auswählen. Die Use Cases zu diesen Aktionen sind wie auch andere weitere Dokumentationsartefakte im github repository[XXX] im Ordner „Documentation“ zu finden. An dieser Stelle soll nur auf die Oberfläche eingegangen werden und kurz erläutert werden wie sich der Client für den Nutzer sichtbar verhält und inwiefern der Client mit dem Dateisystem und den anderen Softwarekomponenten interagiert.

Nach dem Start hat der Nutzer zwei Optionen die die grundlegenden Ziele des PTN widerspiegeln. Er kann eine Datei über den Button „Bild zu Upload auswählen“ selektieren und diese im Client öffnen. Dies entspricht der Situation in Abbildung 3. Um die Datei nun in das PTN hochladen zu können, muss sich der Client mit dem Trackerserver verbinden. Dazu muss er die IP-Adresse des Trackerservers und den Port auf den dieser hört kennen. Dies kann über die TextBox über den Buttons auf der rechten Seite konfiguriert werden. Zudem sollte der Client immer die Konfiguration des PTN kennen. Diese umfasst den verwendeten Prüfsummenalgorithmus, die Größe der im PTN gespeicherten Fragmente und eine Liste der im Netzwerk aktiven Torrentservern. Für den Upload sind nur die ersten zwei Werte notwendig, eine Konfiguration wird jedoch immer vollständig heruntergeladen. Die Konfiguration kann nur vom Trackerserver heruntergeladen werden und wird die automatisch, sofern der Client keine Konfiguration lokal gespeichert hat. Um die Konfiguration zu aktualisieren, kann der Nutzer den Button „Torrentinfos aktualisieren“ nutzen. Diese Aktion ist in der Regel jedoch nicht notwendig, da ein aktives PTN seinen Prüfsummenalgorithmus und seine Paketgröße nicht ohne weiteres umstellen kann, da andernfalls bereits gespeicherte Fragment ihre Validität verlieren würden. Wählt der Nutzer nun die Schaltfläche „Bild im Torrentnetz speichern“ aus, wird das gewählte Bild in Fragmente aufgeteilt und diese an den Trackerserver gesendet. Die Informationen welche Dateien bereits im PTN gespeichert wurden, wird lokal beim Nutzer gespeichert. Diese Datei enthält zu jeder Datei die Angaben Dateiname, Dateiprüfsumme, Dateigröße und einen Zeitstempel, wann diese Datei im PTN gespeichert wurde.

Möchte der Nutzer dagegen eine bereits hochgeladene Datei wieder herunterladen, so kann er dies tun, in dem er sich die lokal gespeicherten Dateiinformationen in das Clientprogramm lädt. Dort kann der Nutzer, wie in Abbildung 4 zu sehen, die Dateien nach ihren Attributen sortieren und zu jeder Datei auswählen, ob er diese herunterladen möchte und/ oder die Datei aus dem PTN gelöscht werden soll. Nachdem der Nutzer seine Entscheidung getroffen hat, kann er diese mit dem Button „Gewählte Aktionen durchführen“ umsetzen. Der Client fragt daraufhin für jede herunterzuladende Datei einen zufälligen Torrentserver aus seiner Netzwerkkonfiguration, wo die Fragmente zu dieser Datei liegen. Nachdem diese Information übermittelt wurde kann der Client die Fragmente an den jeweiligen Torrentservern im PTN anfragen, herunterladen, die Fragmente lokal zusammenfügen und die rekonstruierte Datei lokal abspeichern. Sollte die Datei, beziehungsweise eine Datei mit identischem Namen, bereits vorhanden sein, so wird der Nutzer vorher darüber informiert und gefragt, ob er die lokale Datei überschreiben möchte oder nicht.

Sollte ein Fehler innerhalb des Programms auftreten so wird dieser in einem Fenster ausgegeben. Bei einem auslieferbaren System würde man diesen Fehler nicht ausgeben aber im Kontext der Entwurfsarbeit ermöglicht dies eine bessere Fehlerkorrektur und -behebung.

### Trackerserver

Der Trackerserver ist die zentrale Verwaltungskomponente des PTN und damit der Grund warum es sich nicht um ein vollwertiges Torrentnetzwerk, wie zum Beispiel bei BitTorrent[XXX] handelt. Vollwertig meint in diesem Zusammenhang, dass sowohl das Hoch- als auch das Herunterladen von Dateien dezentral realisiert sein soll und es innerhalb des Torrentnetzes nur gleichwertige Torrentserver gibt. Damit gäbe es in diesem Fall keinen einzelnen Punkt an dem das System funktionsunfähig gemacht werden kann, englisch „single point of failure“. Mit der Entscheidung einen Instanz Trackerserver zu konzipieren wird dieses Konzept außer Kraft gesetzt.

Der Trackerserver hat die Aufgabe das Hochladen von Fragmenten, das Verteilen von Fragmenten und die Weitergabe von Verteilungsinformationen zu zentralisieren. Das hat den Vorteil, dass es im Netzwerk immer eine Komponente weiß wie das Netzwerk aktuell aussieht es nicht zu Konflikten zwischen verschiedenen aktuellen Ständen kommen kann. Zudem müssen Informationen über die Fragmentverteilung immer nur über einen Kommunikationskanal an die einen Torrentserver, nämlich vom Trackerserver aus, geschickt werden, was bei einem Netzwerk mit einem Trackerserver und n Torrentservern genau n Kommunikationswege impliziert. Würde jeder Torrentserver mit jedem anderen kommunizieren stiege die Anzahl der Kommunikationskanäle quadratisch mit der Anzahl der Torrentserver an.

Möchte ein Nutzer eine Datei hochladen, so empfängt der Trackerserver von dieser Datei eine Prüfsumme über die gesamte Datei, die Größe der Datei und im Anschluss die Dateifragmente. Dateiprüfsumme und Dateigröße werden in einer Liste hinterlegt, die diese Informationen von allen im Netzwerk gespeicherten Dateien beinhaltet. Diese Liste wird bei Änderungen an alle Torrentserver im PTN geschickt. Die Dateifragmente werden über einen MQTT-Broker ausgeschrieben. Die Torrentserver bewerben sich beim Trackerserver um dieses Fragment, wenn sie es abspeichern wollen und der Trackerserver entscheidet welcher Torrentserver das Fragment bekommt. Wie genau dieses Kommunikationsprotokoll abläuft, kann in den Sequenzdiagrammen im Dokumentationsordner nachgelesen werden der in dem zu diesem Projekt zugehörigen github repository[XXX] zu finden ist.

### Torrentserver

Der Torrentserver ist die Softwarekomponente die für das Herunterladen von Dateien verantwortlich ist und die Datenfragmente abspeichert. Prinzipiell ist ein Torrentserver pro PTN ausreichend, es sollte in jedem PTN aber mehrere Torrentserver geben. Gäbe es nur einen, so wären alle Dateien in einzelnen Stücken auf einem Server gespeichert, was das Prinzip der dezentralen Datenhaltung nicht realisieren würde. Ein Torrentserver kann auf vier Ereignisse reagieren.

Zum einen kann der Trackerserver ein neues Datenfragment ausschreiben, welches der Torrentserver dann möglicherweise annimmt. Dies hängt von dem Verhältnis ab zwischen der Anzahl der Fragmente, die der Torrentserver aktuell abgespeichert hat und der durchschnittlichen Anzahl von Fragmenten die ein Torrentserver im PTN abgespeichert hat.

Ein zweiter Fall kann eintreten, wenn der Trackerserver eine neue Systeminformationsnachricht schickt. Der Torrentserver bekommt dann eine Informationsnachricht, die ihm die Liste aller aktuell im PTN gespeicherten Fragmente und der jeweiligen Torrentserver die dieses Fragment besitzen. Diese Liste speichert der Torrentserver lokal ab.

Sie wird gebraucht falls der dritte Fall eintritt und ein Nutzer bei ihm eine Datei anfragt. Da jeder Torrentserver die gleiche Liste besitzt, ist es egal bei welchem Torrentserver ein Nutzer anfragt. Das macht sorgt für die Dezentralität beim Herunterladen. Der Nutzer gibt an, welche Datei er herunterladen möchte, in dem er die Dateiprüfsumme mit der Anfrage schickt. Der Torrentserver kann nun aus seiner Liste alle Fragmentinformationen heraussuchen die der Nutzer dafür braucht. Er sendet ihm eine Liste zu, die eine nach Reihenfolge-zum-Zusammensetzen-sortierte Liste von Fragmentprüfsummen beinhaltet. Zu jede dieser Prüfsummen ist zusätzlich eine Liste mit den Torrentservern hinterlegt, bei denen dieses Fragment angefragt werden kann. Mit dieser Liste kann der Nutzer für jedes Fragment bei den Torrentservern das Fragment anfragen.

Das ist das vierte Ereignis bei dem der Torrentserver involviert ist. Wenn der Nutzer ein bestimmtes Fragment anfragt, sendet es der Torrentserver als Antwort zurück. Eine Prüfung ob der Nutzer autorisiert ist dieses Paket herunterzuladen gibt es nicht.

## Schnittstellen

Schnittstellen können als Vertrag zwischen den Komponenten und deren interner Funktionalität angesehen werden. Sie garantieren, dass ein Nutzer oder ein anderes Programm eine vorher definierte Reaktion erwarten kann, wenn er eine in der Schnittstelle definierte Anfrage stellt. Darauf basierend wurden für das PTN mehrere Schnittstellen konzipiert und implementiert. Diese sind genauer in den Sequenzdiagrammen der Dokumentationsunterlagen im github repository[XXX] zu diesem Projekt zu finden. In den folgenden Unterkapiteln soll auf die jeweiligen Technologien näher eingegangen werden und wie diese die Kommunikation innerhalb des PTN beeinflussen.

### gRPC

gRPC steht für „google Remote Procedure Calls“ und ist ein von Google in Kooperation mit anderen Beteiligten entwickeltes RPC-Framework das besonderes Augenmerk auf Plattformunabhängigkeit legt. Es wird als sehr leichtgewichtig, gut skalierbar und sprach- und plattformunabhängig beschrieben [XXX]. Dies sind auch für viele andere RPC-Frameworks wie z.B. CORBA und RMI die wesentliche Zielstellung der Technologie. Die Unterschiede zwischen CORBA und gRPC liegen vor allem in zwei Bereichen. Der eine Bereich sind die verwendeten Transportprotokolle, die die Technologien verwenden. Während CORBA auf einem TCP/IP-artigen Protokoll aufbaut, setzt gRPC auf HTTP/2 auf. Als ein weiterer Unterschied kann der Umgang mit Objektreferenzen gesehen werden. Bei CORBA können Referenzen auf Remote-Objekte als Parameter in Funktionen verwendet werden. Diese werden dann für einen Funktionsaufruf ausgewertet. In gRPC gibt es keine solchen Remote-Objektreferenzen, hier sind alle Parameter für Funktionsaufrufe in und außerhalb der Schnittstellenfunktionen Datenstrukturen, keine Objekte. Entsprechend ist in gRPC keine Vererbung innerhalb von definierten Schnittstellen möglich und Schnittstellenparameter sind immer call-by-value-Parameter.

Im Kontext des Projektes wurde gRPC zur Umsetzung der Kommunikation aus mehreren Gründen gewählt. Zum einen ist gRPC eine neuere Technologie als CORBA. Dies ist nicht zwangsweise eine Aussage darüber welche Technologie geeigneter ist, ist aber aus pragmatischer Sicht eine bessere Entscheidung in Bezug auf einen aktuellen Stand von Dokumentationen und das Ziel sich im Rahmen dieses Projektes mit aktuellen Technologien zu befassen. Es entsteht kein Nachteil daraus, dass für gRPC keine Remote-Objektreferenzen unterstützt werden, da diese für die Schnittstellen im PTN nicht benötigt werden.

Anwendung findet gRPC an zwei Stellen. Sowohl die Kommunikation zwischen dem Client und dem Trackerserver, als auch die Kommunikation zwischen Client und Torrentserver basieren auf gRPC-Schnittstelle. Die Kommunikation zwischen Client und Trackerserver wird immer vom Client aus begonnen und tritt in drei Fällen ein. Zum einen wenn der Client die aktuellen PTN-Informationen, Prüfsummenalgorithmus, etc. anfragen möchte, zum anderen das Hochladen von Dateifragmenten in das PTN und wenn der Nutzer während des Uploads den Status des Uploads abfragen möchte. Der Fall des Hochladens von Dateifragmenten ist dabei ein besonderer Fall, denn hier läuft die Kommunikation über einen Stream ab. Das heißt der Nutzer kann seine Fragmente nacheinander versenden ohne auf eine Antwort warten zu müssen und dann mehrerer Bestätigungen auf die Fragmente bekommen.

Die Kommunikation zwischen Client und Torrentserver läuft ebenfalls über gRPC ab. Auch hier gehen alle Kommunikationsvorgänge immer vom Client aus. Dieser kann zum einen zu einem seiner Datei eine Liste mit Fragmentprüfsummen und den Torrentservern anfragen, die dieses Fragment besitzen und kann im Anschluss an den entsprechenden Torrentservern mit der zweiten Nachricht anfragen, mit der ein konkretes Fragment angefordert und heruntergeladen wird.

Der Einsatz von gRPC war im Verlauf des Projektes sehr vorteilhaft. Eine aktuelle Dokumentation und eine starke Nutzercommunity erleichterten die Technologierecherche und die einfache Einbindung in C# und Java-Code ermöglichten einen schnellen Fortschritt in der Implementierungsphase. Aus der bisherigen Erfahrung mit gRPC konnten keine nachteiligen Eigenschaften erkannt werden die eine spätere Erneute Verwendung der Technologie für diesen oder einen ähnlichen Kontext ausschließen würde.

### MQTT

Die Abkürzung MQTT steht für „Message Queuing Telemetry Transport“ und ist ein Protokoll, welches zur Kommunikation zwischen Maschinen im „Internet of Things“ konzipiert wurde. Es zeichnet sich durch seine sehr leichtgewichtige Kommunikationsstruktur aus und wird vor allem dort eingesetzt wo Ressourcen, wie Strom und Bandbreite, nur knapp vorhanden sind [1]. Dies würde es normalerweise nicht als ein geeignetes Kommunikationsprotokoll klassifizieren, das diese Bedingungen im Fall des PTN nicht gegeben sind. Der Grund warum es dennoch als Protokoll zur Kommunikation zwischen Trackerserver und Torrentservern verwendet wird, liegt in der effizienten 1:N-Kommunikation, welche im PTN regelmäßig vom Trackerserver initiiert wird. Die Kommunikation über MQTT läuft dabei so ab, dass es einen MQTT-Broker gibt, welcher verschieden Kommunikationskanäle, auch Topics genannt bereitstellt. Diese Topics können wie Emailverteilerlisten betrachtet werden, in welche man sich als Nutzer eintragen kann. Sendet ein Teilnehmer eine Nachricht an ein solches Topic, bekommen alles Teilnehmer an der MQTT-Kommunikation, welche dieses Topic abonniert haben die entsprechende neue Nachricht. Eine zusätzliche Besonderheit ist, dass die Topics hierarchisch aufgebaut sind, sodass ein Teilnehmer nur ein bestimmtes Topic abonnieren kann, oder auch mehrere auf einmal und dann alle Nachrichten dieses Topics und seiner Untertopics erhält. Im PT gibt es drei verschiedene Anwendungsfälle, in denen eine Kommunikation über MQTT stattfindet.

Zum einen wenn ein neuer Torrentserver in das PTN aufgenommen werden möchte. Dieser abonniert ein bestimmtes Topic und wartet sendet auf dieses auch einen Eintrittswunsch, welcher vom Trackerserver beantwortet wird. Je nachdem ob alle Vorbedingungen erfüllt sind, wird der Torrentserver aufgenommen oder auch nicht.

Die zweite Situation entsteht, wenn ein Nutzer eine neue Datei in des PTN hoch lädt. Die Fragmente die dabei nach und nach beim Trackerserver ankommen werden über ein MQTT-Topic im PTN ausgeschrieben und die Torrentserver können auf diese Nachricht, die an alle Torrentserver gesendet wurde, antworten, wenn sie das ausgeschriebene Fragment erhalten möchten.

Der dritte Fall ist die regelmäßige Aktualisierung der Netzwerkinformationen und der Übersichtsliste mit allen Fragmenten und deren Eigentümern. Dies ist besonders dafür wichtig, dass jeder Torrentserver den aktuellen Stand im Netzwerk kennt. So kann er einem Nutzer, wenn dieser eine Datei zum Herunterladen anfragt, immer die aktuellen Informationen zum PTN und dessen Fragmentverteilung geben.

Der Einsatz von MQTT im PTN wurde erfolgreich umgesetzt und erfüllt die Anforderungen, die an ihn gestellt wurden. Ob die Wahl, MQTT zu nutzen kann kritisch gesehen werden. Zum einen Bringt MQTT mit seiner Möglichkeit der 1:N-Kommunikation einen großen Vorteil mit sich, da der Trackerserver in einem PTN mit 100 Torrentservern die gleiche Nachricht mit den Fragmentverteilungen nur einmal versenden muss statt über 100 separate 1:1-Kommunikationskanäle mehrfach versendet werden muss. Andererseits wird das typische Publisher-Subscriber-Konzept von MQTT nicht komplett eingehalten, da über bestimmte Teile der Topic-Hierarchie 1:1-Nachrichtenaustausch bidirektional stattfindet, oder in andern Fällen die über Topics geschieht, die auch andere Torrentserver abonniert haben und dort Nachrichten als nur-für-einen-Empfänger-bestimmt gekennzeichnet werden. Die anderen Torrentserver erhalten diese Nachricht dann trotzdem und ignorieren sie. Insgesamt überwiegt aber der Vorteil durch die 1:N-Kommunikation, die den Nachrichtenaustausch deutlich vereinfacht und Fehlerquellen minimiert.

# Diskussion der Ergebnisse

Im Rahmen der Entwurfsarbeit ist es gelungen, einen funktionsfähigen Prototyp zu implementieren. Dieser weist allerdings noch Mängel auf, die in dem bestehenden Kontext nicht mehr gelöst werden können/konnten. So fehlen beispielsweise noch einige grundlegende Funktionen wie das Entfernen einer Datei aus dem PTN. Momentan ist nur das Speichern im PTN und das Herunterladen aus diesem realisiert. Der Client bietet zwar bereits die graphische Oberfläche an, um eine Datei löschen zu können, die Implementierung ist jedoch nicht umgesetzt worden, war im Rahmen dieses Projektes aber auch nicht vorgesehen. Auch das Entfernen eines Torrentservers ist bisher noch nicht implementiert. Sollte ein Torrentserver entscheiden nicht länger im PTN zu agieren, müssten die Fragmente die dieser Torrentserver abgespeichert hat neu verteilt werden, damit es nicht dazu kommt, dass eine Datei nicht wieder zusammengesetzt werden kann. Das Hinzufügen eines Torrentservers ist kein Problem, allerdings ist dieser von da an zu Beginn leer. Vorteilhaft wäre es, wenn jeder Torrentserver der bereits im PTN war einige seiner Fragmente an den neuen Torrentserver abgeben würde. In der aktuellen Situation geschieht dies nicht und würde eine neue Datei hochgeladen werden, so würden mit großer Wahrscheinlichkeit alle Fragmente dieser Datei einmal auf dem neuen Torrentserver abgespeichert werden. Was dem Prinzip der verteilten Speicherung in einem Torrentnetzwerk nicht entsprechen würde.

# Fazit

Im Kontext dieser Entwurfsarbeit wurden der Aspekt der Kommunikation von mehreren Computerprogrammen untereinander genauer betrachtet. Dabei wurden die Protokolle zur Kommunikation entsprechend der Anforderungen selbst entworfen, implementiert und getestet. Dadurch konnte ein tieferes Verständnis über Rechnerkommunikation erworben werden und vor allem mit den Technologien gRPC und MQTT angewandt werden. Zugleich wurde mit der Thematik „Torrentnetzwerk“ ein besonderes Augenmerk auf die Punkte der Datensicherheit, darunter Redundanz, Integrität und Anonymität gelegt. Diese sind vor allem im Kontext der zunehmenden Digitalisierung von Daten und Prozessen besonders wichtig. Zudem beinhaltet die Thematik „Torrentnetzwerk“ auch Prinzipien von verteilten Systemen, welche auch in anderen zukunftsweisenden Themen wie Grid, Cloud und Edge Computing umgesetzt werden. Aufgrund einer Technologierecherche vor Beginn der Implementierungsphase, konnte Aufwand eingespart werden der eine Technologieänderung von CORBA zu gRPC mitten in der Implementierungsphase bedeutet hätte. Ab Beginn der Implementierungsphase wurde inkrementell gearbeitet und der dabei entstehende Stand mit jedem Inkrement mittels Modultests getestet. Ein abschließender Systemtest wurde am Ende der Implementierungsphase angeschlossen.

Mit Ablauf der Projektlaufzeit ist damit ein funktionsfähiger Prototyp eines abgewandelten Torrentnetzwerkes unter Berücksichtigung der zu Beginn aufgestellten Anforderungen konzipiert, realisiert und getestet. Die Tests haben das Funktionieren des Systems bestätigt. Für einen professionellen Einsatz ist das entwickelte System jedoch nicht geeignet. Zum einen fehlen noch weitere Sicherheitsfunktionen wie Verschlüsselung und Authentifikation, zum anderen wurde noch keine Belastungstests durchgeführt, um die Verfügbarkeit des Systems zu überprüfen und gegebenenfalls zu verbessern. Das selbstimplementierte System würde zudem den permanenten Betrieb mehrerer Server als Torrent- und Trackerserver erfordern sowie eingetragene Domains oder statische IP-Adressen, u die Funktionalität durchgehend zu gewährleisten. Dieser Kostenfaktor stünde nicht in Relation zum erbrachten nutzen und ist daher abzulehnen.

Als grundlegende Erkundung dieses Gebiets ist das Projekt dennoch erfolgreich gewesen. Es konnte gezeigt werden, dass es möglich ist in kurzer Zeit ein solches System mit Schnittstellenbeschreibungen zu entwerfen, zu implementieren und zu testen. Ob und wie diese Erkenntnis in Zukunft genutzt werden wird ist vorerst nicht ersichtlich, weitere Möglichkeiten was und wie verbessert werden kann wird im Ausblick dargestellt.

# Ausblick

Nach erfolgreicher Konzeption und Realisierung eines prototypischen Systems kann diese Grundlagenerkundung genutzt werden um ein potenziell marktreifes Produkt weiter zu entwickeln. Dabei ist vordergründig das Eliminieren des Trackerserver das Ziel das verfolgt werden muss, um das implementierte PTN als normales Torrentnetzwerk nennen zu können. Mittels eines dezentralen Hochladevorgangs von Nutzdaten könnte dieses Ziel weiterverfolgt werden. Des Weiteren müsste mit Abschaffung eines Trackerservers auch der regelmäßige Datenaustausch zwischen den Torrentservern gewährleistet sein, was zusätzliche Kommunikationsprotokolle erfordert. Ein Fall der bisher ebenfalls nicht betrachtet wurde ist die dynamische Veränderung der Anzahl von Torrentservern im Netzwerk. Fällt ein Torrentserver aus dem Netzwerk weg, müssen die Datenpakete die dieser mit sich führte neu verteilt werden ohne z.B. die Redundanz zu gefährden. Kommt ein neuer Server in das Netzwerk hinzu, müssen Pakete von den anderen Servern auf diesen neuen umverteilt werden. Die Funktionalität des Client kann in diesem Fall wieder verwendet werden und der Torrentserver fragt in der Rolle eines Clients Fragmente an, die er von anderen Torrentserver herunterladen möchte. Die graphische Oberfläche des Clients sollte interoperabler implementiert werden, damit diese auch unter anderen Betriebssystemen wie z.B. Mac OS oder Linux laufen kann.

Generell ist die

# Quellen

## Literaturquellen und Verweise

1. <https://mqtt.org/>
2. <https://www.grpc.io/> [Zugriff: 25.02.2020]
3. <https://github.com/NeumannDirk/MyTorrentAnhang>
4. <https://github.com/NeumannDirk/MyTorrent/blob/master/Source/CSharp/MyTorrent.gRPC/MyTorrent.proto>

## Glossar

|  |  |
| --- | --- |
| Wort | Bedeutung |
| BitTorrent | Eines der weltweit führenden Torrentsysteme, welches für den privaten und kommerziellen Gebrauch nutzbar ist  und  Ein Protokoll welches von dem gleichnamigen Unternehmen entwickelt wurde, genutzt wird und Open-Source freigegeben wurde zur offenen Weiterentwicklung |
| Nutzer | Person, die Daten in das PTN hoch lädt und/ oder wieder herunterlädt |
| Client | Die grafische Anwendung die der Nutzer verwenden kann um mit dem PTN zu kommunizieren |

## Verwendete Programme

[PG1] XnViewMP: https://www.xnview.com/de/xnviewmp/

[Zugriff: 28.07.2019]

[PG2] PuTTY https://www.putty.org/

[Zugriff: 30.08.2019]

[PG3] Etcher https://www.balena.io/etcher/

[Zugriff: 30.08.2019]

[PG4] Testprogramm Modultest https://github.com/NeumannDirk/Wolfsabschreckung

[Zugriff: 30.08.2019]

## Projektdateien

Link: <https://github.com/NeumannDirk/MyTorrent>