BABEȘ-BOLYAI UNIVERSITÄT CLUJ-NAPOCA

FAKULTÄT FÜR MATHEMATIK UND INFORMATIK

INFORMATIK IN DEUTSCHER SPRACHE

BACHELORARBEIT

ANWENDUNG FÜR DEZENTRALISIERTE SPEICHERUNG

Betreuer

Lektor Dr. Cristea Diana

Eingereicht von

Paul-Cristian Iacob

2020

Abstract

Technology is nowadays an extremely important part of our daily life. Most of the activities we do are simplified by the technology. We are using a lot of applications or online services in order to facilitate our everyday life, but using these services implies sending a lot of information about us. Sending this information is not inherently a bad thing, but it might cause a lot of problems if it is not used in a proper way or it is used for bad purposes.

In this paper I will present a method for avoiding or minimizing these risks. The following system will prevent third-part agents from accessing the user's information and will decrease the risks of accessing information from providers' servers by unauthorized agents.

The solution is building a network, in which the transferred files or information are not stored on external servers - like most of the applications do right now. Instead of this, the information will be divided in smaller pieces and stored on other users' devices. In order to restrict the information readability by the users on whose device the information is stored, the data will be split bitwise. Therefore it will be a peer-to-peer network, in which the information is split in smaller pieces and stored on user's devices instead of an external server.

In this paper I will present a basic application – named Scindo - which implements the above described network. This work is divided in three parts: describing the current context and presenting the advantages Scindo brings, details about the used technologies and implementation and a chapter containing performance tests and simulations.

There are already another peer-to-peer networks implemented in different contexts and for various purposes. The probably most known example are the torrents. The difference between these ones and the one presented in this paper is that Scindo is focused on the data integrity and security, in comparison with the first ones, which aim to reduce the maintenance costs or to improve the download speed. Although the application presented in this paper is focused on file transferring, its benefits can be extended to other information categories, such as personal data or metadata transfers.

Building this network is a very complex process. The target of this paper is not building a perfectly functional network, but to present some preliminary results in this field. There is a lot of research to be done in this field, in order to optimize this type of network.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Einführung

Status quo

Die Technologie hat sich in den letzten Jahren sehr viel und schnell entwickelt. Täglich erschienen neue nützliche Technologien und Services. Für viele Meschen haben diese die Art und Weise sie arbeiten oder kommunizieren geändert. Trotzdem aber, brauchen diese Technologien sehr viele Daten und Information von dem Benutzer. Heutzutage spielen unsere Daten eine sehr wichtige Rolle in der Wirtschaft und in der Gesellschaft. Diese sind die Basis für sehr große wirtschaftliche und gesellschaftliche Systeme, wie zum Beispiel der Bereich der targetierten Bewerbungen. Für fast alle Menschen ist es sehr leicht ihre Daten den Serviceanbietern zu geben, ohne wenigstens die Geschäftsbedingungen vorher zu lesen.

Problem

Die ethische Verpflichtung der Sicherheit unserer Daten fällt also auf die Schultern der Anbietern. Manche aber respektieren diese Normen nicht und so werden diese Daten für verschiedene unethische Zwecke benutzt (z. B. der Skandal mit Cambridge Analytica).

Auch falls diese gute Zwecke haben, bleibt noch die Möglichkeit der Angriffe der Hackers. Solche große Datenbanken mit Informationen über Personen sind sehr attraktiv für Hackers. Darum ist es genug eine kleine Sicherheitsleck in diesen Systemen zu finden, um Zugriff an Daten von Tausende oder Millionen vom Personen zu erhalten. Mehr als das, Data Mining ist heutzutage immer mehr benutzt, um spezifische Informationen über User herauszufinden. Ohne aber solche großen Datenbanken ist dieser Miningprozess beschwierigt. Diese Tatsache ist ein sehr wichtiger Aspekt betreffend der User Data Privacy.

Ideal wäre es sein die third-party Datenspeicherung zu vermeiden. Damit kann man sehr viele Risiken umgehen und eine bessere Sicherheit der Daten zu versichern.

State of the art

Eine ziemlich gute Lösung sind die Blockchain-basierten Applikationen, die die third-party Entitäten eliminieren. Trotzdem aber sind die Blockchainnetzwerke ziemlich schwer zu pflegen. Sie brauchen eine sehr große Rechenleistung, die nicht für alle Benutzer bequem sind. Im Kontrast, die einzige wichtige Ressource die in dieser Arbeit präsentierte Anwendung braucht, ist die Speicherplatz. Mehr über diese Tatsache wird in der Beschreibung Unterkapitel vorkommen.

Beispiele von Anwendungen die Kommunikation auf Blockchain vermitteln sind Dust [1] und Sense [2].

Die Blockchain Technologie ist so entworfen, um die Transaktionen zuverlässiger zu machen. Sie ist eine potentielle Lösung für die Leistung der Dataintegrität, die mit Hilfe von Kryptographie die Datamanipulation versichern kann. Der Blockchain ist dezentralisiert und keine zentralisierte Entität kann Transaktionen genehmigen. Alle Teilnehmer der Netzwerk müssen einen Konsens treffen, um Transaktionen sicherlich zu validieren. Beiseite, können die alten Aufzeichnungen nicht modifiziert sein. Externe Angreifer müssen die Kontrolle aller Teilnehmer des Netzwerkes haben, um sie zu schäden, was praktisch unmöglich ist. Diese Tatsache macht aber die Netzwerk auch sehr schwer aktualisierbar und stützt weitere Entwicklung sehr schwach. [3]

Im Kontrast mit dieser Ansatz bringt diese Anwendung den Vorteil, dass die Pflege des Netzwerkes leichter durchgeführt werden kann. Also ist die Integrität der Daten von ihren Zersplittung versichert, so dass dieser Abtausch vorteilhaftig ist. Ein anderer Vorteil ist, dass die Informationen nicht auf allen Devices der Netzwerkteilnehmer gespeichert sein muss, sondern nur auf ein paar Devices. Das bringt eine sehr wichtige Verbesserung des notwendigen Speicherplatzes.

Eine andere Möglichkeit ist von den direkten user-to-user Anwendungen, die die Daten nur auf den Devices der Teilnehmer speichert. Hier gibt es aber die Möglichkeit einer angesteckter Device, die schädlich für die Informationen ist.

Lösung

Meine Anwendung, Scindo, sollte die Probleme der oben genannten Prozesse lösen. Sie ist auf dem peer-to-peer Konzept basiert, den man auch bei den Torrenten benutzt ist. Die Idee ist, die gesendeten Daten in kleineren Teilen zu teilen und diese Teile auf mehreren Devices speichern (verschlüsselt und archiviert). Die Informationen werden bitwise zersplittet. Wenn man die gesendete Information unterladen möchtet, sammelt die Anwendung die kleinere Teile aus diesen Devices und bildet die ursprüngliche Information zurück. Also die wichtigste Charakteristik der Anwendung ist, dass sie keine Datenbanken für die Speicherung der Daten benutzen wird.

Beschreibung

Scindo ist eine Klient-Server Anwendung, die auch ein peer-to-peer-Kommunikationssystem integriert. Der Server wird nur die Verbindungen zwischen den Klienten versichern und wird die gesendeten Daten nicht speichern. Die einzigen Informationen, die er speichern wird, sind Verbindungsinformationen für den Klienten, um die Kommunikation zwischen diesen zu erleichtern.

Die von einem Klient gesendeten Daten werden in mehreren Teilen geteilt und auf verschiedene Devices der anderen Benutzern gespeichert. Der Server ist mit dem Auswahl der Devices, auf den die Informationsstücke gespeichert werden, verantwortlich. Ich werde mehrere Strategien für den Auswahl versuchen und die Ergebnisse vergleichen.

Ein wichtiger Aspekt, der in diesem Prozess vorkommt, ist die Größe der gebrauchten Datenspeicherungressourcen. Man muss aber betrachten, dass die Technologie sich in den letzten Jahren so schnell entwickelt hat, dass die Speicherungressourcen sich jährlich verdoppeln. Mehr als das, die Dimension unserer Personalgeräte ist immer kleiner und wir haben immer mehr Akzess zu Speicherungressourcen [4].

Eine andere wichtige Perspektive ist die Geschwindigkeit des Netzwerkes. Es ist sehr wichtig eine gute Geschwindigkeit und Stabilität der Konnexion für ein Peer-to-Peer System zu haben. Glücklicherweise aber ist dieses Technologiegebiet auch in einer kontinuierlichen Entwicklung. Man schätzt, dass in den nächsten 2-3 Jahren die 5G Technologie weltweit implementiert wird, die Geschwindigkeiten von 50-100 Mbps vermitteln kann [5].

Sicher, um ein stabilen Peer-to-Peer System zu bauen, braucht man mehrere Kopien jedes einzelnen Informationstückes. In dieser Arbeit werde ich auch eine Analyse der benötigten Speicherungsressourcen im Verhältnis mit der Anzahl der Benutzer des Netzwerkes (online oder offline). Dadurch möchte ich analysieren, welche die kleinste Anzahl der Kopien eines Informationsstücks ist, die die Funktionierung der Anwendung versichert.

Vorteile

- Privacy. Die Daten werden auf keinen Server gespeichert, so dass keine third-party Agenten Zugriff an diesen haben. Obwohl sie auf die Geräte verschiedener Personen gespeichert sein werden, werden sie verschlüsselt gespeichert sein und auch wenn diese (in einem hypothetischen Situation) decrypted sein werden, erhält man nur einen kleinen Teil der Information (und man kann sie bitwise teilen, so dass diese Informationen nutzlos sind).
- Sicherheit. Auch wenn einer der Devices angesteckt ist, ist die Information nicht verloren oder für schlechte Zwecken benutzt. Der Angreifer kann nur an einem kleinen Teil der Information Zugriff haben, und auch wenn er die Verschlüsserung löst, kann er nichts mit den erhaltenen Informationen machen. Um die Information wiederzustellen braucht er alle Teile, also er braucht Zugriff an allen Devices die Teile der Information beinhalten (fast unmöglich).
- Sehr geeignet für Informationen, die für mehreren Personen relevant sind (z. B. Chat Groups). Im Vergleich mit den Anwendungen die nur eine sichere peer-to-peer Konnexion zwischen den direkten Kommunikanter, die die Daten

vollständig auf dem Device des Empfängers speichert (also auf den Devices aller Empfängers), meine Anwendung wird diese Information nur einmal speichern und sie wird den Empfängern sagen, woher sie die kleinen Teilen nehmen können.

- Der Speicherplatz ist nicht ein großes Problem. Obwohl die Daten mehrfach gespeichert werden (wegen Persistenz- und Zugriffsgründe), wird das nicht ein sehr großes Problem sein (solange der Speicherplatz effizient benutzt wird). Sowieso gibt es sehr viel unbenutzter Speicherplatz auf den meisten Devices.

Technologien

Java

Java ist eine Objekt-orientierte Programmiersprache, die 1995 von James Gosling von Sun Microsystems gebaut wurde. Sie wurde 2010 von Oracle gekauft.

Ich habe Java für die Realisierung des Projektes ausgewählt wegen ihre cross-platform Fähigkeiten. Weil die Informationübertragung heutzutage vital ist und weil sie von jedwelchem Device verwendet ist, vermittelt Java die beste Methode diese Anwendung auf mehrere Platformen zu portieren. Die Portierung auf mehreren Devices ist für diese Anwendung sehr relevant, weil je größer die Netzwerk ist desto besser wird sie funktionieren.

Auf der anderen Seite, Java ist heutzutage die meist verwendete High-Level Programmiersprache. Der Vorteil damit ist dass, auf langem Zeit, diese Anwendung leichter weiter entwickelt sein kann. Diese Tatsache ist von den meisten Rangfolgen der Programmiersprachen geprüft, wie zum Beispiel der von TIOBE [6] (Abbildung 1.)

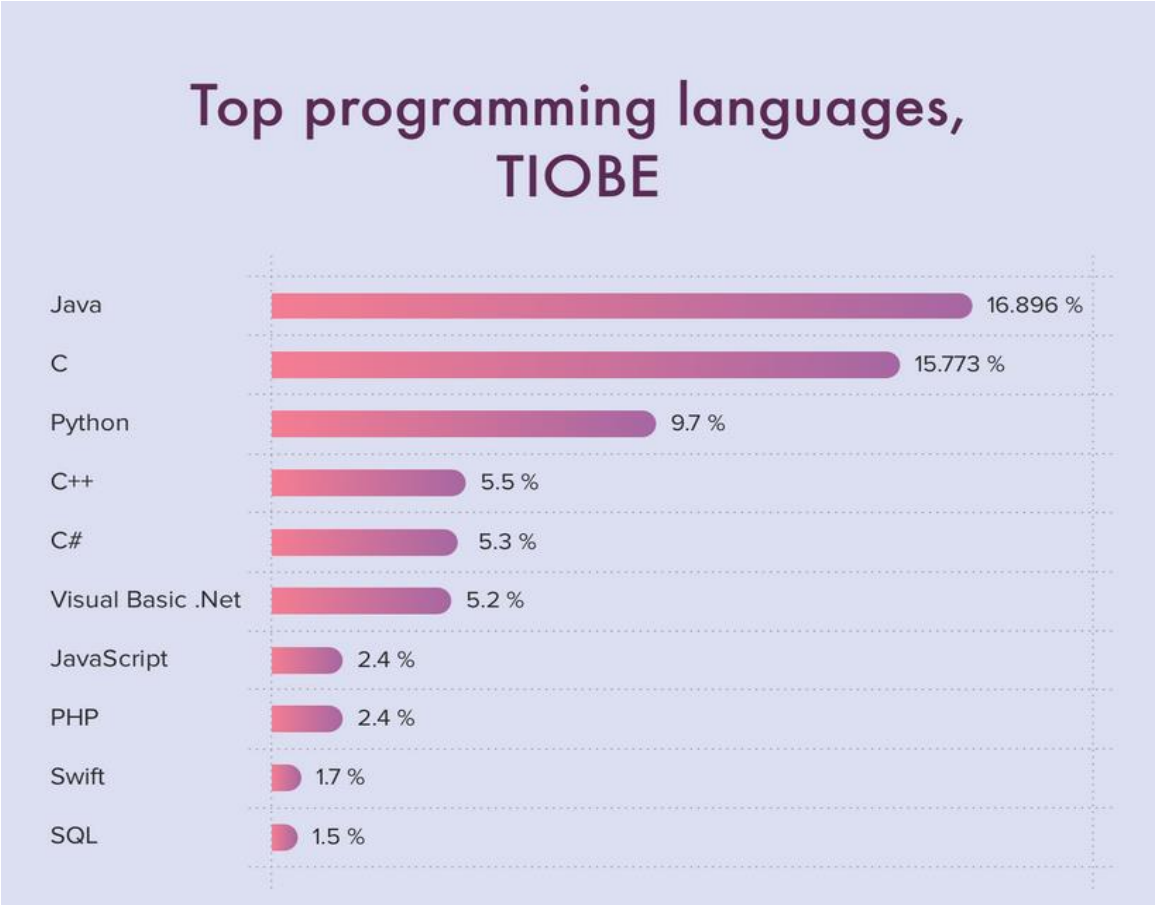

Abbildung 1 – Programmiersprachenranking realisiert von TIOBE [7]

Java 8 ist heutztage die meist verbreitete Java Version. Fast alle Java-spezifische Framework versichern Support für diese Version. Darum werde ich auch Java 8 für diese Anwendung benutzen. Die meisten Umfragen auf diesen Topik zeigen, dass Java 8 in wenigstens 65% der Java Anwendungen verwendet ist. Beispielsweise zeigt eine Umfrage, die von Jetbrains realisiert wurde, dass 83% der Projekten, die in ihrem IDE, IntelliJ, gebaut sind, Java 8 verwenden.

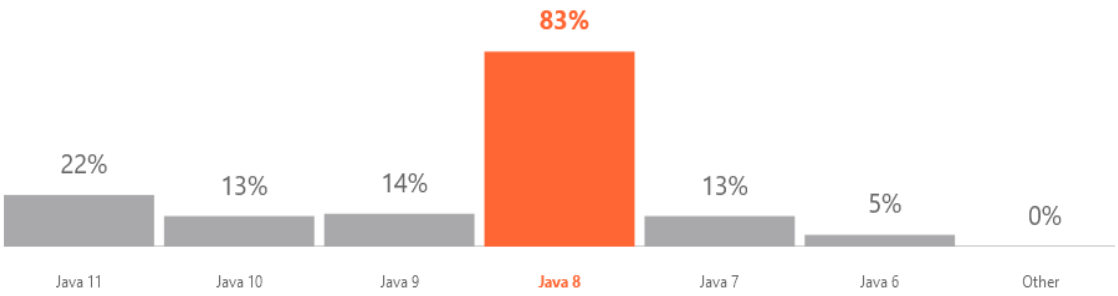

Abbildung 2 - Ranking der Java-versionen nach ihrer Anwendung [8]

Gradle

Gradle ist ein open-source System, der den Buildprozess automatisiert. Er erschien 2007 und ist einer der besten solchen System.

Ein solches System ist für die Entwicklungs eines informatischen Systems sehr wichtig. Damit kann man alle Bibliotheken oder Frameworks, von denen das System abhängig ist, leicht verwalten. Es ist sehr leicht zu benutzen. Man muss nur einen build.gradle Datei erstellen, indem die Informationen über den nötigen Modulen hinzufügen muss.

Dieser Prozess ist von publischen Repositories, wo man sehr viele publische Module veröffentlicht sind, unterstützt. Für diese Anwendung ist Maven Repository [9] benutzt.

MySQL

MySQL ist das aktuell berühmteste Datenbankverwaltungssystem. Er wurde von MySQL AB in 1995 gebaut und ist jetzt opensource.

Die wichtigste Fähigkeit, außer der Speicherung, die MySQL hat, ist die Geschwindigkeit der Suchen. Diese Tatsache ist sehr wichtig für diese Anwendung, wenn man die besten Peers für einen Transfer suchen möchtet. Mehr darüber werde ich in den folgenden Kapiteln sprechen.

Für diese Anwendung wird MySQL minimal auf dem Server Side benötigt, nur um die Informationen über die Bindungen zwischen den Klienten und die Statistiken der Benutzer zu speichern.

Hibernate

Hibernate ist einen Object-relationalen Mapping Framework, der zusammen mit Java funktioniert. Es wurde von Red Hat in 2001 gebaut. Es ist benutzt, um die Entitäten aus dem Model der Anwendung automatisch zu ihren analogen Tabellen aus der Datenbank zu mapieren.

Diese Prozedur bringt sehr viele Vorteile, wie zum Beispiel leichtere Behandlung der Persistenz-layer, predefinierte Datenbankoperationen (Insert, Delete, Update usw.) oder Prepared Statements, die sehr wichtig gegen der SQL-Injection Angriffe sind.

Spring Boot

Spring Boot ist ein open-source Modul der Spring Framework. Er ist von Pivotal Team entwickelt und ist verwendet, um eigenständige Anwendungen zu bauen. Dieser Framework ist sehr oft verwendet, um RESTful API zu vermitteln.

Die Eigenschaften eines RESTful APIs sind die folgenden:

1. Klient-server: es ist wichtig die server- oder Klientspezifische Operationen zu trennen und sie in dem passenden Platz zu implementieren. So wird es auch in diesem Projekt passieren, denn zum Beispiel, die Persistenz der Verbindungsinformationen wird vom Server durchgeführt, während die Splittung der Daten wird von dem Klient durchgeführt.
2. Stateless: jede Anfrage des Klientes soll ale nötige Informationen beinhalten und soll sich nicht auf einem Server Kontext stützen. In dieser Situation wird der Klient seinem ID und Informationen über die angefordete Aktion.
3. Cacheable: Informationen aus der Anfragen, die als cacheable markiert sind, können für zukünftige Anfragen verwendet sein.
4. Uniform interface: REST ist von vier Zwänge definiert: Identifikation der Ressourcen, Manipulation der Ressourcen durch Representation, selbstbeschreibende Nachrichte und Hypermedia als Motor der Anwendungsstatus.
5. Layered system: jede Schicht kann nur Informationen aus den benachbarten Schichten sehen.

Für diesen Projekt wird Spring Boot auf dem Server-Side verwendet, um einen API den Klienten zu vermitteln. Es werden Routen für allen nötigen Kommunikationen zwischen den Klienten und dem Server veröffentlicht, wie zum Beispiel für das Erkenntinis der Peers für Upload and Download.

Python

Python ist eine interpretierte, high-level Programmiersprache. Sie wurde 1991 von Guido van Rossum veröffentlicht. Ein paar Vorteile diser Programmiersprache sind die dynamische Tippen, der Garbage-collector und die leichte und kurze Syntax.

In Abbildung 1 kann man sehen, dass Python, nach Java, die meist verwendete high-level Programmiersprache ist. Mehr als das ist Python immer mehr verwendet. In Abbildung 3 kann man den aszendenten und beschleunigten Trend von Python sehen.

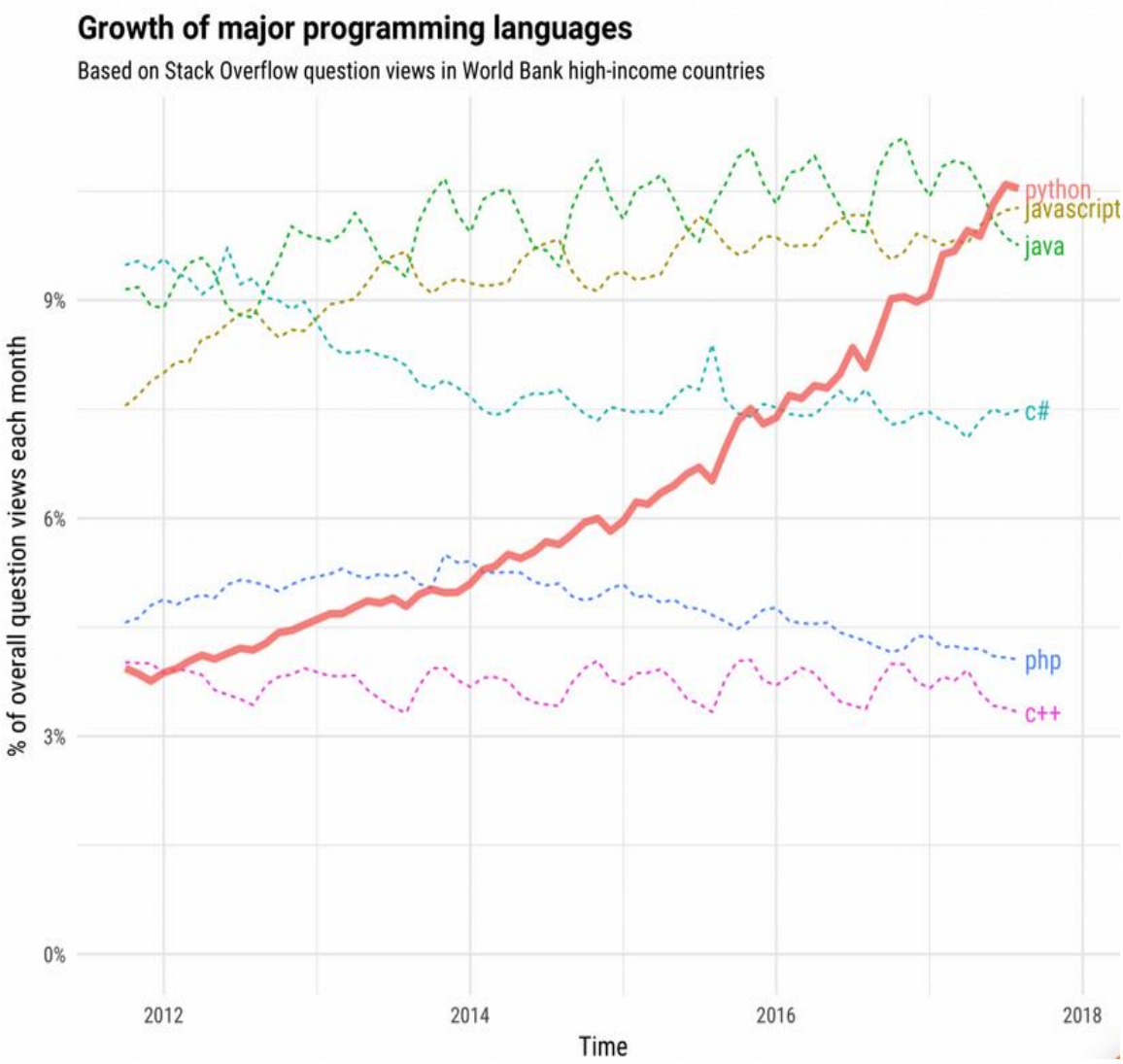

Abbildung 3 - Evolution von Python [10]

Python ist von vielen Leuten als eine Programmiersprache mit ,,enthaltenen Batterien'', wegen ihre sehr umfassende standard Bibliothek.

In dieser Projekt werde ich Python für den Analyseteil verwenden, um verschiedene Situationen aus der Wirklichkeit zu simmulieren. Dafür werde ich ins Besonderen die requests Bibliothek [11] aus Python verwenden. Dadurch kann man Anfragen der Clienten (Login, Logout, Upload, Download usw.) simmulieren, um den ganzen Projekt zu testen.

Weil dieses System ziemlich viele Benutzer braucht – und darum ist es sehr schwer auf mehreren Devices zu testen – werde ich das Verhalten der Klienten simmulieren und dafür werde ich Python verwenden.

JavaFX

JavaFX ist ein Framework mit Hilfe von dem plattformübergreifender Java-Applikationen erstellen kann. Dieser ist eine Java Technologie, die von Oracle hergestellt wurde und hat als primären Ziel die Erleichterung des professionellen Erstellens und Verteilens von interaktiven, multimedialen Inhalten und grafischen Benutzeroberflächen über Java-Plattformen. Seit 2014 ist JavaFX die Standardlösung für grafische Anwendungen unter Java. Diese hat AWT und Swing ersetzt.

Die Benutzeroberfläche von Scindo wird mit Hilfe von JavaFX implementiert. Der wichtigste Vorteil von JavaFX ist, dass sie auf jedwelcher Java-Plattforme funktioniert.

Architektur

Grobe Architektur

Diese Anwendung wird eine Klient-Server Anwendung sein, aber sie wird auch einen Peer-to-Peer Teil haben.

Wie auch in den vorrigen Seiten hingewiesen, wird der Server die Rolle haben, die beste Verbindungen zwischen den Klienten zu versichern. Er wird keine Daten der Klienten speichern, sondern nur Informationen, die notwendig für die Verbindungen sind, wie zum Beispiel die Zeit in der ein Benutzer häufig online ist oder wie viel Speicherungsplatz er für diese Netzwerk veröffentlicht oder benutzt.

Der größte Teil der Logik der effektiven Datentransaktionen wird von dem Klienten realisiert. Er wird die Daten in kleineren Stücke zersplittern, wird die Verbindung mit den Peers öffnen (mit Hilfe der Informationen, die er von dem Server bekommen hat) und die Bildung der originalen Dataien realisieren.

Die Kommunikation eines Klienten mit anderen Klienten wird Peer-to-Peer realisiert werden, während die Zunahme der nötigen Informationen über den Peers von dem Server vermittelt sein wird. Diese Tatsache ist in der unteren Abbildung dargestellt.

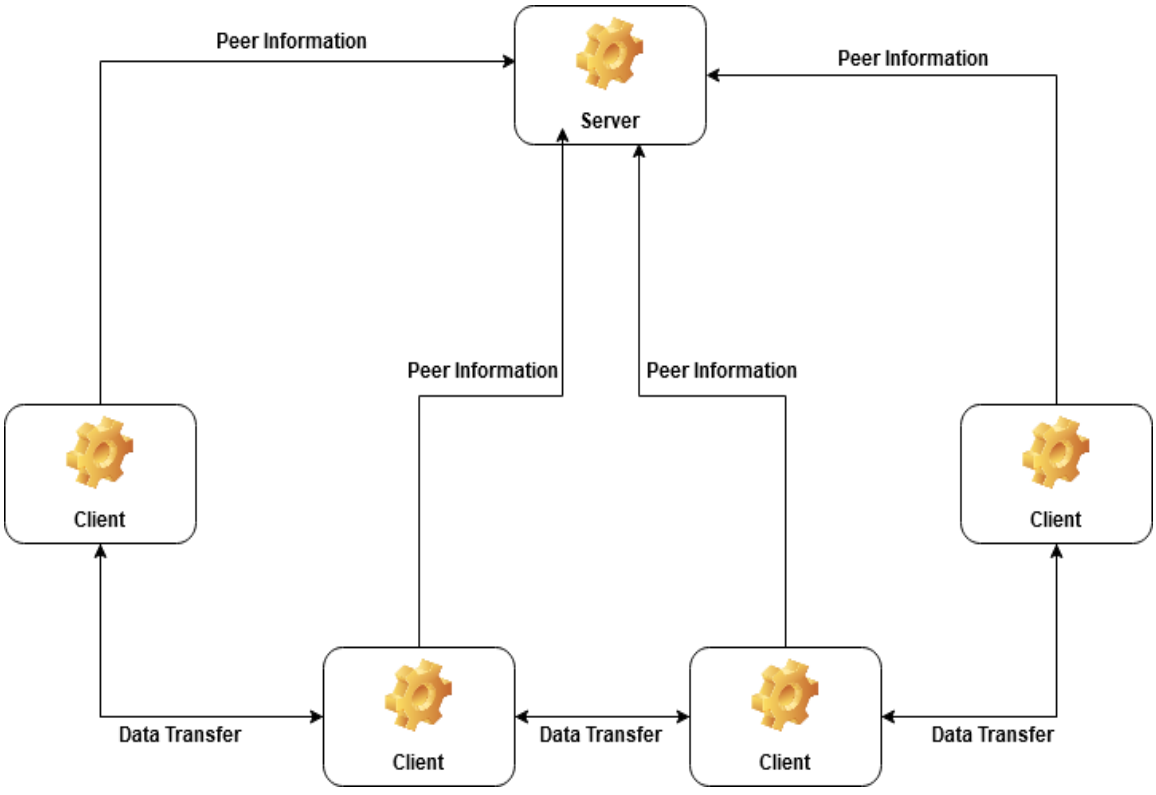

Abbildung 4 - Grobe Architektur

Der Model wird von dem Klient und dem Server gemeinsam benutzt. Diese beinhalten die entsprechenden Klassen für den User und den Transfer. Diese beiden Klassen implementieren die HasId Schnittstelle. Diese Implementation erleichtert die Implementation der Persistenzschnittstelle. Ich werde mehr darüber in den folgenden Abschnitte sprechen.

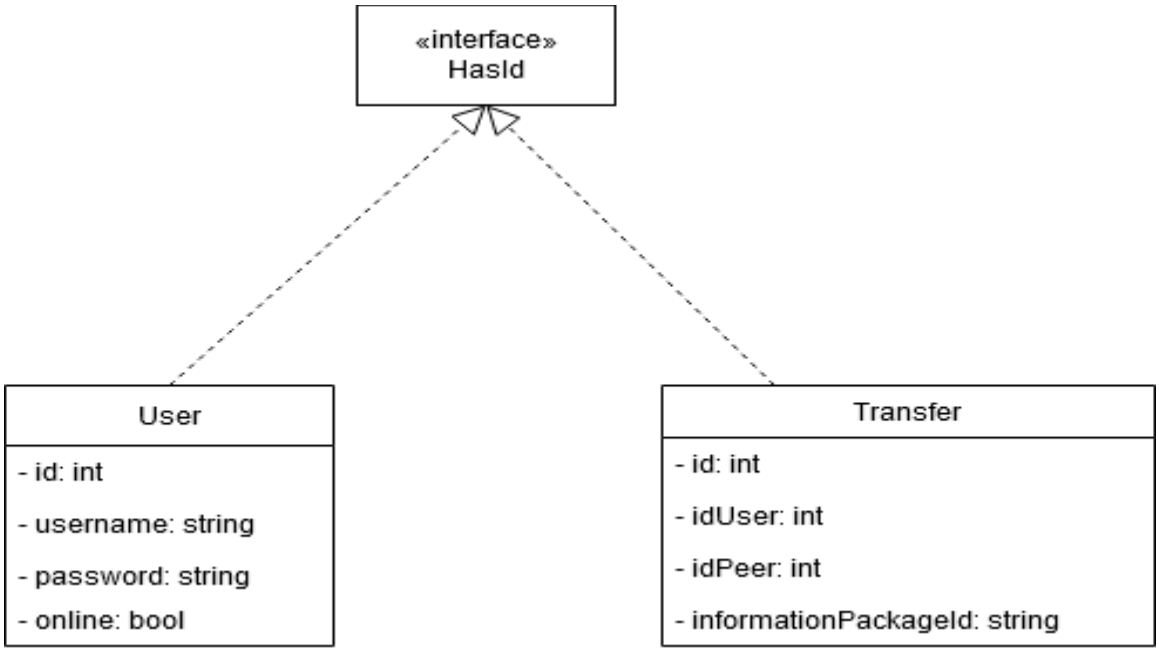

Abbildung 5 - Gemeinsames Model

Die User Klasse ist die Representation eines User. In dieser Klasse wird man Informationen über den Benutzer finden, wie seinen Nutzername, Kennwort oder ob er online ist oder nicht. Der Id ist die eindeutige Kennung eines Benutzers. Dieser wird von dem Persistenzschicht generiert.

Die Transfer Klasse ist die Representation eines Transferes. Für einen Transfer ist es wichtig zu wissen, wer der Sender ist, welche seine Peers sind und für jeden Peer, welchen Teil eines Dokumentes er bekommt.

Die InformationPackageId ist die Name des gesendeten Informationstücks. Um die Eindeutigkeit der Namen zu versichern, wird diese Name folgendermaßen generiert: für den *teil*-ten Teil der Datei datei, die vom User user gesendet wurde, wird der InformationPackageId

<*datei*><*unix timestamp*><*user*>_<*teil*> sein, wobei *unix timestamp* die Unix Time ist. Dadurch versichert man die Eindeutigkeit des Namen jedes einzelnen Dateistückes.

Server-side

Persistenz-layer

Über dem Modelschicht kommt die Persistenzschicht. Dieser Schicht vermittelt die Kommunikation mit der Datenbank und implementiert die Datenbankoperationen. Dafür habe ich eine templatisierte Klasse, Repository < Entität > gebaut, die die gemeinsamen Methoden aller Entitätaten implementiert (add, delete, update usw.). Aus dieser Klasse werden die Entität-spezialisierten Repository Klassen vererben und mehr spezifische Datenbankoperationen implementieren.

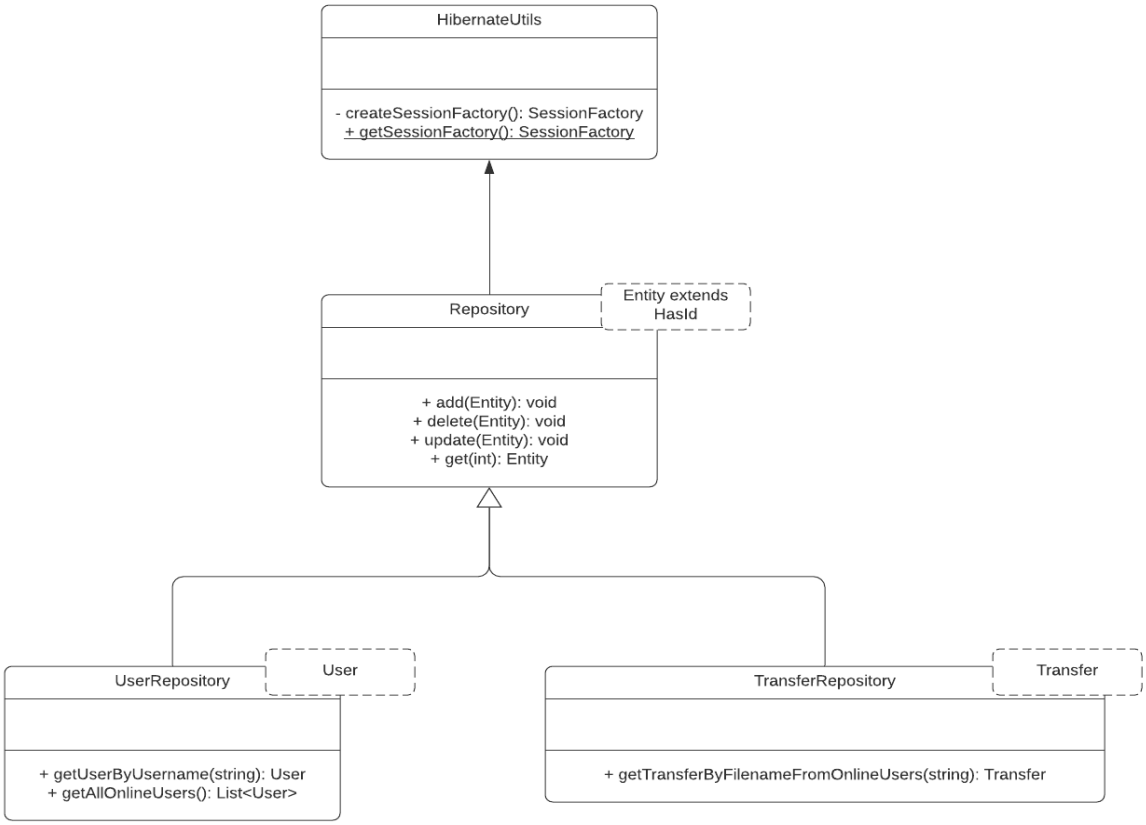

Abbildung 6 - Persistence layer

Die Repository Klasse benutzt die HibernateUtils Klasse. Diese Klasse implementiert die Singleton Design Pattern und versichert die Aufbau einer SessionFactory, die eine wichtige Komponente von Hibernate ist. Damit kann man neue Sessionen öffnen, mit Hilfe von denen die Datenbankoperationen durchgeführt werden können.

Hibernate vermittelt eine sehr bequem benutzbare Schnittstelle, mit Hilfe von der die meisten Datenbankoperationen sehr leicht durchgeführt sein können. Mehr als das kann man für komplexere SQL-Anfragen Queries schreiben. Hibernate vermittelt Prepared statements, die wichtig gegen die SQL-Injection Angriffe sind.

Auf der anderen Seite vermittelt Hibernate eine sehr intuitive Bindung der Entitäten aus dem Model der programmierter Seite mit ihren korespondenten Tabellen aus der Datenbank. Hibernate vermittelt diese Bindungen durch die Verwendung von spezifischen Adnotationen. Ein Beispiel dafür kann man in der unteren Abbildung sehen.

```
@Entity
@Table(name = "User")
public class User implements Serializable, HasId {
    @Id
    @Column(name = "idUser", unique = true)
    private int id;

    @Column(name = "email")
    private String email;

    @Column(name = "password")
    private String password;

    @Column(name = "online")
    private boolean online;
```

Abbildung 7 - Hibernate Adnotationen Beispiel - User Klasse

In der obigen Abbildung kann man die Hibernate-spezifischen Adnotationen sehen: @Entity zeigt Hibernate, dass die folgende Klasse mappiert sein soll, @Tabel gibt den Namen des entsprechenden Tabelles aus der Datenbank, @Column mappiert jede Attribut der Klasse mit der entsprechenden Spalte aus dem Tabellen und @Id zeigt, dass der folgende Attribut der Primärschlüssel entspricht.

Service Layer

Der Service Layer von der Serverseite vermittelt die Operationen, die von dem Server durchgeführt sein werden. Wie auch oben gesagt, die Rolle des Servers wird nur für die Realisierung der Verbindungen zwischen den Klienten sein. Trotzdem aber spielt er eine sehr wichtige Rolle in diesem Netzwerk. Die Auswahl der besten Peers für jeden Transfer ist sehr wichtig, um die gute Funktionalität zu versichern. Dafür werde ich mehrere Strategien für den Auswahl der Peers implementieren und sie analysieren (mehr darüber in dem Ergebnisse Kappitel).

Dieser Layer bringt Funktionalitäten, die notwendig für die Benutzer und für die Transferen sind. Er ist auch die Bindung zwischen dem REST API und dem Persistenzlayer.

Die UserService Klasse implementiert die nötigen Operationen für die Benutzer. Diese Klasse bringt die Funktionalität für die Login-, Logout- und Registrierungsanfragen. Diese nimmt die von dem Benutzer gesendeten Daten von dem REST API und realisiert die Bindung mit den entsprechenden Methoden aus der Persistenzlayer, so dass die Informationen immer aktualisiert zu sein.

Die TransferService Klasse implementiert die nötigen Operationen für die Identifizierung der besten Peers für einen Transfer. In dieser Arbeit habe ich zwei Strategien (mit Hilfe der Strategy Entwurfsmuster) für diesen Prozess implementiert: eine naive Methode, die die Peers zufällig auswählt, die aber sehr wichtig als einem Startpunkt in der Analyse des Netzwerkes ist und eine Methode, die auf den Zeitzonen der Benutzer achtet und die Peers aus der selben Zeitzone wie der Klient, der den Transfer initialisiert hat, priorisiert. Für die zweite Methode habe ich noch einen Attribut in die User Klasse hinzugefügt, der die Zeitzone des Benutzers darstellt. Die Sortierung der Benutzer wird von der Datenbank realisiert, weil sie für solche SELECT + SORT BY Operationen optimiert ist.

Die REST API ist die Schnittstelle, die die Kommunikation zwischen dem Klienten und dem Server. Diese veröffentlicht Endpunkte, in den der Server auf Anfragen von den Klienten wartet. Der Server bearbeitet die Anfragen und gibt eine entsprechende Antwort dem Klienten zurück. Die Antwort ist als einen ResponseEntity Objekt dargestellt und beinhaltet einen entsprechenden http Status Kode und für manche Operationen auch einen entsprechenden Nachricht.

Die Endpunkte aus dieser REST API sind:

- /register – POST Methode

  Durch diesen Endpunkt kann ein Benutzer sich in der Anwendung registrieren.

In der folgenden Abbildung kann man die Implementation dieses Endpunktes sehen. Spring Boot Server vermittelt hilfreiche Adnotationen, mit denen diese Endpunkte leicht implementiert sein können.

```
@PostMapping(value = "/register")
public ResponseEntity<Object> register(@RequestParam("username") String username,
                                       @RequestParam("password") String password,
                                       HttpServletRequest request) {
    try {
        userService.register(username, password);
        return new ResponseEntity<>(HttpStatus.OK);
    } catch (UserException e) {
        return new ResponseEntity<>("There is another user with the given email.", HttpStatus.BAD_REQUEST);
    }
}
```

Abbildung 8 - Implementation des Register Endpunktes

- /login – POST Methode

  Durch diesen Endpunkt kann ein Benutzer sich in der Anwendung einloggen. Als Antwort wird er einen Security Token bekommen, der für die Realisierung der weiteren Operationen nötig ist.

- /logout – POST Methode

  Durch diesen Endpunkt sendet der Benutzer eine Anfrage, um den Server zu notifizieren, dass er ausgeloggt hat.

- /upload – POST Methode

  Durch diesen Endpunkt kann ein Benutzer dem Server die Liste mit den Peers (und mit den entsprechenden Dateistücken) für einen spezifischen Transfer verlangen.

- /download – POST Methode

  Durch diesen Endpunkt kann ein Benutzer dem Server die Liste mit den Peers (und mit den entsprechenden Dateistücken) für einen spezifischen Download verlangen. Falls keine verfügbare Peers gibt, wird der Server einen Fehlernachricht senden.

Um die Sicherheit der Anwendung zu versichern, habe ich einen Tokensystem implementiert. Seiner Zweck ist nicht authorisierte Anfragen zu vermeiden.

Ein Benutzer wird nach dem Login einen Token bekommen, den er für jede andere Anfrage senden muss, um seine Identität zu prüfen.

Diese Tokens werden mit Hilfe der java.util.UUID Standardklasse hergestellt. Die Tokens sind zufällige hex Strings auf 32 Bytes.

Klient-side

Auf der Klientseite werden die interessantesten Operationen durchgeführt. Hier werden die Dateien zersplittet und zusammengesammelt und werden die Verbindungen mit den Peers realisiert.

Zersplittung der Dateien

Wie auch in der vorrigen Kappiteln gesagt wurde, die Zersplittung der Dateien wird bitwise realisiert. Zum Beispiel wird eine Datei mit dem Inhalt $b_1b_2b_3b_4b_5b_6$ (b_n ist der n-te Byte aus dieser Datei) in 3 Teilen folgendermaßen unterteilt: (b_1, b_4), (b_2, b_5), (b_3, b_6). Diese Zersplittungsmethode versichert die Integrität der Daten, weil sie nicht kontinuierliche Datensegmenten beinhalten wird.

Im Vergleich würde eine ,,klassische" Methode diese Datei in (b_1, b_2), (b_3, b_4), (b_5, b_6) unterteilen. Der Nachteil einer solchen Methode ist, dass in kontinuierlichen Segmenten Informationen die lesbar sind, hereinkommen können. Das ist sehr schädlich in der Perspektive eines externes Angriffs, wenn die Daten sensitiv sind (z. B. Personaldaten oder bankare Daten).

Obwohl diese Methode deterministisch und ziemlich banal ist, versichert sie den Schutz der Daten. Mehr als das kann man die Dateien auch durch eine nicht deterministische Methode zersplitten, zum Beispiel nach einer zufälligen Permutation. Die Möglichkeit der Implementierung dieser Alternativen ist in dieser Anwendung durch die Vermittelung eines spezifischen Schnittsstelle, SplitManager, vermittelt. Diese Schnittstelle veröffentlicht zwei Methoden, splitFileBytes(), die eine Datei Zersplittet, und revertSplit(), die die originale Datei zusammenfasst.

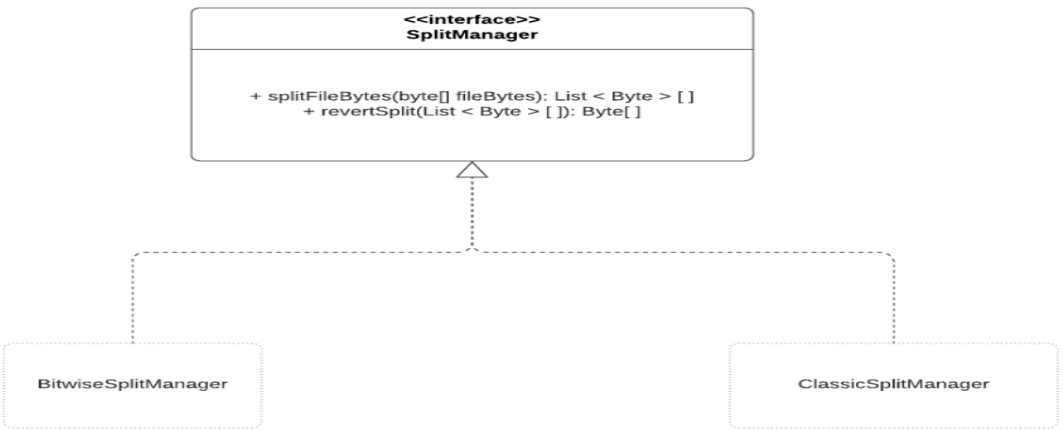

Abbildung 9 - Zersplitterung der Daten

Die ClassicSplitManager spielt in dieser Arbeit nicht eine wichtige Rolle, es wird nur implementiert, um die Vorteile der bitwise Zersplitterung zu erleuchtern.

Die Implementation der bitwise Splitter kann man in der folgenden Abbildung sehen.

```
public List < Byte >[] splitFileBytes(byte[] fileBytes) {
    List < Byte > splittedFile[] = new ArrayList[parts];
    for (int i = 0; i < parts; ++ i) {
        splittedFile[i] = new ArrayList<>();
    }

    for (int i = 0; i < fileBytes.length; ++ i) {
        splittedFile[i % parts].add(fileBytes[i]);
    }

    return splittedFile;
}

@Override
public Byte[] revertSplit(List<Byte>[] splitted) {
    ArrayList < Byte > reverted = new ArrayList<Byte>();
    Boolean finished = false;

    while (!finished) {
        for (int i = 0; i < splitted.length; ++i) {
            finished = true;
            if (splitted[i].size() > 0) {
                reverted.add(splitted[i].get(0));
                splitted[i].remove(0);
                finished = false;
            }
        }
    }

    Byte[] ret = new Byte[reverted.size()];
    return reverted.toArray(ret);
}
```

Abbildung 10 - Bitwise Splitter Implementierung

Eine wichtige Erwähnung ist, dass jede Datei als einen byte Array betrachten kann. Auf dieser Observation basiert sich auch die obere Implementation.

Peer to peer Kommunikation

Die Kommunikation zwischen den Klienten wird durch ein peer to peer System implementiert. Das heißt, dass jeder Klient direkt mit einem anderen kommunizieren wird. Diese Methode ist für die Sicherheit der Daten sehr geeignet, weil diese nicht auf einem externen Server gespeichert werden.

Diese Kommunikation wird mit Hilfe der Sockets implementiert. Jeder Klient wird auf einem Port für vorkommende Anfragen hören und sie durchführen. Ein Klient kann Anfragen um neue Informationstücke zu speichern oder alte Informationstücke zurückgeben.

Sockets wurden für diese Operation ausgewählt, weil sie wenige Ressourcen brauchen. Alternative für die Sockets konnten Message Brokers oder Remote Procedure Calls Systems, aber diese sind leistungsvoller als die Sockets.

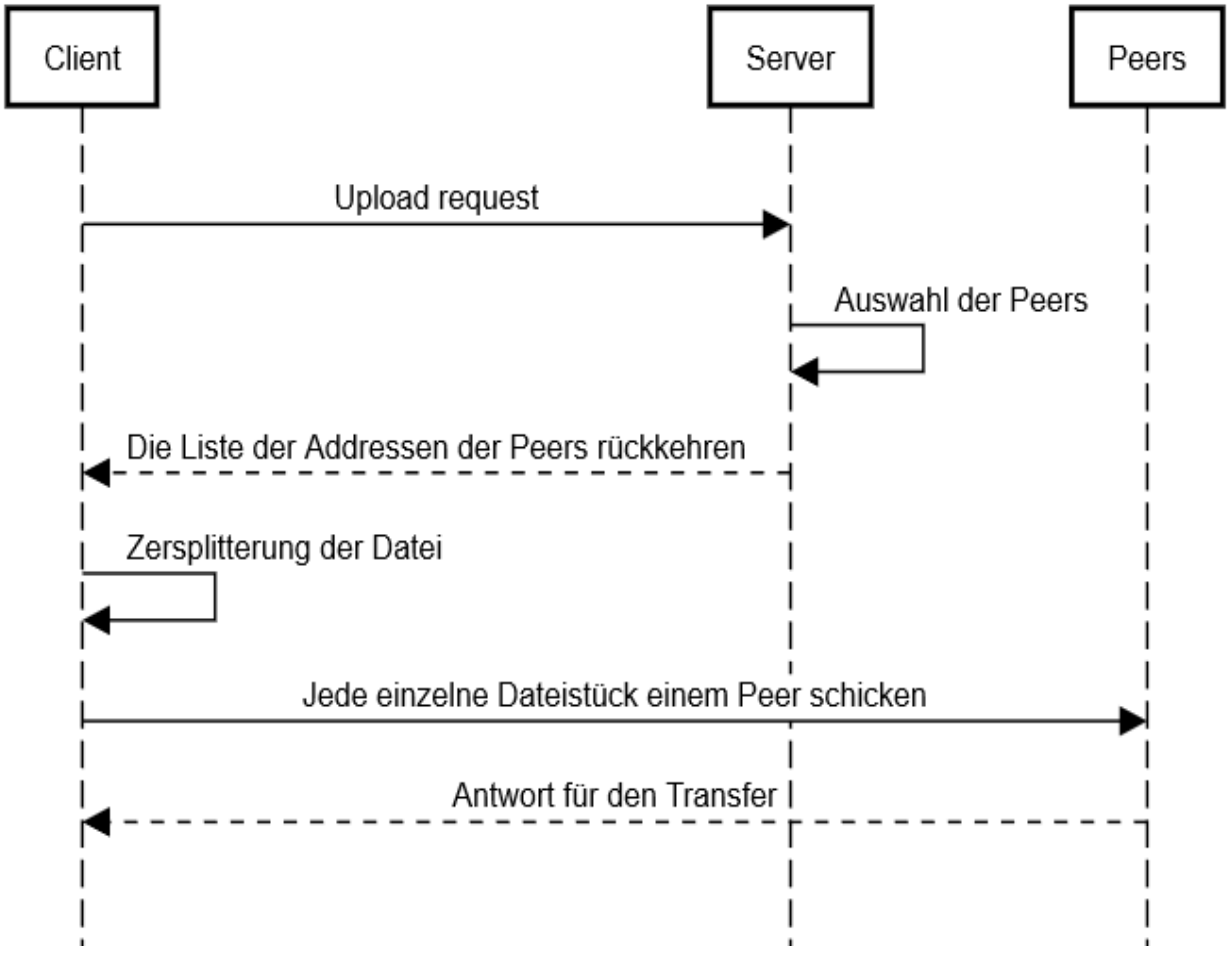

Abbildung 11 - Sequenzdiagramm für einen Upload

In Abbildung 11 kann man die Reihenfolge der Operationen für einen Upload sehen. Der Klient sendet eine Anfrage dem Server. Der Server wählt die besten Peers für diesen Transfer und kehrt eine Liste mit Verbindungsinformationen über diesen zurück. Der Klient wird die Datei in kleineren Stücken teilen und wird diese Teile jedem einzelnen bekommenen Peer schicken. Dafür wird er eine Socketverbindung mit dem Peer öffnen. Als Antwort wird der Peer einen entsprechenden Nachricht dem Klienten schicken. Der Prozess für eine Downloadoperation ist ähnlich.

Die Kodifizierung der Nachrichte wird von einem Nachrichtesystem realisiert. Diese Nachrichte beinhalten spezifische Informationen für die angeforderte Operation und die Verbindungsinformationen des Absenders.

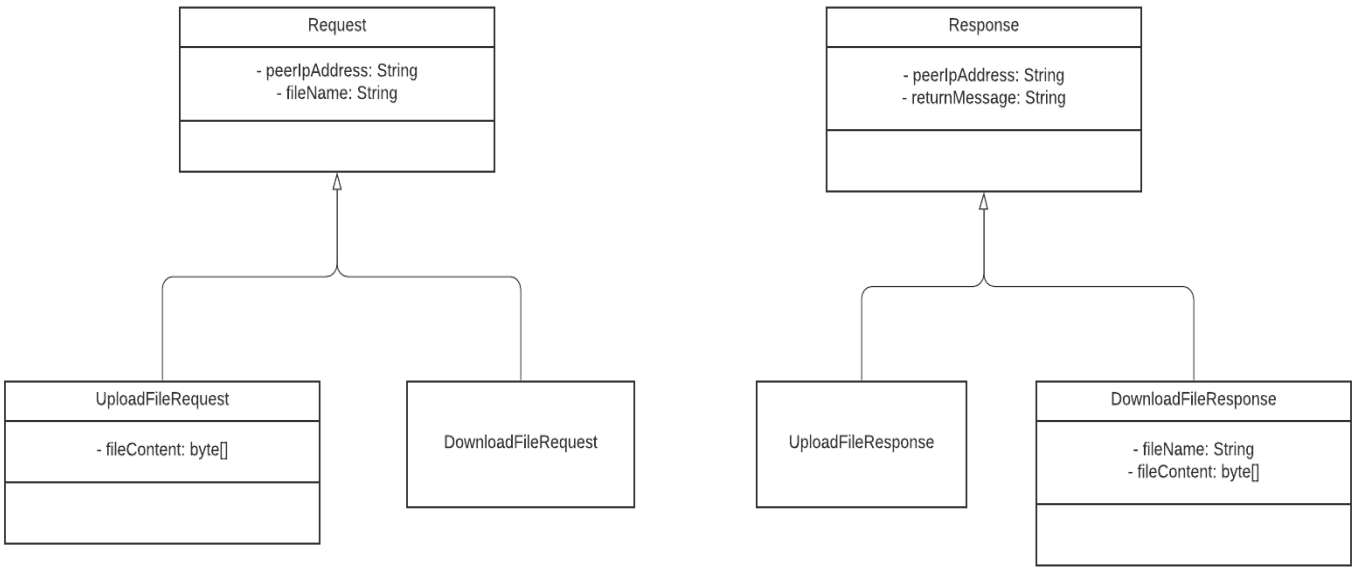

Abbildung 12 - Hierarchie der Nachrichte

Im Falle einer Uploadoperation wird der Klient den Namen der Datei auch speichern, um sie später herunterladen zu können. Die Namen der gesendeten Dateien werden in einer speziellen Datei von der Klientseite gespeichert.

GUI

Die grafische Schnittstelle wird mit Hilfe von JavaFX implementiert. Diese ist ziemlich einfach und stellt die Operationen, die ein Klient durchführen kann dar. Der Schwerpunkt der Anwendung ist nicht die grafische Schnittstelle, so dass sie nicht sehr komplexe oder kunstvolle Elemente hat.

Der Benutzer kann sich in der Anwendung einloggen, oder falls er kein Konto hat, kann er sich registrieren.

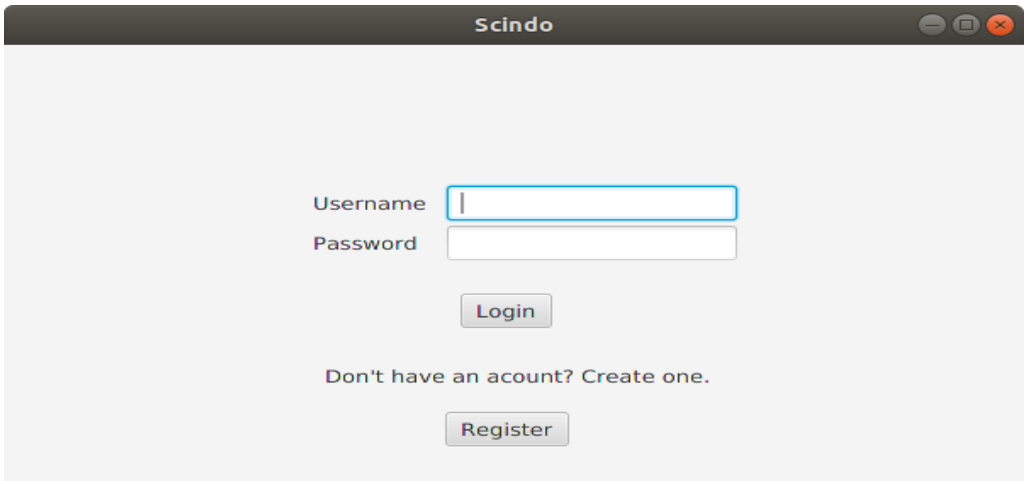

Abbildung 13 - Login Bildschirm

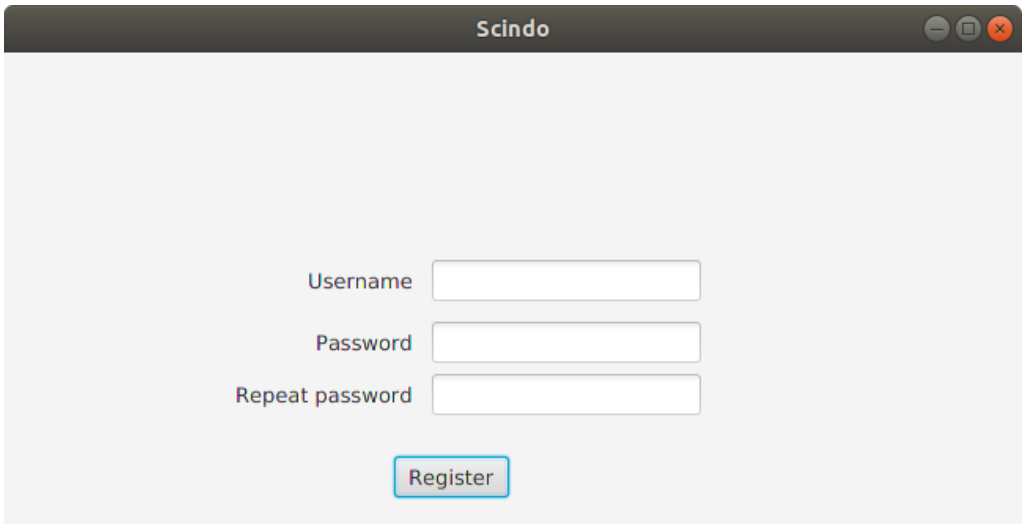

Abbildung 14 - Registrierungs Bildschirm

Nach dem Einloggen wird das Uploadbildschirm sich öffnen. Hier kann der Benutzer eine Datei hochladen. Für die Exemplifizierung der Anwendung, habe ich hier einen TableView hinzugefügt, die den Namen und die Adresse der gefundenen Peers für diesen Transfer beinhaltet. Aus diesem Bildschirm kann der Benutzer das Downloadbildschirm öffnen.

Die *Select file* Taste wird ein Fenster für die Auswahl der Datei öffnen.

Nach dem Druck der Upload Taste wird sich ein Informierungsfenster öffnen, das einen Erfolg- oder Fehlernachricht beinhaltet.

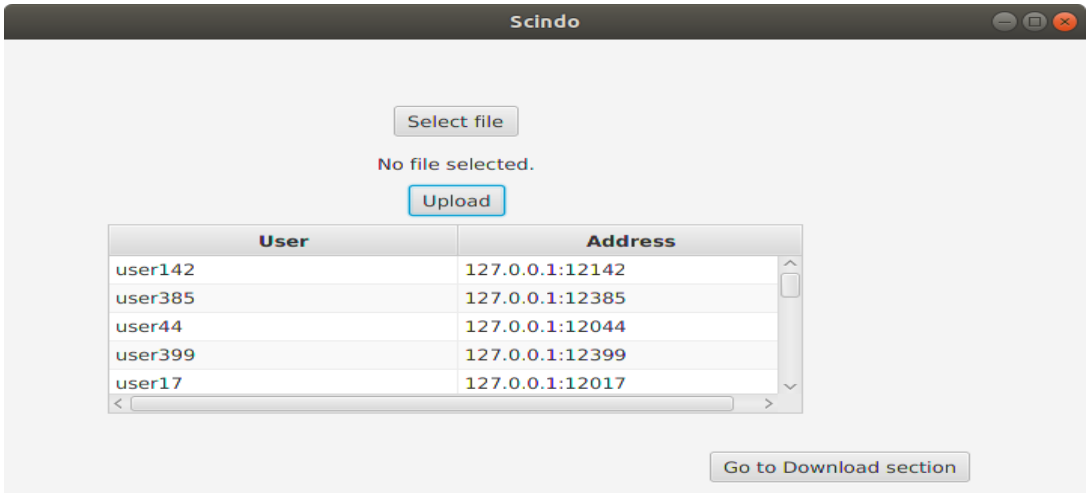

Abbildung 15 - Upload Bildschirm

In dem Downloadbildschirm kann der Benutzer seine hochgeladenen Dateien sehen. Er kann sie von dem ListView von der linken Seite selektieren und sie herunterladen. Der Benutzer kann von diesem Bildschirm zurück zu dem Uploadbildschirm gehen.

Nach dem Druck der Download Taste wird sich ein Informierungsfenster öffnen, das einen Erfolg- oder Fehlernachricht beinhaltet.

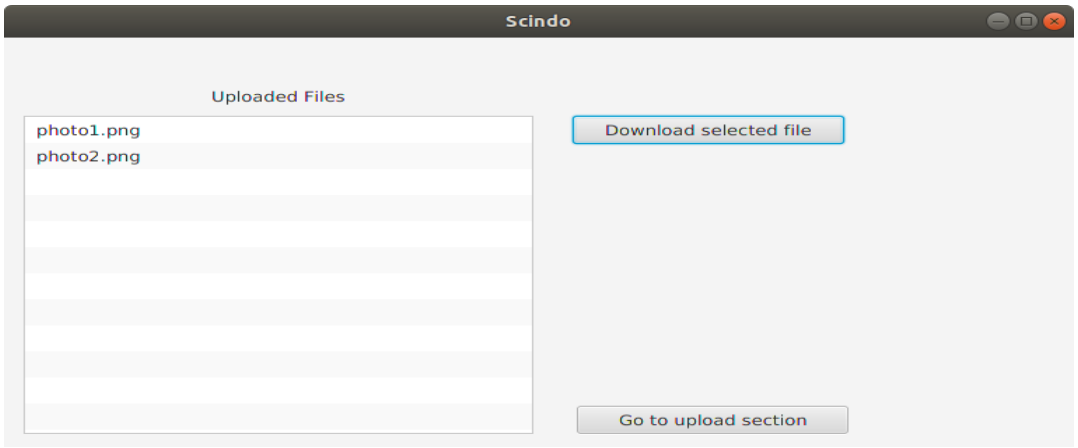

Abbildung 16 - Download Bildschirm

Auswertung

Beschreibung des Testens und der Simulation

Als ich in der Kapitel mit dem benutzten Technologien gesagt habe, wurde das Testen der Anwendung mittels einer Python-Anwendung durchgeführt. Das Testen wurde durch die Simulation der Benutzung des Netzwerkes in der Wirklichkeit realisiert. Konkreter habe ich Anfragen dem Server geschickt, um die Interaktion der reellen Klienten zu simulieren.

Weil es sehr schwer die Netzwerk, ohne Akzess auf sehr viele Devices zu haben, zu testen, wurden die Simulationen lokal getestet. Mehr konkret wurden Login-, Logout-, Upload- und Downloadanfragen dem Server gesendet und die Ergebnisse herausgezogen und analysiert.

Dafür wurden ein paar Pythonmodule implementiert, um diesen Prozess zu vereinfachern:

- *request_handler.py*

  Dieser Modul beschäftigt sich mit der effektiven Sendung der Anfragen dem Server. Dieser kann Anfragen für Login, Logout, Upload und Download senden, aber auch für den Simulationprozess wichtige Operationen durchführen, wie zum Beispiel Logout Anfragen für alle Klienten zu senden.

  Die Anfragen sind POST oder GET Anfragen, die mit Hilfe des requests Moduls aus der standar Python Bibliothek gesendet wurden.

- *command_file_generator.py*

  Um komplette Prozesse zu simulieren braucht man einen Führungsplan. Dieser Modul beschäftigt sich mit der Generierung der Simulationszenarien. Hier werden die Operationen kodifiziert dargestellt.

```
li,user303,user303
lo,user166
d,user443
lo,user178
u,user443
```

Abbildung 17 - Operationen Beispiel

In Abbildung 17 ist ein Beispiel von ein paar Operationen, die in Weiterem simuliert werden. Durch li,user,password wird eine Loginanfrage der User user mit

dem Kennwort password kodifiziert; lo,user ist eine Logoutanfrage der User user; d,user steht für eine Downloadanfrage für user und u,user ist eine Uploadanfrage des Users user. Eine wichtige Bemerkung ist, dass für diese Simulation die Dateien nicht so wichtig sind, so dass sie zufällig ausgewählt sind. Das heißt nicht, dass keine Dateien gesendet sind, sondern dass für den Upload- und Downloadanfragen zufällige Dateien ausgewählt sind. Für den Downloadanfragen muss man aber sicher sein, dass der Benutzer diesen Datei schon hochgeladet hat. Dafür wird man ein Wörterbuch benutzt, das alle hochgeladenen Dateien eines Benutzers speichert.

- *command_file_executor.py*

  Dieser Modul wird die Daten aus einer von command_file_generator generierter Datei lesen und die entsprechende Anrufe durchführen. Dafür werden die Methoden aus request_handler benutzt.

- *plotter.py*

  Um die erhaltenen Daten leichter analysieren zu können, werden auch entsprechende Diagrammen realisiert. Dieser Modul nimmt die Daten aus der Simulation und anhand von diesen generiert entsprechende Schaubilder.

  Für diesen Modul habe ich den matplotlib.pyplot [12] Pythonstandardmodul benutzt. Dieser veröffentlicht sehr intuitive und leicht benützbare Funktionen, um Diagramme zu generieren.

- *mass_executor.py*

  Um repräsentative Ergebnisse zu erhalten, sind nur einzelne Beispiele nicht sehr relevant. Dafür muss man mehrere Simulationen durchführen und die Mittelwerte der relevanten Werte zu analysieren.

  Dieser Modul vermittelt den oben beschriebenen Prozess: er lässt mehrere Simulationen mit den selben Eingangsdaten automatisch durchführen und die Daten analysieren.

  Die Ergebnisse der Simulationen werden in einer csv-Datei gespeichert und damit werden auch die entsprechenden Diagramme hergestellt.

Die Analyse eines so komplexen Netzwerkes ist nicht leicht. Es gibt sehr viele Bedingungen, auf den man achten muss. In dieser Arbeit werde ich ein paar Basisbendingungen analysieren und die Ergebnisse präsentieren.

Ein paar der wichtigsten Kriterien dieses Netzwerkes sind: der totale zusätzliche Speicherplatz, der nötig für die Speicherung der Dateien ist (oder anders gesagt, die Anzahl der Kopien jedes Dateistückes), die Anzahl der Benutzer dieses Netzwerkes, die Zeitspannen in denen diese aktiv sind und der Prozentsatz der erfolglosen Upload- und Downloadanfragen. Der Kost, den man minimieren möchte ist den zusätzlichen

Speicherplatz, während man die erfolgreichen Upload- und Downloadanfragenprozentsatz maximieren möchte.

Um Situationen aus der Wirklichkeit zu simulieren, habe ich ein Wahrscheinlichkeitssystem implementiert, die die Fakten aus der Wirklichkeit simuliert.

Konkreter, bei der Bildung der Befehle, die durchgeführt sein sollen, jede Operation hat eine prezisse Wahrscheinlichkeit vorzukommen. Dadurch kann man eine wirkliche Situation simulieren und relevante Informationen extrahieren.

Man definiert den Tupel (p_{li}, p_{lo}, p_u, p_d) mit $p_{li} + p_{lo} + p_u + p_d = 1$ als Wahrscheinlichkeitstupel, wobei:

- p_{li} ist die Wahrscheinlichkeit eine Loginanfrage vorzukommen;
- p_{lo} ist die Wahrscheinlichkeit eine Logoutanfrage vorzukommen;
- p_u ist die Wahrscheinlichkeit eine Uploadanfrage vorzukommen;
- p_d ist die Wahrscheinlichkeit eine Downloadanfrage vorzukommen.

Mit der Hilfe eines solchen Wahrscheinlichkeitstupel kann man Operationendateien generieren (durch den command_file_generator Modul). Ein solcher Datei beinhaltet für diese Simulationen 500 Operationen, die wie vorher erklärt, generiert werden.

Ein paar Bemerkungen über diese Wahrscheinlichkeiten:

- Die Logoutanfragen sind die schädlichsten. Das Netzwerk wurde 100% funktionieren, falls alle Benutzer immer aktiv wären (das ist aber unmöglich). Darum sind die Logoutanfragen schlecht, weil sie die Möglichkeit ein paar Dateistücke von hier zu unterladen zerstört. Das heißt, dass für das schlechteste Szenario, in dem das Netzwerk funktioniert, die Logoutwahrscheinlichkeit so groß wie möglich sein soll.
- Es ist angemessen anzunehmen, dass die Anzahl der Login- und Logoutanfragen ungefähr gleich ist.
- Die Anzahl der Upload- und Downloadanfragen ist größer als die der Login- und Logoutanfragen. Wenn man diese Tatsache chronologisch betrachtet, heißt dass jeder Benutzer in mehreren Upload- und Downloadanfragen per Session teilnimmt, was intuitiv ist.

Weiter werde ich die Ergebnisse dieser Simulationen präsentieren. Die Simulationen stehen für verschiedene Werte des Wahrscheinlichkeitstupels und der Anzahl der Kopien, aber auch auf verschiedenen Strategien der Auswahl der Peers für jeden einzelnen Transfer. Die Strategien für die Auswahl der Teilnehmer eines Transfers sind diese zufällig auswählen oder diese aus einer ähnlichen Zeitzone zu priorisieren. In beiden Fällen sind diese mit einem kleineren schon benutzten Speicherplatz priorisiert.

Jede Strategie wird durch 20 Simulationsprozesse von je 500 Anfragen, die anhand des entsprechenden Wahrscheinlichkeitstupels generiert sind. Das System beinhaltet 500

Benutzer. Am Anfang der Simulation werden 50 Benutzer online sein (also ist das Netzwerk auf 10% aus seiner vollen Kapazität).

Ergebnisse

Zufällige Auswahl der Peers

Für je 3 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.1, 0.1, 0.4, 0.4) erhält man die folgende Ergebnisse.

Die Wahrscheinlichkeitstupel (0.1, 0.1, 0.4, 0.4) entspricht, dass für jede Logoutanfrage je 4 Upload- und Downloadanfragen gibt. Intuitiv heißt das, dass durchschnittlich, jeder Benutzer in 8 Upload- oder Downloadanfragen per Session impliziert ist.

In den folgenden Diagrammen entspricht der Simulation Index den Index der kurrenten Simulation. Das heißt, dass für jede der 20 Wiederholungen der Simulation die entsprechende Anzahl der erfolglosen Downloadanfragen dargestellt ist.

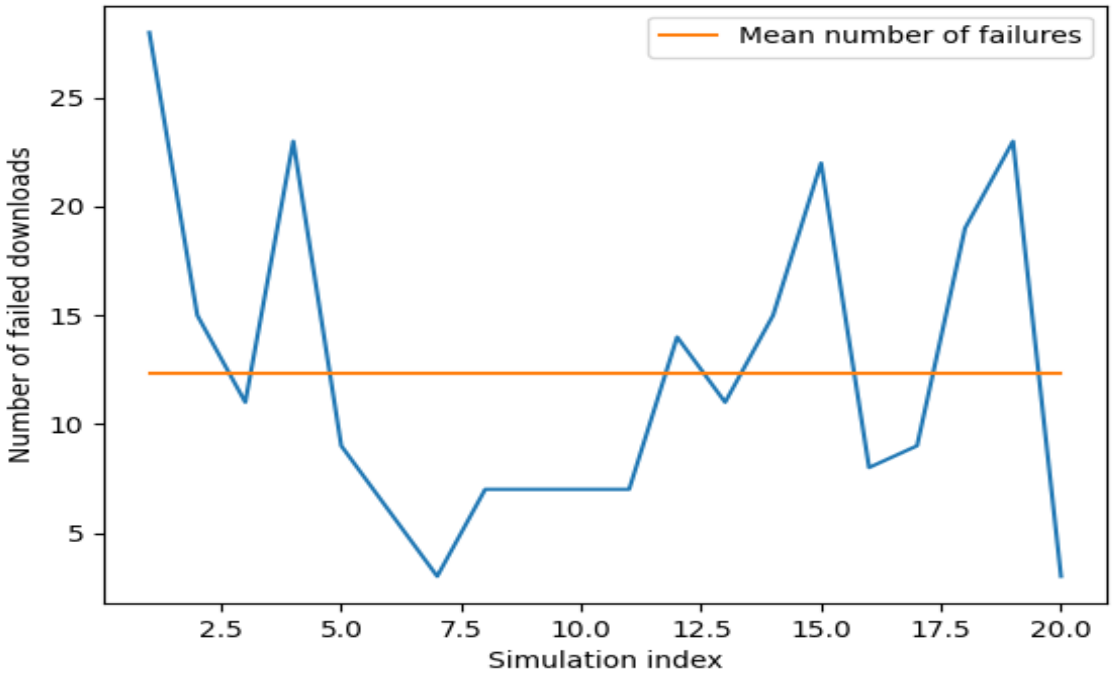

Abbildung 18 - Simulation für 3 Kopien und (0.1, 0.1, 0.4, 0.4)

Mittlere Anzahl der erfolgreichen Downloads: 193.95

Mittlere Anzahl der erfolglosen Downloads: 12.35 (~6% aus allen Downloadanfragen)

Mittlere Anzahl der Uploadanfragen: 194.9

Für je 3 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.05, 0.05, 0.45, 0.45) erhält man die folgende Ergebnisse.

Die Wahrscheinlichkeitstupel (0.05, 0.05, 0.45, 0.45) entspricht, dass für jede Logoutanfrage je 9 Upload- und Downloadanfragen gibt. Intuitiv heißt das, dass durchschnittlich, jeder Benutzer in 18 Upload- oder Downloadanfragen per Session impliziert ist.

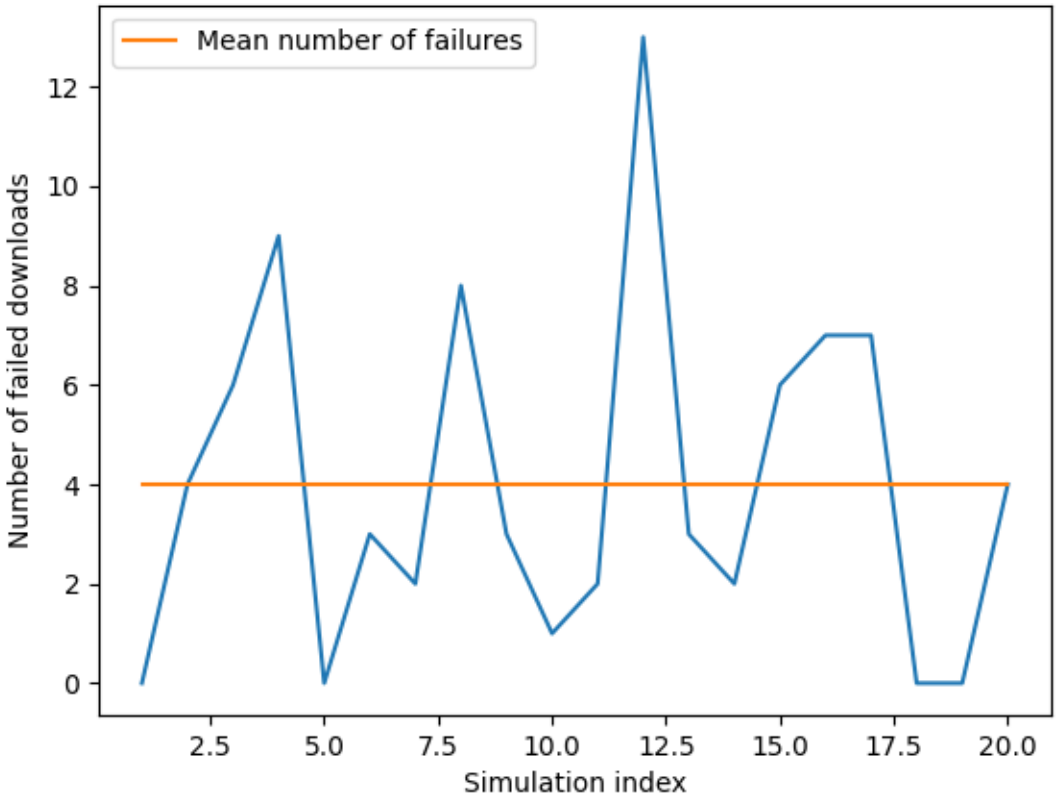

Abbildung 19 - Simulation für 3 Kopien und (0.05, 0.05, 0.45, 0.45)

Mittlere Anzahl der erfolgreichen Downloads: 220.75

Mittlere Anzahl der erfolglosen Downloads: 4 (~1.8% aus allen Downloadanfragen)

Mittlere Anzahl der Uploadanfragen: 224.9

Für je 4 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.1, 0.1, 0.4, 0.4) erhält man die folgende Ergebnisse.

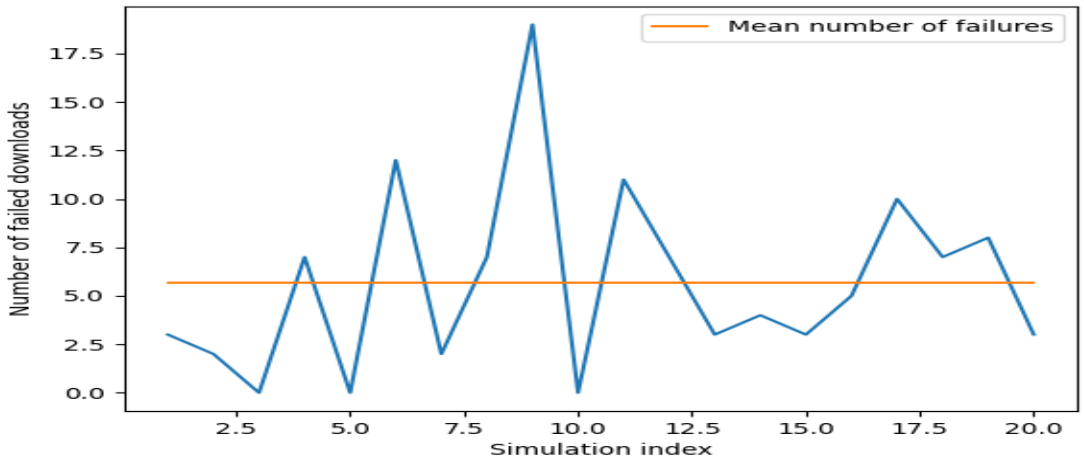

Abbildung 20 - Simulation für 4 Kopien und (0.1, 0.1, 0.4, 0.4)

Mittlere Anzahl der erfolgreichen Downloads: 194.6

Mittlere Anzahl der erfolglosen Downloads: 5.65 (~3% aus allen Downloadanfragen)

Mittlere Anzahl der Uploadanfragen: 199.65

Für je 4 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.1, 0.1, 0.4, 0.4) erhält man die folgende Ergebnisse.

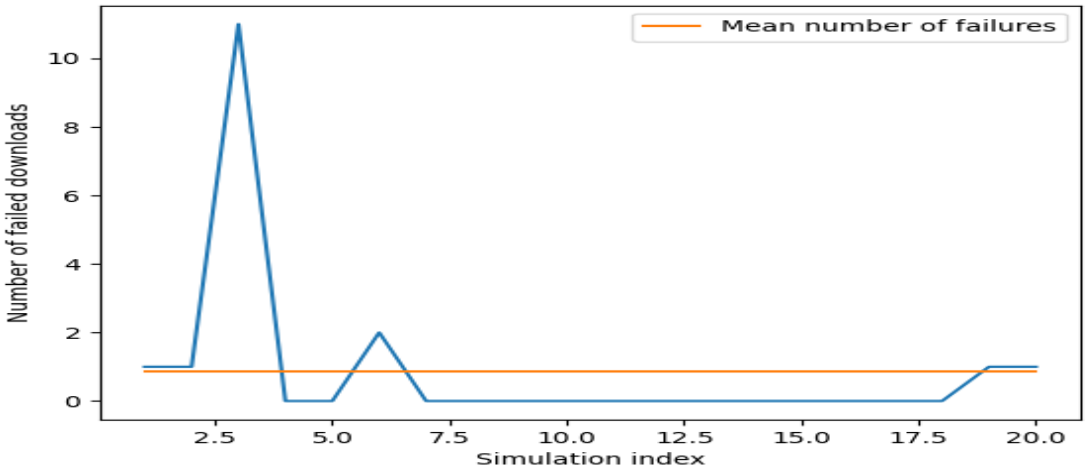

Abbildung 21 - Simulation für 4 Kopien und (0.05, 0.05, 0.45, 0.45)

Mittlere Anzahl der erfolgreichen Downloads: 221.35

Mittlere Anzahl der erfolglosen Downloads: 0.85 (~0.3% aus allen Downloadanfragen)

Mittlere Anzahl der Uploadanfragen: 227.85

Für je 5 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.1, 0.1, 0.4, 0.4) erhält man die folgende Ergebnisse.

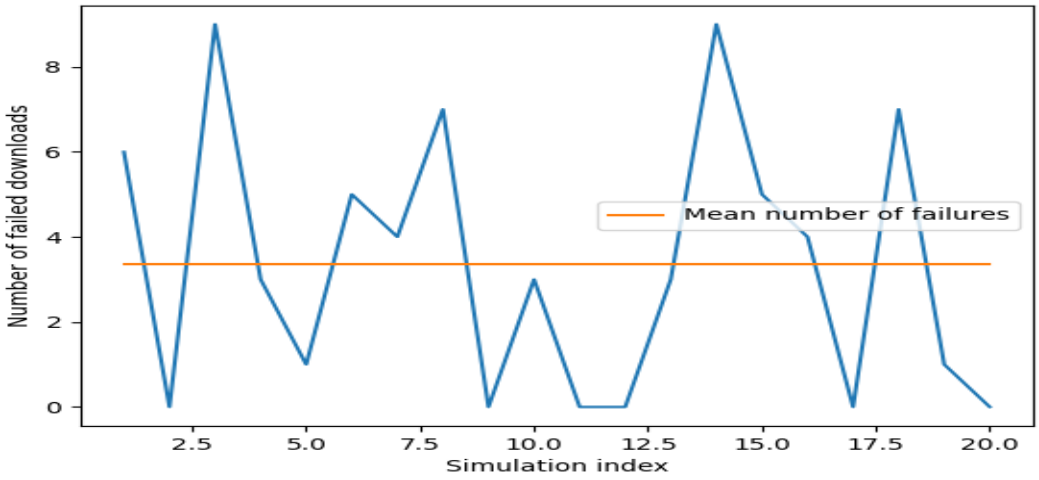

Abbildung 22 - Simulation für 5 Kopien und (0.1, 0.1, 0.4, 0.4)

Mittlere Anzahl der erfolgreichen Downloads: 198.7

Mittlere Anzahl der erfolglosen Downloads: 3.35 (~1.65% aus allen Downloadanfragen)

Mittlere Anzahl der Uploadanfragen: 198.7

Für je 5 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.1, 0.1, 0.4, 0.4) erhält man die folgende Ergebnisse.

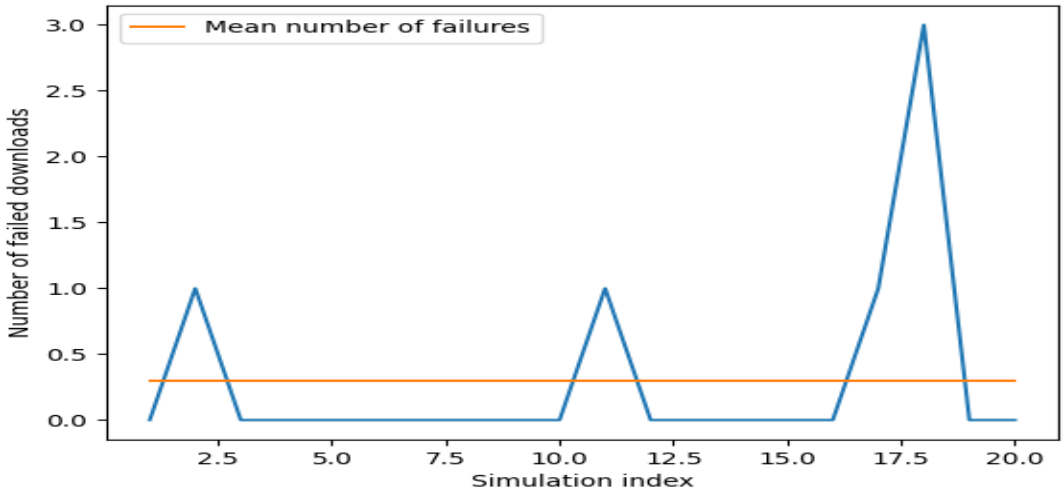

Abbildung 23 - Simulation für 5 Kopien und (0.05, 0.05, 0.45, 0.45)

Mittlere Anzahl der erfolgreichen Downloads: 226.35

Mittlere Anzahl der erfolglosen Downloads: 0.3 (~0.13% aus allen Downloadanfragen)

Mittlere Anzahl der Uploadanfragen: 225.85

Priorisierung der Auswahl der Peers aus der selben Zeitzone

Um diese Simulierung zu realisieren, musste man ein paar Modifikationen der Befehlgenerierungsskript machen. Die Zeitzonen sind mit Zahlen zwischen -11 und 11 dargestellt. Darum kann man dieses Intervall zu den Generierungsintervall (0-500) mapieren. Mit der Laufe der Zeit werden Loginanfragen von Benutzern aus der kurrenten Zeitzone und Logoutanfragen von Benutzern aus den entfernten Zeitzonen mehr wahrscheinlich, und umgekehrt.

Auf dem Serverside wird die Zeitzone entsprechende Paaarungstrategie benutzt.

Für je 3 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.1, 0.1, 0.4, 0.4) erhält man die folgende Ergebnisse.

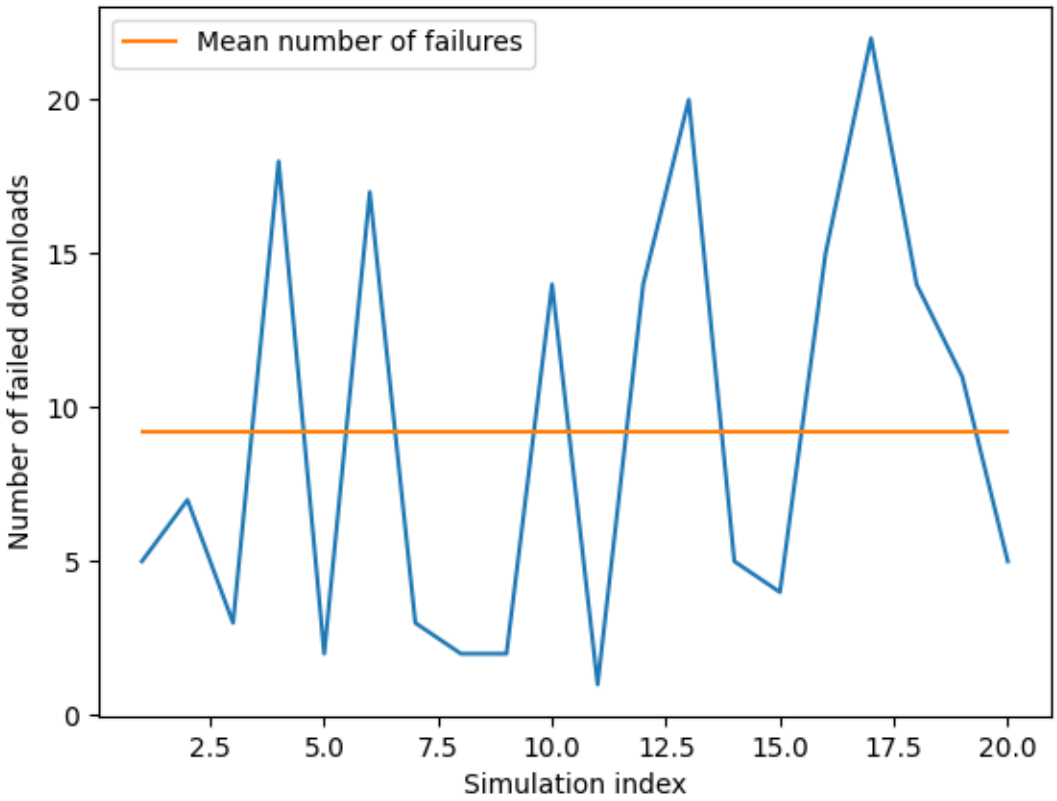

Abbildung 24 - Simulation für 3 Kopien und (0.1, 0.1, 0.4, 0.4) mit Zeitzonepriorisierung

Mittlere Anzahl der erfolgreichen Downloads: 195.35

Mittlere Anzahl der erfolglosen Downloads: 9.2 (~4.5% aus allen Downloadanfragen und mit ~25% besser als die selbe Methode, ohne Zeitzonepriorisierung)

Mittlere Anzahl der Uploadanfragen: 195.6

Für je 3 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.05, 0.05, 0.45, 0.45) erhält man die folgende Ergebnisse.

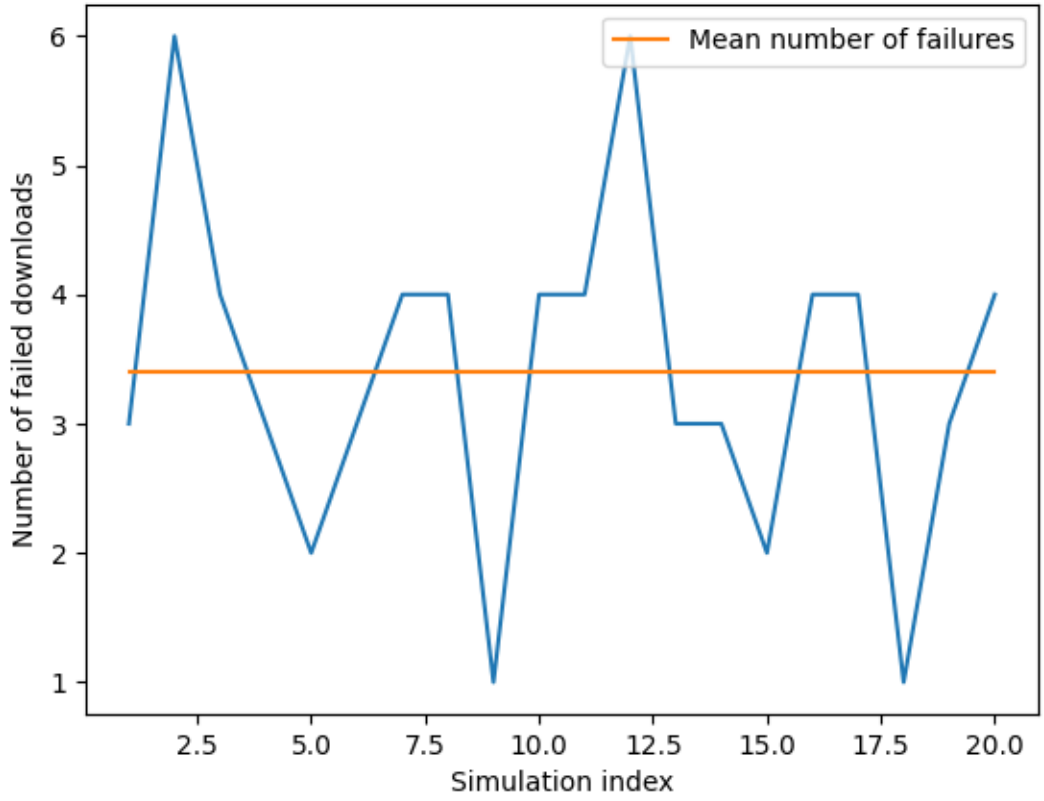

Abbildung 25 - Simulation für 3 Kopien und (0.05, 0.05, 0.45, 0.45) mit Zeitzonepriorisierung

Mittlere Anzahl der erfolgreichen Downloads: 221.6

Mittlere Anzahl der erfolglosen Downloads: 3.4 (~1.5% aus allen Downloadanfragen und mit ~16% besser als die selbe Methode, ohne Zeitzonepriorisierung)

Mittlere Anzahl der Uploadanfragen: 225.9

Für je 4 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.1, 0.1, 0.4, 0.4) erhält man die folgende Ergebnisse.

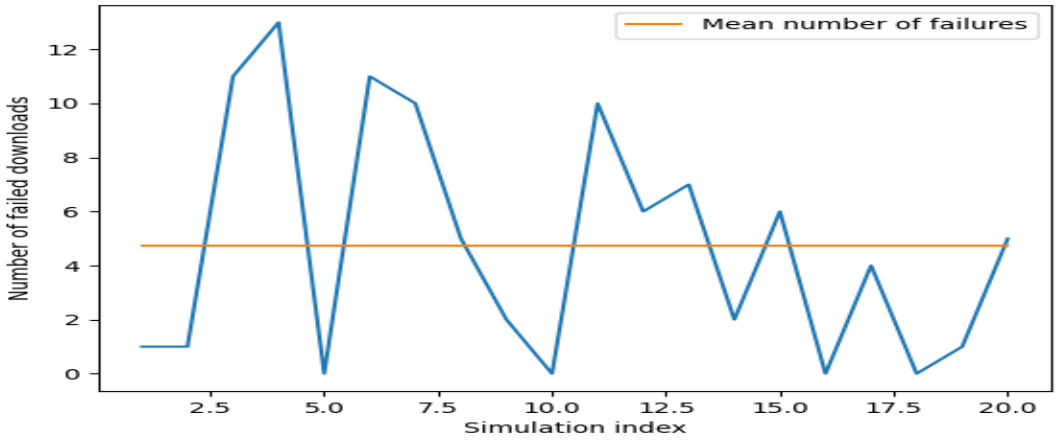

Abbildung 26 - Simulation für 4 Kopien und (0.1, 0.1, 0.4, 0.4) mit Zeitzonepriorisierung

Mittlere Anzahl der erfolgreichen Downloads: 193.9

Mittlere Anzahl der erfolglosen Downloads: 4.75 (~1.8% aus allen Downloadanfragen und mit ~16% besser als die selbe Methode, ohne Zeitzonepriorisierung)

Mittlere Anzahl der Uploadanfragen: 193.4

Für je 4 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.05, 0.05, 0.45, 0.45) erhält man die folgende Ergebnisse.

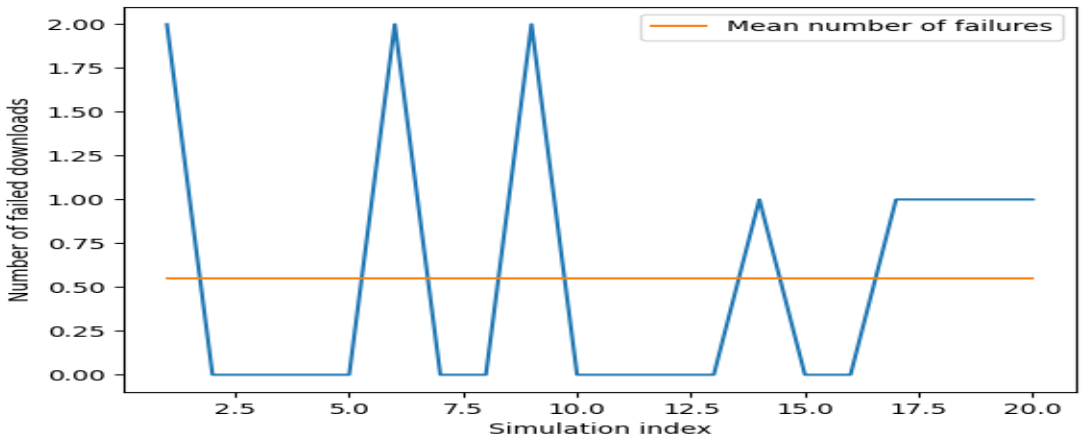

Abbildung 27 - Simulation für 4 Kopien und (0.05, 0.05, 0.45, 0.45) mit Zeitzonepriorisierung

Mittlere Anzahl der erfolgreichen Downloads: 223.1

Mittlere Anzahl der erfolglosen Downloads: 0.55 (~0.24% aus allen Downloadanfragen und mit ~20% besser als die selbe Methode, ohne Zeitzonepriorisierung)

Mittlere Anzahl der Uploadanfragen: 228.75

Für je 5 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.1, 0.1, 0.4, 0.4) erhält man die folgende Ergebnisse.

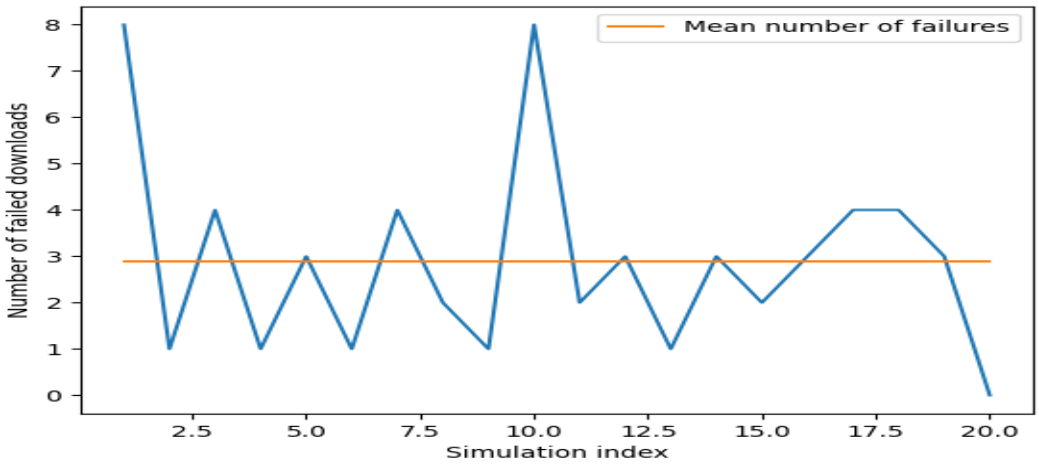

Abbildung 28 - Simulation für 5 Kopien und (0.1, 0.1, 0.4, 0.4) mit Zeitzonepriorisierung

Mittlere Anzahl der erfolgreichen Downloads: 198.35

Mittlere Anzahl der erfolglosen Downloads: 2.9 (~1.43% aus allen Downloadanfragen und mit ~13% besser als die selbe Methode, ohne Zeitzonepriorisierung)

Mittlere Anzahl der Uploadanfragen: 201.2

Für je 5 Kopien jedes Dateistücks, mit einem Wahrscheinlichkeitstupel von (0.05, 0.05, 0.45, 0.45) erhält man die folgende Ergebnisse.

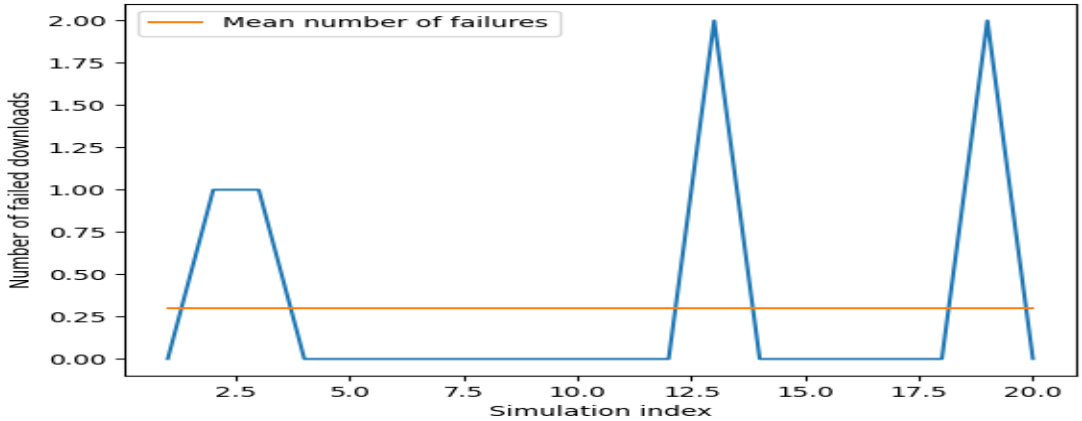

Abbildung 29 - Simulation für 5 Kopien und (0.05, 0.05, 0.45, 0.45) mit Zeitzonepriorisierung

Mittlere Anzahl der erfolgreichen Downloads: 219.9

Mittlere Anzahl der erfolglosen Downloads: 0.3 (~0.13% aus allen Downloadanfragen, der gleiche Wert wie die selbe Methode, ohne Zeitzonepriorisierung)

Mittlere Anzahl der Uploadanfragen: 201.2

Analyse und Schlussfolgerungen

Aus den vorigen Simulationen kann man ein paar wichtige Schlussfolgerungen herausziehen. Bevor diese vorzustellen und erklären, möchte ich noch einmal erinnern, dass ein solches Netzwerk sehr schwer perfekt funktional machen kann, weil sie von seinen Benutzer abhängig ist. Sie funktioniert 100%-ig nur in der idealen Situation, in der (fast) alle Benutzer online sind oder falls jedes Dateistück auf jedem einzelnen Device gespeichert wird. Diese sind aber praktisch unmögliche oder unbequeme Situationen.

Man soll die Leistung des Netzwerkes durch den Prozent der erfolgreichen Transferen, die von diesem Netzwerk unterstützt sind. In Allgemeinem, eine Leistung, die in über 99% der Fälle funktioniert ist eine ziemlich gute Leistung. Mehr als das, die in dieser Arbeit vorgestellte Methode stellt nur ein paar vorläufige Ergebnisse dar und kann sicher durch andere Strategien auch weiter verbessert sein.

Die Wahrscheinlichkeitstupeln, die für die Simulationen verwendet wurden sind Werte für eine mittlere Benutzung des Netzwerkes. Sicherlich kann es vorkommen, dass in der Wirklichkeit auch schwerere Situationen vorkommen, wenn wenige online Benutzer gibt und viele Upload- oder Downloadanrufe vorkommen. Diese Wahrscheinlichkeitstupeln beschreiben ,,normale" Situationen, die bei den oberen und unteren Grenzen einer mittleren Situation stehen.

In dieser Arbeit habe ich eine zufällige Methode für die Auswahl der Peers für jeden Transfer implementiert, um den Konzept zu prüfen. Dadurch habe ich bewiesen, dass ein solches Netzwerk funktionell und realisierbar ist. Auf der anderen Seite, habe ich auch eine mehr effizientere Methode, die sich auf die Zeitzonen der Benutzer verlässt, um die Performanz des Netzwerkes zu verbessern.

In den folgenden Abbildungen kann man eine Zusammenfassung der Ergebnisse sehen. Die zweite Methode bringt eine durchschnittliche Verbesserung von etwa 10-15% als die zufällige Methode.

In Abbildung 30 kann man den Prozent der erfolglosen Downloadanfragen aus den totalen Downloadanfragen. 3.1, 4.1, 5.1 sind die Ergebnisse des Netzwerkes für die zufällige Auswahl, für je 3, 4, respektiv 5 Kopien jedes Datastücks, während 3.2, 4.2, 5.2 die Ergebnisse des Netzwerkes für die Strategie mit Zeitzonebetrachtung.

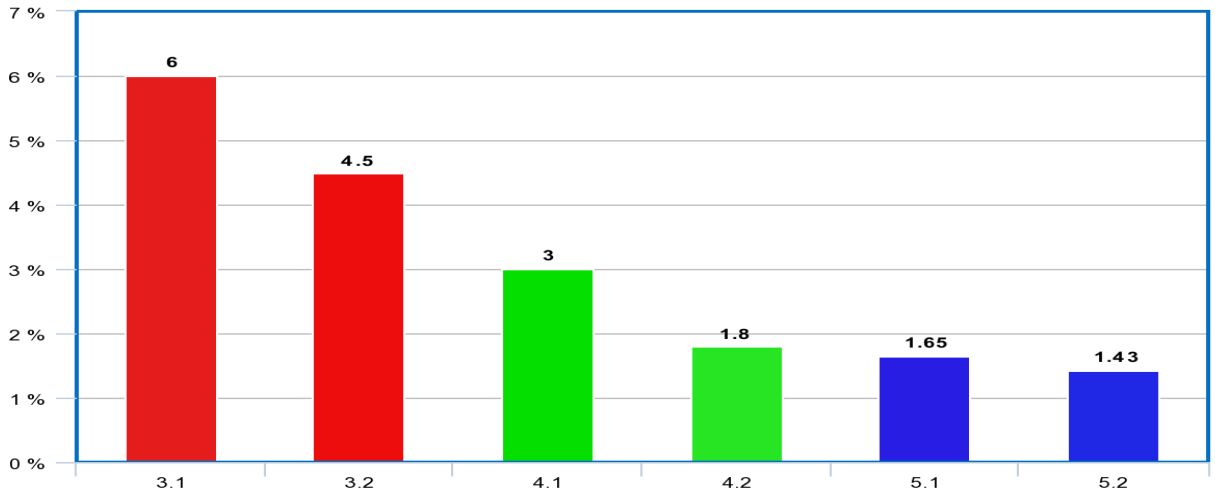

Abbildung 30 - Ergebnisse für (0.1, 0.1, 0.4, 0.4)

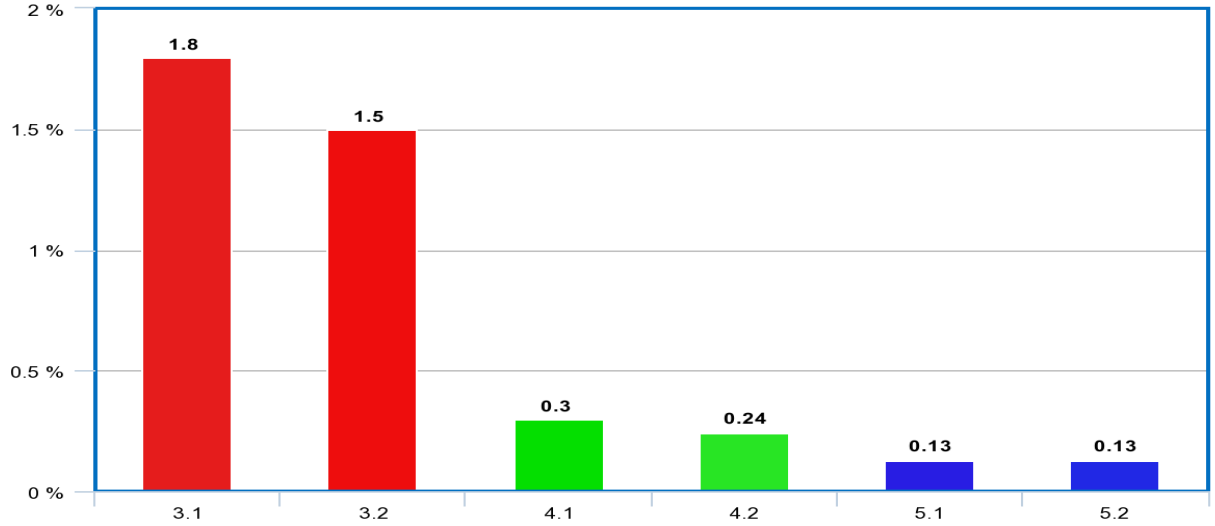

Abbildung 31 - Ergebnisse für (0.05, 0.05, 0.45, 0.45)

Die Notationen aus der Abbildung 31 sind die selben mit denen von Abbildung 30.

Aus diesen Ergebnissen kann man schlussfolgern, dass mit 4 oder 5 Kopien jedes Dateistücks, dieses Netzwerk ziemlich gute Ergebnisse erhält. Trotzdem aber ist der totale Speicherungsplatz immer größer.

Man muss aber merken, dass je größer das Netzwerk wird desto bessere Ergebnisse wird diese erhalten. Mehr als das kann man auch andere Optimierungssysteme in dem Netzwerk implementieren, um diese zu verbessern. Ein paar Möglichkeiten für die zukünftige Arbeit sind:

- Backup Mechanismen implementieren. Falls ein Benutzer für eine lange Zeit inaktiv ist, soll man die Daten, die auf seinem Device gespeichert sind auf dem

Device eines anderen Benutzers kopieren. Damit versichert man eine bessere Chance um die anfängliche Datei für eine längere Zeit zusammensammeln.

- Soziale Analyse der Benutzer. Man kann das Verhalten der Benutzer analysieren, um ihre bevorzugten online Zeitintervalle zu vorhersehen. Damit kann man die Transfers besser realisieren, um die Downloadmöglichkeit zu vergrößern.
- Implementierung einer personalen Benutzergruppe. Dadurch können die Benutzer ihr eigenes Netzwerk, mit ihren Freunden oder Familienmitglieder, definieren. Damit verbessert man auch die Sicherheit der Datenspeicherung (diese werden die anderen Mitglieder vertrauen), aber auch die Performanz des Netzwerkes für diesen (weil sie wahrscheinlicherweise ähnliche online-Verhalten haben).

Bibliographie

[1] „Dust,“ [Online]. Available: https://usedust.com/.

[2] „Sense Chat,“ [Online]. Available: https://www.sense.chat/.

[3] K. Khacef und G. Pujolle, „Secure Peer-to-Peer communication based on Blockchain,“ Matsue, Japan, 2019.

[4] M. Petkovic und W. Jonker, Security, Privacy and Trust in Modern Data Management, New York: Springer, 2007.

[5] A. Osseiran, J. F. Monserrat und P. Marsch, 5G Mobile and Wireless Communications Technology, Cambridge University Press, 2016.

[6] „TIOBE,“ TIOBE, [Online]. Available: https://www.tiobe.com/.

[7] [Online]. Available: https://www.tiobe.com/tiobe-index/.

[8] [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2019/java/.

[9] „MVN Repository,“ Apache Software Foundation, [Online]. Available: https://mvnrepository.com/.

[10] [Online]. Available: http://www.datawtech.com/evolution-of-python/.

[11] „Python requests,“ [Online]. Available: https://pypi.org/project/requests/.

[12] „Pyplot,“ [Online]. Available: https://matplotlib.org/api/pyplot_api.html.